Universidade Federal de Goiás
Regional Catalão
Unidade Acadêmica Especial de Biotecnologia
Curso de Bacharelado em Ciências da Computação

Análise de operadores de cruzamento genético aplicados ao problema do Caixeiro Viajante

Samuel Wanberg Lourenço Nery

Catalão – GO
2017

Samuel Wanberg Lourenço Nery

Análise de operadores de cruzamento genético aplicados ao problema do Caixeiro Viajante

Monografia apresentada ao Curso de Bacharelado em Ciências da Computação da Universidade Federal de Goiás – Regional Catalão, como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação. *VERSÃO REVISADA*

Orientadora: Prof. Dr. Sérgio Francisco da Silva

Catalão – GO
2017

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Nery, Samuel Wanberg Lourenço
Análise de operadores de cruzamento genético aplicados ao problema do Caixeiro Viajante [manuscrito] / Samuel Wanberg Lourenço Nery. – 2017.
83 p.: il.

Orientadora: Prof. Dr. Sérgio Francisco da Silva
Monografia (Graduação) – Universidade Federal de Goiás, Unidade Acadêmica Especial de Biotecnologia, Ciências da Computação, 2017.
Bibliografia.

1. Algoritmos Genéticos. 2. Operadores de Cruzamento. 3. Problema do Caixeira Viajante. I. Silva, Sérgio Francisco da, orient. II. Título.

CDU 004

Samuel Wanberg Lourenço Nery

Análise de operadores de cruzamento genético aplicados ao problema do Caixeiro Viajante

Monografia apresentada ao curso de Bacharelado em Ciências da Computação da Universidade Federal de Goiás – Regional Catalão.

Trabalho aprovado em 05 de Abril de 2017.

Sérgio Francisco da Silva
Orientadora

Núbia Rosa da Silva
UFG/IBiotec

Tércio Alberto dos Santos
UFG/IBiotec

Catalão – GO
2017

Este trabalho é dedicado a Deus, minha família, amigos e professores.

“Combati o bom combate,
acabei a carreira, guardei a fé.”
(Timóteo 4:7)

RESUMO

NERY, S. W. L.. **Análise de operadores de cruzamento genético aplicados ao problema do Caixeiro Viajante**. 2017. 83 p. Monografia (Graduação) – Unidade Acadêmica Especial de Biotecnologia, Universidade Federal de Goiás – Regional Catalão, Catalão – GO.

O problema do Caixeiro Viajante é um clássico problema de otimização combinatória com grande interesse acadêmico, atraindo pesquisadores de vários campos de pesquisas. Formalmente, o problema pode ser descrito como um ciclo hamiltoniano em um grafo completo que represente a menor rota a ser percorrida por um caixeiro (vendedor). Em termos gerais, o problema consiste em determinar um percurso de menor distância para percorrer um conjunto de cidades, sem que haja cidades repetidas no caminho traçado e retornando sempre à cidade de origem. A inspiração natural para formulação do problema é, projetar um plano de vendas no qual um vendedor entregue todos os produtos em diferentes localidades (cidades) e que o custo do transporte seja mínimo. Devido sua simplicidade de enunciação e aplicabilidade em diversas situações do mundo real, o problema tem sido utilizado como uma base de prova importante para o desenvolvimento de novas ideias ou métodos computacionais, resultando em avanços relevantes em pesquisas relacionadas a algoritmos de otimização.

Este trabalho é o resultado de um estudo de revisão da literatura sobre o desenvolvimento de algoritmos genéticos como um modelo de otimização para ser aplicado ao problema do caixeiro viajante. Algoritmos genéticos são algoritmos bioinspirados de otimização e busca, que combina conceitos de busca cega e busca aleatória para encontrar soluções de alta qualidade para problemas difíceis. Em nossa análise, iremos priorizar o estudo de operadores de cruzamento genéticos, projetados especificamente com o proposito de serem aplicados ao problema do caixeiro viajante. Dentro das perspectivas teórica dos algoritmos genéticos, o operador de cruzamento é o operador responsável pela construções de novas soluções candidatas, por meio de recombinação gênica entre um par de soluções codificadas previamente estabelecidas. Então, apresentamos um estudo de 7 operadores de cruzamento apontados pela literatura especializada como operadores clássicos para o problema do caixeiro viajante, discutindo o modo de representação e codificação das soluções candidatas de cada um desses operadores. Os operadores estudados foram implementados e submetidos ao processo avaliação em 20 instâncias de testes para o problema do caixeiro viajante, selecionadas da base de testes TSPLIB. No final os resultados obtidos são demonstrados e os melhores operadores de cruzamento genético aplicado ao Problema do Caixeiro Viajante são enunciados.

Palavras-chave: Algoritmos Genéticos, Operadores de Cruzamento, Problema do Caixeira Viajante.

ABSTRACT

NERY, S. W. L.. **Análise de operadores de cruzamento genético aplicados ao problema do Caixeiro Viajante**. 2017. 83 p. Monografia (Graduação) – Unidade Acadêmica Especial de Biotecnologia, Universidade Federal de Goiás – Regional Catalão, Catalão – GO.

The Travelling Salesman Person (TSP) is a classic problem of combinatorial optimization with great academic interest, attracting researchers from various fields of research. Formally, the problem can be described as a Hamiltonian cycle in a complete graph representing the smallest route to be traveled by a clerk (seller). In general terms, the problem is to determine a course of lesser distance to travel through a set of cities, without there being repeated cities in the way traced and returning always to the city of origin. The natural inspiration for formulating the problem is to design a sales plan in which a seller delivers all products in different locations (cities) and the cost of transport is minimal. Due to its simplicity of enunciation and applicability in several real world situations, the problem has been used as an important proof base for the development of new ideas or computational methods, resulting in relevant advances in research related to optimization algorithms.

This paper is the result of a review of the literature on the development of genetic algorithms as an optimization model to be applied to the problem of the traveling salesman. Genetic algorithms are bioinspired optimization and search algorithms that combine blind search and random search concepts to find high quality solutions to difficult problems. In our analysis, we will prioritize the study of genetic crossing operators, designed specifically for the purpose of being applied to the problem of the traveling salesman. Within the theoretical perspectives of genetic algorithms, the crossover operator is the operator responsible for constructing new candidate solutions, through gene recombination between a pair of previously established coded solutions. Then, we present a study of 7 intersection operators pointed out by the specialized literature as classic operators for the traveling salesman problem, discussing the mode of representation and coding of the candidate solutions of each of these operators. The operators studied were implemented and submitted to the evaluation process in 20 instances of tests for the traveling salesman problem, selected from the TSPLIB test database. In the end the results obtained are demonstrated and the best genetic crossing operators applied to the Traveler Salesman Problem are stated.

Keywords: Genetic Algorithms, Crossover Operators, Travelling Salesman Person.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ALGORITMOS

LISTA DE TABELAS

SUMÁRIO

CAPÍTULO

1

INTRODUÇÃO

Neste capítulo, apresenta-se informações gerais sobre o contexto deste trabalho acadêmico, define-se o problema a ser estudado, discute-se a técnica empregada ao problema enunciado, elabora-se os objetivos para o trabalho, e por fim, trata-se da organização deste documento.

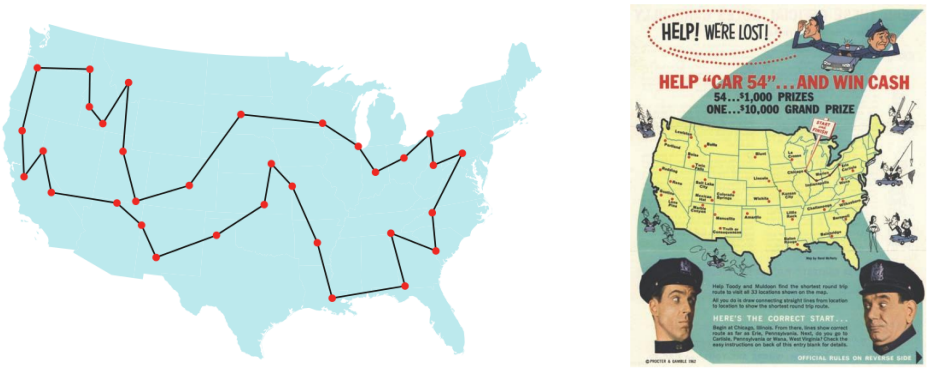
1.1 Considerações iniciais

O Problema do Caixeiro Viajante, do inglês *Travelling Salesman Person* (TSP), destaca-se por ser um dos mais relevantes problema de otimização combinatória, com um amplo campo de interesse acadêmico (LAWLER *et al.*, 1991). O enunciado do problema pode ser empregado na representação de diversas situações que demandam um complexo pensamento matemático para serem solucionadas. Como por exemplo, problemas de engenharia, logística, computação, genética, entre outros. Portanto, a busca de técnicas eficientes para resolução do problema é algo de grande interesse e valor para comunidade acadêmica em geral.

O TSP trata-se de um desafio baseado na definição de uma permutação de cidades entre um conjunto de n cidades, que represente um percurso fechado de tamanho mínimo a ser seguido por um vendedor, em seu trajeto de vendas. O vendedor precisa visitar cada cidade uma única vez, retornando sempre à cidade de origem de modo que o custo total do trajeto seja mínimo.

Como exemplo de um percurso para o TSP, na Figura 1 é apresentado o percurso ótimo para o problema histórico do TSP denominado *Car 54 poster*. O problema *Car 54 poster* foi noticiado em 1962 como um desafio para encontrar um caminho de menor distância, no qual dois oficiais de policia devem percorrer 33 cidades dos Estados Unidos, iniciando e finalizando o trajeto a partir de Chicago. Com base neste problema, foi proposto um concurso e quem encontrasse o percurso de menor comprimento receberia uma recompensa de 10 mil dólares. Os pesquisadores Robert Karg e Gerald Thompson foram os primeiros vencedores para este

Figura 1 – Representação do percurso ótimo para o problema *Car 54 poster* de 33 cidades.



(a) Percurso ótimo (b) Cartaz do problema

Fonte: Cook (2012).

problema, eles desenvolveram uma estratégia heurística denominada *hit-or-miss* que produziu a melhor solução para o problema naquele momento (COOK, 2012).

Por fim, esta pesquisa visa analisar o desempenho dos algoritmos genéticos, sobretudo de operadores de cruzamento genéticos projetados especificamente, para o problema do TSP. Acredita-se que algoritmos genéticos, seja um bom método para resolução do problema, devido a plena capacidade desses algoritmos em lidar com problemas complexos, com espaços de busca não polinomiais (HAUPT, 2004).

O restante deste capítulo é estruturado da seguinte maneira. Na Seção 1.2 é apresentado uma descrição formal do problema. A Seção 1.3 apresenta a complexidade computacional do TSP. Na Seção 1.4 são listadas algumas aplicações reais do TSP. A Seção 1.5 expõe os objetivos do trabalho. Na Seção 1.6, discute-se alguns trabalhos relacionados. A Seção 1.7 são apresentadas a estrutura do trabalho e as considerações finais sobre o capítulo.

1.2 Formulação do problema do caixeiro viajante

Um vendedor (caixeiro) necessita visitar n cidades, sendo que cada cidade só pode ser percorrida uma única vez, iniciando o seu trajeto a partir de qualquer cidade e retornando sempre ao ponto de origem. O desafio para este problema é, encontrar um percurso que satisfaça as restrições impostas e, que ao mesmo tempo, minimize a distância total do trajeto percorrido pelo vendedor.

Uma formulação matemática para o TSP, é enunciada conforme a definição 1 (HELSGAUN, 2000):

Definição 1. Dado um conjunto de n cidades e uma matriz de custos $C = (c_{ij})$, com c_{ij} representando o custo (distância) entre as cidades i e j para $i, j = (1, ..., n)$, deve-se encontrar uma

permutação (i_1,i_2,i_3,...,i_n) de cidades que minimize o custo total do trajeto [$c_{i_1 i_2} + c_{i_2 i_3} + ... + c_{i_{n-1} i_n} + c_{i_n i_1}$].

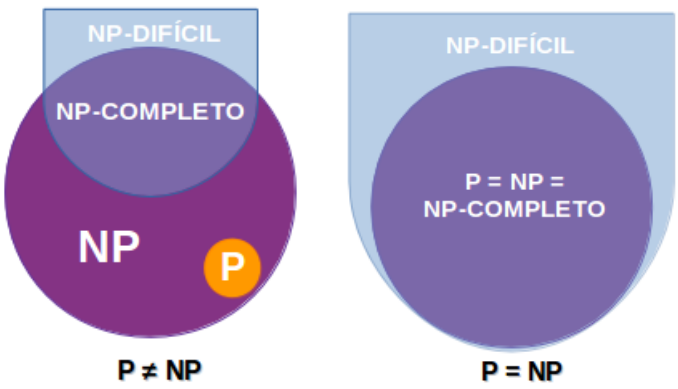
Conforme as propriedades da Definição 1, o TSP é classificado em três tipo específicos (HELSGAUN, 2000):

- Para casos em que $c_{ij} = c_{ji}$ $\forall$ (i,j), o problema é definido como TSP simétrico, caso contrário, o problema é definido como TSP assimétrico.
- Se $c_{iz} \leq (c_{ij} + c_{jz})$ $\forall$ (i,j,z), as distâncias satisfazem as condições de desigualdade de triângulos, então o problema é definido como TSP métrico.
- Se c_{ij} é definido por coordenadas euclidianas, o TSP é considerado como um problema Euclidiano, e também são extensíveis a casos simétricos e assimétricos.

1.3 Complexidade computacional

Conforme a teoria da complexidade computacional, a classe P compreende o conjunto de problemas computacionais com soluções que podem ser encontradas de forma eficiente em tempo polinomial. Enquanto que, a classe NP contém problemas computacionais que não possuem soluções imediatas em tempo polinomial, porém dada uma solução, esta solução pode ser verificadas de forma eficiente em tempo polinomial. A classe de problemas NP-Completo estão contidos os problemas considerados mais difíceis em NP. Ao mesmos instante, tais problemas são NP-Difíceis e NP. A classe NP-Difícil, é uma classe especial onde são considerados problemas tão difíceis quantos os mais difíceis contidos em NP (como exemplo Figura 2), porém, os problemas contidos nesta classe não necessariamente precisam estar em NP e nem precisam ser considerados problemas de decisão.

Figura 2 – Diagrama de Venn para representação dos problema P, NP, NP-COMPLETO e NP-DIFÍCIL.



(a) (b)

Fonte: Elaborada pelo autor.

O problema do caixeiro viajante pertence a classe de problemas NP-Difícil (LAWLER *et al.*, 1991). Considerando que P é diferente de NP (Figura 2a), logo, não é possível afirmar que existe algum algoritmo capaz de encontrar uma solução ótima para qualquer instância do problema, em tempo polinomial. Porém, caso uma solução exista, esta solução pode ser verificada em tempo polinomial.

Para encontrar o percurso de menor comprimento, é necessário avaliar os custos de todos os percursos possíveis no espaço de busca do problema. Por exemplo, para instâncias do tipo TSP simétrico com n cidades, o tamanho do espaço de busca é definido por $(n-1)! \div 2$. Se n for igual à 20, existem aproximadamente 10^{18} percursos distintos no qual um caixeiro pode percorrer (YUAN; LI, 2009).

Nota-se então, que o espaço de busca do TSP cresce de forma substancial, levando a crer que o problema em questão possui uma natureza não solucionável, mesmo para casos em que o número de cidades é relativamente pequeno. Este fato, torna difícil o desenvolvimento de bons algoritmos, capazes de prover soluções de qualidade para o problema. Considera-se que um bom algoritmo, retorne um valor ótimo para qualquer instância do problema em tempo médio n^k, sendo k um valor fixo não muito elevado (por exemplo $k = 2$ ou 3) (COOK, 2012).

Em (JOHNSON; MCGEOCH, 1997; LAPORTE, 1992) são apresentados estudos de revisão relacionando diferentes abordagens de otimização aplicadas ao TSP. Algoritmos para o TSP são classificados em dois tipos, algoritmos exatos e algoritmos (heurísticas) de aproximação.

Algoritmos exatos, são algoritmos discretos capazes de encontrar a solução exata (ótima) para um conjunto delimitado de passos, tais algoritmos são demasiadamente complexos para serem implementados (com codificações constituindo por milhares de linhas de códigos) e demandam alta configuração computacional para exercerem suas capacidades de otimização (HELSGAUN, 2000).

Contrapondo-se aos algoritmos exatos as heurísticas de aproximação são geralmente simples de serem implementadas e obtém boas soluções em tempo de execução relativamente curto quando aplicadas de forma eficiente ao problema. Embora esses algoritmos não garantem encontra o valor ótimo em todos os casos, alguns retornam soluções que diferem do valor ótimo em grau percentual mínimo (ALBAYRAK; ALLAHVERDI, 2011). Portanto se uma pequena variação em relação ao valor ótimo é aceitável, pode ser propício utilizar um algoritmo aproximado como método de resolução do problema.

1.4 Aplicações

O TSP pode ser aplicado em diversas áreas do conhecimento, atuando para solucionar problemas no mundo real. Em (LAWLER *et al.*, 1991; COOK, 2012) são listados vários problemas que podem ser modelados pelo TSP:

- **Logística e roteamento de veículos**: O problema pode ser entendido como encontrar um conjunto de rotas a serem percorridas por uma frota de veículos, a fim de efetuar entregas a um determinado número de clientes com um custo mínimo. Alguns problemas reais deste tipo de aplicação são: problema de roteamento de ônibus escolar (HASHI; HASAN; ZAMAN, 2016), auxílio no planejamento de viagens (MA, 2016; BEIRIGO; SANTOS, 2016; VARGAS; SENDALES, 2016), problemas de entrega de correspondências (LAWLER *et al.*, 1991), entre outros.

- **Sequenciamento do genoma**: O TSP é útil para obter informações precisas sobre a ordem em que marcadores aparecem no genoma a ser analisado. Em um mapa genômico, os marcadores são segmentos de DNA que aparecem exatamente uma vez no genoma. Para cada cromossomo existe uma sequência de marcadores com estimativas das distâncias entre marcadores adjacentes. Um ordenamento do genoma pode ser modelado como um TSP por um caminho hamiltoniano, percorrendo cada marcador na coleção em busca da distância mais curta entre os marcadores. Como exemplo de aplicação real temos, o uso do TSP em pesquisas agronômicas para desenvolver estratégias que melhorem a resistência à doenças ou aprimore o rendimento de culturas agrícolas (MICLAUS; PRATT; GALATI, 2012).

- **Perfuração de circuitos integrados**: Em uma placa de circuitos integrados existe uma quantidade relativamente grande de perfurações (furos), que são utilizadas para montagem de *chips* ou para interligação de circuitos em camadas. Tais perfurações são produzidas por máquinas de perfurações automáticas, que se movem em posições previamente especificadas, para realizar um furo após o outro. Neste modelo, o TSP poder ser aplicado para minimizar o tempo de locomoção total da broca de perfuração, onde as cidades correspondem as localizações dos furos na placa e o custo do percurso são estimativas do tempo gasto pelo deslocamento da broca entre os locais das perfurações. Como exemplo de aplicação, a otimização do tempo gasto no processo de inspeção de perfurações realizadas em circuitos integrados (KATAGIRI *et al.*, 2016)

- **Rotacionamento de Telescópio**: Auxiliar no processo rotação (*slewing*) de grandes telescópios, minimizando o tempo gasto no giro total para um conjunto de observações a serem implementadas. Para este modelo, as cidades são os objetos a serem visualizados e os custos do percusso total são estimativas de tempo de giro de um estado para outro (LAWLER *et al.*, 1991).

1.5 Objetivo

Devido o fato de não haver algoritmo capaz de solucionar o TSP em tempo polinomial (Seção 1.3), e os métodos exatos que existem na literatura tendem a serem complexos e de difícil codificação, exigindo um alto custo computacional (HELSGAUN, 2000), vários algoritmos de

aproximação são aplicados ao problema com o objetivo de obter soluções próximas do valor ótimo, porém com um custo de execução aceitável.

Em (JOHNSON; MCGEOCH, 1997), apresenta-se um estudo de caso, com diferentes métodos de aproximação que podem ser aplicados ao TSP. Eles são, busca tabu (GLOVER, 1989), recozimento simulado (LAARHOVEN; AARTS, 1987), redes neurais (ARBIB, 2002) e algoritmos genéticos (HOLLAND, 1975), métodos de busca local (HELSGAUN, 2000) e entre outros. Para esta pesquisa, o interesse restringe-se ao uso de algoritmos genéticos aplicados ao TSP.

Esta pesquisa tem por objetivo medir o desempenho de alguns operadores de cruzamento genético durante o processo evolutivo de um algoritmo genético projetado especificamente para solucionar o Problema do Caixeiro Viajante. A ideia de manter uma população de percursos distintos para o caixeiro, alocados em diversas pontos dentro do espaço de busca para o TSP, e que novos pontos possam ser alcançados por meio de transições probabilísticas efetuadas pelos operadores artificiais, evidenciam as capacidades de otimização global dos algoritmos genéticos. Portanto, considera-se que os algoritmos genéticos, constituem um modelo sólido para construção de heurísticas competitivas aplicadas ao TSP.

Os algoritmos genéticos, caracterizam-se por reunir conceitos de matemática e teoria evolucionária, com propósito de simular aspectos de adaptação visto em organismo vivos, por meio de operadores artificiais. Esses operadores, são aplicados em uma população de indivíduos artificiais, que representam soluções para algum problema específico. Para este trabalho, considera-se o TSP como o problema específico.

Nos algoritmos genéticos aplicados ao TSP, o operador de cruzamento é responsável por construções de novos percursos, combinando os percursos já conhecidos dentro do espaço de busca para o problema. Um bom operador de cruzamento, deve ser concebido tendo como considerações as características do domínio do problema, projetando regras que inibam a perda de características de qualidade presentes nos percursos conhecidos durante a fase de construção de novos percursos (LIU; ZENG, 2009).

Por fim, será apresentado no contexto do trabalho, uma revisão dos principais operadores de cruzamento genético projetados especificamente para o problema do caixeiro viajante enunciados na literatura especializada. Por fim, tais operadores serão analisados, submetidos a um conjunto de instâncias de teste, e os resultados serão comparados. O objetivo é estabelecer os operadores de cruzamento mais competitivos quando aplicados ao TSP.

1.6 Trabalhos relacionados

Nesta seção apresentamos uma breve revisão de alguns trabalhos relacionado com o tema proposto por este trabalho.

Em Larranaga *et al.* (1999), são discutidos diferentes modos de codificar uma solução para o TSP, como por exemplo, representação binária, repor adjacência, representação baseada em matriz, representação ordinal e representação por percurso. Também é apresentada uma revisão de diversos operados de cruzamento e mutação projetados especificamente para o TSP. Por fim, é feito um GA empregando o conceito de *GENITOR* no qual apenas um cromossomo é criado por geração e esse cromossomo é reinserido na população, se e somente se, o valor de aptidão for melhor que o valor de aptidão do pior cromossomo da população corrente.

Em seguida, um conjunto variado de operadores de cruzamento e mutação que empregam o modelo de codificação baseado em representação por percurso são comparados empiricamente, sendo submetidos a uma base de experimentos com 3 instâncias do TSP variando entre 24, 47 e 48 cidades. Os operadores *Genetic Edge Recombination Crossover* (ER), *Order Crossover 1* (OX1), *Order Based Crossover* (OX2), *Position Based Crossover* (POS), *Partially-Mapped Crossover* (PMX) e *Cycle Crossover*(CX) obtiveram os melhores resultados, sendo que o ER, OX1, OX2 e POS demonstrou maior superioridade em qualidade em relação aos demais.

Em Nagata e Kobayashi (1997) é apresentado um operador de cruzamento genético denominado *Edge Assembly Crossover*. Este operador encontra novas soluções para o TSP por meio da construção de subciclos com as arestas existentes na representação de dois percursos progenitores e o emprego de busca gulosa durante o processo construtivo nos resultados o operador se mostrou bastante promissor. Em Nagataa e Kobayashia (2013) é proposto uma nova estratégia de seleção de subciclos com o propósito de inibir a perca de diversidade da população. O operador foi submetido a vários experimentos, precisamente 57 instâncias com até 87000 cidades, tendo um desempenho considerável e obtendo soluções de alta qualidade para as instâncias testadas.

Em Andreica e Chira (2014) é demonstrado um estudo sobre diversos operadores de cruzamento genético com representação baseada em percurso. Os operadores são, *Order Crossover 1* (OX1), *Order Based Crossover* (OX2), *Cycle Crossover* (CX), *Partially-Mapped Crossover* (PMX), *One-Point Crossover* (1X), *Two-Point Crossover* (2X), *Uniform Crossover* (UX), *Position Based Crossover* (POS). Também é proposto um novo operador de cruzamento denominado *best-order crossover* (BOX). Os operadores enunciados no trabalho, foram aplicados a três problemas de otimização clássicos, eles são, o problema do caixeiro viajante (TSP), problemas de roteamento de veículos (VRP) e problemas de programação de projetos com restrição de recursos (RCPSP). Esses dois últimos são derivações do primeiro problema. No total 9 operadores de cruzamento genético foram aplicados em 63 instância de teste para o TSP com até 1000 cidades, 94 instâncias de teste para o VRP e 20 instâncias para o RCPSP. Conforme os resultados numéricos do trabalho, para as instâncias do TSP OX1, OX2 e BOX obtiveram as melhores desempenhos.

1.7 Considerações finais e estrutura do trabalho

Neste capítulo foi apresentado a definição do Problema do Caixeiro Viajante e suas aplicações práticas. Também foi discutido sobre a complexidade para solucioná-lo. Em seguida, foi apresentada uma introdução superficial sobre algoritimos genéticos como o método de otimização aproximado a ser aplicado na busca de soluções para o problema Caixeiro Viajante.

No Capítulo 2 será abordado de forma mais detalhada a teoria geral sobre algoritmos genéticos.

No Capítulo 3 discute-se um conjunto de operadores de cruzamento propostos para o problema do Caixeiro Viajante. Também é apresentado o esboço metodológico aplicado no processo de investigação dos operadores listados no âmbito desta pesquisa.

O Capítulo 4 apresenta os experimentos realizados, detalha os procedimentos adotados e discute os resultados obtidos.

No Capítulo 5 conclui-se os resultados obtidos do trabalho e esboça-se algumas ponderações sobre a pesquisa e projetando interesses sobre trabalhos futuros.

CAPÍTULO

2

ALGORITMOS GENÉTICOS

Neste capítulo apresenta-se os conceitos fundamentais, definições, características específicas e terminologias técnicas sobre a teoria de algoritmos genéticos.

2.1 Considerações iniciais

No âmbito do trabalho, algoritmo genético foi aplicado ao problema do caixeiro viajante com o propósito da pesquisa direcionado especificamente para o estudo de operadores de cruzamento genético projetados para o TSP. Porém, é necessário salientar que para além da fase de cruzamento o processo evolutivo de um GA envolve outros conceitos relevantes, tais como operadores auxiliares que juntos ao operador de cruzamento genético completam a fase de reprodução de um GA. Neste capítulo será apresentando a estrutura de um GA típico adotado por este trabalho e aplicado ao TSP.

O capítulo é dividido em conceitos básicos (Seção 2.2), definição de um algoritmo genético típico (Seção 2.3), também é apresentando o esboço de um algoritmo genético híbrido (Seção 2.4) e por fim considerações finais do capítulo (Seção 2.5).

2.2 Conceitos básicos

Algoritmo genético, do inglês *Genetic Algorithm* (GA), é uma simples e eficiente meta-heurística[1] de propósito geral, baseada em princípios de seleção natural e genética evolutiva, cujo objetivo é encontrar soluções de alta qualidade para problemas complexos (HAUPT, 2004). Os GAs operam sobre uma população finita de elementos. Um elemento é um indivíduo codificado na forma de uma estrutura cromossômica, que representa uma solução em potencial para um

[1] Uma meta heurística é um algoritmo aproximado de alto nível projetado para solucionar uma ampla gama de problemas de busca e otimização combinatória sem haver necessidade de uma adaptação rígida em sua estrutura para cada tipo de problema (BOUSSAïD; SIARRY, 2013).

tipo específico de problema. A cada ciclo evolutivo do algoritmo, os indivíduos presentes na população são submetidos a avaliação por algum critério que visa mensurar um valor de aptidão para cada indivíduo em relação ao objetivo a ser solucionado: este valor também é conhecido como *fitness*.

Com base no valor do *fitness*, a cada geração os indivíduos mais aptos da população são selecionados por um mecanismo que simula o processo de seleção natural (operador de seleção), para serem submetidos a operadores de transformação estocástica inspirados em genética elementar e reprodução sexuada, com a finalidade de criar novos cromossomos mais aptos para próxima geração (GEN, 2000; MELANIE, 1998). Existem dois tipos essenciais de operadores de transformação, operação de cruzamento (*crossover*), que gera novos indivíduos pela combinação genética entre dois ou mais cromossomos progenitores, e operação de mutação, que gera um novo indivíduo por meio de troca aleatória de algumas pequenas características genéticas de um indivíduo (MICHALEWICZ, 1996).

A escolha adequada da codificação dos cromossomos em conjunto com a definição de uma medida de aptidão que avalie criteriosamente os indivíduos da população são requisitos fundamentais para um bom desempenho de um GA (HAUPT, 2004). Avaliar de forma eficiente os indivíduos é importante para caracterizar quais cromossomos representam as melhores soluções. Sendo assim, é essencial que cada indivíduo seja codificado corretamente e que o mesmo contenha maior quantidade de informações sobre o problema.

Uma medida de aptidão adequada define de forma precisa quais cromossomos deverão possuir as maiores probabilidades de serem selecionados para reproduzirem, transmitindo suas características hereditárias às gerações futuras. Neste contexto, tanto a codificação dos cromossomos quanto a medida de aptidão devem ser projetadas em conformidade com as restrições naturais intrínsecas ao problema, como garantia de eficácia e eficiência do processo evolutivo do algoritmo.

A cada geração do GA uma nova população, com os indivíduos melhores adaptados ao problema, é reinserida (operador de reinserção) no processo evolutivo. A interação entre avaliação, seleção de indivíduos, transformação estocástica e composição de uma nova população é repetida até que um critério de parada seja satisfeito. Tal critério pode variar do esgotamento do número finito de gerações pré-estabelecidas ou estagnação do processo de busca em encontrar novas soluções com maiores aptidões (SILVA, 2011). Quando há estagnação do processo de busca dizemos que o GA convergiu, e como consequência, o ciclo evolutivo é encerrado e o GA retorna o melhor indivíduo da população atual com a expectativa que o mesmo seja próximo do valor ótimo para o problema (MICHALEWICZ, 1996).

Algoritmos genéticos tendem a serem eficientes quando bem projetados para problemas com espaço de busca muito extenso (GOLDBERG, 1989). Eles atuam sobre um conjunto de pontos, explorando múltiplas direções do espaço de busca simultaneamente e utilizam regras de transições probabilísticas nos operadores de transformação e seleção. Somadas juntas, estas duas

características dos *GA*s simplificam a implementação de algoritmos paralelos e previne que a busca fique estagnada em um ótimo local (GOLDBERG, 1989; YANG, 2014).

Uma ponderação importante a respeito dos GA, está relacionada a adequação dos parâmetros de controle, tais como, tamanho da população de indivíduos, taxa de cruzamento e de mutação, e critério de reinserção dos novos indivíduos na população de cromossomos (YANG, 2014). Esta tarefa é extremamente importante, pois uma configuração equivocada de tais parâmetros pode provocar um efeito indesejável no processo de busca, gerando resultados de baixa qualidade ou um ganho de custo computacional sem resultar em uma melhora significativa na qualidade da população de cromossomos.

É importante ressaltar que, assim como em qualquer meta-heurística, algoritmos genéticos não garantem encontrar ótimos globais para todos os problemas ou até mesmo em todos os casos de um determinado problema. Todavia, a aplicação dos *GA*s em um conjunto numeroso de problemas complexos tem mostrado que os algoritmos genéticos cumprem o seu papel com rigor e eficiência na busca de soluções competitivas (MELANIE, 1998; GOLDBERG, 1989).

2.2.1 Computação evolutiva e as inspirações biológicas dos Algoritmos Genéticos

Na natureza, a evolução biológica é um mecanismo completo e sofisticado, capaz de criar organismos autônomos extremamente complexos com habilidades de sobreviverem em ambientes imprecisos e em constante transformação. Deste modo, o processo de evolução se manifesta como um processo de adaptação robusto, onde indivíduos superiores, são equipados com habilidades de auto-aprendizagem, reconhecimento de padrões, treinamento e tomada de decisão, e são bem sucedidos na busca pela sobrevivência em diferentes ecossistemas (DRÉO, 2006).

A grande variedade de situações em que o processo de adaptação natural demonstra eficiência na solução de problemas do mundo real, serviu como inspiração na concepção de processos computacionais baseados na abstração dos conceitos envolvidos nas leis naturais (DRÉO, 2006). O objetivo é expandir as funcionalidades originais dos fenômenos da evolução biológica a fim de serem aplicados no desenvolvimento de sistemas artificiais robustos e eficientes para solucionar problemas que não sejam somente a proliferação da vida na terra.

Entre as décadas de 1960 e 1970, foram desenvolvidas três técnicas relacionando a evolução biológica como uma ferramenta de otimização para problemas de computação e engenharia (MELANIE, 1998). Elas são estratégias evolucionárias desenvolvidas por Rechenberg (1965) as quais são utilizadas para otimização de parâmetros variáveis e auto-adaptáveis em problemas contínuos; programação evolucionária desenvolvida por Fogel, Owens e Walsh (1966), no qual as soluções candidatas são representadas como máquinas de estado finito, que são evoluídas por mutações aleatórias e seleção dos candidatos mais aptos e algoritmos genéticos desenvolvidos

por Holland (1975) com o objetivo de compreender mecanismos auto adaptáveis por meio de simulação em sistemas artificiais.

Embora no princípio houvesse pouca interação entre estas abordagens, todas apresentam características semelhantes envolvendo reprodução, comportamento aleatório, competição e seleção de indivíduos pertencentes à uma determinada população (FOGEL, 1995). Porém, cada grupo havia desenvolvido sua própria notação e nomenclatura de forma isolada (ROZENBERG; BCK; KOK, 2011). À medida que a interação entre os grupos se tornou mais frequente, ficou evidente a necessidade de agrupá-los em uma perspectiva mais ampla, na qual as estratégias evolucionárias, a programação evolucionária e os algoritmos genéticos fossem vistos como instâncias importantes dentro de uma estrutura de computação padronizada (ROZENBERG; BCK; KOK, 2011; BäCK, 1996). Então, em 1991 a Computação Evolutiva (CE) surge como um campo de pesquisa avançando, representando um esforço para reunir diversas abordagens que visam simular aspectos diversos da evolução biológica em sistemas computacionais (BäCK DAVID B FOGEL, 2000).

Algoritmo Genético é a técnica mais proeminente da Computação Evolutiva (DAVIS, 1991). Eles foram desenvolvidos pelo pesquisador John Holland em 1975 na Universidade de Michigan, e aprimorados em conjunto com seus alunos e colegas na década 1970. A princípio, o objetivo de Holland em suas pesquisas iniciais não era projetar algoritmos para solucionar problemas específicos, mas formalizar um estudo teórico sobre o fenômeno de adaptação natural, bem como ocorre na natureza, e propor um esquema que torne possível incorporar tais mecanismos em sistemas computacionais (MELANIE, 1998).

Conforme a teoria de Seleção Natural de Darwin (DARWIN, 1859), indivíduos pertencentes a um mesmo ecossistema competem entre si por recursos naturais limitados tais como água, comida e espaço físico. Ao longo do tempo, alguns indivíduos podem sofrer transformações devido as variações que ocorrem no ambiente. Essas transformações, agem no fenótipo desses indivíduos alterando algumas de suas características observáveis. Então, indivíduos com fenótipos favoráveis que se adaptam melhor as variações do ambiente, tem maiores chances de sobreviver e reproduzir do que indivíduos com fenótipos menos favoráveis. Ao reproduzirem, os indivíduos transmitem suas características hereditárias que foram imprescindíveis a sua sobrevivência para seus descendentes, aumentando a ocorrência desses fenótipos nas gerações futuras e criando uma forma de adaptação especializada desses indivíduos em relação ao nicho ecológico que os mesmos sobrevivem (DARWIN, 1859). Ao mesmo tempo, indivíduos que compõem fenótipos menos favoráveis tendem a não obter êxito em sua descendência, ocasionando a extinção dos mesmos, juntamente com suas características menos adaptáveis.

Em seu livro "*Adaptation in Natural and Artificial Systems*" (HOLLAND, 1975), Holland apresenta os *GA*s à comunidade científica como uma abstração da teoria de Darwin (DARWIN, 1859), em conjunto com um modelo de sustentação matemático possível de ser aplicado em um ambiente artificial. A ideia é que assim como ocorre com os seres vivos na

natureza as máquinas também poderiam se adaptar ao ambiente (SILVA, 2011).

2.2.2 Definição

Nesta seção, apresenta-se três definições para os algoritmos genéticos:

Definição 2. Conforme Goldberg (1989), algoritmos genéticos são baseados em mecanismos de seleção natural e genética elementar, que combinam a sobrevivência dos mais fortes com a troca aleatória de informação estruturada, formando um algoritmo com um faro de busca inovador. Os *GA*s exploram de forma eficiente informações históricas contidas na população corrente para especular novos pontos no espaço de busca, com a expectativa de melhora de performance.

Definição 3. Conforme Michalewicz (1996), algoritmos genéticos pertencem a classe dos algoritmos estocásticos, cujos métodos de busca baseiam-se em modelos relacionados a fenômenos naturais, como herança genética e seleção natural. Porém, os *GA*s diferem de outros métodos aleatórios no modo em que combinam elementos de busca dirigida e estocástica. Isto posto, os *GA*s se apresentam de maneira robusta, mantendo uma população de solução em potenciais para o problema ao invés de atuarem em um único ponto dentro do espaço de busca.

Definição 4. Segundo Rozenberg, Bck e Kok (2011), algoritmos genéticos dão enfase ao uso de genótipo como forma de representação das soluções candidatas, sendo decodificados e avaliados a cada interação do algoritmo. Os genótipos são apresentados em estruturas de dados simples e são definidos como cromossomos artificiais. Em boa parte das aplicações envolvendo GAs, os genótipos são formados por cadeias de bits que são recombinados em uma forma simplificada de reprodução sexuada e podem sofrer mutações com trocas aleatórias de bits dentro da cadeia ao longo das gerações.

2.2.3 Espaço de busca

Algoritmos genéticos são essencialmente algoritmos de busca em um espaço de soluções candidatas (HAUPT, 2004). O processo de busca considera que, soluções candidatas com alta qualidade contidas em diferentes regiões do espaço de busca possam ser recombinadas via operadores genéticos e, ao longo do tempo (das interações), este processo produza novas soluções com melhores aptidões (MITCHELL, 1996).

Um importante conceito no que se refere ao espaço de busca dentro das perspectivas dos *GA*s, está associado a definição do ambiente de busca (*landscape*). No contexto das populações genéticas, o *landscape* refere-se a representação de todos os genótipos possíveis em conjunto com suas aptidões (MITCHELL, 1996). Seguindo a lógica para formulação do *landscape*, o processo de evolução induz que uma determinada população se mova ao longo do ambiente de forma singular. Portanto, o fenômeno de adaptação é visto como uma caminhada em direção a ”picos” locais dentro do espaço de busca (MITCHELL, 1996).

O processo de busca dos *GA*s empregam as técnicas *exploitation* e *exploration* (HOLLAND, 1975). *Exploitation* caracteriza-se por recombinar informações já extraídas do *landscape* durante a fase de cruzamento genético. A análise de convergência ocorre em baixa pressão seletiva, criando com comportamento aleatório do processo de busca. *Exploration*, guia a busca por novos pontos nas regiões do *landscape*, durante a fase de mutação gênica. A análise de convergência ocorre am alta pressão seletiva criando um movimento de busca guiada com comportamento similar ao método Hill Climbing

2.2.4 Terminologia

Algoritmos genéticos possuem um elo de ligação entre genética elementar e evolução natural. Em função disso, alguns termos técnicos entres as abordagens artificias dos *GA*s e as abordagens naturais da evolução biológica, possuem denotação sintática similar, porém possuem conotação semântica distintas. Para exemplificar, o Quadro 1 (GOLDBERG, 1989), apresenta o significado de alguns termos técnicos da evolução natural aplicados aos *GA*s.

Quadro 1 – Definição de termos da evolução natural aplicados aos *GA*s.

Evolução natural (termo)	**Algoritmos genéticos (significado)**
Cromossomo	*string*, cadeia, indivíduo
Gene	característica
Alelo	valor de uma característica
Locus	posição
Genótipo	estrutura codificada
Fenótipo	conjunto de parâmetros, estrutura decodificada

Nos algoritmos genéticos, os termos indivíduo, cromossomo ou solução são intercambiáveis; ambos possuem o mesmo significado ao processo e tipicamente se referem a uma solução candidata para o problema (LINDEN, 2006). Os cromossomos são conceitualmente subdivididos em genes. Um gene, por sua vez, na perspectiva dos GA, representa um fator de controle responsável por codificar uma determinidade característica do indivíduo (SIVANANDAM; DEEPA, 2008). As diferentes configurações (valores) possíveis que um gene pode assumir são denominadas de alelo (MITCHELL, 1996), e cada posição do gene dentro da estrutura do cromossomo é denominada de locus (LINDEN, 2006).

Dois outros termos de grande relevância aos *GA*s são, genótipo e fenótipo. O genótipo representa a estrutura geral do cromossomo, ou seja, a forma de codificação do cromossomo, enquanto o fenótipo corresponde à interação do conteúdo genético com o ambiente, relacionado ao conjunto de parâmetros do algoritimo (LINDEN, 2006).

2.2.5 Principais Características dos Algoritmos Genéticos

Em Goldberg (1989), são apresentadas algumas características que distinguem os algoritmos genéticos dos demais métodos de otimização. Essas características são responsáveis pela robustez embutida nos *GA*s. Tais características são enunciadas conforme abaixo:

- *GA*s trabalham com codificação dos parâmetros definidos e não apenas com os próprios parâmetros em si;
- *GA*s atuam sobre um conjunto de pontos dentro do espaço de busca, não apenas em um único ponto;
- *GA*s não necessitam de informações auxiliares e nem de informações derivadas dos problemas, mas apenas informações extraídas da função objetivo;
- *GA*s utilizam regras de transições probabilísticas ao invés de regras de transições determinísticas.

2.3 Algoritmo Genético Canônico

Segundo SILVA (2011), um *GA* simples, caracteriza-se por atuar em uma única população de cromossomos e otimizar somente um único objetivo, sem o emprego de métodos de busca local durante o processo evolutivo do algoritmo.

De acordo com Goldberg (1989), os mecanismos envolvidos em um algoritmo genético simples são relativamente triviais, não envolvendo nada mais complexo que a cópia e a troca de sequências parciais entre os cromossomos. A eficiência de tais algoritmos são compostos basicamente por três componentes fundamentais:

- **Seleção**: Processo em que os cromossomos são selecionados em acordo com as aptidões obtidas pela função objetivo.
- **Reprodução**: Processo que realiza o cruzamento entre os cromossomos selecionados com o objetivo de criar novos indivíduos.
- **Mutação**: Processo de inserção de novas características que evita a perca de diversidade genética de uma população de cromossomos.

A Figura 3 esboça o fluxo de operação básico do processo evolutivo executado por um *GA* simples. O primeiro passo do processo evolutivo de um *GA* é definir uma população inicial de cromossomos (*Pc*), e submeter os cromossomos gerados a um critério de avaliação definido pela função objetivo. Então, seleciona-se um conjunto de cromossomos (*tc*) a partir de *Pc* e submeta-os a operação de cruzamentos aos pares. Em seguida, submeta os descendentes

gerados após operação de cruzamento à operação de mutação, com uma probabilidade pré-definida (*pm*). Avalie as aptidões dos descendentes por meio da função de avaliação (função objetivo), submeta os cromossomos descendentes junto com os pais ao operador de reinserção que definirá os sobreviventes para próxima geração, e substitua os cromossomos em *Pc* pelos sobreviventes. Caso o critério de parada seja satisfeito, pare o processo de execução do *GA* e retorne o cromossomo de *Pc* com o maior valor de aptidão.

Figura 3 – Ciclo evolutivo de um GA simples.

Fonte: Elaborada pelo autor.

2.3.1 Representação dos cromossomos

A representação dos cromossomos compreende o processo de codificar as soluções candidatas contidas no espaço de busca para o problema proposto. Este processo, envolve estabelecer um alfabeto finito de símbolos para serem utilizados na composição dos cromossomos (*coding space*), de maneira que melhor atenda as restrições impostas pela função objetivo (*solution space*) (SILVA, 2011).

Segundo Aguilar-Ruiz, Giráldez e Santos (2007), é desejável que o processo de codificação das soluções candidatas, satisfaça as seguintes propriedades:

- **Coerência**: impossibilidade de codificação de cromossomos que não possuam um significado semântico;
- **Uniformidade**: todos os cromossomos devem ser representados pelo mesmo critério de codificação, se possível, o cromossomo deve ser o único;
- **Simplicidade**: a função de codificação deve ser simples de ser implementada tanto para codificar quanto para decodificar;
- **Localidade**: pequenas modificações nos valores (fenótipos) devem corresponder a pequenas modificações na estrutura de solução candidatas codificadas (genótipo);
- **Consistência**: codificações improdutivas não devem ser produzidas;
- **Minimalidade**: a dimensão da estrutura do cromossomo codificado deverá ser tão curta quanto possível.

A representação binária (HOLLAND, 1975) é a abordagem mais clássica para codificação dos cromossomos. Foi amplamente utilizada nas primeiras pesquisas sobre GAs e atualmente tem uma grande aceitação pela comunidade científica por sua simplicidade de operação.

Entretanto, existem várias situações em que a representação binária apresenta obstáculos ao conjunto de restrições naturais impostos pelo problema (TANG *et al.*, 1996), podendo ocasionar um comportamento de convergência prematura do processo de busca, perca de qualidade ou previsibilidade das soluções codificadas ou um custo de avaliação dos cromossomos mais elevado.

Com a expansão do uso de GAs, outras formas de representação dos cromossomos foram introduzidas, algumas delas são, representação por números reais em ponto flutuantes aplicados a problemas contínuos com exigência de um grau de precisão elevado (JANIKOW; MICHALEWICZ, 1991), representação baseada em permutações aplicados em problemas que envolvem otimização combinatória (DAVIS; MITCHELL, 1991), representação baseada em lista de elementos para problema de agendamento de tarefas (TANG *et al.*, 1996; CHENG; GEN; TSUJIMURA, 1996).

2.3.2 População inicial

Uma população inicial de tamanho finito, deve ser gerada de forma totalmente aleatória ou parcialmente aleatória por meio de algum método de pertubação baseado em busca local, tal que, os cromossomos gerados sejam espalhados, contemplando todas as regiões do espaço de busca (SIVANANDAM; DEEPA, 2008).

É importante que a população inicial cubra a maior parte possível do espaço de busca, para que haja diversidade de características na representação dos cromossomos. Em analogia

com a natureza, para que exista evolução deve haver pluralidade, logo é necessário que os cromossomos tenham variação de características genéticas para que possa ocorrer à seleção natural (SILVA, 2011).

2.3.3 Função de avaliação

A fase de avaliação é responsável por determinar quais cromossomos presentes na população representam as melhores soluções para o problema proposto (NELSON; BARLOW; DOITSIDIS, 2009). Ela deve avaliar os cromossomos individualmente e atribuir um valor de aptidão, que é calculado com base no nível de adaptação em que cada cromossomo se encontra em relação a função objetivo.

A função de avaliação deve refletir os objetivos a serem alcançados na resolução do problema, sendo abstraída diretamente do conjunto de restrições imposta pelo mesmo. Portando, esta função deve ser projetada com máximo de rigor, para que obtenha uma maior quantidade de informações sobre o problema. Quanto maior o grau de conhecimento da função objetivo maior será o nível de avaliação de aptidão imposta à população de cromossomos (LINDEN, 2006).

Segundo Tang *et al.* (1996), com o propósito de estabelecer um nível de adequação padronizada entre as aptidões dos cromossomos, faz-se necessário mapear o valor de objetivo que cada cromossomo da população representa para um valor de *fitness* associado por algum método de normalização. Para tal mapeamento, são apresentados em seguida duas técnicas baseadas no cálculo de *fitness*:

- ***Windowing***: Nesta técnica, cada cromossomo da população é assinalado com um valor de *fitness* $f_{(i)}$ que é proporcional à diferença do valor objetivo entre o i-ésimo cromossomo com o pior cromossomo da população, conforme a Equação 2.1.

$$f_{(i)} = c \pm (V_i - V_w). \tag{2.1}$$

  Considere V_w como o valor de objetivo do pior cromossomo presente na população, V_i o valor objetivo referente ao i-ésimo cromossomo e c uma constante qualquer. Se o problema for de maximização, então considere o sinal positivo para a equação 2.1, se o problema for de minimização considere o sinal negativo.

- **Normalização linear**: Os cromossomos são classificados em ordem crescente ou decrescente, conforme o valor objetivo de cada cromossomo. Dado o cromossomo com a melhor aptidão extraída da função objetivo, o valor de *fitness* do i-ésimo cromossomo na ordenação é calculado por uma função linear dada pela Equação 2.2, sendo d uma taxa de decrescimento.

$$f_{(i)} = f_{melhor} - (i-1) \cdot d. \tag{2.2}$$

2.3.4 Operação de seleção

A operação de seleção desempenha uma função similar ao da seleção natural na teoria da evolução, determinando quais indivíduos (cromossomos) serão selecionados para fase de reprodução. Neste processo, os indivíduos com os maiores valores de aptidão devem possuir as maiores probabilidades de serem selecionados (DRÉO, 2006).

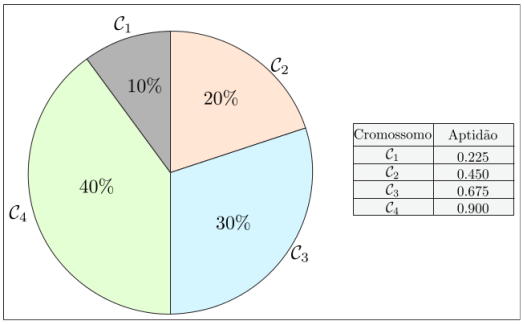
Em seguida, são listados alguns operadores de seleção tradicionais dentro da perspectiva da teoria dos algoritmos genéticos:

- **Seleção proporcional (método da roleta)**: Neste modelo, é descrito um esquema de seleção natural que seleciona os sobreviventes para a fase de reprodução conforme os valores de *fitness* extraídos da função objetivo durante a fase de avaliação. Neste esquema, a probabilidade de seleção para um indivíduo qualquer, é definida pela Equação 2.3,

$$P_{(i)} = \frac{f_{(i)}}{\sum_{j=1}^{n} f_{(j)}} \tag{2.3}$$

onde $P_{(i)}$ é a probabilidade de seleção do i-ésimo indivíduo da população corrente, dada por meio da divisão do seu valor de *fitness* ($f_{(i)}$) pela soma total de todos os *fitness* da população corrente. O esquema de seleção é baseado no método da roleta, onde é definida uma roleta imaginária de tamanho N, sendo $N = \sum_{i=1}^{n} P_{(i)}$. Para cada indivíduo da população é assinalado uma "fatia" desta roleta com tamanho proporcional ao valor de *fitness*. A roleta é girada T vezes, sendo T a quantidade de indivíduos selecionados. Em cado giro da roleta, um valor entre [0,N] é sorteado e o indivíduo com o segmento que abriga este valor na roleta deve ser marcado como selecionado e submetido a fase de reprodução. A figura 4 exemplifica o funcionamento da rotela durante a fase de seleção.

Figura 4 – Ilustração de uma roleta imaginária utilizada no processo de seleção proporcional



Fonte: SILVA (2011).

As desvantagens em relação a este método de seleção esta associado a uma rápida estagnação do processo de busca em uma GA. Normalmente, no inicio do processo evolutivo de um GA, a variação do nível de aptidão em relação aos cromossomos de população inicial é alta

e somente um pequeno número de cromossomos são muito mais aptos que os demais. Sob o critério de seleção do método de seleção proporcional ao nível de aptidão, os indivíduos mais aptos e seus descendentes se sobressairão em relação aos demais, multiplicando-se rapidamente na população de cromossomos e impedindo que o processo de busca do GA tenha acesso a outras região do espaço de busca do problema. Este comportamento da população de cromossomos é conhecido como convergência prematura. De forma sucinta, o modelo de seleção proporcional muitas vezes enfatiza demasiadamente a exploração de ótimos locais representandos pelos cromossomos com os melhores níveis de aptidão, em detrimento da exploração de outra regiões do espaço de busca. Porém, a medida que o GA converge, e os cromossomos da população são muito semelhantes (a variância de aptidão é baixa), não existe diversidade de informação genética suficientes para o operador de seleção explorar, então o processo de evolução é guiado para uma parada prematura. Deste modo, a tava de evolução para este operador de seleção depende diretamente da variação de aptidão apresentada pela população de cromossomos (MITCHELL, 1996).

- **Seleção por *Rank***: Neste modelo de seleção, a cada geração, os indivíduos presentes na população corrente são ordenadas conforme o valor de *fitness* para cada indivíduo em relação aos demais. O indivíduo com o pior valor de aptidão recebe o *rank* 1 enquanto o melhor indivíduo recebe o *rank* N (para N = o tamanho da população). Seja $\{ x_1^{(t)}, x_2^{(t)}, ..., x_N^{(t)} \}$ o conjunto de indivíduos da população na geração t. Logo, a probabilidade de seleção para um indivíduo é calculada por um ranqueamento linear (Equação 2.4) (CHAKRABORTY; CHAKRABORTY, 1999):

$$p(x_i^{(t)}) = \frac{1}{N}\left(min + \frac{(max - min)\left(rank[x_i^{(t)}] - 1\right)}{N-1} \right) \tag{2.4}$$

onde $max + min = 2$ e $1 \leq max \leq 2$. Então, $p(x_i^{(t)})$ representa a distribuição de probabilidade apropriada para $x_i^{(t)}$, em um espaço amostral de N indivíduos.

- **Seleção por Torneio**: Seleciona-se aleatoriamente um número predeterminado de indivíduos da população corrente. Promova um torneio, tomando para sobrevivência o melhor indivíduo entre os candidatos pré-selecionados de maneira determinística ou probabilística. Repita os procedimentos descritos anteriormente t vezes, sendo t igual a quantidade de indivíduos que serão selecionados para reprodução.

- **Elitismo**: Neste modelo de seleção, um número fixo de indivíduos com os melhores níveis de aptidão são retidos diretamente a cada geração (MITCHELL, 1996). O objetivo é evitar que indivíduos com grau de aptidão elevado, não se perca durante a fase de reprodução ou que sejam deformados por construções nocivas causadas por operadores de cruzamento ou mutação.

- **Seleção truncada**: Um subconjunto dos melhores indivíduos são selecionados para fase de reprodução com a mesma probabilidade (SILVA, 2011).

2.3.5 Operação de Cruzamento

O operador de cruzamento utiliza a combinação entre dois cromossomos progenitores para gerar cromossomos descendentes (DRÉO, 2006). Deste modo, os cromossomos da população corrente são selecionados aos pares de forma distinta e submetidos ao processo de reprodução com a troca aleatória de características genéticas entre os cromossomos selecionados. O objetivo é proporcionar uma mesclagem entre as soluções candidatas que cada cromossomo representa para criar novos cromossomos mais aptos (YANG, 2014).

O operador de seleção é responsável por definir quais cromossomos serão submetidos à operação de cruzamento, e a taxa de cruzamento determina a proporção exata da quantidade de cromossomos que serão submetidos ao operador cruzamento (DRÉO, 2006). A taxa de cruzamento é definida por um parâmetro de controle e, em geral, varia entre 70% à 100% da população corrente (YANG, 2014).

Não existe uma regra absoluta determinística a ser seguida na construção de um operador de cruzamento (DRÉO, 2006). A ideia é que os parâmetros para construção de tal operador sejam guiados pelo domínio e restrições do problema em questão (SILVA, 2011). Em geral, as operações de cruzamento seguem tendências e conceitos inspirados biologicamente, como à troca direta de material genético entre os pares de cromossomos (SILVA, 2011).

Nesta categoria existem três operadores de cruzamento considerados como clássicos na literatura (DRÉO, 2006):

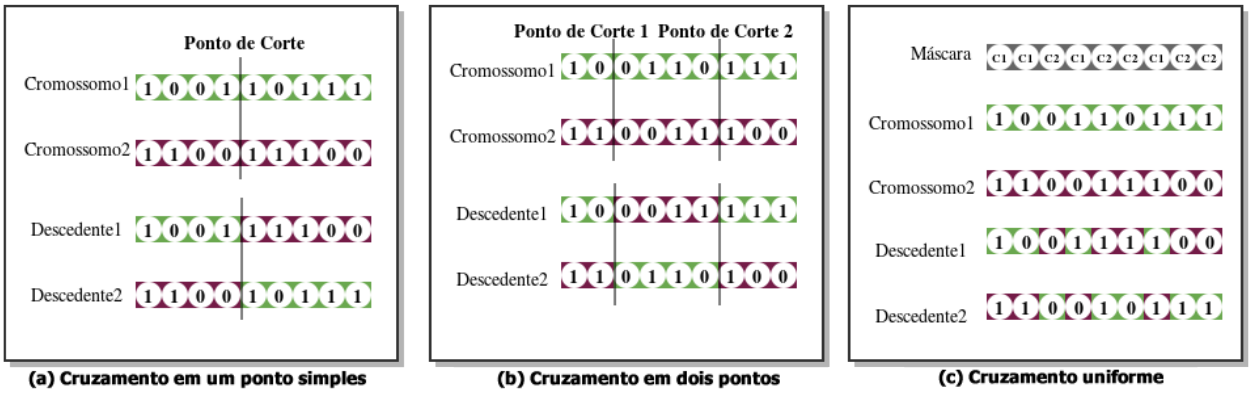
- **Cruzamento de um ponto simples**: Proposto originalmente por (HOLLAND, 1975), dado um par de indivíduos (cromossomos), primeiro é definido aleatoriamente um ponto de corte que particionará ambos os cromossomos em duas partes. Posteriormente, a primeira parte do primeiro cromossomo será agrupada à segunda parte do segundo cromossomo, formando assim o primeiro cromossomo descendente, depois, a primeira parte do segundo cromossomo será agrupada à segunda parte do primeiro cromossomo formando o segundo cromossomo descendente (DRÉO, 2006). A Figura 5-(a) ilustra o cruzamento de um ponto.

- **Cruzamento múltiplo**: Múltiplos pontos de cortes são definidos aleatoriamente. As características genéticas contidas entre os pontos de corte são trocadas alternadamente entres os cromossomos progenitores (SILVA, 2011; DRÉO, 2006). A Figura 5-(b) ilustra uma operação de cruzamento com dois pontos de corte.

- **Cruzamento Uniforme**: Análogo ao cruzamento múltiplo, porém o número de pontos de cortes não é especificado a priori (DRÉO, 2006). Utiliza-se uma estrutura de máscara com o mesmo comprimento que os cromossomos e sorteia-se os genes que cada cromossomo

progenitor fornecerá ao primeiro filho; o outro filho é gerado pelo complemento da máscara (SILVA, 2011). A Figura 5-(c) ilustra o cruzamento uniforme.

Figura 5 – Alguns exemplos de operadores de cruzamento genético.



Fonte: Elaborada pelo autor.

2.3.6 Operação de Mutação

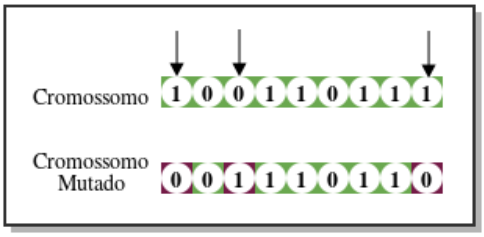
No contexto dos algoritimos genéticos, o processo de mutação ocorre ocasionalmente alterando aleatoriamente algumas características genéticas de um conjunto de cromossomos previamente selecionados por um critério probabilístico (com um fator de probabilidade muito pequeno) (GOLDBERG, 1989).

A motivação para utilizar o operador de mutação no processo evolutivo de um algoritmo genético está associado a manutenção da diversidade genética, com a possibilidade de explorar novos pontos no espaço de busca até então sem precedentes na população de cromossomos (ALBAYRAK; ALLAHVERDI, 2011).

A quantidade de cromossomos submetidos ao operador de mutação é proporcional à taxa de mutação, que possui uma probabilidade de ocorrência variando geralmente entre 0.01 à 0.05 (YANG, 2014). Uma taxa de mutação devidamente equilibrada, desempenha um papel importante na prevenção de efeitos negativos ocasionados por uma pressão seletiva demasiadamente elevada, atuando principalmente na preservação da diversidade genética, algo extremamente útil na exploração eficiente do espaço de busca do problema (DRÉO, 2006).

O operador de mutação deve ser definido de forma a não violar as restrições do problema em questão. Na literatura especializada, são listados alguns métodos de mutação (HOLLAND, 1975; GOLDBERG; LINGLE JR., 1985). Como exemplo, inversão aleatória de alguns genes na estrutura do cromossomo para o caso da representação binária conforme a Figura 6.

Figura 6 – Operador de mutação básico.



Fonte: Elaborada pelo autor.

2.3.7 Reinserção

Posterior ao processo de reprodução que envolve, seleção, cruzamento genético, mutação e avaliação das aptidões dos cromossomos descendentes da população corrente, à próxima etapa do ciclo evolutivo de um algoritmo genético é selecionar os sobreviventes para a próxima geração. Este processo consiste em definir um operador de reinserção com um mecanismo de seleção apropriado para executar tal procedimento.

Em (SILVA, 2011) são listados alguns operadores de reinserção amplamente empregado na construção de um algoritmo genético:

- **Reinserção pura**: consistem na substituição de todos os cromossomos progenitores da população corrente por todos os cromossomos descendentes. Geralmente este método é acompanhado por elitismo.

- **Reinserção uniforme**: seleciona os cromossomos sobreviventes da população total (pais + filhos) para a próxima geração por meio de algum operador de seleção tradicional, conforme mencionado na subseção 2.3.4.

- **Reinserção por elitismo**: uma parte dos melhores cromossomos progenitores da população corrente são mantidos para próxima geração.

- **Reinserção baseada na aptidão** ($\mu + \lambda$): todos os cromossomos da população corrente (pais e filhos) são ordenados conforme suas aptidões e os t melhores cromossomos são selecionados diretamente para a próxima geração.

2.3.8 Condição de parada e parâmetros de controle

A adequação dos parâmetros de controle dentro das perspectivas dos GAs é fundamental para uma boa convergência do processo de busca. A taxa de convergência é fortemente influenciada pela seleção de um conjunto de parâmetros, tais como o tamanho da população de cromossomos, probabilidade de mutação e probabilidade de cruzamento.

Entretanto, a definição de tais parâmetros de controle derivam das características específicas relacionadas ao problema a ser solucionado, como tamanho do espaço de busca ou até mesmo a definição da função de avaliação.

Alguns fatores relevantes na decisão de tais parâmetros devem ser considerados, como por exemplo, uma população de cromossomos pequena expressa uma baixa variação de diversidade genética, podendo resultar em uma convergência prematura do processo de busca. Todavia, uma população demasiadamente grande pode representar um aumento de custo computacional considerável sem resultar em aumento qualitativo significativo das soluções encontradas. Outro ponto de ponderação considerável é a respeito da taxa de cruzamento. Quanto maior a probabilidade de cruzamento, mais intensificado o processo de construções de novas soluções se torna. Em contrapartida, uma taxa de cruzamento muito elevada pode ser nociva a cromossomos de alto grau de aptidão, ao passo que a operação de seleção não consegue mantê-los em mesmo nível de intensidade (SILVA, 2011).

O operador de mutação é importante para manutenção da diversividade genética da população de cromossomos, exercendo a função de explorar novas regiões dentro do espaço de busca e prevenindo contra convergência prematura. Porém, uma taxa de mutação elevada torna a busca demasiadamente aleatória.

Para codição de parada dos GAs, é desejável que para todos os casos, a melhor solução para o problema seja sempre encontrada. Porém, os GAs são essencialmente métodos de busca aproximada, logo, não é garantido que em todos os casos os valores ótimos serão alcançados. Devido a este fato relevante, são listados alguns critérios de parada que podem ser adotados pelos GAs (SILVA, 2011):

- Esgotamento do número máximo de gerações (iterações) previamente estabelecidas;
- Esgotamento do tempo máximo de processamento previamente estabelecido;
- Encontro de um cromossomo com um valor de aptidão maior ou igual a um valor pré-definido;
- Estagnação do processo de busca em encontrar cromossomos com maiores aptidões em relação aos cromossomos presentes na população atual, após um determinado número de gerações.

2.4 Algoritmos genéticos híbridos

Algoritmos genéticos híbridos, por vezes também referenciados como Algoritmos Meméticos, são tipos especiais de *GA*s, que combinam caracteristicas dos *GA*s tradicionais (Seção 2.3) em conjunto com heurísticas de busca local (MOSCATO, 2003). O termo memético é inspirado pela teoria de *Dawkins* sobre o meme, que é definido como um gene cultural adaptado

por aspectos comportamentais adquiridos pelas pessoas que o carregam antes mesmo de serem transmitidos para a próxima geração (WANG *et al.*, 2011). No Algoritmo 1, é ilustrado o modelo de um GA Memético padrão.

Algoritmo 1 – Algoritmo genético híbrido padrão (WANG *et al.*, 2011)

Requer: Solução Intermediária.
1: $t \leftarrow 0$;
2: $P[t] \leftarrow PopulacaoInicial()$;
3: $P[t] \leftarrow BuscaLocal(P[t])$;
4: $AvaliacaoDeAptidao(P[t])$;
5: **enquanto** (condição de parada) **faça**
6: $P'[t] \leftarrow select(P[t])$;
7: $P'[t] \leftarrow cruzamento(P'[t])$;
8: $P'[t] \leftarrow mutacao(P'[t])$;
9: $P'[t] \leftarrow BuscaLocal(P'[t])$;
10: $AvaliacaoDeAptidao(P'[t])$;
11: $P[t+1] \leftarrow selecNovaPop(P[t], P'[t])$;
12: **se** $CriterioDeConvergencia(P[t+1])$ **então** **retorna** $melhor(P[t+1])$;
13: **fim se**
14: $t \leftarrow t+1$;
15: **fim enquanto**

O Algoritmo 1 difere do GA canônico basicamente em dois aspectos básicos, no emprego de métodos de busca local após a criação da população inicial, e posteriormente, a cada processo reprodutivo do ciclo evolutivo do GA. O emprego de busca local atua como um mecanismo de pertubação da ordem aleatória estabelecida durante o processo de busca evolucionária, atribuindo também um comportamento de busca dirigida ao processo de busca global efetuado pelo GA.

2.5 Considerações finais

Neste capítulo foram discutidos conceitos, terminologias técnicas e fundamentos que envolvem o ciclo evolutivo dos GAs. Também foram apresentadas as estruturas básicas de operação de um GA canônico e dos GAs híbridos. O objetivo de tal feito é evidenciar um arcabouço teórico robusto para uma melhor compreensão dos capítulos seguintes.

Os GAs, são algoritmos essencialmente baseados em técnicas de busca, sendo que, um algoritmo de busca recebe como parâmetro de entrada um problema a ser solucionado e retorna como solução um conjunto de ações que ele recomenda que sejam tomadas para que se atinja o objetivo desejado (RUSSELL; NORVIG, 2004). Neste contexto, os GAs são capazes de lidar com problemas que possuem um espaço de busca complexo, buscando soluções de alta qualidade por meio de técnicas de transições probabilísticas, conduzindo a população de cromossomos em direção a picos globais.

Em termos funcionais, nos GAs a operação de cruzamento (*crossovers*), representa um fator protagonista na construção de novas soluções candidatas com expectativa que sejam mais promissoras que as demais. O operador de cruzamento é o principal operador genético para formação de novas soluções descendentes, combinado soluções já conhecidas, criando uma forma especializada de busca em novas regiões do espaço de busca. O objetivo desta pesquisa é analisar o conjunto de operadores de cruzamento genético listados no Quadro 2, aplicados à instâncias do Problema do caixeiro viajante.

Quadro 2 – Tabela com os operadores de cruzamento estudados.

Operador	**Abreviação**	**Descrição**
Partially-Mapped Crossover	PMX	Dois pontos de corte são selecionados aleatoriamente em ambas estruturas dos cromossomos pais, em seguida a sequência de genes delimitada em um cromossomo pai é mapeando para o outro cromossomo formando um cromossomo descendente (subseção 3.2.1).
Cycle Crossover	CX	Os genes são subdivididos em ciclos, e os filhos são gerados pela seleção destes ciclos nos cromossomos pais (subseção 3.2.2).
Order Crossover 1	OX1	Um novo cromossomo é criado selecionando uma porção de genes de um cromossomo pai com o complemento dos genes do outro cromossomo pai, preservando relativamente a ordem de ocorrência dos genes nos pais (subseção 3.2.3).
Order based Crossover	OX2	Seleciona-se posições aleatoriamente dentro da estrutura de um cromossomo pai, e copie os genes presentes nestas posições para cromossomo filho preservando a ordem de precedência do outro cromossomo pai (subseção 3.2.4).
Position Based Crossover	POS	Seleciona-se aleatoriamente vários genes na estrutura do cromossomo pai copiando os genes para o cromossomo filho, mateando os genes no filho na mesma posição do cromossomo pai original (subseção 3.2.5).
Genetic edge recombination crossover	ER	Construa um cromossomo filho, selecionando arestas dos cromossomos pais com base em informações extraídas da tabela de adjacência de arestas dos pais (subseção 3.3.1).
Edge Assembly Crossover	EAX	Construa cromossomos filhos por meio da remoção de arestas de um cromossomo pai definidas pela criação de subciclos com arestas dos dois cromossomos pais e conectando os subciclos formados por meio de método de busca local(subseção 3.3.2).

Com esta tarefa em curso, no próximo capítulo é apresentado um esquema metodológico e experimental no qual o conjunto de operadores listados na Quadro 2 devem ser submetidos. Com base neste esquema e nos resultados obtidos será possível analisar o conjunto de operadores de maneira qualitativa.

CAPÍTULO

3

METODOLOGIA

Neste capítulo serão abordados operadores de cruzamento genético apontados na literatura especializada, como operadores clássicos para o TSP. Em nossa análise, iremos caracterizar o modelo matemático, detalhando as especificidades do modo de operação desses operadores no processo de criação de novas soluções para instâncias do TSP.

3.1 Considerações iniciais

Conforme supracitado no Capítulo 2, na ordem cronológica dos GAs, a operação de cruzamento situada entre as operações de seleção e mutação, caracteriza-se pela fase da formação de cromossomos filhos por meio da combinação genética dos cromossomos pais. Neste contexto, o operador de cruzamento genético define o modelo no qual esta combinação irá ocorrer. Um fator preponderante que influência no desempenho dos operadores de cruzamento quando aplicados ao TSP, está relacionado à modelagem dos cromossomos, ou mais precisamente, à escolha adequada da representação da codificação dos cromossomos conforme as características intrínsecas ao problema (CHATTERJEE; CARRERA; LYNCH, 1996).

Em (LARRANAGA *et al.*, 1999), são descritos vários tipos de codificação de cromossomos que são utilizados na modelagem de um GA típico aplicado ao TSP. Também são apresentados operadores de cruzamento e mutação que aplicam o modelo de representação de cromossomos por representação em ordem ou também conhecido por representação baseada em percurso. Este modelo de codificação é o mais natural para representar soluções do TSP em um GA, no qual um percurso para o caixeiro é representado por uma lista contendo n cidades, sendo que, a cidade j contida na i-ésima posição da lista, deverá ser a i-ésima cidade a ser visitada pelo caixeiro (ANDREICA; CHIRA, 2014).

Um outra forma para representar soluções candidatas para o TSP são baseadas em abordagens analíticas de arestas (JUNG; MOON, 2002), que caracterizam-se pela composição

dos cromossomos descendentes por meio da análise dos percursos formados nos cromossomos pais. Embora nesta abordagem a representação dos cromossomos não depende de um estilo único de codificação fixa, tais como ocorrem com os modelos baseado em percurso, as abordagens analíticas ainda são processos de recombinação no sentindo que combinam as características dos pais para gerarem os seus descendentes (JUNG; MOON, 2002).

Neste presente estudo, os operadores de cruzamento são classificados conforme a estrutura de codificação das soluções candidatas supracitadas.

O restante do capítulo está estruturado da seguinte forma. Na Seção 3.2 são apresentados operadores de cruzamento para o *TSP* com representação baseada em ordem. Na Seção 3.3, são apresentados operados de cruzamento para o *TSP* com representação baseada em abordagens analíticas. A Seção 3.4 é discutido um ensaio metodológico com o critério de avaliação dos operadores de cruzamento. Por fim, na Seção 3.5 são apresentadas as considerações finais sobre o capítulo.

3.2 Operadores de cruzamento genético com representação baseada em ordem

Nesta seção serão abordados alguns dos mais relevantes operadores de cruzamento genético para o TSP com abordagens baseadas em ordem, enunciados pela literatura. Posteriormente, cada um desses operadores vão ser submetidos a uma base de experimentos para análise e comparação de suas respectivas performances.

Representação baseada em ordem, também referendada como representação baseada em permutação ou representação baseada em percurso, é um dos modos mais naturais de representação cromossomal para instâncias do TSP no contexto dos GAs. Dado um percurso de n cidades, é possível representar este percurso como sendo um vetor de n elementos. Por exemplo, suponha um percurso formado pela ordem de cidades (2 4 5 1 3 8 7 6), podemos representá-lo por esta abordagem da seguinte forma:

2	**4**	**5**	**1**	**3**	**8**	**7**	**6**

.

Conforme a abordagem baseada em ordem, a cidade contida na i-ésima posição do vetor, deverá ser a i-ésima cidade na ordem de cidades percorridas. Como exemplo, a cidade 8 que está presente na sexta posição do vetor acima, deverá ser a sexta cidade visitada no trajeto a ser percorrido pelo caixeiro.

Em geral, abordagens baseadas em ordem são boas opções para representação dos cromossomos, porque são relativamente simples de serem implementadas e operadores com este estilo de codificação constituem operações fechadas, que não geram cromossomos incompletos ou com cidades repetidas (LARRANAGA *et al.*, 1999). Vários operadores de cruzamento

empregam este estilo de representação cromossomal. Entre os mais relevantes estão, *partially mapped crossover* (PMX), *cycle crossover* (CX), *order crossover* (OX_1), *order based crossover* (OX_2) e *position based crossover* (POS) (LARRANAGA *et al.*, 1999).

3.2.1 Partially Mapped Crossover

O *Partially Mapped Crossover* (PMX) foi proposto por (GOLDBERG; LINGLE JR., 1985). O PMX opera sobre duas soluções progenitoras (pais) para gerar duas soluções descendentes (filhos), transmitindo um segmento de informação dos progenitores para os descendentes. Uma partição de cidades contidas na estrutura cromossômica de uma solução pai é selecionada de forma aleatória e mapeada em conjunto com cidades remanescentes de outra solução pai que não estão contidas neste segmento para gerar um descendente. O Algoritmo 2 descreve o procedimento realizado pelo PMX na geração das soluções descendentes.

Algoritmo 2 – Operador de cruzamento PMX

```
1:  procedimento PMX(Pai_1[], Pai_2[])
2:      Filho_1 ← []                          ▷ Inicie uma lista vazia para o Filho_1
3:      Filho_2 ← []                          ▷ Inicie uma lista vazia para o Filho_2
4:      i, j ← selec_Aleator_Dois_Pontos_De_Corte_Do(Pai_1)
5:      Filho_1[i : j] ← Pai_2[i : j]
6:      Filho_2[i : j] ← Pai_1[i : j]
7:      para todo v ∈ [1,2] faça
8:          Lista_(v) ← selec_Cidades_Nao_Copiados_Para(Filho_(v), Pai_(v mod 2+1))
9:          para todo c ∈ Lista_(v) faça
10:             aux ← c
11:             enquanto i <= indexOf(c, Pai_(v)) <= j faça
12:                 c ← Filho_(v)[indexOf(c, Pai_(v))]
13:             fim enquanto
14:             Filho_(v)[indexOf(c, Pai_(v))] ← aux
15:         fim para
16:     fim para
17:     retorna Filho_1, Filho_2
```

O Algoritmo 2 recebe como entrada duas listas contendo cidades de duas soluções pais e retorna como saída duas listas representando as soluções filhas. Na linha 3 do algoritmo são selecionados aleatoriamente dois pontos de corte (i, j) entre o intervalo inferior e superior de uma das listas que representam as soluções pais, sendo i menor que j. Nas linhas 4 e 5, são efetuados mapeamentos dos pais para os filhos, entre as cidades pertencentes aos segmentos delimitados pelos pontos (i, j) de maneira cruzada, isto é, do Pai_1 para o $Filho_2$ e do Pai_2 para o $Filho_1$.

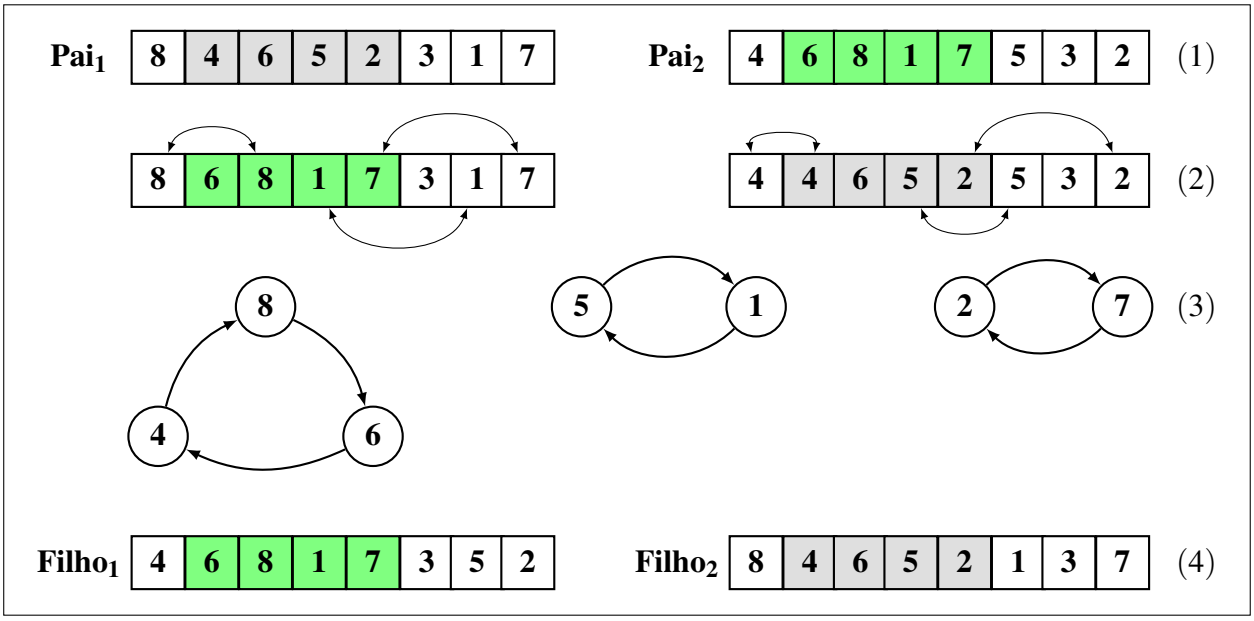
Entre as linhas 6 e 15 do Algoritmo 2, as cidades remanescentes dos pais que não estão contidas na região de corte (i, j) dos respectivos filhos, são selecionadas e inseridas nas devidas posições dos filhos de acordo com a ordem realizada pelo mapeamento, sem que cidades

duplicadas sejam deixadas na cadeia de cidades de cada solução filho. A linha 16 retorna às duas novas solução descendentes.

Como exemplo, na Figura 7 temos dois pais (**8 4 6 5 2 3 1 7**) e (**4 6 8 1 7 5 3 2**), respectivamente. O primeiro ponto de corte (i) está entre o primeiro e o segundo elemento e o segundo ponto de corte (j) está entre o quinto e o sexto elemento em cada solução pai. Neste caso, a sublista (**4 6 5 2**) delimitada por (i, j) no $\mathbf{Pai_1}$ é herdada pelo $\mathbf{Filho_2}$. Consecutivamente, a sublista (**6 8 1 7**) delimitada por (i, j) no $\mathbf{Pai_2}$ é herdada pelo $\mathbf{Filho_1}$.

Após as trocas realizadas (passo (2) da Figura 7), as posições externas aos segmentos nos filhos com cidades duplicadas são alteradas conforme a ordem do mapeamento realizada no passo (3) da mesma figura. Por exemplo, começando pelo $\mathbf{Pai_1}$, a cidades **1**, **7** e **8** estão duplicados após as trocas realizado pelo mapeamento, então devemos trocar as cidades nas posições repetidas externas a região de corte por cidades que estão nas mesmas posições das repetidas internas na região de corte do progenitor inicial, neste caso a cidade 1 que ocupa a sétima posição deve ser substituída pela cidade 5 que ocupa a mesma posição no progenitor inicial da cidade repetida 1 dentro da região de corte, o mesmo se aplica às outros cidades repetidas. Como resultado temos (**4 6 8 1 7 3 5 2**) e (**8 4 6 5 2 1 3 7**).

Figura 7 – *Partially Mapped Crossover - PMX*



Fonte: Elaborada pelo autor.

3.2.2 Cycle crossover

O operador *Cycle crossover* (CX) gera sua descendência por meio de dois pais, sendo que cada posição na estrutura cromossômica de um descendente deve ser preenchida por um elemento na mesma posição correspondente há um dos pais (OLIVER; SMITH; HOLLAND,

1987). Desta forma, o CX garante que os elementos presentes nos pais serão herdados pelos filhos mantendo a ordem de ocorrência. No Algoritmo 3 é explicitado um procedimento em que o CX gera sua descendência.

Algoritmo 3 – Operador de cruzamento CX

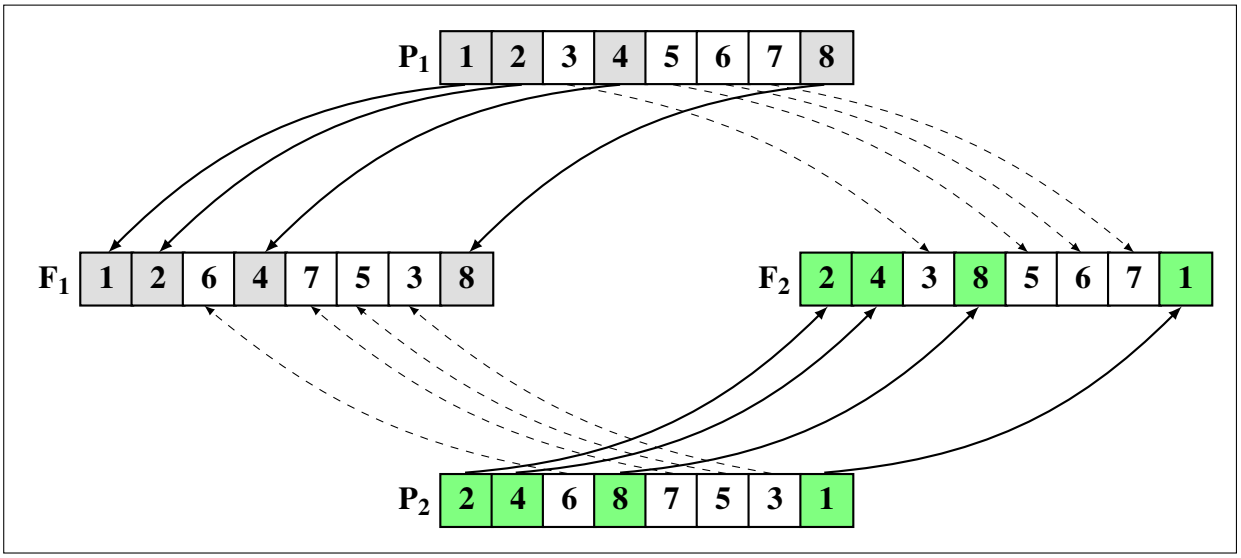
```
1:  procedimento CX(Pai_1[], Pai_2[])
2:      Filho_1 ← []                          ▷ Inicie uma lista vazia para o Filho_1
3:      Filho_2 ← []                          ▷ Inicie uma lista vazia para o Filho_2
4:      para todo i ∈ [1,2] faça
5:          Filho_(i) ← copy(Pai_(i mod 2+1))
6:          index ← 0
7:          repita
8:              Filho_(i)[index] ← Pai_(i)[index]
9:              index ← indexOf(Pai_(i mod 2+1), Filho_(i)[index])
10:         até index ≠ 0
11:     fim para
12:     retorna Filho_1, Filho_2
```

O Algoritmo 3 recebe como entrada duas soluções Pais e retorna como saída duas soluções Filhos. O algoritmo seleciona inicialmente de uma solução pai, a cidade que encontra-se na primeira posição e insere esta cidade na primeira posição do filho. No passo seguinte, seleciona-se a posição em que a cidade do passo anterior se encontra na cadeia de cidades do outro pai, a partir desta posição, a próxima cidade será selecionada do pai e inserida no filho. Este processo é repetido até que a primeira posição do filho seja novamente alcançada, completando assim o ciclo. Consequentemente, as posições remanescentes do filho que não fazem parte deste ciclo, são preenchidas por cidades com mesma posição pertencentes a outra solução pai. O processo interativo do Algoritmo 3 é aplicado aos dois pais.

A Figura 8 apresenta um esquema ilustrativo do Algoritmo 3. Suponha como exemplo duas soluções pais, $\mathbf{P_1}$ igual a [**1 2 3 4 5 6 7 8**] e $\mathbf{P_2}$ igual a [**2 4 6 8 7 5 3 1**].

Para gerar o primeiro filho ($\mathbf{F_1}$) consideramos a primeira posição de $\mathbf{P_1}$ ou $\mathbf{P_2}$; neste caso temos as cidades **1** e **2**. Suponha agora que selecionamos a cidade **1** e inserimos esta cidade na primeira posição de $\mathbf{F_1}$. Agora temos a cidade **8** devido a cidade **1** encontrar-se na oitava posição de $\mathbf{P_2}$. Inserimos então a cidade **8** na oitava posição de $\mathbf{F_1}$. De forma análoga ao passo anterior, inserimos as cidades **4** na quarta posição de $\mathbf{F_1}$ e **2** na segunda posição de $\mathbf{F_1}$. Ao fim deste passo, o primeiro ciclo está formado; neste ponto, o segundo ciclo deve ser iniciado a partir de $\mathbf{P_2}$. Considere a terceira posição de $\mathbf{F_1}$, ela deve ser ocupada pela cidade **6** de $\mathbf{P_2}$, isto implica que a quinta, sexta e sétima posições devem ser preenchidas pelas cidades **7**, **5** e **3** consecutivamente. No fim deste processo, $\mathbf{F_1}$ conterá as cidades [**1 2 6 4 7 5 3 8**]. Para gerar o $\mathbf{F_2}$, o processo é similar ao descrito no parágrafo anterior.

Figura 8 – *Cycle crossover - CX*



Fonte: Elaborada pelo autor.

3.2.3 Order crossover

O operador *Order crossover* (OX_1) foi proposto por (DAVIS, 1985). O OX_1 é uma estratégia baseada em ordem, onde a sequência em que as cidades aparecem é mais relevante do que simplesmente as posições que as mesmas devem ocupar dentro da estrutura de uma solução descendente.

Uma nova solução deve ser construída selecionando um segmento de cidades que está contido em uma solução pai. Com base neste segmento, seleciona-se cidades do outro progenitor que não estão neste segmento, preservando relativamente a ordem de precedência destas cidades neste progenitor. Junto com as cidades contidas no segmento do outro progenitor, estas cidades formam uma solução descendente (Algoritmo 4).

No Algoritmo 4, seleciona-se dois pontos de corte (i, j) de forma aleatória dentro da cadeia de cidades pertencentes a um pai, sendo que i deve ser menor que j. Com base nestes pontos de corte, dois segmentos de cidades são selecionados dos pais para os filhos consecutivamente, conforme as linhas 4 e 5 no algoritmo.

No passo seguinte do Algoritmo 4, inicia-se o processo que preenche as posições vazias deixadas nos filhos com cidades remanescentes de cada pai, sendo o pai escolhido oposto ao filho. Começando do **Filho$_1$** a partir do ponto j, as cidades são copiadas na ordem em que aparecem no **Pai$_2$** a partir também do ponto j, ignorando apenas as cidades que já estão no **Filho$_1$**. Quando a última posição do **Filho$_1$** ou **Pai$_2$** for alcançada, o processo deve continuar a partir da primeira posição. De forma similar, procedimento anterior deve ser replicado ao **Filho$_2$** em conjunto com o **Pai$_1$** para completar a solução do **Filho$_2$**.

Algoritmo 4 – Operador de cruzamento OX_1

procedimento $OX_1(Pai_1[], Pai_2[])$
  $Filho_1 \leftarrow []$ ▷ Inicie uma lista vazia para o $Filho_1$
  $Filho_2 \leftarrow []$ ▷ Inicie uma lista vazia para o $Filho_2$
  $i, j \leftarrow selec\_Aleator\_Dois\_Pontos\_De\_Corte\_Do(Pai_1)$
  $Filho_1[i:j] \leftarrow Pai_1[i:j]$
  $Filho_2[i:j] \leftarrow Pai_2[i:j]$
  para todo $v \in [1,2]$ **faça**
    $index \leftarrow j+1$
    $city \leftarrow Pai_{(v \bmod 2+1)}[j]$
    repita
      se $index > tamanho(Pai_{(v \bmod 2+1)})$ **então**
        $index \leftarrow 1$
      fim se
      enquanto $city \in Filho_{(v)}$ **faça**
        $aux \leftarrow indexOf(city, Pai_{(v \bmod 2+1)})$
        se $aux = tamanho(Pai_{(v \bmod 2+1)})$ **então**
          $city \leftarrow Pai_{(v \bmod 2+1)}[1]$
        senão
          $city \leftarrow Pai_{(v \bmod 2+1)}[aux+1]$
        fim se
      fim enquanto
      $Filho_{(v)}[index] \leftarrow city$
      $index \leftarrow index+1$
    até $index \neq i$
  fim para
  retorna $Filho_1, Filho_2$

Como exemplo do Algoritmo 4, considere as duas soluções progenitoras (pais) para o OX_1, ilustradas em seguida:

1 2 3 4 5 6 7 8 e **4 8 5 2 6 3 1 7** .

Suponha que os pontos de cortes (i, j) estejam na segunda e quarta posição de cada solução pai respectivamente, gerando dois segmentos de corte na estrutura dos cromossomos pais, conforme ilustrado em seguida:

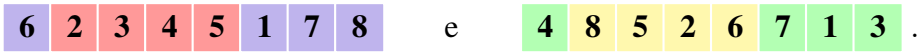
1 2 3 4 5 6 7 8 e **4 8 5 2 6 3 1 7** .

Na próxima etapa, duas soluções intermediarias são criadas, copiando ordenadamente às cidades que estão contidas nos segmentos delimitados pelos pontos (i, j) dos pais para duas listas vazias distintas entre si, respectivamente temos:

2 3 4 5 e **8 5 2 6** .

Conforme o Algoritmo 4, as posições vazias contidas nas soluções intermediárias criadas

no passo anterior, devem ser preenchidas com cidades remanescentes dos pais. O processo de cópia deve-se iniciar a partir da posição posterior ao ponto j em cada solução candidata, copiando ordenadamente e também a partir da posição posterior ao ponto j dos pais, as cidades contidas na solução pai oposta e que não pertencem ao respectivo segmento da solução candidata em questão. Quando a última posição de um pai ou um intermediário é alcançada, o processo deve continuar a partir da primeira posição até que todas as posições vazia dos filhos sejam preenchidas. Ao final do processo temos as duas novas soluções, conforme ilustrado em seguida:



3.2.4 Order based crossover

O *Order based crossover* ou OX_2 (SYSWERDA, 1991), seleciona aleatoriamente várias posições de uma estrutura do cromossomo progenitor. As cidades contidas nessas posições, devem ser sobrepostas no outro progenitor com a mesma ordem de precedência do progenitor inicial. No Algoritmo 5, temos um procedimento no qual o OX_2 opera para gerar soluções descendentes.

Algoritmo 5 – Operador de cruzamento OX_2

```
1:  procedimento OX2(Pai1[], Pai2[])
2:      Filho1 ← []                          ▷ Inicie uma lista vazia para o Filho1
3:      Filho2 ← []                          ▷ Inicie uma lista vazia para o Filho2
4:      para todo i ∈ [1,2] faça
5:          pos1 ← selec_Aleator_Conjunto_De_Posi_Do(Pai(i mod 2+1))
6:          Filho(i) ← copy(Pai(i))
7:          pos2 ← []
8:          j ← 1
9:          enquanto j <= tamanho(pos1) faça
10:             pos2[j] ← indexOf(Pai(i mod 2+1)[pos1[j]], Pai(i))
11:             j ← j + 1
12:         fim enquanto
13:         ordena(pos2)
14:         j ← 1
15:         enquanto j <= tamanho(pos2) faça
16:             Filho(i)[pos2[j]] ← Pai(i mod 2+1)[pos1[j]]
17:             j ← j + 1
18:         fim enquanto
19:     fim para
20:     retorna Filho1, Filho2
```

No Algoritmo 5, duas soluções pais são inseridas como entrada. Seleciona-se de cada solução pai, um conjunto de posição dentro da estrutura que armazenam as cidades. As cidades contidas nas posições selecionadas de cada pai são sobrepostas na solução pai oposta conforme

a ordem de precedência destas cidades no pai original, gerando duas novas soluções. Como exemplo, suponha duas soluções progenitoras, $\mathbf{Pai_1}$ e $\mathbf{Pai_2}$ respectivamente:

| 6 | 3 | 5 | 1 | 2 | 4 | 8 | 7 | e | 8 | 5 | 2 | 4 | 3 | 1 | 7 | 6 | .

No passo seguinte, conforme a linha 4 do Algoritmo 5, a cada passo interativo do algoritmo uma lista de posições é selecionada aleatoriamente de uma solução pai. Para o $\mathbf{Pai_1}$ e $\mathbf{Pai_2}$, considere que foram selecionadas as seguintes posições:

$\mathbf{Pai_1(pos_1)}$ e $\mathbf{Pai_2(pos_1)}$

(2) (5) (7) (1) (3) (8) .

No caso do $\mathbf{Pai_2}$, a primeira, terceira e oitava posições, contém as cidades **8**, **2** e **6** respectivamente. Essas cidades estão na sétima, quinta e primeira posição do $\mathbf{Pai_1}$. Desta forma, uma solução intermediária ($\mathbf{Filho_1}$) é criada copiando as cidades no $\mathbf{Pai_1}$ para uma lista vazia, com exceção das cidades que estão na primeira, segunda e sétima posição:

| | 3 | 5 | 1 | | 4 | | 7 | .

No $\mathbf{Pai_1}$, a segunda, quinta e sétima posições contém as cidades **3**, **2** e **8** respectivamente. Essas cidades estão presentes na quinta, terceira e primeira posições no $\mathbf{Pai_2}$. Uma segunda solução intermediária ($\mathbf{Filho_2}$) é criada então, copiando as cidades que estão em $\mathbf{Pai_2}$, com exceção das cidades contidas na primeira, terceira e quinta posições:

| | 5 | | 4 | | 1 | 7 | 6 | .

Na próxima etapa, as cidades que estão faltando nas soluções intermediárias devem ser inseridas com a mesma ordem que aparecem nas soluções pais. Começando pelo $\mathbf{Filho_1}$, são inseridas as cidades **8**, **2** e **6** conforme a ordem de incidência no $\mathbf{Pai_2}$. De maneira similar à regra anterior, a partir do $\mathbf{Filho_2}$ são inseridas as cidades **3**, **2** e **8**, obedecendo a ordem de incidência no $\mathbf{Pai_1}$. Como resultado, temos duas novas soluções:

$\mathbf{Filho_1}$ | 8 | 3 | 5 | 1 | 2 | 4 | 6 | 7 | e $\mathbf{Filho_2}$ | 3 | 5 | 2 | 4 | 8 | 1 | 7 | 6 | .

3.2.5 Position based crossover

O *Position based crossover* (*POS*) foi proposto por (SYSWERDA, 1991). Análogo ao OX_2, o POS também seleciona um conjunto variado de posições a partir de uma solução pai. Contudo, as cidades presentes nestas posições devem ser realocadas na outra solução pai, mantendo as posições que as mesmas ocupavam no progenitor original. No Algoritmo 6, apresenta-se o modo como o *POS* opera para gerar sua descendência.

Algoritmo 6 – Operador de cruzamento *POS*

```
1:  procedimento POS(Pai1[], Pai2[])
2:      posicoes ← selec_Aleator_Conjunto_De_Posi_Do(Pai1)
3:      Filho1 ← copy(Pai1)
4:      Filho2 ← copy(Pai2)
5:      para todo i ∈ posicoes faça
6:          Filho1[i] ← Pai2[i]
7:          Filho1[indexOf(Pai1, Filho1[i])] ← Pai1[i]
8:          Filho2[i] ← Pai1[i]
9:          Filho2[indexOf(Pai2, Filho2[i])] ← Pai2[i]
10:     fim para
11:     retorna Filho1, Filho2
```

Para exemplificar o processo descrito no Algoritmo 6, considere duas soluções pais, $\mathbf{P_1}$ e $\mathbf{P_2}$, respectivamente, e suponha que as posições selecionadas aleatoriamente estejam na segunda, terceira e sétima posições dos pais conforme ilustrado em seguida:

1	2	3	4	5	6	7	8

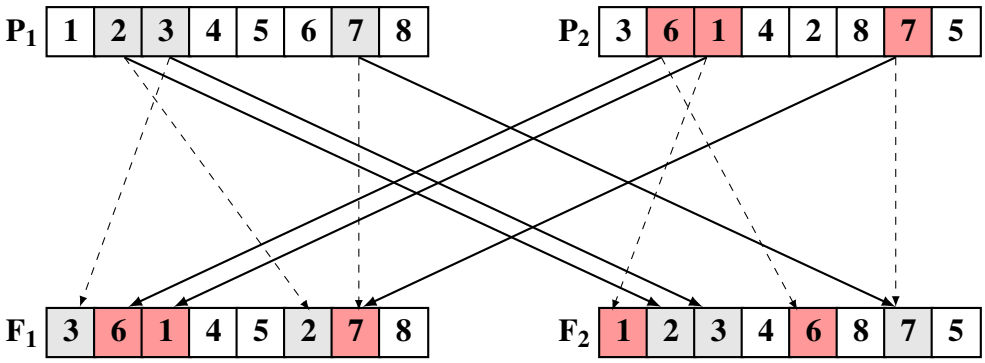
e

3	6	1	4	2	8	7	5

.

Para criar novas soluções, o próximo passo é efetuar a troca entre as cidades presentes nas posições selecionadas dos pais (Figura 9). Começando por $\mathbf{P_1}$, a cidades **6**, **1** e **7** presentes na segunda, terceira e sétima posições do $\mathbf{P_2}$ devem ser sobrepostas consecutivamente na segunda, terceira e sétima posição do $\mathbf{P_1}$, gerando o primeiro filho ($\mathbf{F_1}$). Após as trocas serem efetuadas em $\mathbf{F_1}$, as cidades duplicadas **1** e **6** presentes na primeira e sexta posições são realocadas pelas cidades **3** e **2** de $\mathbf{P_1}$, respectivamente, na primeira e sexta posições de $\mathbf{F_1}$, conforme o Algoritmo 6. De maneira similar, o $\mathbf{F_2}$ é gerado a partir de $\mathbf{P_2}$, igualmente ilustrado na Figura 9.

Figura 9 – *Position based crossover - POS*



Fonte: Elaborada pelo autor.

3.3 Operadores de cruzamento genético com representação baseada em abordagem analítica

Operadores de cruzamento com representação cromossomal baseados em abordagens analíticas, geram seus descendentes com ênfase em características extraídas dos cromossomos (indivíduos) pais. Para este estilo de representação, temos os operadores clássicos *genetic edge recombination crossover* (ER) e *edge assembly crossover* (EAX).

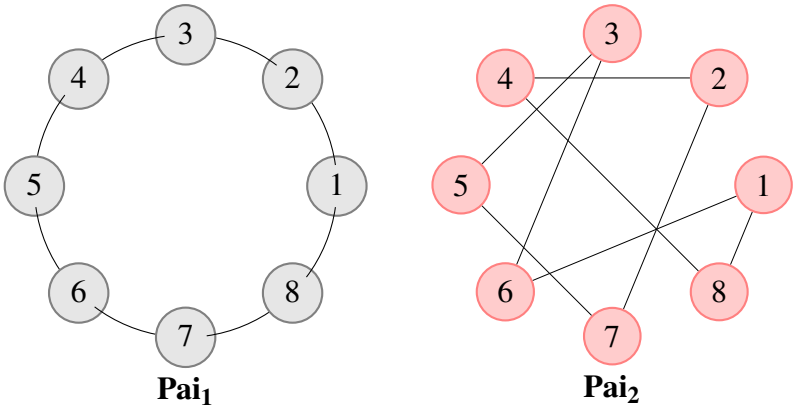
3.3.1 Genetic edge recombination crossover

O *Genetic edge recombination crossover* (ER) foi proposto por (WHITLEY; STARKWEATHER; FUQUAY, 1989). O ER opera com ênfase em informações obtidas de uma matriz de adjacência. Esta matriz de adjacência, também referenciada como tabela de arestas, é composta por interligações (arestas) de entrada e saída presentes nas cidades que estão contidas em duas soluções pais quaisquer.

A tabela de arestas, funciona como um mapa auxiliando o ER na composição das arestas presentes na solução descendente. Neste operador de cruzamento, as arestas são vistas como portadoras de informação hereditária e as informações assumidas por estas arestas são de extrema relevância no processo de composição de novos indivíduos. Desta forma, o ER atua com impeto para preservar as arestas dos pais com o objetivo de transmitir o máximo de informação hereditária para os descendentes (FUQUAY; WHITLEY, 1990).

Como exemplo, considere duas soluções pais, $\mathbf{Pai_1}$ e $\mathbf{Pai_2}$ conforme ilustrado na Figura 10.

Figura 10 – Duas soluções pais para o operador *ER*.



Fonte: Elaborada pelo autor.

Na Tabela 1, é dada a composição da tabela de arestas com todas as interligações entre as cidades presentes nas soluções progenitoras da Figura 10.

Tabela 1 – Tabela de arestas referente ao **Pai$_1$** e **Pai$_2$** - **ER**.

Cidades	**Cidades vizinhas**
1	2,6,8
2	1,3,4,7
3	2,4,5,6
4	2,3,5,8
5	3,4,6,7
6	1,3,5,7
7	2,5,6,8
8	1,4,7

A fim de preservar as arestas existentes nos pais, o *ER* opera em conjunto com a tabela de aresta e por meio do algoritmo descrito pelos passos abaixo para criar soluções descendentes (LARRANAGA *et al.*, 1999; STARKWEATHER *et al.*, 1991):

1. Selecione a primeira cidade de alguma das soluções pais e armazene esta cidade na solução descendente. Esta seleção pode ser feita aleatoriamente ou pelo mesmo critério de seleção do passo 4. Defina esta cidade como sendo a cidade corrente.

2. Remova todas as ocorrências da cidade corrente que se encontram na lista de cidades no lado direito da tabela de aresta. Marque a cidade corrente como sendo "visitada".

3. Caso exista alguma entrada na lista de cidades vizinhas referente a linha da cidade corrente na tabela de aresta prossiga para o passo 4, caso contrário, prossiga para o passo 5.

4. Selecione a cidade presente na lista de cidades vizinhas referente a cidade corrente que contém a menor quantidade de entradas em sua respectiva lista de cidades vizinhas na tabela de arestas. Atualize a cidade corrente para esta cidade. No caso de duas ou mais cidades com o menor conjunto de entradas na lista de vizinhos, escolha a cidade corrente de forma aleatória.

5. Se todas as cidades já foram visitadas, então pare o algoritmo. Caso contrário, escolha aleatoriamente uma cidade que não foi marcada como "visitada"e vá para o passo 2.

Prosseguindo o exemplo, a cidade inicial para solução descendente foi selecionada de um dos pais da Figura 10. Assumindo que **1** e **8** são as cidades inciais dos pais, pelo critério de menor número de vizinhos conforme o passo 4 do algoritmo, as cidades **1** e **8** possuem ambas três entradas na Tabela 1. Então, selecione a cidade **8** aleatoriamente para ser a cidade inicial da solução descendente. A cidade **8** é excluída da lista de vizinhos na Tabela 1.

A lista de cidades vizinhas para cidade **8** é composta pelas cidades **1**, **4** e **7**. As cidades **4** e **7**, possuem três arestas, enquanto que a cidade **1** possui duas arestas na Tabela 1. Então a

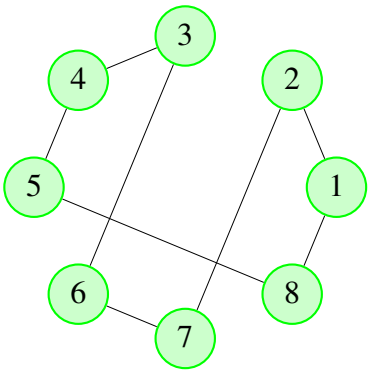
cidade **1** é selecionada como próxima cidade corrente e é removida da lista de cidades vizinhas da Tabela 1.

Na lista de cidades vizinhas para cidade **1** restaram as cidades **2** e **6**. Ambas as cidades **2** e **6** possuem três entradas na lista de vizinhos. Selecionamos a cidade **2** aleatoriamente como sendo a cidade corrente, removemos então esta cidade da lista de vizinhos. Na lista de vizinhos para cidade **2**, restaram as cidades **3**, **4** e **7**. Para este caso, as cidades **4** e **7** tem duas entradas na lista de arestas enquanto a cidade **3** tem três entradas. A cidade **7** é selecionada aleatoriamente como sendo a próxima cidade corrente para solução descendente.

Prosseguindo para a lista de vizinhos da cidade **7**, a cidade **5** possui 3 entradas na Tabela 1 e a cidade **6** possui 2 entradas na Tabela 1. A cidade **6** é selecionada como cidade corrente. A lista de vizinhos da cidade **6** é composta pelas cidades **3** e **5**, sendo que, ambas as cidades possuem a mesma quantidade de entradas na tabela de arestas. Aleatoriamente a cidade **3** é selecionada como a nova cidade corrente.

Por fim, restaram as cidades **4** e **5** na lista de arestas para a cidade **3**. Suponha que a cidade **4** seja selecionada aleatoriamente primeiro e, consequentemente, a cidade **5** por último. No fim o processo de formulação da nova solução descente está completo, conforme ilustrado na Figura 11.

Figura 11 – Solução descendente resultante do $\mathbf{Pai_1}$ e $\mathbf{Pai_2}$ aplicado ao *ER*.



Fonte: Elaborada pelo autor.

3.3.2 Edge Assembly Crossover

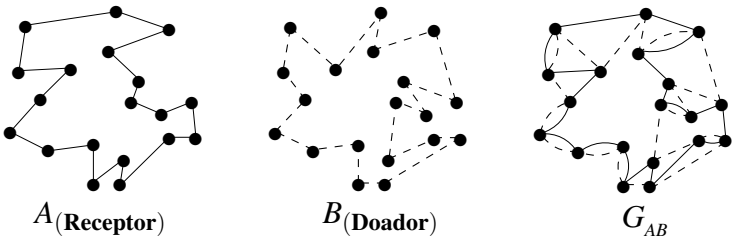
O *Edge Assembly Crossover* (EAX) é uma técnica proposta por (NAGATA; KOBAYASHI, 1997). Dado um par de soluções progenitoras quaisquer (pais), sendo que cada solução progenitora representa um determinado percurso para o caixeiro viajante, o EAX gera sua prole por meio de uma permutação de arestas, envolvendo a substituição de um conjunto reduzido de arestas do primeiro progenitor por uma quantidade similar de arestas relacionadas ao segundo progenitor. Este processo gera um conjunto de percursos desconexos, denominados de subciclos. Cada subciclo é formado por arestas de ambos os progenitores. Por fim, os subciclos devem

ser agrupados até formar um único percurso, de modo que para cada subciclo são adicionados novas arestas definidas por um mecanismo simplificado de busca gulosa, conectando o subciclo corrente com os subciclos adjacentes (LIU; ZENG, 2009).

Seja uma instância qualquer para o TSP, o primeiro passo é defini-la como sendo um grafo completo $G = (V, E, \omega)$ onde V é um conjunto de vértices que representam as cidades, E é o conjunto de arestas que representam as interligações entre todas as cidades de V e ω é o conjunto contendo os pesos das arestas em E. Selecione dois ciclos hamiltoniano A e B de G. Defina A como receptor e B como doador e faça $E_A \subset E$ e $E_B \subset E$ como os conjuntos que armazenam as arestas para as soluções A e B, respectivamente. Defina E_I como um conjunto auxiliar para armazenar as arestas da solução intermediária durante o processo de criação da solução descendente.

Apartir de A e B, gere um multigrafo não direcional $G_{AB} = (V, E_A \cup E_B)$ com as arestas de E_A e E_B sobrepostas de forma distintas (Figura 12).

Figura 12 – Um par de soluções progenitoras com o G_{AB} gerado - EAX.



Fonte: Elaborada pelo autor.

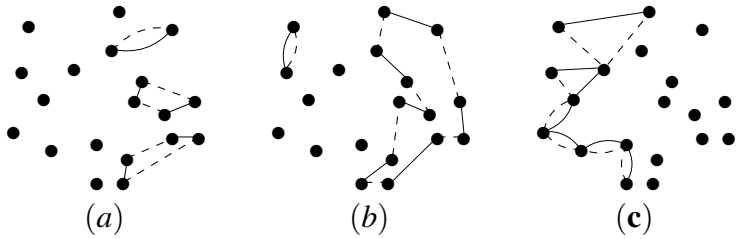
O EAX gera uma solução intermediária por meio da construção de pequenos ciclos (subciclos) em G_{AB}, com o complemento das arestas contidas em E_A gerando percursos desconexos. Em seguida, esses ciclos são conectados com a adição de uma quantidade reduzida de novas arestas formando uma nova solução descendente (NAGATAA; KOBAYASHIA, 2013). O processo básico do modo de operação do EAX é descrito em seguida (NAGATA; KOBAYASHI, 1999):

1. Gere *AB-ciclos* de G_{AB} e faça $C_{(j)}$ um *AB-ciclo* gerado. Um *AB-ciclo* pode ser definido por um circuito fechado em G_{AB} da seguinte forma, seja $C_{(j)} = \{ ar_1, ar_2, \ldots, ar_{2n} \}$ tal que, $\forall_i = (1,2,\ldots,n)$, $ar_{(2i-1)} \in E_A$ e $ar_{(2i)} \in E_B$ então ar_i e ar_{i+1} são adjacentes entre si (Figura **??**).

   O procedimento para gerar *AB-ciclos* é descrito em seguida (NAGATAA; KOBAYASHIA, 2013): Selecione um vértice v em V aleatoriamente. A partir de v, trace um percurso em G_{AB} selecionando arestas alternadas de E_A e E_B, aleatoriamente (caso duas possibilidades seja possível), até um *AB-ciclo* for alcançado no percurso traçado. Remova imediatamente de G_{AB} cada aresta alcançada no percurso traçado. Caso um *AB-ciclo* exista no percurso traçado, armazene e remova as arestas que constituem o percurso traçado. Se o percurso

traçado corrente não é vazio, reinicie o processo de busca pelo final do percurso atual. Caso contrário, reinicie o processo de busca selecionando um novo vértice v aleatoriamente, que esteja conectado pelo menos a uma aresta em G_{AB}. Caso não exista nenhuma aresta em G_{AB} termine o procedimento de busca.

Figura 13 – Exemplos de *AB-ciclos* gerados - EAX.



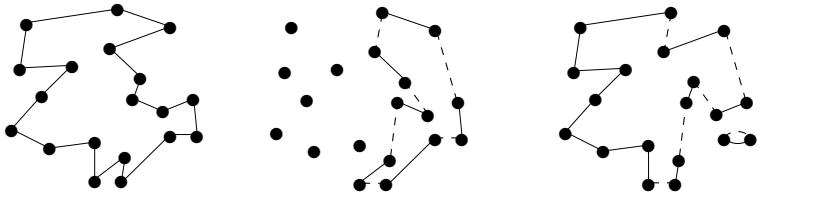
Fonte: Elaborada pelo autor.

Fonte: O Autor

2. Selecione *AB-ciclos* aleatoriamente com uma probabilidade p. Defina D como o conjunto união dos *AB-ciclos* selecionados.

3. Gere uma solução intermediária a partir de A (receptor), removendo as arestas de E_A que estão contidas em D e adicionando as arestas de E_B contidas em D em E_A, isto é, faça $E_I = (E_A \setminus (D \cap E_A)) \cup (D \cap E_B)$ (Figura 14). Em uma solução intermediária podem ocorrer um ou mais subciclos.

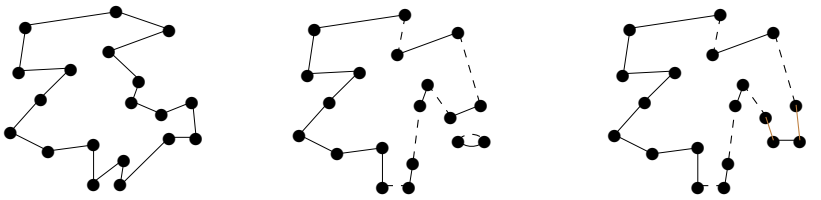
Figura 14 – Geração de uma solução intermediária - EAX.



Fonte: Elaborada pelo autor.

4. Gere uma nova solução descendente (Figura 15), conectando os percursos desconexos que foram criados na solução intermediária com arestas definidas por meio de procedimento de busca local explicitado no Algoritmo 7 (NAGATA; KOBAYASHI, 1999).

Figura 15 – Percurso final validado - EAX.



Fonte: Elaborada pelo autor.

Algoritmo 7 – Procedimento de busca local - EAX

Requer: Solução Intermediária.
1: Faça k ser o número de subciclos contidos na solução intermediária;
2: $s \leftarrow k$;
3: Faça U_i (i = 1,2, ... , k) o conjunto de arestas do subciclo i;
4: **repita**
5: $i^* \leftarrow \arg_{i=(1,\ldots,s)} \min |U_i|$;
6: $j^* \leftarrow \arg_{j(\neq i^*)} \min \omega_{(i^* j)}$;
7: $\{e, e'\} \leftarrow \arg_{(e \in U_{i^*}, e' \in U_{j^*})} \min \omega_{(i^* j^*)}$;
8: $U_{i^*} \leftarrow (\, U_{i^*} \cup U_{j^*} - \{e, e'\}\,) \cup \{e'', e'''\}$;
9: $U_{j^*} \leftarrow U_s$;
10: $s \leftarrow s - 1$;
11: **até** $s = 1$
retorna U_s
12: Onde $|U_i|$ representa a quantidade de arestas incluidas em U_i,
13: $\omega_{(ij)}$ = $\min_{e \in U_i, e' \in U_j}$ -dist(e) -dist(e') +dist(e'') +dist(e'''), sendo ($e = uv, e' = u'v'$), então (e'', e''') é definido por ($e'' = uv', e''' = u'v$) ou ($e'' = u'v, e''' = uv'$) . =0

3.4 Metodologia de avaliação

No Algoritmo 8 é apresentado o esboço da estratégia evolucionária conforme descrito no Capítulo 2 e que será adotado neste trabalho. O objetivo é submeter o conjunto de operadores de cruzamento descritos nas Seções 3.2 e 3.3, a uma base diversificada de instâncias para o TSP. Posteriormente, os resultados obtidos serão descritos conforme análise de performance dos operadores.

Os parâmetros de controle (taxa de mutação, taxa de cruzamento, tamanho da população e condição de parada), assim como outros operadores auxiliares (operador de seleção, mutação e reinserção), foram definidos conforme as características específicas dos operadores de cruzamento testados, restringindo-se o mais próximo das condições relatadas nos trabalhos originais ou em pesquisas relacionadas com o escopo deste trabalho:

- **Tamanho da população** (λ): Foi usado o valor λ = 600 para instâncias com até 1060 cidades.

- **Operador de Seleção**: Seleção baseada em seleção por roleta (2.3.4).

- **Operador de Mutação (γ)**: Este processo é baseado em mutação inversa (HOLLAND, 1975), em que, dois pontos na estrutura do cromossomo são selecionados aleatoriamente e permutados entre si. A taxa de mutação foi fixada em, $\gamma = 0.01$. O EAX não utiliza operação de mutação em seu ciclo evolutivo (NAGATA; KOBAYASHI, 1997).

- **Operador de Reinserção**: Foi utilizado o método de reinserção baseada em aptidão, no qual o conjunto dos melhores cromossomos pais e filhos são selecionados para a próxima geração de forma direta (2.3.7).

- **Condições de parada**: para os testes realizados, o número de avaliações é estabilizado em 50.000 ciclos evolutivos do GA. O algoritmo é finalizado se a cada mil ciclos evolutivos consecutivos não houver um aperfeiçoamento de aptidão em relação à melhor solução encontrada.

Algoritmo 8 – Algoritmo evolucionário básico

```
1: gere uma população de tamanho N
2: avalie cada indivíduo da população
3: repita
4:     selecione Pai_1 e Pai_2 a partir da população;
5:     Filho ← cruzamento(Pai_1, Pai_2);
6:     mutação(Filho);
7:     avalie(Filho);
8:     reinserção(população,Filho);
9: até (condição de parada);
       retorna o melhor indivíduo da população;
```

Para os experimentos realizados, foram selecionados 20 instâncias para o TSP, variando entre 16 e 1060 cidades (Tabela 2), derivadas da base dados **TSPLIB** (REINELT, 1991). A **TSPLIB** é uma extensa biblioteca, que provê um conjunto variado de arquivos para testes, para uma classe variada de problemas de otimização combinatória derivadas do TSP, tais como, TSP simétricos, TSP assimétricos, ciclos Hamiltonianos, entre outros.

As instâncias contidas na Tabela 2 representam um conjunto de dados específicos para casos de testes baseados em TSP simétricos (Seção 1.2), restringindo-se o escopo deste trabalho. As cidades presentes nestas instâncias são apresentadas em forma de coordenadas da seguinte maneira. Seja c_i uma cidade definida pelas coordenadas (x_i,y_i), então a distância entre duas cidades i e j ($dist\ (c_i, c_j)$) pode ser obtida pelo cálculo da distância euclidiana conforme Equação 3.1.

$$dist(c_i, c_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \tag{3.1}$$

Logo, o computo de todas as distâncias para um determinado percurso (π) percorrido pelo caixeiro é dado pela Equação 3.2:

$$Percurso(\pi) = \sum_{i=1}^{N-1} dist(c_{\pi(i)}, c_{\pi(i+1)}) + dist(c_{\pi(N)}, c_{\pi(1)}). \tag{3.2}$$

Em geral, análise de desempenho de algoritmos aplicado ao TSP são fragmentadas por dois aspectos fundamentais. Pela qualidade da solução encontrada ou pelo tempo de execução gasto no processo de busca da solução ótima para TSP (LIU; ZENG, 2009). Todavia, no contexto deste trabalho, foram considerados somente comparações feitas pela qualidade da solução encontrada. Este fato é devido a dificuldade encontrada em estabelecer o mesmo estado do ambiente de execução computacional, para todos os testes realizados.

A medida de qualidade das soluções resultantes dos métodos aqui relatados é dada como o erro percentual (%) em relação a melhor solução encontrada e em relação a média das soluções obtidas em 10 execuções, para os operadores de cruzamento aplicados as instâncias da Tabela 2.

O valor do erro percentual em relação a melhor solução encontrada é dada pela diferença com o valor ótimo conhecido de cada instância, conforme a Equação 3.3

$$\textbf{erro}(\Psi) = [(\text{melhor - ótimo}) \div \text{ótimo}] * 100 \tag{3.3}$$

onde a melhor solução representa a solução de menor custo em 10 ensaios de execução. O valor ótimo corresponde ao valor previamente conhecido na base de testes para o melhor percurso mínimo conhecido de cada instância.

A medida de qualidade das soluções em relação a média, é também dada pelo erro percentual (%), definido por

$$\textbf{erro}(\Phi) = [(\text{média - ótimo}) \div \text{ótimo}] * 100 \tag{3.4}$$

onde a média é o cálculo do valor médio das soluções encontradas em 10 ensaios de execução. Novamente, o valor ótimo representa o melhor valor conhecido para um percurso mínimo de uma determinada instância.

O desvio padrão (σ), dado pela Equação 3.5, é utilizado para quantificar o valor de dispersão em relação ao valor do cálculo do **erro(Ψ)** das 10 soluções encontradas por cada operador de cruzamento, onde N representa o número de execuções e x_i representa o i-ésimo valor amostral no espaço de 10 soluções.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}, \text{ onde } \mu = \frac{1}{N}\sum_{i=1}^{N}x_i \tag{3.5}$$

3.5 Considerações finais

Neste capítulo foram apresentados os operadores de cruzamento genético que serão aplicados ao Problema do Caixeiro Viajante. O operador de cruzamento é um componente

Tabela 2 – Tabela com 20 instâncias selecionadas da base de dados TSPLIB.

Instâncias	**Cidades**	**Valor ótimo**
ulysses16	16	6859
att48	48	10628
berlin52	52	7542
eil76	76	538
kroC100	100	20749
bier127	127	118282
gr137	137	69853
pr144	144	58537
kroB150	150	26130
kroA200	200	29368
pr264	264	49135
a280	280	2579
pr299	299	48191
lin318	318	42029
d493	493	35002
att532	532	27686
p654	654	34643
d657	657	48912
gr666	666	294358
u1060	1060	224094

importante na construção de GAs para obter soluções de boa qualidade no espaço de busca do TSP dado que ele é o principal mecanismo para a geração de novas soluções. No Capítulo 4 serão feitos experimentos com esses operadores, aplicando-os às instâncias de testes listadas na Tabela 2.

CAPÍTULO

4

IMPLEMENTAÇÃO E RESULTADOS

Neste capítulo, apresenta-se informações gerais sobre os resultados numéricos do trabalho, discute-se o comportamento e o desempenho dos operadores de cruzamento estudo e aplicado ao TSP. São demonstrados gráficos explicitando o comportamento desses operadores conforme seus respectivos desempenhos.

4.1 Considerações iniciais

Os operadores de cruzamento genético descritos no Capítulo 3 foram experimentados em 20 instâncias do TSP simétrico com a quantidade de cidades variando entre 16 e 1016 (Tabela 2). Todas as instâncias foram selecionadas da base para testes *benchmark* TSP:TSPLIB (REINELT, 1991).

Os códigos do GA proposto, foram implementados em *python*, que é uma linguagem de programação interpretada de tipagem dinâmica, bastante utilizada pela comunidade acadêmia por ser considerada de fácil aprendizado e alto nível de codificação. Para otimizar o processo de execução do GA, o código do programa foi executado utilizando *Pypy*, um interpretador e compilador *just-in-time* [1] com o foco no aumento de velocidade e eficiência na execução de código *python*.

Os experimentos foram executados em um ambiente computacional com 20 processadores Intel Xeon de 2.0 *GHz* e 36 *GBytes* de memória RAM. Para cada experimento em execução é alocada um único núcleo, e cada operador de cruzamento inserido no GA descrito no Algoritmo 8, é executado 10 vezes sobre cada instância da Tabela 2, coletando 10 resultados por experimento.

Por fim, neste Capítulo, são descritos os resultados alcançados pelos experimentos

[1] Em computação, a compilação *just-in-time* (JIT) trate-se de um processo de compilação feita durante o tempo de execução do código fonte, em vez de antes do processo de execução.

realizados. São apresentadas tabelas e gráficos com os resultados obtidos, para análise de comportamento dos operadores de cruzamento testados. Também apresenta-se conclusões pontuais sobre a pesquisa realizada e sobre o presente capítulo.

4.2 Resultados numéricos do trabalho

Os resultados experimentais do trabalho foram obtidos por meio da execução do GA para cada operador de cruzamento genético relatado no trabalho, aplicando tal operador às 20 instâncias do TSP listados na Tabela 2. Os operadores *Partially-Mapped Crossover* (PMX), *Cycle Crossover* (CX), *Order Crossover 1* (OX1), *Order Based Crossover* (OX2), *Position Based Crossover* (POS), *Genetic Edge Recombination Crossover* (ER) e *Edge Assembly Crossover*(EAX) foram analisados entre si, e os aspectos fundamentais dos dados obtidos são relatados na Tabela 3.

Basicamente na Tabela 3, a primeira coluna é reservada para classificar as instâncias do TSP com os seus respectivos valores ótimos (menor percurso conhecido) assinalados entre parenteses. Em seguida, para cada operador de cruzamento são mostrados os respectivos valores do erro percentual do valor ótimo em relação a melhor solução encontrada (Equação 3.3), o erro percentual do valor ótimo em relação a média das 10 melhores soluções encontradas (Equação 3.4), e o desvio padrão (Equação 3.5).

Por fim, para cada instância de teste listada na Tabela 2 foi aplicado o mesmo critério de avaliação, sendo 10 ciclos de execução para cada operador de cruzamento, totalizando 70 simulações por instância. Logo os resultados finais da Tabela 3 são obtidos por um conjunto total de 1400 avaliações.

Os campos da Tabela 3 onde os valores são assinalados por (*), indicam que o tempo máximo de 168 horas preestabelecido para a execução das 10 amostras foi excedido. Neste caso, os cálculos são realizados levando em consideração os dados gerados até o exato momento da interrupção do processo de execução.

Tabela 3 – Resultado do processo de busca evolutiva obtidos para instâncias TSP.

Instâncias (TSPLIB)		**PMX**	**CX**	**OX1**	**OX2**	**POS**	**ER**	**EAX**
ulysses16	**Melhor** (Ψ)	0.52	0.23	0.00	0.00	0.00	0.00	0.00
(6859)	**Média** (Φ)	6.40	3.55	0.08	2.14	1.62	0.11	0.00
	Desv/Pad (σ)	6.27	5.27	0.16	2.17	1.54	0.19	0.00
att48	**Melhor** (Ψ)	19.30	29.24	1.34	15.04	25.15	0.18	0.00
(10628)	**Média** (Φ)	38.01	42.84	3.83	33.73	38.44	1.97	0.00
	Desv/Pad (σ)	9.23	9.92	2.19	12.88	14.10	1.12	0.00

continua na próxima página

Tabela 3 – continuação

Instâncias (TSPLIB)		**PMX**	**CX**	**OX1**	**OX2**	**POS**	**ER**	**EAX**
berlin52	**Melhor** (Ψ)	22.42	24.34	0.00	19.84	24.52	0.00	0.00
(7542)	**Média** (Φ)	36.66	38.63	6.97	32.74	34.81	0.78	0.00
	Desv/Pad (σ)	8.79	9	4.04	8.64	9.67	0.97	0.00
eil76	**Melhor** (Ψ)	33.45	32.34	6.13	35.31	31.59	3.15	0.00
(538)	**Média** (Φ)	49.05	50.18	9.70	45.81	46.78	5.85	0.00
	Desv/Pad (σ)	10.53	10.42	2.36	5.99	8.37	1.26	0.00
kroC100	**Melhor** (Ψ)	59.02	70.10	7.91	84.04	80.11	3.33	0.00
(20749)	**Média** (Φ)	85.18	103.88	14.35	102.62	97.87	7.98	0.00
	Desv/Pad (σ)	16.71	18.78	3.70	14.16	12.81	3.05	0.00
bier127	**Melhor** (Ψ)	41.98	40.26	5.34	47.85	41.38	2.43	0.00
(118282)	**Média** (Φ)	52.48	55.08	11.26	59.27	58.68	5.48	0.00
	Desv/Pad (σ)	7.81	9.17	3.18	6.08	8.35	2.95	0.00
gr137	**Melhor** (Ψ)	64.64	86.50	12.09	86.40	74.41	9.03	0.00
(69853)	**Média** (Φ)	102.04	113.36	20.60	111.93	98.12	15.60	0.00
	Desv/Pad (σ)	21.79	18.40	4.07	14.95	14.58	4.00	0.00
pr144	**Melhor** (Ψ)	166.26	155.39	21.37	167.48	165.47	4.22	0.00
(58537)	**Média** (Φ)	218.19	198.73	28.7	211.22	221.11	14.33	0.00
	Desv/Pad (σ)	26.85	28.74	5.15	22.95	28.60	5.87	0.00
kroB150	**Melhor** (Ψ)	76.75	90.57	11.74	86.02	76.58	9.03	0.00
(26130)	**Média** (Φ)	120.72	121.73	18.12	117.25	110.13	12.80	0.00
	Desv/Pad (σ)	29.04	19.21	2.98	17.32	21.71	2.70	0.00
kroA200	**Melhor** (Ψ)	108.94	109.18	19.58	129.49	109.30	20.94	0.00
(29368)	**Média** (Φ)	137.29	133.42	28.24	153.66	133.53	26.34	0.00
	Desv/Pad (σ)	16.70	15.31	6.63	19.98	13.82	3.82	0.00
pr264	**Melhor** (Ψ)	338.63	308.35	34.17	330.53	340.72	27.31	0.00
(49135)	**Média** (Φ)	413.38	372.93	40.10	398.60	388.89	38.16	0.00
	Desv/Pad (σ)	42.17	34.59	5.80	36.39	36.04	5.72	0.00

continua na próxima página

Tabela 3 – continuação

Instâncias (TSPLIB)		PMX	CX	OX1	OX2	POS	ER	EAX
a280	**Melhor** (Ψ)	141.10	142.34	34.85	158.23	145.59	32.49	0.00
(2579)	**Média** (Φ)	166.93	171.31	45.72	180.88	167.3	45.25	0.00
	Desv/Pad (σ)	14.71	16.72	7.01	12.04	13.22	7.27	0.00
pr299	**Melhor** (Ψ)	159.28	153.90	40.79	199.56	157.46	42.27	0.00
(48191)	**Média** (Φ)	187.57	186.77	49.87	215.64	187.34	54.36	0.00
	Desv/Pad (σ)	19.30	21.42	4.53	9.55	18.66	7.76	0.00
lin318	**Melhor** (Ψ)	176.17	164.10	41.50	178.19	158.43	47.26	0.00
(42029)	**Média** (Φ)	191.80	191.68	51.36	212.78	181.56	58.01	0.00
	Desv/Pad (σ)	11.72	15.79	6.44	13.46	14.59	6.18	0.00
d493	**Melhor** (Ψ)	140.25	142.05	29.24	136.03	129.89	33.01	0.00
(35002)	**Média** (Φ)	157.01	154.33	40.58	160.06	150.44	43.69	0.00
	Desv/Pad (σ)	10.05	7.68	6.19	13.79	9.50	5.00	0.00
att532	**Melhor** (Ψ)	209.62	199.40	51.54	227.67	195.52	40.94	0.00
(27686)	**Média** (Φ)	228.48	233.19	62.85	252.01	235.90	63.62	0.00
	Desv/Pad (σ)	14.39	13.52	7.52	12.74	19.18	14.09	0.00
p654	**Melhor** (Ψ)	801.44	723.60	83.78	810.46*	834.41	125.02*	0.00
(34643)	**Média** (Φ)	897.94	895.78	97.31	1017.15*	979.18	131.19*	0.00
	Desv/Pad (σ)	56.01	102.77	11.81	127.11	97.07	4.70	0.00
d657	**Melhor** (Ψ)	213.33	228.82	85.41	246.75*	213.8	95.09*	0.00
(48912)	**Média** (Φ)	237.55	249.23	97.13	256.97*	238.55	100.98*	0.00
	Desv/Pad (σ)	14.67	21.78	8.56	8.03	13.19	4.50	0.00
gr666	**Melhor** (Ψ)	206.98	219.44	58.56	258.76*	197.37	80.69*	0.00
(294358)	**Média** (Φ)	234.60	230.46	74.79	261.79*	229.07	88.89*	0.00
	Desv/Pad (σ)	17.26	9.77	8.93	3.03	17.23	8.20	0.00
u1060	**Melhor** (Ψ)	378.40	355.58	120.44	417.57*	373.81*	186.76*	0.00
(224094)	**Média** (Φ)	402	394.96	133.68	417.57*	392.24*	186.76*	0.00
	Desv/Pad (σ)	10.27	30.6	8.71	0.00	20.65	0.00	0.00

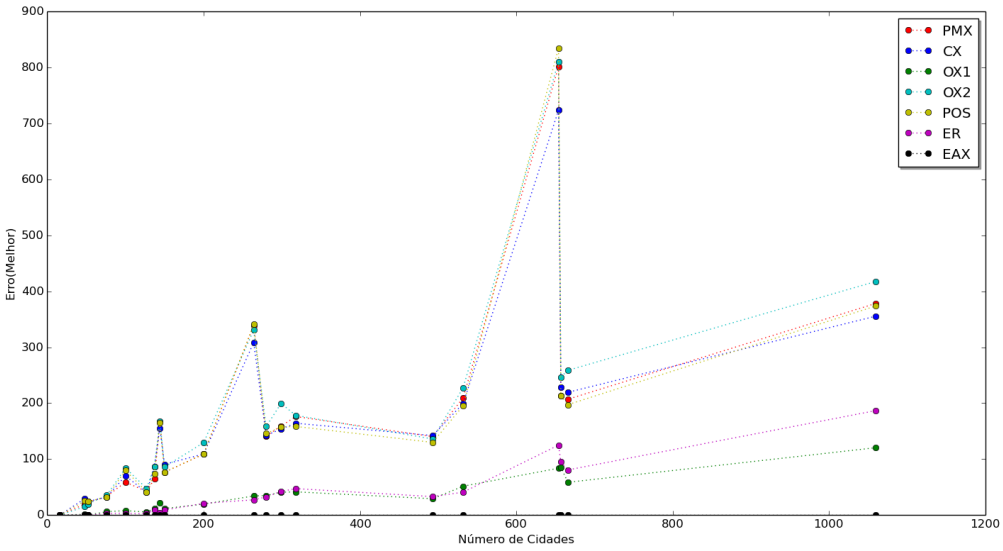
Conforme os resultados numéricos da Tabela 3, o operador EAX obteve o melhor desempenho alcançando o valor ótimo para todas as instância ao qual foi submetido. Os operadores OX1 e ER conseguiram obter o valor ótimo em relação ao erro da melhor solução encontra por cada operador, somente para duas instâncias da Tabela 3, porém vale ressaltar que estes operador obtiveram resultados satisfatória com o valor do erro próximo de zero para várias instância da tabela. Os operadores POS e OX2 alcançaram o valor ótimo em relação ao erro da melhor soluções encontrar somete para uma instância, os demais resultados destes operadores foram insatisfatórios quando comparados aos operadores EAX, OX1 e ER. Por fim, os operadores PMX e CX não obtiveram resultados ótimos para nenhuma instância ao qual foram submetidos, sendo os operadores com os piores resultados do trabalho.

4.3 Comportamento dos operadores de cruzamento genético no escopo do trabalho

Na Tabela 3 estão os resultados numéricos obtidos pelos experimentos realizados no trabalho. Nesta seção, os operadores de cruzamentos relatados no Capítulo 3 são analisados conforme os resultados encontrados, reportados na Tabela 3.

A Figura 16, apresenta o comportamento dos operadores de cruzamento genético quando aplicados ao conjunto de 20 instâncias de testes na Tabela 2. A análise do gráfico ocorre pelo valor do erro da melhor solução encontrada por cada operador em relação ao número de cidades das instâncias.

Figura 16 – Gráfico do comportamento dos operadores em relação ao erro da melhor solução.



Fonte: Elaborada pelo autor.

Com base na dispersão dos pontos no gráfico da Figura 16, é possível distinguir a formação de três grupos de operadores de cruzamento com comportamentos distintos. Os grupos são delimitados entre os operadores (PMX, CX, OX2, POS), os operadores (OX1, ER) e por último, o operador EAX.

O EAX obteve o melhor desempenho entre os demais operadores relatados no trabalho, encontrando o valor ótimo para todas as instâncias de testes ao qual foi submetido. O argumento factível para interpretação do bom desempenho do EAX está associado a concepção do processo de reprodução dos descendentes sob dois aspectos fundamentais. Em primeiro plano, temos que, somente arestas já concebidas nas soluções pais podem ser transmitidas para as soluções filhos, preservando assim características preexistentes nos pais também nos filhos. Esse comportamento se dá por meio da criação dos *AB-ciclos* (Figura **??**) e durante a fase de formação da solução intermediária (Figura 14). Em segundo plano, no EAX, qualquer adição de novas características (arestas) nos cromossomos filhos, é feito por uso de busca local (Figura 15), evidenciado também o comportamento híbrido deste operador. É importante salientar que diferente dos GAs híbridos tradicionais (Seção 2.4), no EAX o processo híbrido acontece junto ao processo de cruzamento, e apenas uma porção da estrutura dos cromossomos são alteradas com a adição de um conjunto reduzido de novas arestas pelo método de busca local (Algoritmo 7).

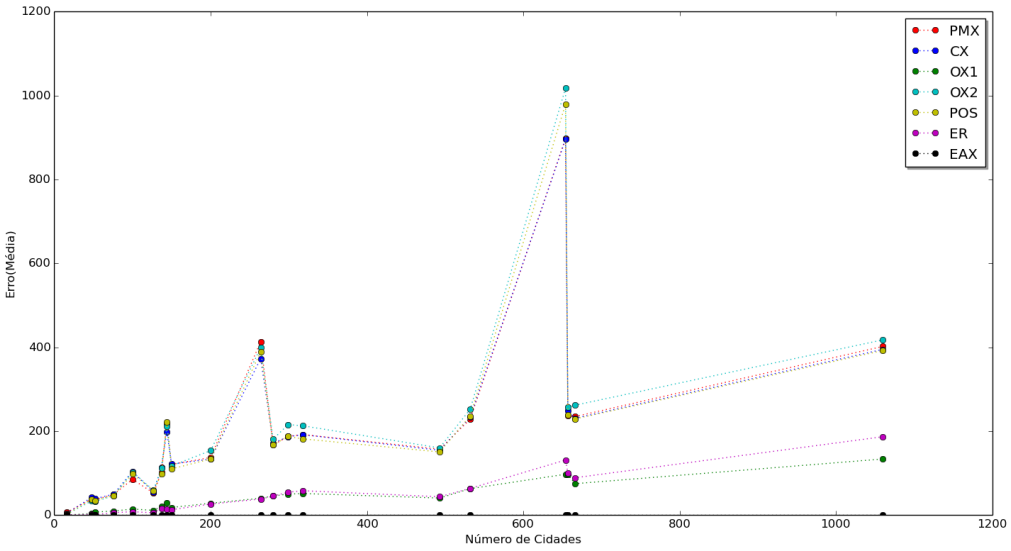
Os operadores OX1 e ER, obtiveram soluções de boa qualidade para instâncias com até 300 cidades, e soluções razoáveis para instâncias que excederam o número de 500 cidades. Na estimativa total, o ER obteve melhores resultados para as instâncias em que o número de cidades é relativamente menor (instâncias com até 280 cidades), enquanto o OX1 obteve melhores resultados para instâncias em que o número de cidades é mais elevado. Entretanto, em relação as projeções no gráfico na Figura 16, os dois operadores possuem um comportamento congruente, sendo os respectivos desempenhos dos dois operadores estão próximos entre si.

Os operadores restantes (PMX, CX, OX2 e POS) conseguiram obter resultados satisfatórios somente para instâncias pequenas, com até 52 cidades. Ambos os operadores possuem comportamento semelhante no gráfico da Figura 16, porém, possuem os piores resultados numéricos do trabalho.

Analisando o processo de formação das soluções filhos pelos dois grupos de operadores de cruzamento, PMX (Seção 3.2.1), CX (Seção 3.2.2), OX2 (Seção 3.2.4), POS(Seção 3.2.5), OX1 (Seção 3.2.3) e ER (Seção 3.3.1), é possível observar que tanto o OX1 quanto o ER priorizam que sejam transmitidos para os seus descendentes, a ordem relativa em que as cidades aparecem nos percursos representados pelas soluções pais. Os demais operadores como por exemplo, PMX e CX, se esforçam para transmitir aos filhos, a posição absoluta de cada cidade dentro da estrutura dos cromossomos pais (LARRANAGA *et al.*, 1999). Logo, mesmo que empiricamente pelos resultados obtidos da Tabela 3 e do comportamento do gráfico da Figura 16, é factível afirmar que, transmitir a ordem relativa de precedência das cidades nos cromossomos pais para os filhos é mais relevante do que simplesmente transmitir a posição absoluta que cada

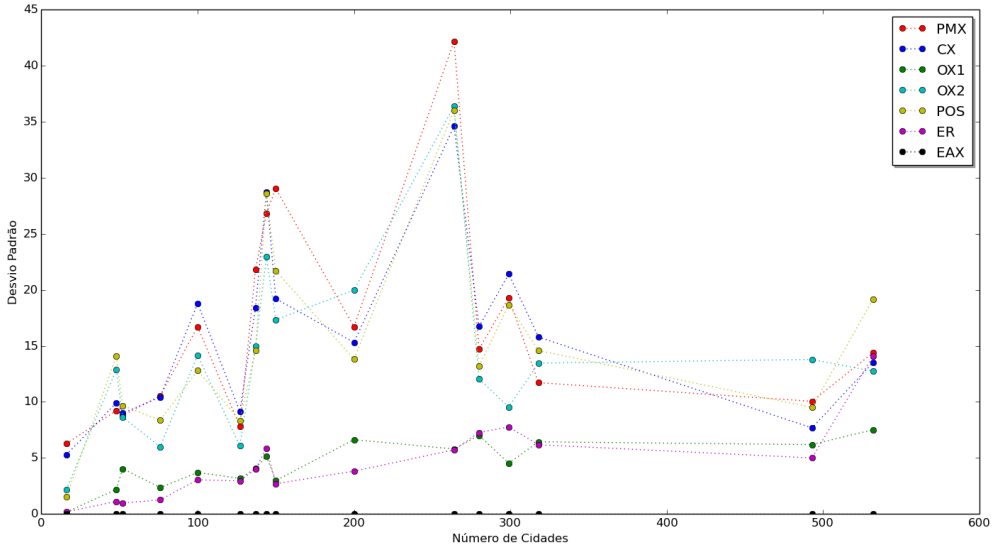
cidade ocupa.

Figura 17 – Comportamento dos operadores em relação ao erro médio.



Fonte: Elaborada pelo autor.

Figura 18 – Comportamento dos operadores em relação desvio padrão do cálculo do erro pelas 10 soluções encontradas.



Fonte: Elaborada pelo autor.

A Figura 18 apresenta o gráfico do desvio padrão (Equação 3.5) do erro percentual das 10 execuções realizadas por cada operador de cruzamento genético. Pode-se notar que o operador EAX produz um desvio padrão do erro igual a 0 para todas as instâncias, evidenciando o fato

dele encontrar a solução ótima em todas as execuções, para todas as instâncias. Novamente, com exceção do operador EAX, podemos notar dois grupos de operadores, o grupo composto pelos operadores OX1 e ER, com um desvio padrão do erro percentual em torno de 5%, e o grupo composto pelos demais operadores, com desvios padrão do erro mais elevados que aqueles de OX1 e ER.

Analisando o erro médio (Figura 17) em conjunto com o desvio padrão do erro (Figura 18) temos que EAX é o melhor operador, pois encontrou erro médio 0, e consequentemente, desvio padrão 0, para todas as instâncias testadas. O grupo com a segunda melhor performance consiste dos operadores OX1 e ER, em que o erro médio e o desvio padrão do erro é inferior ao dos demais operadores (com exceção do EAX). Isso significa que em média OX1 e ER encontra melhores resultados que os demais operadores (exceto EAX), com uma maior consistência do erro encontrado entre as múltiplas execuções.

Vale lembrar que todos os experimentos foram executados com tamanho fixo de população e o número fixo de gerações (Seção 3.4). Se o tamanho da população aumentasse, é provável que os operadores de cruzamento tivessem um comportamento diferente em relação ao erro percentual dos gráficos. Porém, o aumento no tamanho da população, resulta no aumento do tempo computacional na busca de novas soluções.

4.4 Considerações finais

Neste capitulo, foram aprestandos os resultados numéricos do trabalho (Tabela 3). A análise dos gráficos do erro percentual médio e erro percentual do desvio padrão, mostrou que o EAX possue o melhor desempenho entres os operadores de cruzamento estudados. Vale ressaltar que o ER e o OX1 obtiveram um desempenho semelhantes entre si, conquistando alguns bons resultados para um conjunto de instâncias.

Porém, deve-se salientar que os operadores PMX, CX, OX2 e POS, obtiveram um comportamento bem inferior aos demais operadores de cruzamento, encontrando soluções distante do valor ótimo com o erro bem alto. No próximo capítulo, são apresentados as conclusões gerais do trabalho.

CAPÍTULO

5

CONCLUSÕES

Neste trabalho, propõe-se um estudo de revisão da literatura sobre alguns dos mais relevantes operadores de cruzamento genético projetados para serem aplicados ao Problema do Caixeiro Viajante. No Capítulo 3 foram apresentados dois modelos de representação da codificação das soluções candidatas para o TSP. Os modelos são, representação por percurso e representação por abordagens analítica de arestas.

Representação por percurso é o modo mais natural para representar soluções para o problema do caixeiro viajante. Para este estilo de representação, foram selecionados os operadores de cruzamento *Partially-Mapped Crossover* (PMX), *Cycle Crossover* (CX), *Order Crossover 1* (OX1), *Order Based Crossover* (OX2), *Position Based Crossover* (POS).

A representação por abordagens analítica utiliza as informações extraídas de soluções progenitoras já conhecidas para criar novos indivíduos. Para este estilo de representação foram selecionados os operadores de cruzamento *Genetic Edge Recombination Crossover* (ER) e *Edge Assembly Crossover* (EAX).

Os operadores de cruzamento foram testados em 20 instâncias do problema com o número de cidades variando de 16 até 1060. Após a fase de avaliação, foi constatado que o operador EAX obteve os melhores resultados, encontrando soluções ótimas para todas as instâncias. Também é importante ressaltar que os operadores OX1 e ER obtiveram um bom desempenho, encontrando soluções com um percentual do erro considerável, mesmo para instâncias com até 532 cidades.

O grupo de operadores PMX, CX, OX2 e POS não conseguiram um desempenho satisfatório, resultando em um percentual de erro alto mesmo para instâncias menos complexas, com poucas cidades, onde os outros operadores obtiveram melhores resultados. Esse conjunto de operadores, apenas obtiveram resultados satisfatórios para instâncias de menores dimensões com até 48 cidades, e mesmos para estes casos, os resultados alcançados foram os piores em relação ao erro médio e o desvio padrão se comparados com os operadores ER e OX1.

É importante enfatizar que o EAX difere dos demais operadores de cruzamento, no sentido que utiliza método de busca local durante a fase de reprodução dos descendentes. Desta forma, o processo de hibridização dos operadores de cruzamento podem ser considerados como um fator relevante de melhoria de desempenho dos mesmos.

5.1 Trabalhos futuros

Com o objetivo de melhorar o desempenho dos operadores ER e OX1, e utilizar das capacidades de otimização demonstradas pelo EAX demonstrado pelos resultados obtidos, alguns trabalhos futuros que possam dar continuidade a esta pesquisa são sugeridos:

- Desenvolver e experimentar um GA hibrido (Seção 2.4) com nos operadores ER e OX1.
- Desenvolver e experimentar um GA aplicando o modelo de ilha (WHITLEY; RANA; HECKENDORN, 1998), com múltiplas populações de cromossomos, aplicando os operadores ER e OX1.
- Desenvolver um GA para o Problema de Localização e Roteamento (LAPORTE; NOBERT; TAILLEFER, 1988), com operador de cruzamento baseado no EAX.

REFERÊNCIAS

AGUILAR-RUIZ, J. S.; GIRáLDEZ, R.; SANTOS, J. C. R. Natural encoding for evolutionary supervised learning. **IEEE Trans. Evolutionary Computation**, v. 11, p. 466–479, 2007. Citado na página 38.

ALBAYRAK, M.; ALLAHVERDI, N. Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. **Expert Syst. Appl.**, v. 38, n. 3, p. 1313–1320, 2011. Citado nas páginas 26 e 44.

ANDREICA, A.; CHIRA, C. Best-order crossover for permutation-based evolutionary algorithms. **Applied Intelligence**, 2014. Citado nas páginas 29 e 49.

ARBIB, M. A. (Ed.). **The Handbook of Brain Theory and Neural Networks**. 2nd. ed. Cambridge, MA, USA: MIT Press, 2002. Citado na página 28.

BäCK DAVID B FOGEL, Z. M. T. **Evolutionary Computation 1: Basic Algorithms and Operators**. Philadelphia, USA: Institute of Physics Publishing, 2000. Citado na página 34.

BäCK, T. **Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms**. [S.l.]: Oxford University Press, 1996. Citado na página 34.

BEIRIGO, B. A.; SANTOS, A. G. dos. Application of nsga-ii framework to the travel planning problem using real-world travel data. In: **2016 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2016. p. 746–753. Citado na página 27.

BOUSSAïD, J. L. I.; SIARRY, P. A survey on optimization metaheuristics. **Information Sciences**, v. 237, p. 82 – 117, 2013. Citado na página 31.

CHAKRABORTY, M.; CHAKRABORTY, U. K. Branching process analysis of linear ranking and binary tournament selection in genetic algorithms. **Journal of Computing and Information Technology (Zagreb)**, v. 7, p. 107–113, 1999. Citado na página 42.

CHATTERJEE, S.; CARRERA, C.; LYNCH, L. Genetic algorithms and traveling salesman problems. **European Journal of Operational Research**, v. 93, n. 3, p. 490–510, 1996. Citado na página 49.

CHENG, R.; GEN, M.; TSUJIMURA, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation. **Computers Industrial Engineering**, v. 30, n. 4, p. 983 – 997, 1996. Citado na página 39.

COOK, W. J. **In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation**. [S.l.]: Princeton University Press, 2012. Citado nas páginas 24 e 26.

DARWIN, C. **On the Origin of Species by Means of Natural Selection, Or, The Preservation of Favoured Races in the Struggle for Life**. [S.l.]: J. Murray, 1859. Citado na página 34.

DAVIS, L. Applying adaptive algorithms to epistatic domains. In: **Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1985. p. 162–164. Citado na página 54.

DAVIS, L. **Handbook of genetic algorithms**. [S.l.]: Van Nostrand Reinhold, 1991. (VNR computer library). Citado na página 34.

DAVIS, L. D.; MITCHELL, M. Handbook of Genetic Algorithms. **Van Nostrand Reinhold**, 1991. Citado na página 39.

DRÉO, J. **Metaheuristics for Hard Optimization: Methods and Case Studies**. [S.l.]: Springer, 2006. Citado nas páginas 33, 41, 43 e 44.

FOGEL, D. B. **Evolutionary Computation: Toward a New Philosophy of Machine Intelligence**. Piscataway, NJ, USA: IEEE Press, 1995. Citado na página 34.

FOGEL, L.; OWENS, A.; WALSH, M. **Artificial intelligence through simulated evolution**. Chichester, WS, UK: Wiley, 1966. Citado na página 33.

FUQUAY, D.; WHITLEY, D. Genetic algorithm solutions for the traveling salesman problem. In: **Proceedings of the 28th Annual Southeast Regional Conference**. New York, NY, USA: ACM, 1990. p. 143–150. Citado na página 59.

GEN, R. C. M. **Genetic Algorithms and Engineering Optimization**. [S.l.]: John Wiley & Sons, Inc., 2000. Citado na página 32.

GLOVER, F. Tabu search — part i. **ORSA Journal on Computing**, v. 1, n. 3, p. 190–206, 1989. Citado na página 28.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. 1st. ed. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1989. Citado nas páginas 32, 33, 35, 36, 37 e 44.

GOLDBERG, D. E.; LINGLE JR., R. Alleles loci and the traveling salesman problem. In: **Proceedings of the 1st International Conference on Genetic Algorithms**. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985. p. 154–159. Citado nas páginas 44 e 51.

HASHI, E. K.; HASAN, M. R.; ZAMAN, M. S. U. Gis based heuristic solution of the vehicle routing problem to optimize the school bus routing and scheduling. In: **2016 19th International Conference on Computer and Information Technology (ICCIT)**. [S.l.: s.n.], 2016. p. 56–60. Citado na página 27.

HAUPT, S. E. H. R. L. **Practical Genetic Algorithms**. [S.l.]: John Wiley & Sons, Inc., 2004. Citado nas páginas 24, 31, 32 e 35.

HELSGAUN, K. An effective implementation of the lin-kernighan traveling salesman heuristic. **European Journal of Operational Research**, v. 126, p. 106–130, 2000. Citado nas páginas 24, 25, 26, 27 e 28.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. Cambridge, MA, USA: MIT Press, 1975. Citado nas páginas 28, 34, 36, 39, 43, 44 e 65.

JANIKOW, C. Z.; MICHALEWICZ, Z. An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms. In: BELEW, R. K.; BOOKER, L. B. (Ed.). **Proc. of the 4th International Conference on Genetic Algorithms**. [S.l.]: Morgan Kaufmann, 1991. p. 151–157. Citado na página 39.

JOHNSON, D. S.; MCGEOCH, L. A. The Traveling Salesman Problem: A Case Study in Local Optimization. In: ____. **Local Search in Combinatorial Optimization**. [S.l.: s.n.], 1997. Citado nas páginas 26 e 28.

JUNG, S.; MOON, B. Toward minimal restriction of genetic encoding and crossovers for the two-dimensional euclidean tsp. **Evolutionary Computation, IEEE Transactions on**, v. 6, n. 6, p. 557–565, 2002. Citado nas páginas 49 e 50.

KATAGIRI, H.; GUO, Q.; WU, H.; HAMORI, H.; KATO, K. A route optimization problem in electrical pcb inspections: Pickup and delivery tsp-based formulation. In: ____. **Transactions on Engineering Technologies: International MultiConference of Engineers and Computer Scientists 2015**. [S.l.]: Springer Singapore, 2016. Citado na página 27.

LAARHOVEN, P. J. M.; AARTS, E. H. L. (Ed.). **Simulated Annealing: Theory and Applications**. Norwell, MA, USA: Kluwer Academic Publishers, 1987. Citado na página 28.

LAPORTE, G. The traveling salesman problem: An overview of exact and approximate algorithms. **European Journal of Operational Research**, v. 59, n. 2, p. 231 – 247, 1992. Citado na página 26.

LAPORTE, G.; NOBERT, Y.; TAILLEFER, S. Solving a family of multi-depot vehicle routing and location-routing problems. **Transportation Science**, v. 22, n. 3, p. 161–172, 1988. Citado na página 78.

LARRANAGA, P.; MURGA, C. M. H. K. R.; INZA, I.; DIZDAREVIC, S. Genetic algorithms for the travelling salesman problem: A review of representations and operators. **Artificial Intelligence Review**, v. 13, p. 129–170, 1999. Citado nas páginas 29, 49, 50, 51, 60 e 74.

LAWLER, E. L.; LENSTRA, J. K.; KAN, A. H. G. R.; SHMOYS, D. B. **The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization**. [S.l.]: John Wiley & Sons, 1991. Citado nas páginas 23, 26 e 27.

LINDEN, R. **Algoritmos genéticos: uma importante ferramenta da inteligência computacional**. [S.l.]: Brasport, 2006. Citado nas páginas 36 e 40.

LIU, F.; ZENG, G. Study of genetic algorithm with reinforcement learning to solve the tsp. **Expert Systems with Applications**, v. 36, n. 3 PART 2, p. 6995–7001, 2009. Citado nas páginas 28, 62 e 66.

MA, X. Intelligent tourism route optimization method based on the improved genetic algorithm. In: **2016 International Conference on Smart Grid and Electrical Automation (ICSGEA)**. [S.l.: s.n.], 2016. Citado na página 27.

MELANIE, M. **An Introduction to Genetic Algorithms**. [S.l.]: MIT Press, 1998. Citado nas páginas 32, 33 e 34.

MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)**. London, UK, UK: Springer-Verlag, 1996. Citado nas páginas 32 e 35.

MICLAUS, K.; PRATT, R.; GALATI, M. **The Traveling Salesman Traverses the Genome: Using SAS ® Optimization in JMP ® Genomics to build Genetic Maps ABSTRACT**. 2012. Citado na página 27.

MITCHELL, M. **An Introduction to Genetic Algorithms**. Cambridge, MA, USA: MIT Press, 1996. Citado nas páginas 35, 36 e 42.

MOSCATO, P. A gentle introduction to memetic algorithms. In: **Handbook of Metaheuristics**. [S.l.]: Kluwer Academic Publishers, 2003. p. 105–144. Citado na página 46.

NAGATA, Y.; KOBAYASHI, S. Edge assembly crossover: A high-power genetic algorithm for the travelling salesman problem. In: BÄCK, T. (Ed.). **Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)**. San Francisco, CA: [s.n.], 1997. p. 450–457. Citado nas páginas 29, 61 e 65.

NAGATA, Y.; KOBAYASHI, S. Analysis of edge assembly crossover for the travelling salesman problem. In: . [S.l.: s.n.], 1999. v. 3, p. III–628 – III–633. Citado nas páginas 62 e 63.

NAGATAA, Y.; KOBAYASHIA, S. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. **INFORMS Journal on Computing**, v. 25, n. 2, p. 346–363, 2013. Citado nas páginas 29 e 62.

NELSON, A. L.; BARLOW, G. J.; DOITSIDIS, L. Fitness functions in evolutionary robotics: A survey and analysis. **Robot. Auton. Syst.**, North-Holland Publishing Co., v. 57, p. 345–370, 2009. Citado na página 40.

OLIVER, I. M.; SMITH, D. J.; HOLLAND, J. R. C. A study of permutation crossover operators on the traveling salesman problem. In: **Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application**. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1987. p. 224–230. Citado na página 53.

RECHENBERG, I. Cybernetic solution path of an experimental problem. In: **Royal Aircraft Establishment Translation No. 1122, B. F. Toms, Trans.** Farnborough Hants: Ministry of Aviation, Royal Aircraft Establishment, 1965. Citado na página 33.

REINELT, G. Tsplib - a traveling salesman problem library. **INFORMS Journal on Computing**, v. 3, n. 4, p. 376–384, 1991. Citado nas páginas 65 e 69.

ROZENBERG, G.; BCK, T.; KOK, J. N. **Handbook of Natural Computing**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2011. Citado nas páginas 34 e 35.

RUSSELL, S.; NORVIG, P. **Inteligência artificial**. [S.l.]: CAMPUS - RJ, 2004. Citado na página 47.

SILVA, S. F. d. **Seleção de características por meio de algoritmos genéticos para aprimoramento de rankings e de modelos de classificação**. Tese (Doutorado) — São Carlos : Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2011. Citado nas páginas 32, 35, 37, 38, 40, 41, 43, 44, 45 e 46.

SIVANANDAM, S. N.; DEEPA, S. N. **Introduction to Genetic Algorithms**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2008. Citado nas páginas 36 e 39.

STARKWEATHER, T.; MCDANIEL, S.; WHITLEY, D.; MATHIAS, K.; WHITLEY, D.; DEPT, M. E. A comparison of genetic sequencing operators. In: **Proceedings of the fourth International Conference on Genetic Algorithms**. [S.l.]: Morgan Kaufmann, 1991. p. 69–76. Citado na página 60.

SYSWERDA, G. Schedule optimization using genetic algorithms. In: DAVIS, L. (Ed.). **Handbook of Genetic Algorithms**. New York, NY: Van Nostrand Reinhold, 1991. Citado nas páginas 56 e 57.

TANG, K. S.; MAN, K. F.; KWONG, S.; HE, Q. Genetic algorithms and their applications. **IEEE Signal Processing Magazine**, v. 13, 1996. Citado nas páginas 39 e 40.

VARGAS, D. F. V.; SENDALES, D. A. P. Optimal scheduling trip plan for tourist. In: **2016 10th International Conference on Intelligent Systems and Control (ISCO)**. [S.l.: s.n.], 2016. p. 1–5. Citado na página 27.

WANG, Y. ting; LI, J. qing; GAO, K. zhou; PAN, Q. ke. Memetic algorithm based on improved inver–over operator and lin–kernighan local search for the euclidean traveling salesman problem. **Computers Mathematics with Applications**, v. 62, n. 7, 2011. Citado nas páginas 17 e 47.

WHITLEY, D.; RANA, S.; HECKENDORN, R. B. The island model genetic algorithm: On separability, population size and convergence. **Journal of Computing and Information Technology**, v. 7, p. 33–47, 1998. Citado na página 78.

WHITLEY, L. D.; STARKWEATHER, T.; FUQUAY, D. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In: **Proceedings of the 3rd International Conference on Genetic Algorithms**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989. p. 133–140. Citado na página 59.

Genetic algorithms - chapter 5. In: YANG, X.-S. (Ed.). **Nature-Inspired Optimization Algorithms**. Oxford: Elsevier, 2014. p. 77 – 87. Citado nas páginas 33, 43 e 44.

YUAN, Y. L. L.; LI, M. Genetic algorithm based on good character breed for traveling salesman problem. p. 234–237, Dec 2009. Citado na página 26.