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 $Filho_1$.

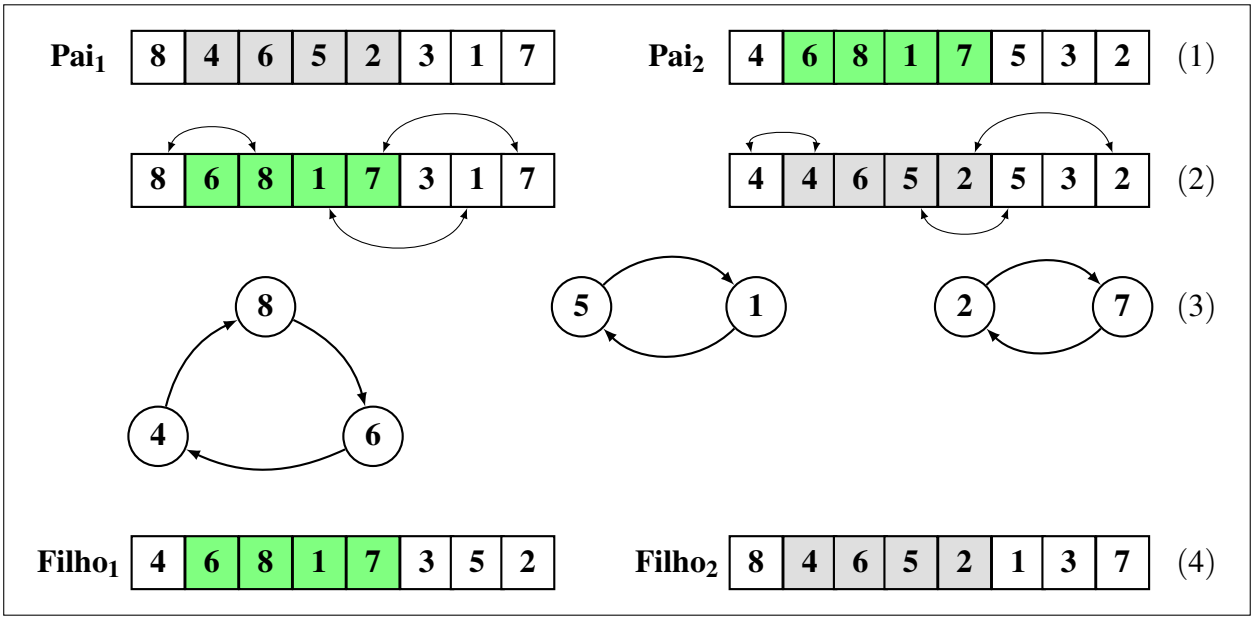
Entre as linhas 6 e 15 do Algoritmo 2, as cidades remanescentes dos pais que não estão contidas na região de corte (i, j) dos respectivos filhos, são selecionadas e inseridas nas devidas posições dos filhos de acordo com a ordem realizada pelo mapeamento, sem que cidades

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

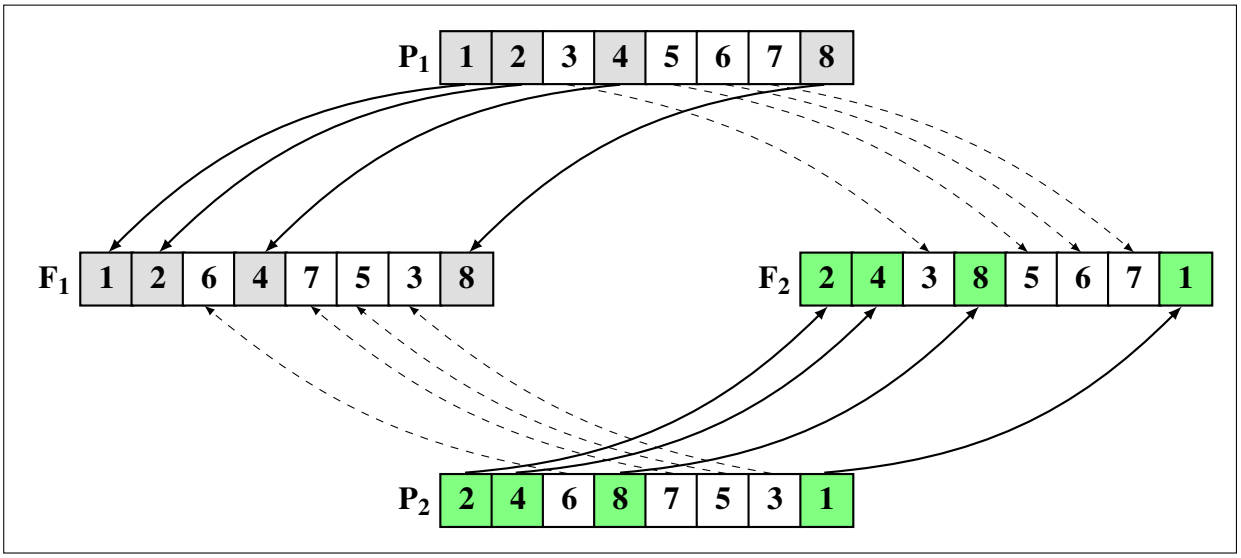
```
1:  procedimento CX(Pai_1[], Pai_2[])
2:      Filho_1 ← []                          ▷ Inicie uma lista vazia para o Filho_1
3:      Filho_2 ← []                          ▷ Inicie uma lista vazia para o Filho_2
4:      para todo i ∈ [1,2] faça
5:          Filho_(i) ← copy(Pai_(i mod 2+1))
6:          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:

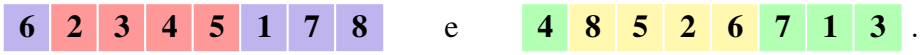
1 2 3 4 5 6 7 8 e **4 8 5 2 6 3 1 7** .

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

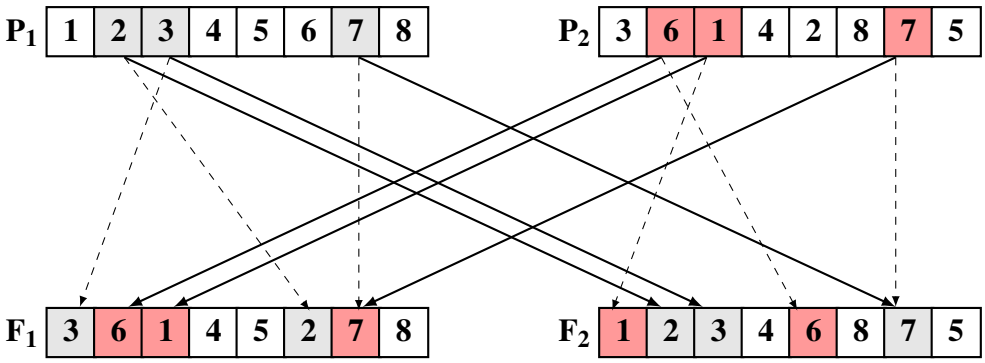
e

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).

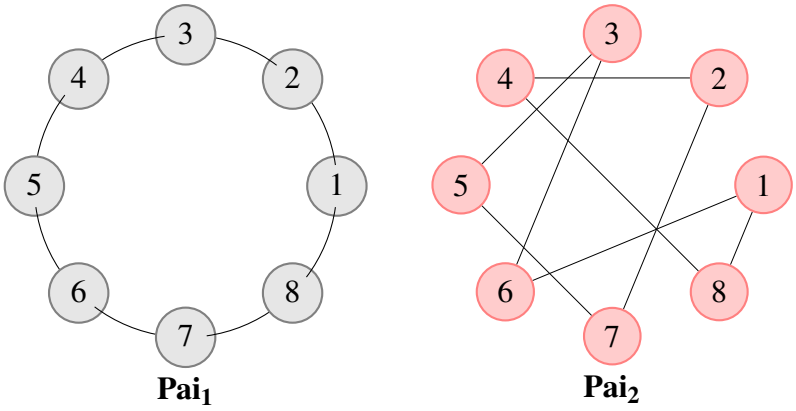
3.3.1 Genetic edge recombination crossover

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

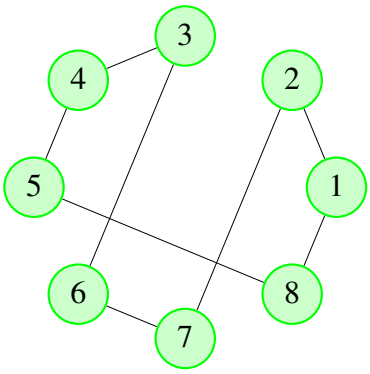
cidade **1** é selecionada como próxima cidade corrente e é removida da lista de cidades vizinhas da Tabela 1.

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

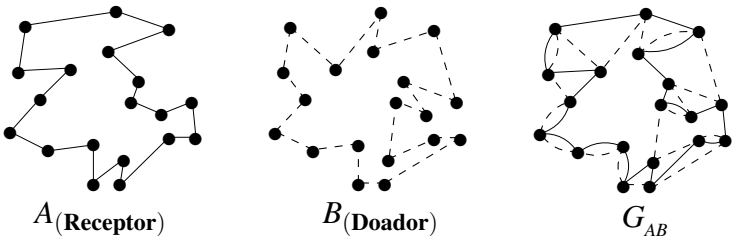
O *Edge Assembly Crossover* (EAX) é uma técnica proposta por (NAGATA; KOBAYASHI, 1997). Dado um par de soluções progenitoras quaisquer (pais), sendo que cada solução progenitora representa um determinado percurso para o caixeiro viajante, o EAX gera sua prole por meio de uma permutação de arestas, envolvendo 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.

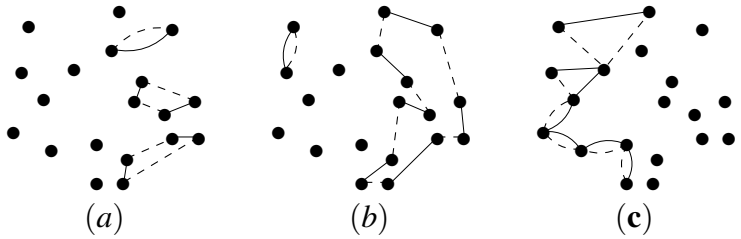
O EAX gera uma solução intermediária por meio da construção de pequenos ciclos (subciclos) em G_{AB}, com o complemento das arestas contidas em E_A gerando percursos desconexos. Em seguida, esses ciclos são conectados com a adição de uma quantidade reduzida de novas arestas formando uma nova solução 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.



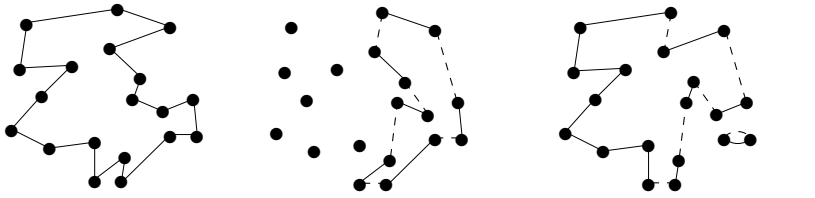
Fonte: Elaborada pelo autor.

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.

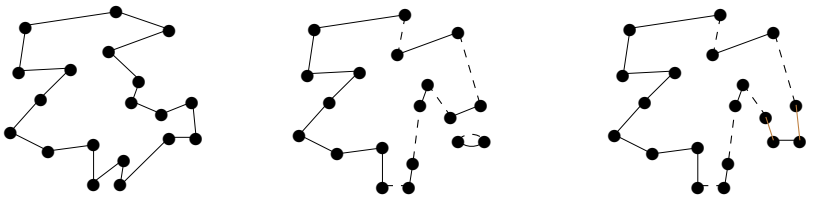
Figura 14 – Geração de uma solução intermediária - EAX.



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.