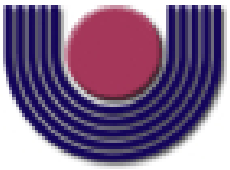

Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Estudo das Metaheurísticas *Ant System* e Algoritmos Genéticos Aplicados na Resolução do Problema do Caixeiro Viajante

Lucas Szeremeta

CASCAVEL
2016

LUCAS SZEREMETA

ESTUDO DAS METAHEURÍSTICAS *ANT SYSTEM* E ALGORITMOS GENÉTICOS APLICADOS NA RESOLUÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE

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

Orientadora: Prof. Rosangela Villwock
Co-Orientador: Prof. Adair Santa Catarina

CASCAVEL 2016

LUCAS SZEREMETA

ESTUDO DA EFICIÊNCIA DAS METAHEURÍSTICAS ANT SYSTEM E ALGORITMOS GENÉTICOS APLICADOS NA RESOLUÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE

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

Prof. Rosangela Villwock (Orientadora)
Colegiado de Ciência da Computação,
UNIOESTE

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

Prof. Adriana Postal
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 15 de Fevereiro de 2017

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AG	Algoritmo Genético
POC	Problema de Otimização Combinatória
PCV	Problema do Caixeiro Viajante

Sumário

Resumo

Problemas de otimização combinatória (POC) pertencem à classe dos problemas NP-difíceis. Tendo isso em vista pesquisadores das mais diversas áreas trabalham não apenas em maneiras de resolver um dado POC, mas também em como resolvê-los de maneira eficiente. O problema do caixeiro viajante (PCV) é um exemplo de POC onde, em dado grafo, o viajante deve passar por todos seus vértices sem repeti-los, retornando ao vértice de partida. Uma opção para resolver o problema seria analisar todas as possibilidades de rotas do problema, o que seria inviável computacionalmente. Como alternativa foram desenvolvidas as Heurísticas e Metaheurísticas, as quais tentam encontrar uma solução considerada boa em tempo adequado. Neste trabalho foram comparadas duas Metaheurísticas inspiradas na natureza; uma é a *Ant-System*, a qual se inspira em uma colônia de formigas; a outra é a conhecida como Algoritmos Genéticos, a qual foi inspirada na teoria da evolução de Darwin. A eficácia de ambas Metaheurísticas foram comparadas em problemas simétricos do Caixeiro Viajante utilizando-se de parâmetros já testados por outros pesquisadores, onde retornando resultados satisfatórios. No presente trabalho o Algoritmo Genético retornou resultados melhores para os problemas escolhidos, no entanto o *Ant System* retornou resultados em tempo mais viável do que o AG.

Palavras-chave: Problema do Caixeiro Viajante, Metaheurísticas, Problema de Otimização Combinatória.

Capítulo 1

Introdução

Os problemas de Otimização Combinatória (POC), pertencentes à classe dos problemas NP-difíceis, são de difícil solução e, dificilmente, serão apresentados algoritmos polinomiais que os solucionem em tempo viável. Tendo isso em vista, pesquisadores de várias áreas não medem esforços no desenvolvimento de algoritmos mais eficientes, cujo foco é resolver estes tipos de problemas. Nos POC o objetivo é atribuir valores a um conjunto de variáveis de decisão, de tal maneira que uma função objetivo seja otimizada (maximizada ou minimizada) atendendo a um certo conjunto de restrições. O Problema do Caixeiro Viajante, o Problema da Mochila e do roteamento de veículos são considerados alguns exemplos de POC [3].

Uma das possíveis maneiras de resolução destes problemas poderia ser simplesmente enumerar e realizar todas as combinações possíveis de soluções do problema e aguardar um resultado ótimo, mas isso seria inviável devido ao tempo e custo computacional gasto neste processo, mesmo utilizando-se de supercomputadores. Ao longo dos anos foram criados algoritmos exatos com propostas diferentes para se resolver os POC, mas ainda assim para problemas de grande porte continuaram inviáveis. Assim, técnicas computacionais mais avançadas foram necessárias para a resolução destes problemas. As heurísticas são algoritmos que não tem garantia de encontrar solução ótima, ou seja, a melhor solução para os problemas; No entanto, são capazes de retornar uma solução em um tempo adequado para as necessidades da aplicação [3] .

O problema do caixeiro viajante (PCV) ou *Traveling Salesman Problem* (TSP), objeto de estudo no presente trabalho, é um dos problemas mais estudados no grupo de problemas de otimização combinatória; seu objetivo é encontrar em um grafo G = (V, A), um circuito hamiltoniano de menor custo (Figura 1.1), sendo que o mesmo é um passeio que percorre todos os vértices de um grafo e retorna ao vértice de origem (ponto inicial), passando por cada vértice

apenas uma vez. Um grafo pode ser definido como um conjunto de vértices e arestas, sendo que os vértices podem representar cidades, postos de trabalho dentre outros. Arestas são linhas que realizam conexões entre os vértices, representando assim ruas, estradas ou outros tipos de caminhos. As arestas possuem um custo ou peso que pode ser medido por vários fatores, sendo estes a distância, sinaleiros, más condições de asfalto, etc [1].

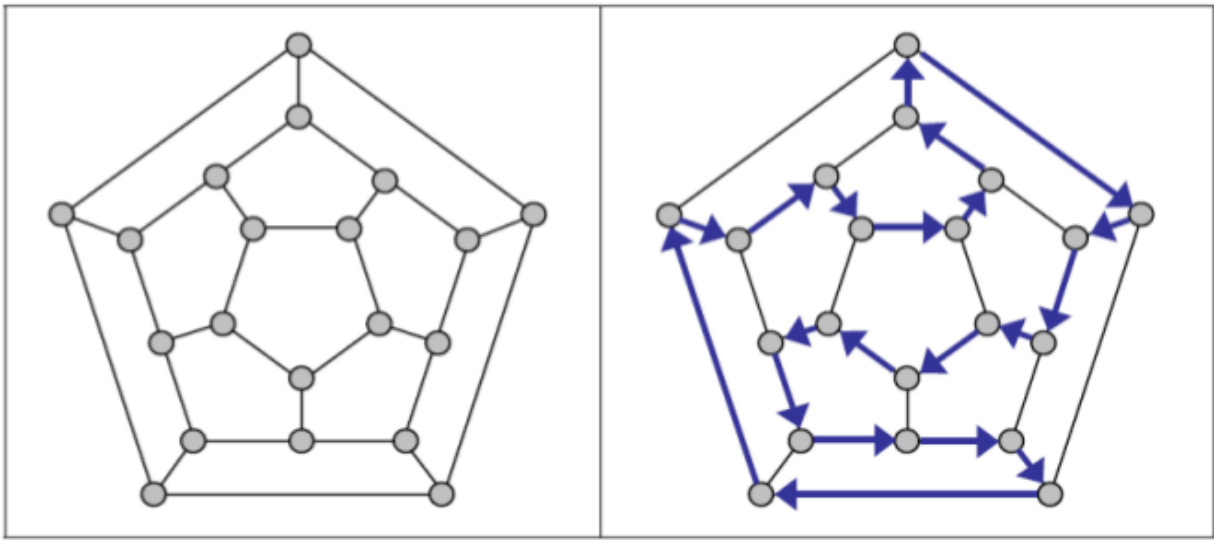

Figura 1.1: Jogo de Hamilton [1].

O PCV é um clássico problema de otimização combinatória da área de Pesquisa Operacional. Pode ser classificado como simétrico ou assimétrico, sendo que ele será simétrico se para todos os pares de nós i, j os custos c_{ij} e c_{ji} forem iguais, ou seja, se os caminhos de ida e volta possuem a mesma unidade de peso. Caso contrário, o problema é assimétrico [4].

Reinelt [5] criou uma biblioteca chamada de TSPLIB, que contém vários tipos de problemas que podem ser modelados como PCV, os quais são testados e discutidos até hoje dentre a comunidade de pesquisadores do problema. Na TSPLIB existem problemas que variam de 14 até 85.900 cidades.

De acordo com Benevides *apud* Raff [6], existem três classes de heurísticas capazes de resolverem o PCV, as quais são:

- Procedimentos de construção de rotas, as quais tem o objetivo de construir rotas ótimas ou quase ótimas.

- Procedimentos de melhorias de rotas, que efetuam melhorias em rotas já existentes.

- Procedimentos compostos, que utilizam de procedimentos de construção de rotas para gerar um resultado inicial e após utilizam de procedimentos de melhorias de rotas para gerarem um resultado final mais eficiente.

Com o passar do tempo surgiram técnicas conhecidas como metaheurísticas dentre as quais se destacam os Algoritmos Genéticos, Busca Tabu, *Simulated Annealing*, *Ant System*, GRASP, dentre outros. As metaheurísticas têm como objetivo encontrar uma boa solução e, eventualmente, a ótima, sendo modeladas para cada problema específico. As heurísticas baseiam-se na melhora do movimento a ser executado, buscando um ótimo local, o que acaba sempre gerando uma mesma solução quando iniciadas sempre em um mesmo ponto inicial; as metaheurísticas suprem esta deficiência, baseando-se na melhoria da solução, deixando um ótimo local para procurar um ótimo global [4].

O objetivo deste trabalho foi comparar as duas metaheurísticas conhecidas como *Ant System* e Algoritmos Genéticos aplicados ao PCV, utilizados os mesmos parâmetros de trabalhos que obtiveram bons resultados na resolução deste tipo de problema. Foram utilizados os trabalhos de Dorigo e Gambardella [7] para o Ant System e Benevides [4] para o Algoritmo Genético.

O *Ant System* é uma metaheurística de sucesso, inspirada no comportamento real de formigas. A partir do *Ant System*, vários algoritmos baseados nas mesmas ideias iniciais foram criados e aplicados com sucesso em uma variedade de problemas combinacionais, inclusive o PCV. Assim estes algoritmos ficaram conhecidos como os algoritmos Colônia de Formigas ou ACO (*Ant Colony Optmization*) [8].

O Algoritmo Genético é um procedimento iterativo que mantém uma população de estruturas chamadas de indivíduos, os quais representam as possíveis soluções para um determinado problema. A cada iteração, chamada de geração, os indivíduos da população passam por uma avaliação que verifica sua capacidade em oferecer uma solução satisfatória para o problema. Tal avaliação é feita pela função de aptidão, ou função de *fitness*. De acordo com esta avaliação, alguns indivíduos são selecionados, através de uma regra probabilística, passando por um processo de reprodução, gerando novos indivíduos. Assim pressupõe-se que a população a cada geração se tornará mais apta em resolver o problema . Assim, após certo número de iterações ou gerações, é selecionado o indivíduo mais apto como uma possível solução do problema [9].

Capítulo 2

Referencial Teórico

2.1 Algoritmos Genéticos

Os Algoritmos Genéticos (AGs), introduzidos pelo trabalho de Holland [10], é uma metaheurística que pode ser utilizada em vários problemas de otimização combinacional inclusive no PCV. Os AGs realizam procedimentos de busca no espaço das soluções viáveis, utilizando regras probabilísticas para combinar soluções com a finalidade de obter melhorias de qualidade. Como o AG é um algoritmo baseado no processo evolutivo da natureza, estabeleceu-se uma analogia entre os termos empregados nas duas áreas de conhecimento. Os AGs possuem as seguintes características gerais [2]:

- Operam sobre pontos, denominados de população, e não a partir de pontos isolados;
- Operam em um espaço de soluções codificadas e não diretamente em um espaço de busca;
- Necessitam como informação somente o valor de uma função objetivo (denominada de função *fitness*);
- Usam transições probabilísticas e não regras determinísticas.

Os AGs são uma classe de algoritmos evolucionários que encontraram a possibilidade de aplicação geral. Os termos a seguir são altamente utilizados quando se fala de AG [11][2]:

- População: conjunto de indivíduos ou soluções para o problema.
- Cromossomo: representação de um dos indivíduos da população, ou seja, de uma possível solução.

- Gene: representa um componente do cromossomo.
- Alelo: descreve um dos possíveis valores de uma variável do problema.
- Locus: representa a posição do atributo no cromossomo.
- Operadores Genéticos: são regras que permitem a manipulação de cromossomos, as quais são de *crossover* (operador que permite a geração de novos indivíduos através do cruzamento de dois pais) e mutação (permite criar um novo indivíduo através de alterações diretas no pai).
- Fenótipo: denota o cromossomo decodificado.
- Genótipo: representa a estrutura do cromossomo codificado.
- *Schema*: é um modelo de representação para uma família de cromossomos, pode ser representado por um "*" dentro do cromossomo.

Através da figura 2.1 é possível ter uma visualização destes termos.

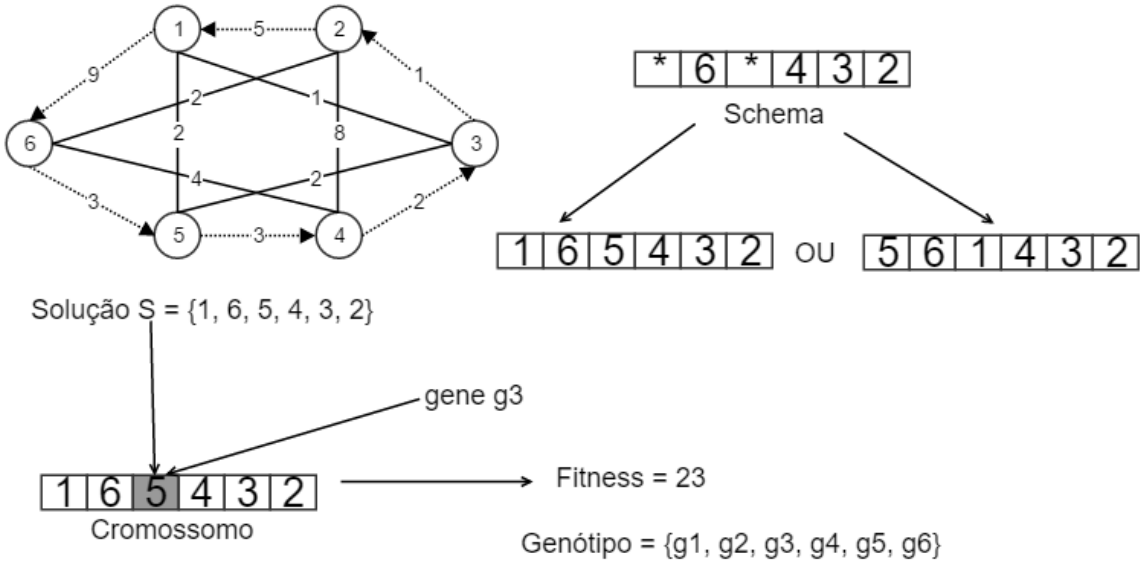

Figura 2.1: Cromossomo de um PCV [2].

Os AG possuem as seguintes características básicas [4][12]: seus indivíduos são representados por cromossomos e competem por recursos e possibilidade de reprodução. Os cromossomos

geralmente são associados a possíveis soluções dos problemas. Uma população inicial geralmente é formada por um mecanismo de avaliação de desempenho. Quando não se possui este mecanismo, a população pode ser gerada de maneira randômica. Os indivíduos geralmente são codificados como uma sequência finita em um alfabeto binário [0,1]. É natural supor que um gene esteja associado a uma variável do problema. A partir da solução inicial, o mecanismo de reprodução será aplicado.

Os indivíduos, que tiverem maiores desempenhos nas competições, terão maior probabilidade de se reproduzir. O mecanismo de reprodução é constituído por uma função de avaliação (função *fitness*) que classifica os indivíduos por seu desempenho e por regras que permitirão que esses melhores indivíduos perpetuem e reproduzam. O método conhecido como *roullette wheel* escolhe um indivíduo para ser um genitor com uma probabilidade igual a sua adequação normalizada, ou seja, o resultado de sua adequação absoluta dividida pela adequação média da população avaliada. O método *tournament* sorteia aleatoriamente k membros da população e, nesse grupo seleciona o indivíduo de melhor desempenho. É importante que se note a natureza probabilística do processo, que procura manter a diversidade da população.

Os genes dos indivíduos considerados bons são propagados através das populações, de modo que sejam aperfeiçoados. Os indivíduos selecionados serão transformados por operadores genéticos em novos indivíduos. Existem operadores associados à reprodução com a utilização de um ou mais indivíduos. O operador primário da reprodução utiliza dois indivíduos sendo denominado de cruzamento ou *crossover*. Existem vários modos de realizar um cruzamento entre os indivíduos representados por cromossomos. Basicamente, o objetivo do cruzamento é formar novos indivíduos a partir da troca de genes entre os indivíduos pais. As regras que definem como essa troca de genes ocorrerá são específicas para cada caso.

A figura 2.2 apresenta dois exemplos de cruzamento simples, onde a posição da permutação é escolhida aleatoriamente.

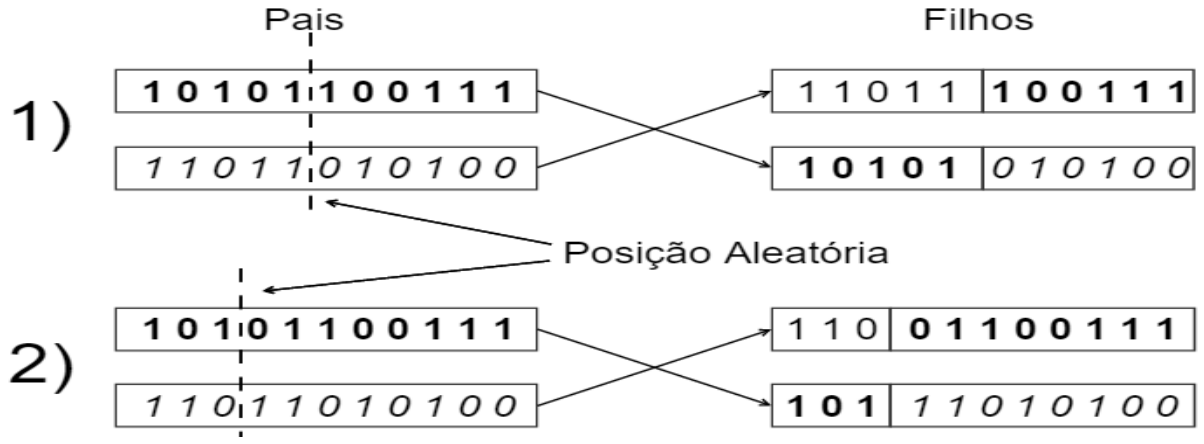

Figura 2.2: Exemplos de Cruzamento Simples de posição aleatória.

Após completo o processo de cruzamento, a população pode ser submetida a mutação, o qual não é um processo obrigatório. Esta alternativa de geração de novos indivíduos pode ser feita diretamente de um outro indivíduo. A figura 2.3 mostra dois tipos possíveis de mutação, onde invertem-se os valores binários do vetor. A única diferença entre eles é se um trecho ou apenas um ponto do cromossomo sofrerá mutação.

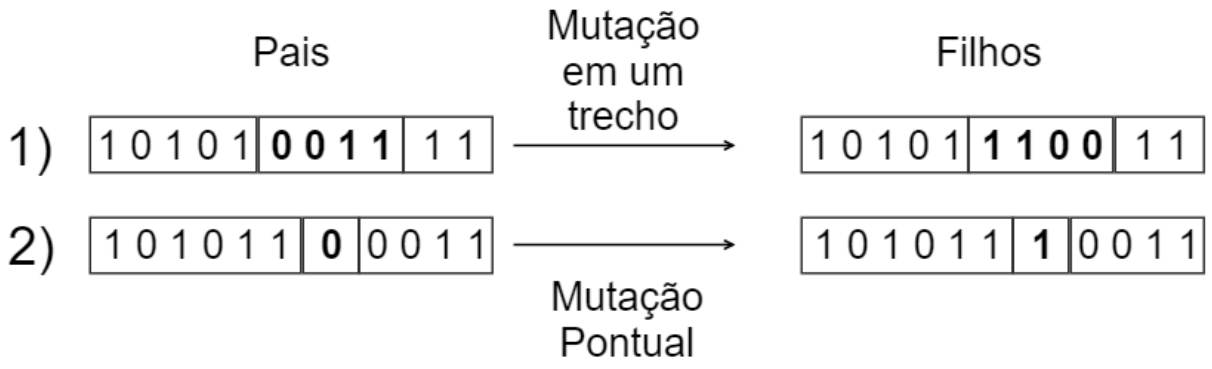

Figura 2.3: Exemplo de mutação.

Descrito o funcionamento do algoritmo, pode-se apresentar o [2]:

Algoritmo 1 META-ALGORITMO GENÉTICO

Gerar uma população Inicial
Avaliar o *fitness* dos indivíduos da população
repita
  Selecionar um conjunto de pais na população
  Cruzar os pais de modo que se reproduzam
  Avaliar o *fitness* dos filhos gerados
  Substituir os filhos julgados inadequados
até *que o critério de parada seja atendido*;

2.1.1 Aplicação ao Problema do Caixeiro Viajante

Alguns autores (Grefenstette [9], Goldberg [13]) obtiveram bons resultados na resolução do PCV. Para explicar melhor como o AG aplicado ao PCV funciona, vamos utilizar a figura 2.4 como referência.

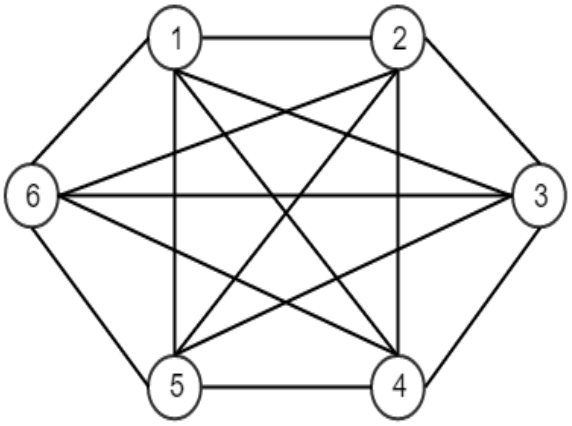

Figura 2.4: Grafo de exemplo para o PCV.

Para resolver o PCV através do AG é necessário que se tenha uma representação genética das soluções viáveis do PCV, associando essa representação ao processo de reprodução. Existem várias formas de representação genética das soluções do PCV sendo uma das mais conhecidas a sequência de inteiros [4]. Nesta forma de representação o cromossomo é um vetor formado pela sequência dos nós percorridos. Baseando-se na figura 2.5, a solução equivale a $p_1 = (1, 4, 2, 3, 6, 5)$.

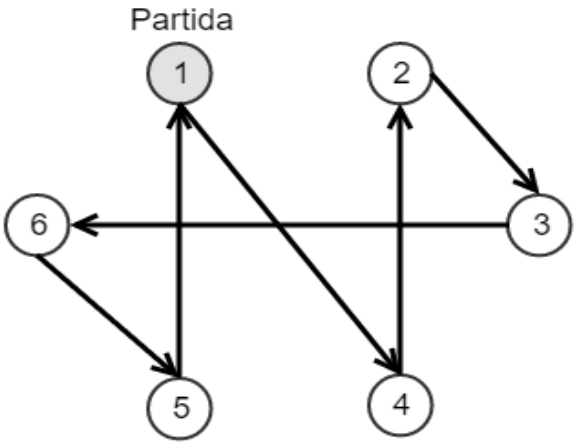

Figura 2.5: Uma solução do PCV.

Esta é uma representação simples, já que a solução é diretamente expressa. Entretanto esta representação possui um problema durante a fase de cruzamento podendo gerar ciclos (filhos) inviáveis. Por exemplo, o cruzamento entre os pais $p_1 = (1,\ 2,\ 3 \mid 4,\ 5,\ 6)$ e $p_2 = (1,\ 2,\ 3 \mid 5,\ 1,\ 6)$, gera os filhos $f_1 = (1,\ 2,\ 3 \mid 5,\ 1,\ 6)$ e $f_2 = (4,\ 3,\ 2 \mid 4,\ 5,\ 6)$ os quais são respostas inválidas para o PCV uma vez que as cidades ou pontos não podem se repetir. Tendo isto em vista foram desenvolvidos operadores especiais que evitam a produção de filhos inviáveis, dado que os pais sejam representações viáveis.

Os operadores são entre os que preservam a posição absoluta das cidades e os que preservam a ordem relativa. Dos que mantém a posição absoluta estão o PMX (*Partially Mapped Crossover*) e o CX (*Cycle Crossover*). E entre os que preservam a ordem relativa está o OX (*Order Crossover*) [4].

O PMX realiza um *crossover* parcialmente mapeado, utilizando dois pontos de corte e realizando o *crossover* entre estes dois pontos. O processo consiste na troca de um vértice do pai 1 por um do pai 2 para cada posição da rota entre os pontos, onde cada uma destas trocas define o mapeamento:

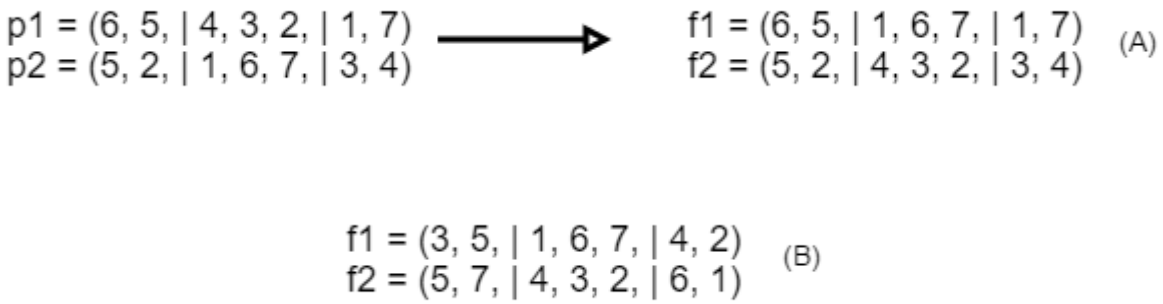

Figura 2.6: Exemplo de crossover PMX.

Nota-se que ambos os filhos gerados são inviáveis, pois em *f1* repete-se os elementos 1, 6, 7 e em *f2* repete-se 4, 3, 2. Assim, a troca entre os pontos de *crossover* define o seguinte mapeamento:$1 \longleftrightarrow 4, 6 \longleftrightarrow 3, 7 \longleftrightarrow 2$. Substituindo as cidades repetidas dos cromossomos pelas mapeadas, resolve-se o problema de inconsistência.

O operador OX consiste em construir um cromossomo filho herdando parcialmente uma sequência de cidades de um dos cromossomos pai mantendo a ordem relativa do outro, conforme o exemplo a seguir:

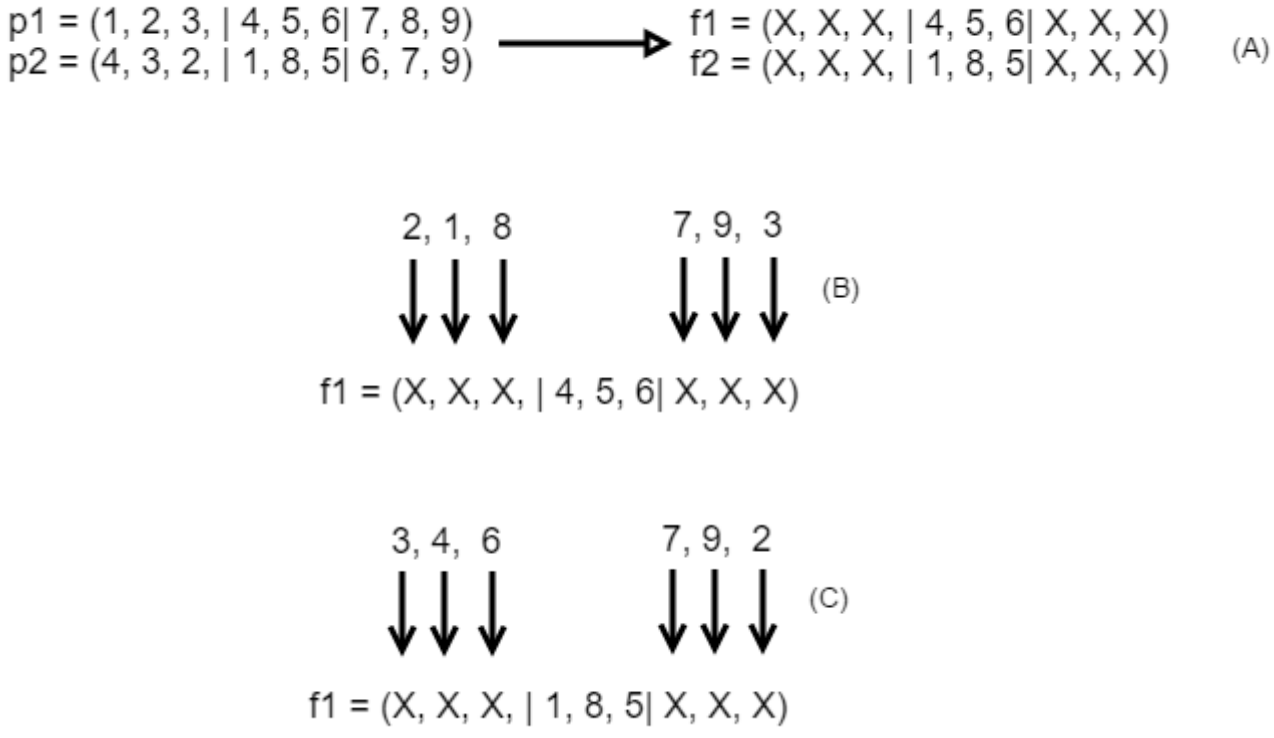

Figura 2.7: Exemplo de crossover OX.

Os filhos herdam o conteúdo que está entre as faixas dos pais. No caso f1 manteve 4, 5, 6 de p1 e f2 manteve 1, 8, 5 de seu pai p2. Partindo do segundo corte do pai $p2$, o cromossomo sofre uma alteração de ordem, inserindo a ultima posição de corte na primeira e avançando os outros cortes uma posição gerando a lista: 6, 7, 9, 4, 3, 2, 1, 8, 5. Desta são removidas as cidades contidas entre os dois cortes no outro pai o qual é 4, 5, 6, obtendo a sequência: 7, 9, 3, 2, 1, 8. Esta sequência é indexada em $f1$ a partir do seu segundo corte, onde 7, 9 ,3 é inserido ao final e 2, 1, 8 ao primeiro:

O CX é um operador que preserva a posição absoluta das cidades nos cromossomos dos pais. Como exemplo serão utilizados os seguintes pais:

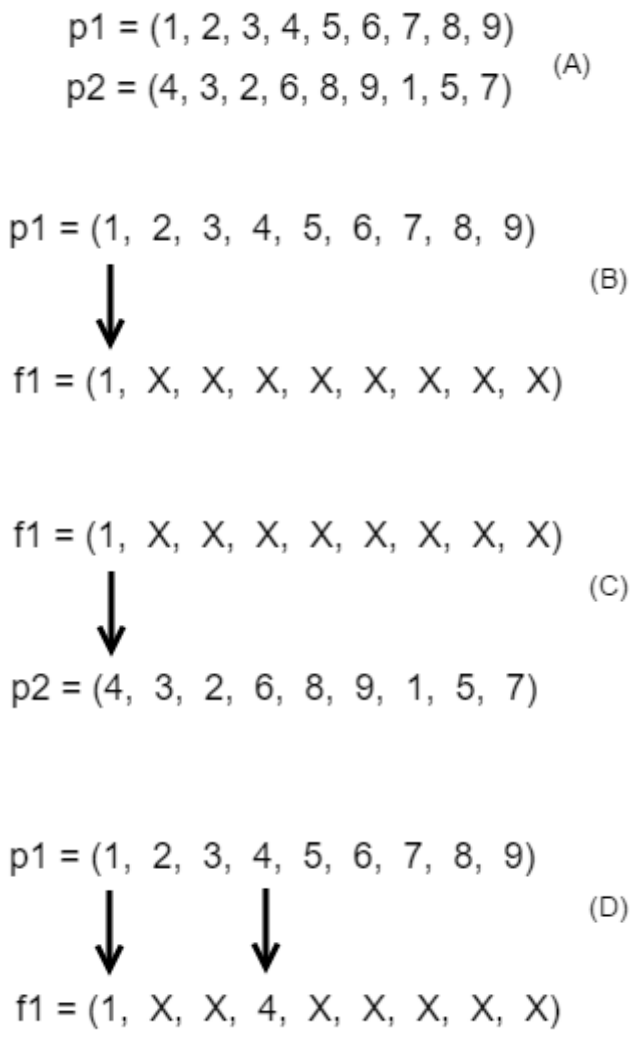

Figura 2.8: Exemplo de crossover CX.

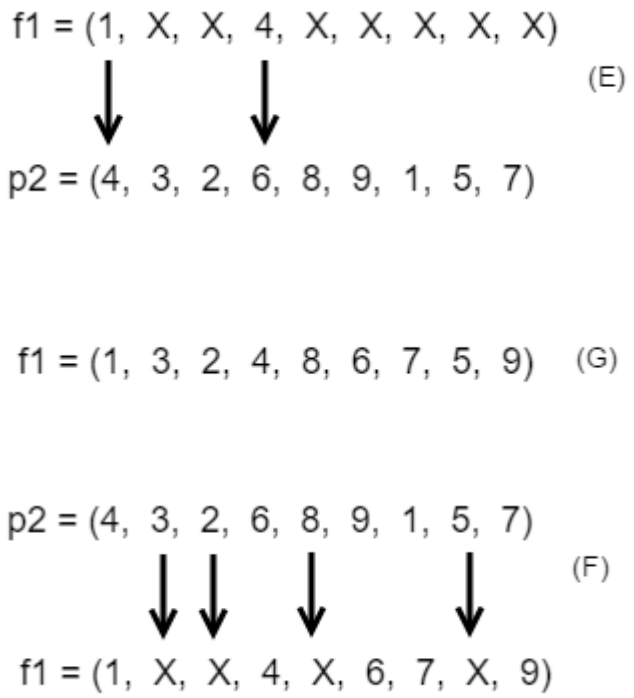

Figura 2.9: Exemplo de crossover CX.

O primeiro filho($f1$) é obtido adicionando a primeira cidade de $p1$ na primeira posição do cromossomo $f1$: Após definir a posição do último elemento adicionado em $f1$, busca-se em $p2$ qual a posição correspondente a este elemento: O elemento da primeira posição de $p2$ será herdado pelo filho $f1$. Entretanto, preservando-se a posição em que esse elemento ocupa no cromossomo de seu pai $p1$. Neste caso, a quarta posição: Na sequência, é encontrado o elemento que ocupa a quarta posição no pai $p2$, que é 6. De maneira semelhante o elemento 6 será herdado por $f1$ na posição que ocupa no $p1$, sexta posição. Procedendo-se assim sucessivamente quando a posição visitada em $p2$ for igual a primeira posição, neste caso ocorrerá na sétima posição, é dito que o ciclo foi completado. O ciclo deste exemplo foi completado com a inclusão do elemento 7. Quando um ciclo foi completo as outras posições são preenchidas do cromossomo $p2$.

2.1.2 Um algoritmo Genético para resolução do PCV

A função de aptidão (*fitness*) é utilizada para avaliar a qualidade da solução a qual está sendo verificada. Assim quanto maior a aptidão, melhor a solução. No caso do PCV quanto menor é o

custo de uma solução melhor será o *fitness* do cromossomo correspondente e por consequência, maiores serão as chances do indivíduo sobreviver e se reproduzir [4][14].

A ideia principal é fazer com que indivíduos com melhor aptidão tenham maior chance de se reproduzir. Considerando uma população **m** de cromossomos, organizados em ordem crescente de custos, sendo $C_1 \leq C_2 \leq \cdots C_m$, onde C_i é o custo associado a um cromossomo r_i, um indivíduo é escolhido para ser submetido aos operadores genéticos através de uma distribuição de probabilidade proporcional aos índices dos cromossomos. Com base nesses pontos a função de seleção é dada pela equação 2.1 [4][15] :

$$Select(R) = \left\{ r_j \in R / j = m + 1 - \left\lceil \frac{-1+\sqrt{1+4.Rnd(m^2+m)}}{2} \right\rceil \right. \tag{2.1}$$

Onde:

- $Rnd[0, 1)$ é um número aleatório uniformemente distribuído;
- $\lceil b \rceil$ é o menor inteiro maior que b;
- $R = (r_1, r_2, \cdots, r_m)$ é o conjunto ordenado de cromossomos, de modo que $C_1 \leq C_2 \leq \cdots C_m$;
- $C_i =$ [*Fitness*(r_i)]

As equações 2.2 e 2.3 representam um exemplo de seleção aplicados a uma população de 5 indivíduos:

$$\begin{aligned} Rnd &= 0,3 \\ j_1 = 5 + 1 - \left\lceil \frac{-1 + \sqrt{1 + 4.(0,3).(5^2+5)}}{2} \right\rceil & \qquad (2.2) \\ j_1 &= 3 \end{aligned}$$

$$\begin{aligned} Rnd &= 0,1 \\ j_2 = 5 + 1 - \left\lceil \frac{-1 + \sqrt{1 + 4.(0,1).(5^2+5)}}{2} \right\rceil & \qquad (2.3) \\ j_2 &= 4 \end{aligned}$$

Como o AG pode convergir rapidamente para uma região específica do espaço de busca é necessário que haja um mecanismo para evitar isso. Se o algoritmo convergir para um mínimo

global não há problema, entretanto existe a tendência de convergência rápida para um mínimo local. Este problema pode ser evitado utilizando os operadores de mutação, os quais impõem uma rotina para que se explore outro espaço de busca. Para isso é tomado um novo indivíduo, resultado recente de um cruzamento, e aplica-se a nele, mudando suas caraterísticas hereditárias de forma aleatória. Convém lembrar que a mutação não ocorre com frequência, sendo a probabilidade próxima de zero [14]. A mutação pode ser feita por *Swap*, que consiste em trocar dois genes aleatoriamente de posição [4].

Com base nas definições anteriores é possível construir o seguinte algoritmo para AG [15]:

Algoritmo 2 AG PARA PCV

P1: Construção da população inicial: gere os m cromossomos aleatórios representando os roteiros Hamiltonianos. Calcule os custos de todos os cromossomos gerados, construindo uma lista $R = (r_1, r_2, \cdots, r_m)$, de forma que $C_1 \leq C_2 \leq ..... \leq C_m$; faça $k = 0$, defina o erro ε e o número máximo de iterações k_{max};
P2: Teste: se $C_m - C_1 \leq \varepsilon$ ou $k \geq k_{max}$, então PARE e apresente o cromossomo r_1 ;
P3: Seleção Natural: elimine da lista R o pior cromossomo (último da lista R) e selecione dois cromossomos $r_p = Select(R)$ e $r_q = Select(R)$ com $r_p \neq r_q$;
P4: Reprodução: faça $r_f = Crossover\ (r_p, r_q)$;
P5: Mutação: se o cromossomo r_f não representa um circuito Hamiltoniano, então proceda a mutação $r_f = Mutate(r_f)$;
P6: Calcule $F_f = Fitness(r_f)$ e insira o cromossomo r_f na lista R, mantendo a ordem crescente dos custos; faça $k = k + 1$, e volte ao P2.

2.2 Colônia de Formigas

Os algoritmos de colônia de formigas tiveram suas origens no trabalho de Dorigo, Maniezzo e Colorni [7]. Estes são compostos de uma população de indivíduos concorrentes assíncronos que cooperam globalmente para que se encontre uma boa solução. Conforme os agentes constroem soluções eles deixam marcações nas estruturas, permitindo assim a comunicação entre os agentes. Estas marcas mimetizam a substância química denominada de *feromônio*. O colônia de formigas mimetiza a construção de uma solução aos passos de uma formiga que busca alimento. Assim a construção da solução é feita utilizando uma trilha de decisões que são tomadas com base nos caminhos que foram marcados com feromônio. Uma boa solução é uma trilha de boas decisões, ou seja, é o menor caminho que liga o formigueiro à comida. Sendo assim, as melhores trilhas devem acumular feromônio rapidamente uma vez que as formigas irão passar por ali com mais frequência deixando o feromônio mais forte no local [8].

Os agentes do algoritmo, que neste caso são as formigas artificiais, modificam informações numéricas armazenadas localmente em cada estado do problema por elas visitados. A informação compartilhada leva em consideração o histórico de busca e o desempenho de cada formiga. No geral os algoritmos de colônia de formigas utilizam um mecanismo de evaporação do feromônio que modifica a informação do feromônio conforme o tempo. As formigas se movimentam de um estado para outro estado vizinho. As definições de estado e vizinhança variam de acordo com o problema. A decisão de mudança de estado depende de um evento probabilístico, o qual está associado à intensidade com que esta mudança foi explorada anteriormente. As formigas analisam apenas informações locais, não sendo capazes de realizar qualquer estratégia que preveja estados futuros [12].

É importante notar que as formigas não se comunicam diretamente. Cada uma constrói uma solução movendo-se por uma sequência finita de estados vizinhos. Os movimentos são feitos de acordo com uma probabilidade que leva em consideração a informação da própria formiga (representa uma memória simples para ações locais), a trilha de feromônio e outras informações locais do problema [12].

As formigas artificias diferem das reais nos seguintes aspectos [8]:

1. As formigas artificiais vivem em um mundo discreto;
2. As formigas artificiais possuem memória de suas ações;
3. Para a formiga artificial a quantidade depositada de feromônio é função da qualidade da solução corrente.

Os algoritmos envolvendo as formigas artificiais podem ser enriquecidos com técnicas extras, como otimização local ou *backtracking* (refinamento de força bruta onde múltiplas soluções podem ser eliminadas sem serem explicitamente examinadas).

A figura 2.10 exemplifica como funciona uma colônia de formigas, ficando explícito o comportamento dos agentes mediante a um obstáculo. Nota-se também que no momento final as formigas otimizaram sua rota, o que foi possível através da trilha de feromônio. Uma vez que a rota era mais curta e as formigas passaram por ali com mais frequência, resultou-se numa trilha de feromônio mais forte do que a do caminho mais distante. Neste, as formigas demoravam

mais para passar, tornando o rastro de feromônio por este caminho cada vez mais fraco e a passagem de agentes por ali menos frequente.

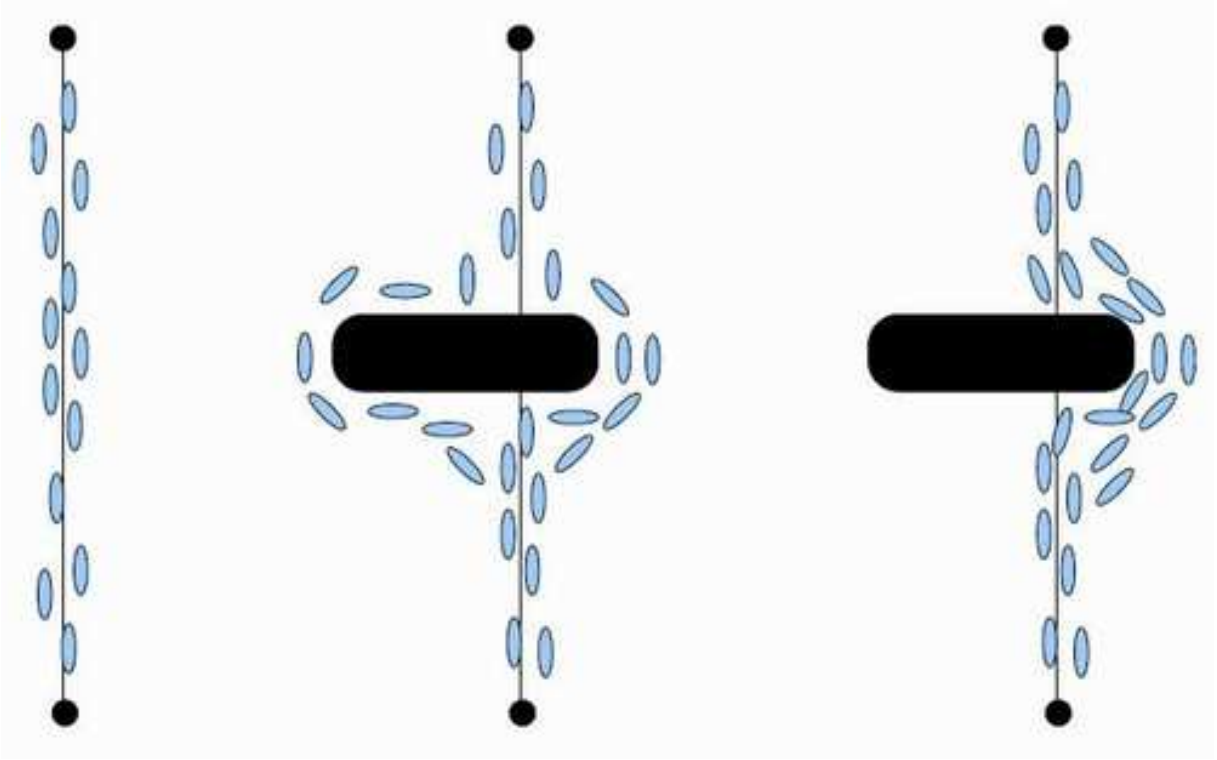

Figura 2.10: Comportamento das Formigas.

2.2.1 Algoritmo *Ant System*

O algoritmo *Ant System* foi uma das primeiras variações dos algoritmos de colônia de formigas aplicado no PCV. O algoritmo consiste na formiga passar por todos os pontos ou cidades de um dado problema liberando assim o feromônio por onde irá passar [16].

Tomando um grafo G (N, A) onde N representa o número de nós e A representando as arestas do problema, uma aresta (i, j) conecta dois nós n_i e n_j com uma distância d_{ij}. O problema consiste em encontrar a combinação de nós que forneça o menor circuito Hamiltoniano possível [4].

Cada formiga é um agente com as seguintes características [7]:

- A formiga decide qual cidade irá a seguir levando em consideração a distância e o nível de feromônio presente em uma aresta;
- Para que a formiga faça um circuito válido, transições para cidades já visitadas não são

permitidas até que o circuito seja completado. Isto é feito através da lista Tabu que cada formiga contém, que consiste em armazenar na memória da formiga por qual caminho ela já passou;

- Quando o circuito é completado, o rastro de feromônio é adicionado a aresta (i, j) no qual a formiga passou.

Tendo estes pontos em vista a probabilidade de uma cidade j ser selecionada, pode ser dada pela seguinte equação [8][4]:

$$p_{ij} = \begin{cases} \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{h\in\Omega}[\tau_{ih}]^{\alpha}[\eta_{ih}]^{\beta}} \text{ , } se\ j \in \Omega \\ \\ 0, \text{ caso contrário} \end{cases} \tag{2.4}$$

onde:

τ_{ij}: Intensidade do feromônio na aresta (i,j);

α: Parâmetro que regula a influência de τ_{ij};

η_{ij}: Representa a visibilidade da cidade j para a i, onde $\eta_{ij} = \frac{1}{d_{ij}}$;

β: Parâmetro que regula a influência de η_{ij};

Ω: Conjunto das cidades que não foram visitadas ainda;

d_{ij}: distância entre as cidades i e j;

Este processo de seleção é repetido até que todas as formigas completem o circuito. A cada passo da iteração uma cidade é removida até que sobre apenas uma cidade a ser selecionada. Neste caso, a probabilidade de se selecionar esta cidade se torna $p_{ij} = 1$. Após é calculada a distância percorrida nos circuitos feitos por cada formiga para que então seja atualizada a melhor rota até o momento [8].

Cada formiga libera uma quantidade Q de feromônio por onde passa. Análogo a natureza, parte da trilha de feromônio evapora, sendo reduzidas à um fator $(1 - \rho)$ antes de novas trilhas serem feitas. Isto é feito para evitar uma convergência prévia e é regulada por um parâmetro ρ. A atualização pode ser feita de duas maneiras, sendo atualizada por toda formiga que percorre

o problema, ou somente pela formiga que fez a melhor rota na iteração atual. A equação 2.5 representa a atualização de feromônio feita por apenas uma formiga enquanto a equação 2.6 atualiza o feromônio de acordo com cada formiga [7]:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \text{ e } \Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} \text{ se a formiga k percorre a aresta (i, j)} \\ 0 \text{ caso contrário} \end{cases} \tag{2.5}$$

onde

$\Delta\tau_{ij}$: Aumento total de feromônio numa aresta (*i*, *j*);

m: Número de formigas

$\Delta\tau_{ij}^{k}$: Aumento de feromônio numa aresta (*i*, *j*) causada por uma formiga k;

Q: quantidade de feromônio liberada por uma formiga em um circuito;

L_k: distância de um circuito realizado por uma formiga k;

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{1}{L_k(t)} \text{ se (i, j)} \in T^k(t) \\ 0 \text{ se (i, j)} \notin T^k(t) \end{cases} \tag{2.6}$$

onde

t é o contador de iterações, $T^k(t)$ é o tour realizado pela formiga k na t-ésima iteração com comprimento $L^k(t)$.

Ao fim a evaporação do feromônio é dada por 2.7.

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij} \tag{2.7}$$

Onde [$\rho \in$ [0,1]:] Parâmetro que regula a redução de τ_{ij}(evaporação);

Tendo como base o que foi descrito, temos o seguinte pseudocódigo do *Ant System* [8]:

Algoritmo 3 PSEUDOCÓDIGO DO "*Ant System*"

Inicialize
para *t = 1 até número de iterações* **faça**
  para *k = 1 até* m **faça**
    repita
      Selecione a cidade j a ser visitada por próximo
      com a probabilidade p_{ij} dada pela equação 2.4;
    até *a formiga* k *completar o ciclo*;
    Calcule o custo total L_k da rota gerada pela formiga k
  Atualize os níveis de feromônio τ_{ij} em todas as arestas de acordo com 2.7

Capítulo 3

Implementação de métodos e Metaheurísticas

3.1 Problemas Utilizados

Para este trabalho os problemas foram extraídos da TSPLIB [5], a qual é uma famosa biblioteca que contém uma grande quantidade de problemas do caixeiro viajante. Estes são arquivos simples de texto, os quais existem em dois formatos: matriz de adjacência ou em uma lista de coordenadas x e y de cada nó ou vértice. Os arquivos possuem a extensão ".tsp" e são nomeados geralmente por uma palavra acompanhada de um número, representando a cidade ou país e seu respectivo tamanho como por exemplo: brazil58.tsp, representando um problema com 58 coordenadas de solo brasileiro. A biblioteca também possui o resultado ótimo para cada problema.

A tabela 3.1 corresponde aos problemas de entradas escolhidas para os testes. Todos estes problemas são simétricos e completos, o que significa que todas as cidades são interligadas entre si e as distâncias de ida e volta de uma cidade para outra iguais. Nos problemas selecionados variam de 51 a 1291 cidades.

Tabela 3.1: Arquivos de entrada para os testes

Arquivo	Nº de Cidades	Solução Ótima
eil51	51	426
eil101	101	629
kroA200	200	29368
linhp318	318	41345
d439	439	35002
d657	657	48912
rat783	783	8806
d1291	1291	50801

3.2 Implementação computacional

Este trabalho busca analisar comparativamente as duas metaheurísticas na resolução dos PCVs: *Ant System* e Algoritmos Genéticos na resolução de PCVs. Pesquisas relacionadas a solução do PCV encontraram parâmetros para os algoritmos responsáveis por encontrar boas soluções para cada problema. Para o formigas utilizou-se os parâmetros de Dorigo [7] : sendo m (nº de formigas) = 10, $\alpha = 1$, $\beta = 2$, e $\rho = 0,98$ e o número de iterações sendo igual ao número de cidades do problema.

Para o AG foram utilizados os mesmos parâmetros vistos em Benevides [4]. Os parâmetros utilizados foram: o tamanho da população de 50, número de gerações igual a 10000 e a probabilidade de mutação 1%. O operador de cruzamento utilizado foi o OX, pois como visto em Rani e Kumar [18] retornou melhores resultados do que o PMX e o CX.

As implementações foram feitas em JAVA em uma máquina com as seguintes especificações:

- Processador: Intel Core i5-4200U, 1.60 GHz
- Memória (RAM): 8,00 GB
- Tipo de Sistema Operacional: 64 bits, processador base em x64.

Para a implementação do algoritmo *Ant System* foram utilizados conceitos de orientação objeto. Foram criadas algumas classes para representar o agente formiga e o grafo do problema em si. Para a formiga foi criada a classe “Ant.class”, onde esta possui um atributo identificador, um atributo para identificar sua posição atual, uma lista tabu a qual armazena as cidades pelas

quais a formiga já passou e a distância total percorrida pela formiga até o momento. Para a representação de um vértice foi criada a "City.class" onde possui as coordenadas x e y e um identificador próprio. Para uma aresta foi utilizada a "Path.class", a qual armazena as duas cidades das suas extremidades, a medida da aresta, um identificador e o nível de feromônio na mesma. E por fim a classe "Tour.class" utilizada para representar uma rota percorrida por uma formiga, a qual contém um vetor de arestas contendo as referências para os vértices pelos quais a formiga passou. A figura 3.1 representa o diagrama de classes do *Ant System*.

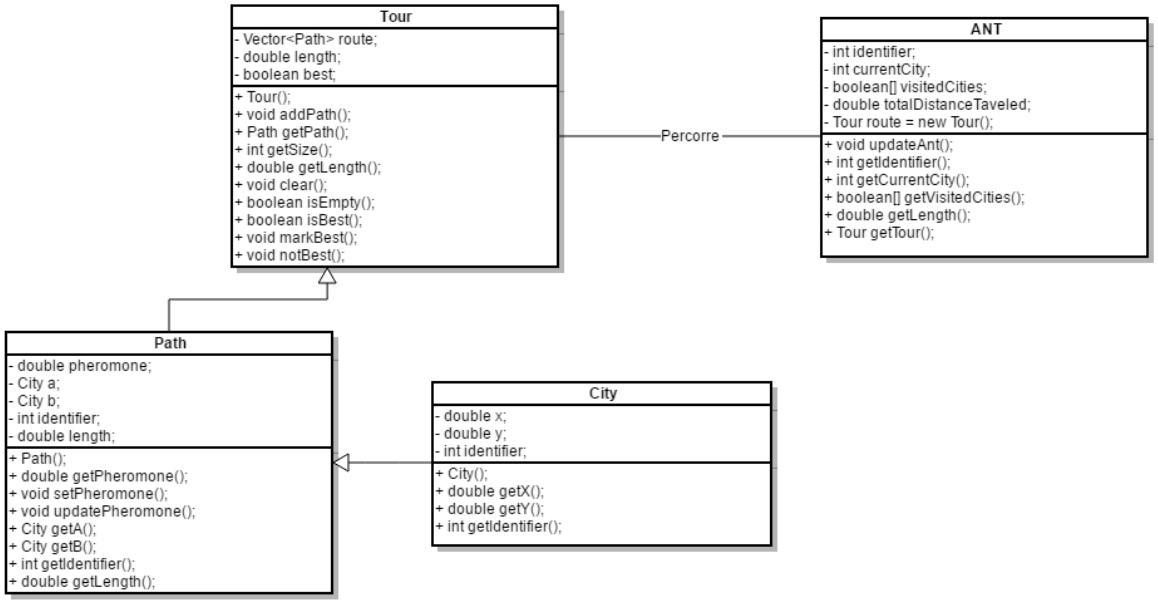

Figura 3.1: Diagrama de Classes do *Ant System*.

Através desta estrutura foi possível a implementação do algoritmo *Ant System*. Onde para a execução do algoritmos são solicitados os seguintes parâmetros: Número de Iterações, número de formigas, α, β, o valor de ρ e por fim o arquivo do problema retirado da TSPLIB. A implementação do *Ant System* não envolveu o uso de nenhuma interface, focando apenas na execução do código e no retorno dos resultados em modo texto.

Formigas são liberadas a cada iteração, selecionando caminhos pela regra probabilística apresentada no capítulo 2 até que se forme uma rota completa, após o fim de cada iteração é analisado o melhor caminho encontrado até o momento, e assim o feromônio é atualizado com base no caminho da formiga que o fez, enquanto os outros são descartados. A evaporação do feromônio é feita para cada aresta antes da próxima iteração ser iniciada. Para o *Ant System*

foram utilizados os parâmetros do pesquisador Dorigo [17].

Para a implementação do Algoritmo Genético, foi criada uma estrutura com as seguintes classes: “City.class”, que representa uma cidade no grafo, com suas respectivas coordenadas. Uma classe “Tour.class”, que possui as informações de uma rota realizada e também métodos para gerar a própria rota e retornar seu *fitness*. Uma classe “TourManager.class”, responsável por manipular várias rotas. Uma classe “Population.class”, que representa a população que possui uma lista de rotas, sendo cada rota um indivíduo da população atual. E por fim a classe “GA.class”, que possui os métodos principais do algoritmo genético, sendo eles de seleção, cruzamento e mutação. A figura 3.2 representa o diagrama de classes sobre o AG.

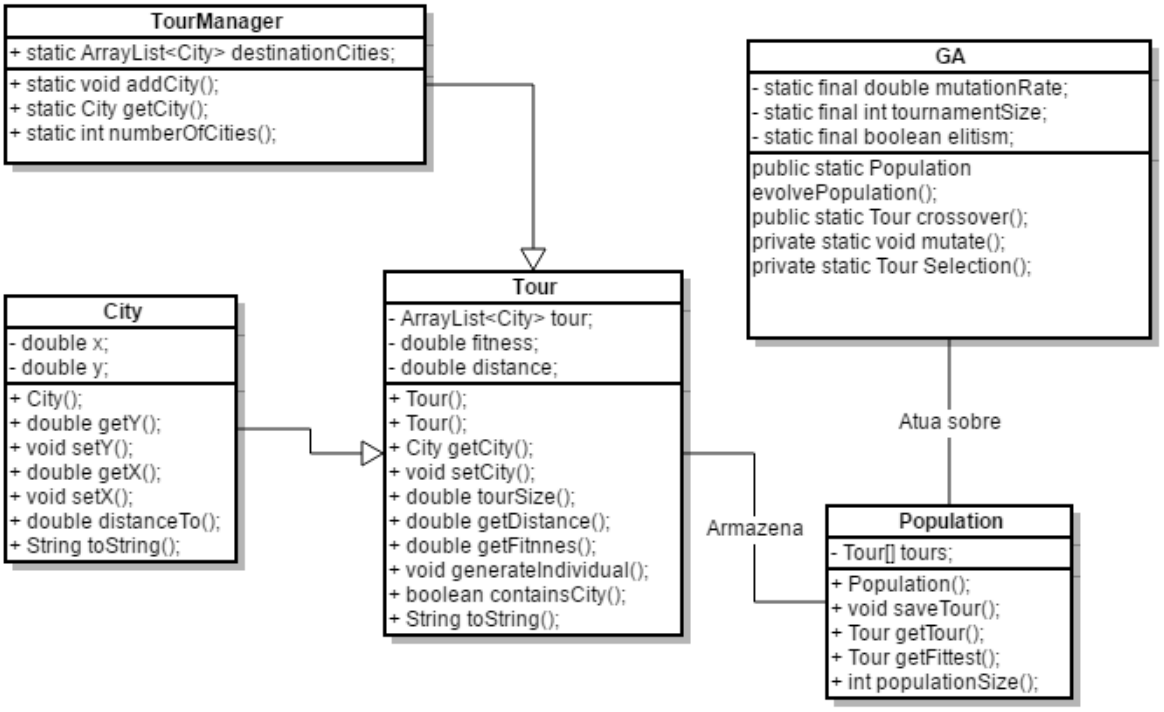

Figura 3.2: Diagrama de Classes do AG.

Para a execução deste algoritmo são solicitados: o número de gerações, o tamanho da população, a taxa de mutação e por fim o arquivo que será utilizado da TSPLIB [18].

Inicialmente é gerada uma população inicial de forma randômica, para então o algoritmo ser executado pelo número de gerações definidas. Em seguida a seleção é feita pelo processo de seleção natural apresentado em 2.1. O cruzamento foi feito pelo operador OX visto na seção 2.1.1. A mutação quando solicitada é feita pelo método *Swap* que consiste na troca aleatória de

dois genes. Antes da seleção seguinte é realizado o processo de elitismo, mantendo o indvíduo com o melhor *fitness* da etapa anterior. Ao final da execução do algoritmo é retornado o custo da rota do indivíduo mais apto.

Capítulo 4

Resultados Obtidos

Para os testes foram utilizados os mesmos parâmetros descritos em Dorigo [17] por terem obtido resultados bons para uma quantidade razoável de problemas, sendo *m* (nº de formigas) = 10, $\alpha = 1$, $\beta = 2$, e $\rho = 0,98$ e por fim o número de iterações sendo igual ao número de cidades do problema.

Para cada problema o algoritmo foi executado 10 vezes, obtendo-se uma média das soluções, uma vez que as formigas são liberadas em cidades diferentes a cada execução podendo ocasionando alguma variação nos resultados. A tabela 4.1 apresenta a solução ótima para os problema, a média das soluções obtidas pelo *Ant System* e a diferença em percentual.

Tabela 4.1: Tabela de médias dos resultados do *Ant System*

Arquivo	Solução Ótima	Média das Soluções	Tempo (Seg.)	Diferença (%)
eil51	426.0000	503.6537	0.8543	18.2285
eil101	629.0000	685.6238	2.9573	9.0022
kroA200	29368.0000	31475.1973	31.2542	7.1751
linhp318	41345.0000	45429.5490	63.1228	9.8791
d493	35002.0000	38798.9786	269.7245	10.8478
d657	48912.0000	58681.9927	300.5373	19.9746
rat783	8806.0000	10785.1718	1243.8274	22.4752
d1291	50801.0000	65599.5383	8393.4410	29.1204

O *Ant System* apresentou, na média boas soluções (na média) para a metade dos problemas, sendo consideradas boas soluções aquelas com uma variação menor ou igual a 10%. O resultado do problema d493 foi incluído com boa solução pelo motivo de ter tido uma pequena variação acima de 10%. Cabe observar que o *Ant System* se mostrou eficiente em problemas de porte

pequeno médio (51 a 439 nós), como observado em Dorigo [17].

Na tabela 4.2 é possível observar os melhores resultados encontrados, para cada problema, pela aplicação do *Ant System*. Apesar do *Ant System* não ter alcançado o resultado ótimo para nenhum problema, o mesmo chegou bem próximo deste para os problemas eil51 e kroA200 (diferença menor que 4%). Porém que o mesmo pode ter uma variação relevante significante entre uma execução ou outra, como é o caso do problema eil51, onde a melhor solução alcançou uma diferença de apenas 3% e no pior chegou a 20% de diferença.

Tabela 4.2: Tabela dos melhores resultados do *Ant System*

Arquivo	Solução Ótima	Melhor Solução Encontrada	Diferença (%)
eil51	426.0000	442.7218	3.9253
eil101	629.0000	672.8448	6.9705
kroA200	29368.0000	30324.9778	3.2585
linhp318	41345.0000	43767.6236	5.8595
d493	35002.0000	37169.1972	6.1916
d657	48912.0000	57680.8735	17.9278
rat783	8806.0000	10173.2184	15.5259
d1291	50801.0000	63648.7557	25.29035

Para a execução do AG foram utilizados os mesmos parâmetros vistos em Benevides [4] uma vez que retornaram bons resultados. O algoritmo foi executado 10 vezes da mesma forma que o *Ant System*, e obteve-se a média das soluções 4.3.

Tabela 4.3: Tabela de médias dos resultados do AG

Arquivo	Solução Ótima	Média das Soluções	Tempo(Seg.)	Diferença (%)
eil51	426.0000	439.0321	90.9423	3.0516
eil101	629.0000	655.4131	170.4401	4.1335
kroA200	29368.0000	30632.9112	480.0029	4.3040
linhp318	41345.0000	43781.3921	792.1242	5.8918
d493	35002.0000	38924.3211	2100.4021	11.2050
d657	48912.0000	60325.2196	3625.1226	23.3337
rat783	8806.0000	9342.3256	4207.9982	6.0867
d1291	50801.0000	67312.3213	9124.7319	32.5013

O AG se mostrou eficiente em todas as variedades de tamanho de problema analisados, onde encontrou boas soluções(diferença menor que 10% da solução ótima). O AG se mostrou

um algoritmo pesado, sendo sua execução um tanto lenta para problemas de médio a grande porte.

A tabela 4.4 mostra os melhores resultados encontrados pelo AG, onde o mesmo não encontrou o resultado ótimo mas em alguns problemas ficou muito próximo deste (diferença menor que 3%), como é o caso dos problemas eil51, eil101 e rat783, demonstrando assim a capacidade do AG de gerar boas soluções para o problema do PCV.

Tabela 4.4: Tabela com os melhores resultados do AG

Arquivo	Solução Ótima	Melhor Solução Encontrada	Diferença (%)
eil51	426.0000	432.0553	1.4214
eil101	629.0000	643.3421	2.2801
kroA200	29368.0000	30339.8462	3.3092
linhp318	41345.0000	42982.3921	3.9603
d493	35002.0000	37842.0032	8.1138
d657	48912.0000	60325.2196	23.3345
rat783	8806.0000	8993.1924	2.1257
d1291	50801.0000	66423.8734	30.7530

4.0.1 Comparação

Comparando os resultados obtidos 4.5, verificou-se que na média o AG foi melhor que o *Ant System*, encontrando soluções melhores em cinco dos oito problemas utilizados, sendo que em dois a redução foi de mais de 10%. O AG ganhou também no quesito de boas soluções (variança menor que 10%), encontrando boas soluções para cinco problemas enquanto que o AS encontrou boas soluções para quatro problemas.

O *Ant System* se mostrou eficiente no quesito de encontrar boas soluções, entretanto não encontrando resultados melhores que o AG na maioria dos casos, mas realizou a execução em tempo melhor que o AG.

Porém, como o objetivo do trabalho foi analisar qual das metaheurísticas retorna uma melhor solução para o problema, pode-se afirmar que o AG revelou-se melhor nos problemas analisados. onde o algoritmo varia muito de uma execução para outra ao retornar um resultado final.

Vale lembrar que ambas as metaheurísticas são bastante modulares, permitindo assim a reimplementação de várias partes e assim possibilitando o retorno de resultados diferentes, bem

como tempos de execução variados. Além disso, o número de problemas aqui avaliados foi reduzido.

Assim foi concluído que, ao se desejar uma resposta de maior qualidade é recomendado o uso o Algoritmo Genético, enquanto quando se deseja uma resposta em tempo menor deve-se se utilizar o *Ant System*,

Tabela 4.5: Comparação entre as Médias do *Ant System* e AG

Arquivo	Solução Ótima	Média das Soluções(*Ant System*)	Média das Soluções (AG)
eil51	426.0000	503.6537	439.0321
eil101	629.0000	685.6238	655.4131
kroA200	29368.0000	31475.1973	30632.9112
linhp318	41345.0000	45429.5490	43781.3921
d493	35002.0000	38798.9786	38924.3211
d657	48912.0000	58681.9927	60325.2196
rat783	8806.0000	10785.1718	9342.3256
d1291	50801.0000	65599.5383	67312.3213

Capítulo 5

Considerações Finais e Sugestões para Trabalhos Futuros

Este trabalho visou a comparação entre duas metaheurísticas: Ant System e Algoritmos Genéticos na resolução do Problema do Caixeiro Viajante.

Nos resultados apresentados a Metaheurística AG se mostrou mais eficiente ao retornar melhores soluções para os problemas analisados, se mostrando eficaz na resolução da maioria dos problemas utilizados neste trabalho, cujos tamanhos eram variados. Com o *Ant System* obteve-se resultados bons para problemas de pequeno porte, como observado no trabalho de Dorigo [17], entretanto existe a possibilidade de se obter resultados variados escolhendo outros parâmetros para a execução da metaheurística. O *Ant System* retornou boas soluções, apesar de serem piores que o AG para a maioria dos casos, mas o tempo de execução do *Ant System* é mais viável.

Neste trabalho foi observado que, o AG pode encontrar boas soluções para problemas de tamanhos variados, porém, o tempo de execução é alto em problemas grandes. O *Ant System* foi capaz de encontrar soluções boas para problemas de pequeno e médio porte, mas não conseguiu o mesmo desempenho para os problemas de porte maior.

Assim, interessados em utilizar o AG devem levar em conta sua eficiência e tempo de execução, buscando alternativas de implementação ou parâmetros diferentes para sua utilização, caso o tempo seja relevante. No caso do *Ant System* o algoritmo retornou bons resultados em tempo viável, entretanto, com o aumento do problema suas soluções foram perdendo qualidade. Sugere-se então para quem deseja utiliza-la realizar teste para ajuste de parâmetros ou aplicar heurísticas de melhoria de rota após sua execução, por exemplo.

Para trabalhos futuros sugere-se mais testes com as mesmas metaheurísticas já que as mesmas permitem que sejam implementadas de maneiras diferentes, as quais podem implicar em uma mudança de resultado e tempo de execução. Recomenda-se também um estudo mais aprofundado na questão dos parâmetros, pois os mesmos podem variar o resultado obtido dependendo do problema a ser resolvido.

Referências Bibliográficas

[1] GANHOTO, M. A. *Abordagens para problemas de roteamento*. Tese (Doutorado) — Dissertação de Mestrado, Universidade Estadual de Campinas, SP, 2004.

[2] GOLDBARG, M. C.; LUNA, H. P. L. *Otimização combinatória e programação linear: modelos e algoritmos*. [S.l.]: Elsevier, 2005. 331-368 p.

[3] PRESTES, Á. N. *Uma análise experimental de abordagens heurísticas aplicadas ao problema do caixeiro viajante*. Tese (Doutorado) — Universidade Federal do Rio Grande do Norte, 2006.

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

[5] REINELT, G. Tsplib a traveling salesman problem library. *ORSA journal on computing*, INFORMS, v. 3, n. 4, p. 376–384, 1991.

[6] RAFF, S. Routing and scheduling of vehicles and crews: the state of the art. *Computers & Operations Research*, Elsevier, v. 10, n. 2, p. 63–211, 1983.

[7] DORIGO, M.; MANIEZZO, V.; COLORNI, A. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, IEEE, v. 26, n. 1, p. 29–41, 1996.

[8] BULLNHEIMER, B.; HARTL, R. F.; STRAUSS, C. *A new rank based version of the Ant System*. SFB Adaptive Information Systems and Modelling in Economics and Management Science, 1997.

[9] GREFENSTETTE, J. J. Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, IEEE, v. 16, n. 1, p. 122–128, 1986.

[10] HOLLAND, J. H. Information processing in adaptive systems. In: *Information Processing in the Nervous System: Proceedings of the International Union of Physiological Sciences*. [S.l.: s.n.], 1962. v. 3, p. 330–339.

[11] LUCAS, D. C. *Algoritmos Genéticos: uma Introdução*. Universidade Federal do Rio Grande do Sul. Apostila, 2002.

[12] GOLDBARG, M. C.; GOLDBARG, E. G.; LUNA, H. P. L. (Ed.). *Otimização Combinatória e Metaheurísticas: Algoritmos e Aplicações*. RJ, Brasil: Elsevier Editora Ltda., 2016.

[13] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675.

[14] MALAQUIAS, N. G. L. *Uso dos algoritmos genéticos para a otimização de rotas de distribuição*. Dissertação de Mestrado em Ciências, Universidade Federal de Uberlândia - UFU, 2006.

[15] MAYERLE, S. F. Um algoritmo genético para solução do problema do caixeiro viajante. *Artigo de circulação interna do departamento de Engenharia de Produção e sistemas da UFSC*, 1994.

[16] COLORNI, A.; DORIGO, M.; MANIEZZO, V. Distributed optimization by ant colonies. In: *Proceedings of the first European conference on artificial life*. Paris, FRA: [s.n.], 1991. v. 142, p. 134–142.

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

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