Heurísticas computacionais para o Problema do Comprador Viajante

Matheus Souza Leão[1], Márcia Cristina Valle Zanetti[1]

[1]Instituto Federal de Educação Ciência e Tecnologia do Sudeste de Minas Gerais – Campus Juiz de Fora

matsouzaleao@gmail.com, marciavalle@gmail.com

Abstract. *This article deals with methodologies to indicate solutions to the Traveling Buyer Problem, a generalization of the classic Traveling Salesman Problem. In the determination of possible methods of solving the proposed problem, a systematic review of the literature was carried out in order to identify computational heuristics capable of solving it. Among the different heuristic algorithms identified, methodological combinations using Genetic Algorithm with Local Search operators and refinement heuristics called Tour Reduction and Path Relinking were selected for this work. The computational tests performed allowed to compare the performance of the proposed combinations and to determine the most robust ones, with respect to the processing cost of each combination, as well as their capacity of convergence for suboptimal solutions.*

Resumo. *Este artigo trata sobre metodologias para indicar soluções para Problema do Comprador Viajante, uma generalização do clássico Problema do Caixeiro Viajante. Na determinação de possíveis métodos de resolução do problema proposto, foi realizada uma revisão sistemática da literatura a fim de identificar heurísticas computacionais capazes de resolvê-lo. Dentre os diferentes algoritmos heurísticos identificados, foram selecionados para esse trabalho combinações metodológicas que utilizam as técnicas, Algoritmo Genético com operadores de Busca Local e heurísticas de refinamento denominadas Tour Reduction e Path Relinking. Os testes computacionais realizados permitiram comparar a performance das combinações propostas e determinar as mais robustas, no que se refere ao custo de processamento de cada combinação, bem como à sua capacidade de convergência para soluções subótimas.*

1. Introdução

De acordo com Bontoux [2008], o Problema do Comprador Viajante, também conhecido na literatura como *Travelling Purchaser Problem* (TPP), é uma generalização do Problema do Caixeiro Viajante, tratado mundialmente por *Travelling Salesman Problem* (TSP). Descrito inicialmente por Ramesh [1981], o TSP consiste em definir a melhor rota (*tour*) de deslocamento entre várias cidades, proporcionando que

cada cidade possa ser visitada ao menor custo possível. No TPP essas cidades podem ser representadas por mercados, nos quais são oferecidos produtos. O objetivo do problema é de posse de uma lista de produtos a serem adquiridos nesses mercados, determinar o tour de deslocamento mínimo que passe por um subconjunto de mercado e que permita a aquisição dos produtos da lista de compras ao menor custo total possível.

Na representação do TPP, tem-se um conjunto de m mercados, i itens e k produtos a serem comprados, sendo k um subconjunto de i, onde apresenta-se:

- $M = \{m_1, m_2, m_3, ..., m_m\}$, conjunto de mercados, acrescidos da origem (m_0);

- $I = \{i_1, i_2, i_3, ..., i_i\}$, conjunto de itens ofertados para cada mercados M;

- $L = \{l_1, l_2, l_3 ... l_k\}$, conjunto de itens que devem ser comprados no *tour*;

- $D = (D_{mn})$, matriz que representa o custo de deslocamento entre os mercados m e n;

- C = (Cim); matriz que representa o custo do item i no mercado m.

O objetivo proposto é estabelecer um tour fechado que parta da origem m_0, passe por um subconjunto (S) de M e retorne a m_0, de forma a adquirir uma unidade de cada um dos itens constantes em L, e cuja soma dos custos de aquisição dos k produtos e do deslocamento entre os S mercados selecionados seja minimizado.

$$Zmin = \sum Smn + \sum Skm$$

No caso de um produto não estar disponível em um mercado lhe será atribuído um custo de valor elevado, com caráter de penalidade associada a incapacidade aquisição do produto, de forma que o procedimento de seleção evite elencar tal mercado para a aquisição do produto.

O TPP é um problema classificado como NP-difícil, pois não apresenta solução ótima em tempo polinomial para instâncias grandes.

Ao buscar pelas palavras chaves *"Travelling Purchaser Problem" OR "Biobjective travelling purchaser problem"* nos motores de busca *IEEE Digital Library*, *Science Direct*, *ACM Digital Library*, *Scopus* e *Google Acadêmico* foram encontrados 132 artigos, dos quais 17 foram selecionados com base em sua pertinência ao tema e importância na literatura.

A literatura aponta que a primeira tentativa para a resolver o TPP foi elaborada Ramesh [1981], que propôs um algoritmo *lexicographic search procedure*, baseado em uma solução exata. Posteriormente, outros autores também apresentaram trabalhos com a mesma finalidade, baseados em soluções heurísticas.

Golden [2005] propôs a Heurística de Economia Generalizada (*Generalized Saving Heuristics*), onde a seleção do *tour* se baseia na inserção de mercados com maior número de produtos ofertados e posterior adição ao caminhamento de mercados que permitissem diminuir o custo total da solução. Teeninga [2004], visando melhorar a heurística proposta por Golden, implementou o procedimento *Commodity Adding*, no qual constrói uma solução válida a partir da inserção de mercados no *tour* baseado no critério de menor custo e posterior busca local *Drop* (remoção de um mercado do *tour*) e *Add* (adição de um mercado ao *tour*), complementando ainda o refinamento com uma heurística de perturbação para escapar de mínimos locais. Ong [1982] desenvolveu uma heurística de poda (*Tour Reduction*) na qual, após a determinação de um *tour* completo

entre todos os mercados, remove mercados da solução parcial e redistribui os itens anteriormente comprados no mercado removido para os outros mercados. A heurística desenvolvida por Boctor [2003] estabelece uma fase construtiva baseada em Golden [1981] e completa essa combinação com uma fase de Refinamento baseada no *Tour Reduction* de Ong [1982].

Ademais, Angelelli [2009] propôs um Algoritmo Construtivo Guloso que realiza busca por mercados baseado em proximidade e maior número de aquisições, Riera-Ledesma [2005] exploram a busca local com Add, Drop e *Exchange* (troca de mercados). Voß [1996] utiliza a estratégia de Busca Tabu. Bontoux [2008] implementa o algoritmo de Colônia de Formigas e obtém resultados muito expressivos, convergindo ao ponto ótimo em menor tempo de processamento.

Outra estratégia que apresenta elevado poder de convergência para boas soluções é o Algoritmo Genético (AG). A literatura aponta estudos com resultados significativos onde foram implementados AG em diferentes versões. Bontox [2010] propõe um algoritmo genético com o uso de criação de clusters nos crossovers combinando métodos com busca local, *Memetic Genetic Algorithm*. Goldbarg [2008] utiliza um Algoritmo Transgenético baseado na evolução dos plasmídios e transpossomos, espelhando a reprodução dos mesmos para a criação de novas soluções. A proposta baseia-se no caso dos plasmídios, na permutação dos anéis de DNA que são replicados independente do cromossomo e para os transpossomos na movimentação de segmentos de DNA ocorrem espontaneamente de uma posição para outra. Os resultados obtidos com o Transgenético mostraram-se expressivos ao alcançar o ótimo das instâncias citadas na literatura.

Inspirado pela literatura, que demostra o sucesso das implementações de Algoritmos Genéticos e métodos de busca local, este trabalho tem por objetivo propor uma combinação de estratégias, baseadas na combinação entre Algoritmo Genético, critérios de busca local e operadores de refinamento como *Path Relinking* e *Tour Reduction* [Ong, 1982] para a resolução do TPP.

2. Algoritmo Genético

Algoritmos genéticos são heurísticas computacionais baseadas na teoria evolucionária das espécies e na seleção natural. Em AG um cromossomo representa os diversos parâmetros que podem ser variados de forma a solucionar o problema. A partir da variação desses parâmetros é gerado um conjunto de soluções possíveis que juntas formam uma população de diferentes indivíduos, chamada geração.

Na geração da população inicial podem ser adotados procedimentos de geração aleatória ou através de procedimentos orientados por heurísticas. Cada indivíduo gerado é submetido a avaliação por meio de uma função objetivo, que avalia a sua aptidão frente ao objetivo a ser alcançado [Reeves, 2003].

Posteriormente adotam-se critérios de seleção de forma a elencar um subconjunto de indivíduos da população capazes de dar origem a uma nova geração de indivíduos, privilegiando na seleção os mais aptos dentro da população. Operações de mutação e cruzamento são aplicadas aos cromossomos elencados de forma a recombinar soluções existentes e criar novas soluções filhas. Na mutação, novas informações são

incorporadas ao indivíduo e no cruzamento ocorre a combinação entre pares de cromossomos pré-existentes [Reeves, 2003].

A cada iteração do AG uma nova geração de indivíduos é criada, e seleciona-se novamente uma porcentagem da população mais apta, ou seja, soluções com a melhor qualidade. Com essa nova população começa novamente o ciclo, repetindo os passos do AG até alcançar o critério de parada [Reeves, 2003 e Fernandes, 2014]. Como critério de parada pode ser usado a fixação de um número finito de gerações ou prolongar o procedimento até que se obtenha um número mínimo de gerações sem melhoria.

2.1 Algoritmo Genético Proposto

Baseado nos procedimentos investigados na literatura, cujos relatos demostram diferentes *inputs* para criação dos operadores, para este trabalho foram selecionados 3 métodos de mutação distintos, 2 métodos de cruzamento e 4 métodos de busca local, de forma que por meio da combinação metodológica se possa potencializar as soluções obtidas.

Nos procedimentos adotados neste trabalho, alguns passos foram sensivelmente diferentes dos propostos no AG tradicional com alguns ajustes baseados nas indicações obtidas na revisão bibliográfica, de forma a aliar combinações de metodologias mais promissoras [Lima, 2017; Bontox, 2010, Riera-Ledesma, 2005].

2.1.1. Cromossomo e População

Para o TPP, o cromossomo representa a sequência de mercados a serem visitados dentre todos os mercados disponíveis, de forma que nesse subconjunto de mercados possa se adquirir os itens da lista de compras. Computacionalmente, o cromossomo é representado por um vetor no qual, em cada uma de suas posições é armazenado o índice do mercado selecionado na ordem sequencial da visitação.

Cabe observar que os cromossomos são dinâmicos e podem apresentar tamanhos distintos, pois foi utilizado um método baseado no empirismo de compras, no qual o número de mercados visitados pode variar. Isso se deve à própria característica do problema, já que percorrer mais mercados nem sempre garante um ganho no custo da solução total. Uma rota de compras pode ser atendida até mesmo por uma visita a um único mercado. Por isso o número de mercados roteados pode crescer ou diminuir devido aos operadores de busca local.

Neste trabalho o AG proposto teve sua população iniciada com 100 cromossomos gerados aleatoriamente.

A função de aptidão utilizada para elencar os indivíduos mais aptos a gerar as populações futuras foi calculada a partir da soma das distâncias percorridas entre mercados, acrescida do custo de aquisição dos produtos da lista de compras.

O processo de seleção que levantou os indivíduos mais aptos foi realizado pelo método de classificação, no qual os indivíduos da população são selecionados a partir da função de aptidão e a probabilidade de escolha é maior para os mais bem classificados.

2.1.2. Mutação

Partindo da mesma essência do algoritmo genético, a mutação se baseia na natureza, onde ocorre de forma aleatória ou induzida. A mudança de um cromossomo em uma cadeia de DNA ou RNA promove mudanças no ser vivo, pode acarretar que essa mudança o torne mais apto a sobreviver.

Seguindo esse conceito, foram implementados três procedimentos de mutação, descritos a seguir por seus pseudocódigos:

- *Single*: troca de um nó (mercado) do cromossomo por outro aleatoriamente, descrito no Código 1.

```
Single (Solucao solucao)
1. Inicio
2. Remove o ultimo mercado
3. A = Sorteia um mercado aleatório diferente da origem;
4. Se(mercado A já estiver na solução)
5.     Adiciona origem no final da solução;
6. Senão
7.     B = Sorteia posição onde o mercado vai ser inserido diferente
   da origem;
8.     Insere A no tour da solução na posição B;
9.     Adiciona origem ao final da solução;
10. Fim
```

Cógido 1 – Pseudocódigo da função Single

Fonte: Elaboração própria

- *Double*: troca de dois nós (mercados) do cromossomo por outros dois, aleatoriamente. Ele repete o procedimento Single duas vezes.
- *Random*: onde sorteamos um número n de vezes em que a função *Single* é chamada.

As implementações de *Single* e *Double*, foram desenvolvidas baseadas em Riera-Ledesma [2005], que propõe um método chamado como *Consecutive Exchange*.

2.1.2. Cruzamento

O cruzamento ocorre na natureza por meio da troca de material genético, onde pela combinação de genes pode-se gerar um indivíduo mais apto a sobreviver. Do mesmo modo, este trabalho segue esse princípio, adotando também algumas variações baseadas em trabalhos realizados anteriormente, por meio das ações a seguir:

- Permuta: dadas duas soluções, ocorre a criação de duas novas soluções baseadas da combinação das metades das soluções pais. (Código 2).

```
Permuta (Solucao solucao1, Solucao solucao2)
1. Inicio
2. tamMenorSolucao = PegaTamanhoMenorSolucao(solucao1, solucao2);
```

```
3. temp = tamMenorSolucao / 2;
4. novaSolucao1 = solucao1.Clone()
5. novaSolucao2 = solucao2.Clone()
6. cria aux
7. Para( x = 0; x < temp; x++)
8.     novaSolucao1 posição x recebe o valor da solucao2 posição x
9.     novaSolucao2 posição x recebe o valor da solucao1 posição x
10.Fim Para
11.Retorna novaSolucao1, novaSoluacao2
12.Fim Inicio
```

Cógido 2 – Pseudocódigo da função Permuta

Fonte: Elaboração própria

- *Shuffle*: dadas duas soluções, ocorre a criação de uma nova solução pela combinação inteira das duas soluções. Para cada posição da solução ocorre um sorteio para identificar se essa posição vem da primeira solução ou da segunda, até finalizar a criação do *tour* (Código 3).

```
Shuffle (Solucao solucao1, Solucao solucao2)
1. Inicio
2. tamMenorSolucao = PegaTamanhoMenorSolucao(solucao1, solucao2) ;
3. novaSolucao
4. Para( x = 0; x < tamMenorSolucao; x++)
5.     A = Sorteia entre 0 ou 1;
6.     Se (A = 0)
7.       novaSolucao na posição x recebe o valor da solucao1[x]
8.     Senão
9.       novaSolucao na posição x recebe o valor da solucao2[x]
10.}Fim Para
11.Se (solucao1.tour > solucao2.tour){
12.    Para (; x < solucao1.tour.count; x++){
13.      novaSolucao na posição x recebe o valor de solucao1[x]
14.    }Fim Para
15.Senão{
16.    Para(; x < solucao2.tour.count; x++)
17.      novaSolucao na posição x recebe o valor de solucao2[x]
18.    }Fim Para
19.Fim Para
20.Fim Inicio
```

Cógido 3 – Pseudocódigo da função *Shuffle*

Fonte: Elaboração própria

Na linha 2 do procedimento verifica-se qual solução tem o menor *tour* e armazena essa informação na variável tamMenorSolucao. Na linha 4, inicia-se a sequência de interações, de 0 até o valor de tamMenorSolucao. Para cada interação, sorteia-se um

número 0 ou 1 (linha 5) e, na sequência, faz-se a verificação do número sorteado. Caso seja 0, executa-se a linha 7 onde o *tour* da nova solução criada, na posição x, recebe o valor da solução 1 posição x, Caso o valor sorteado seja 1, executa-se a linha 9 onde o *tour* da nova solução criada, na posição x, recebe o valor da solução 2.

Devido as soluções não necessariamente terem o mesmo tamanho de *tour* é necessário fazer um tratamento para que não ocorra uma referência de ponteiro nulo, assim, na linha 11 verifica-se em que solução o *tour* é maior e continua a interação até o final da maior solução identificada, copiando o que resta do *tour* da solução maior (linha 16 e 17) para a nova solução.

2.2. Operador de Busca Local

Métodos de busca local se mostraram muito eficazes para a solução do TPP [Boctor, 2003], especialmente quando aliados aos Algoritmos Genéticos. Tal combinação, por se mostrar muito promissora, originou uma nova linha de pesquisa chamada de *Memetic algorithm* [Bontoux, 2010]. Visto isso, 4 métodos de busca local, baseados nos explicitados na literatura, foram utilizados neste trabalho.

SingleDrop: remove um mercado da solução.

DoubleDrop: remove dois mercados da solução

SingleAdd: insere um mercado na solução.

DoubleAdd: insere dois mercados na solução.

3. Refinamento

Visando potencializar a melhoria das soluções, foram utilizadas duas estratégias de refinamento das soluções geradas pelo algoritmo genético:

3.1. Path Relinking

Path Relinking é um método proposto por Gloover [1963] que tem por objetivo, a partir de um conjunto de soluções, gerar novas combinações de soluções, de modo que as novas combinações geradas possam servir de *inputs* para as próximas combinações, de forma interativa.

O objetivo do método é permitir que duas soluções, de qualidade considerável, possam ser combinadas de forma a permitir varreduras em soluções vizinhas, visando determinar outras soluções que possam ser melhores que as iniciais. Na determinação da vizinhança, os vértices das soluções iniciais são permutados de forma a identificar soluções intermediárias entre elas.

Considerando duas soluções 1 e 2, como na Figura 1, o valor em cada uma das posições da solução 1 (excetuando a origem), irá gerar uma nova solução 2 com o valor constante na solução 1. No exemplo, foram geradas 3 novas soluções, trocando o valor de cada posição da solução 2 pela primeira da posição 1.

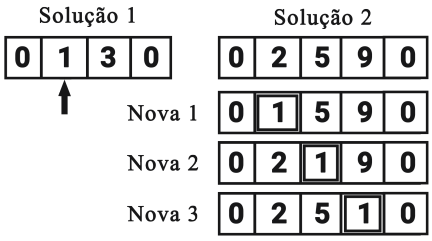

Figura 2 . Permutações *Path Relinking*

Fonte: Elaboração Própria

```
PathRelinking (solucao1, solucao2 )
1. Inicio
2. novaSolucao1 = solucao1.Clone();
3. novaSolucao2 = solucao2.Clone();
4. solucaoAux
5. Lista de solucoesGeradas
6. Para(i=1; i < novaSolucao1.tour.tamanho; i++){
7.     Para(j=1; j < novaSolucao2.tour.tamanho; j++){
8.       solucaoAux = novaSolucao2.Clone()
9.       solucaoAux.tour[j] = novaSolucao1.tour[i]
10.      Se(solucaoAux.VerificaViabilidade()){
11.        solucoesGeradas.add(solucaoAux)
12.      }Fim Se
13.    }Fim Para
14.}Fim Para
15.Retorna solucoesGeradas
16.Fim Inicio
```

Cógido 2 – Pseudocódigo da função Permuta

Fonte: Elaboração Própria

Nas linhas de 2 a 5 são iniciadas as variáveis usadas, que inicialmente serão clones da solução 1 e 2. Nas linhas 6 e 7 iniciam-se as iterações. Na linha 8, a variável soluçãoAux recebe um clone da nova Solução 2. Na linha 9, o tour da soluçãoAux na posição j recebe o valor que está no tour da nova solução 1 na posição i. A seguir, verifica se essa solução é viável. Caso seja, a solução é adicionada na lista de soluções geradas na linha 11, caso não seja viável, ela será sobrescrita na próxima iteração.

3.2. *Tour* Reduction

Tour Reduction [TR] foi uma heurística de poda criada por Ong [1982]. Onde a partir de uma solução completa já criada, se começa a remover mercados do *tour* com o intuito de diminuir seu tamanho, os produtos ali comprados passam a ser redistribuídos entre outros mercados ainda constantes do *tour*. Como o TR é uma heurística que não gera soluções, mas melhora soluções existentes, o mesmo precisa estar aliado a outras heurísticas criacionais, que gerem um inicial *tour* completo utilizado como entrada no procedimento de poda.

Nessa implementação optou-se pelo critério de remoção de mercados com menos produtos oferecidos, de forma a redistribuir esses produtos que seriam comprados nesse mercado para outros mercados.

4. Combinações dos métodos

Os procedimentos descritos anteriormente foram mesclados em 4 combinações metodológicas:

- *Full*: Uma combinação de todos os métodos desenvolvidos.
- *AddDrop_TourReduction*: Uma combinação que visa explorar os métodos de busca local e a inserção de uma perturbação nas soluções com o embaralhamento do método Shuffle. Métodos: *SingleDrop*, *DoubleDrop*, *SingleAdd*, *DoubleAdd*, *Shuffle* e *Tour Reduction*.
- *Single_PathRelinking:* Uma combinação que visa usar apenas operações unitárias e depois fazer um refinamento explorando as soluções próximas. Métodos: *Permuta*, *Single*, *SingleDrop*, *SingleAdd*, *PathRelinking*.
- Refinamento: Explora apenas o uso dos critérios usados para o Refinamento. Métodos: *Path Relinking* e *Tour Reduction*.

5. Instâncias

Para realizar os testes dos algoritmos, foram geradas instâncias de tamanhos distintos, nas quais se variaram o total de mercados e quantidade de produtos,

A matriz de distâncias entre os mercados foi criada sorteando aleatoriamente as coordenadas de cada mercado. A distância entre mercados foi calculada por meio da distância euclidiana entre os mesmos.

$$\sqrt{((x1-x2)^2+(y1-y2)^2)}$$

O mesmo critério de aleatoriedade foi utilizado para gerar a matriz de produtos por mercado. Para cada mercado foram sorteados os custos ou ainda a impossibilidade de aquisição do produto no mercado, indicada por valor muito elevado (9999999), fora do domínio dos preços praticados.

A geração lista de compras seguiu 3 parâmetros distintos:

- Grande: visa comprar todos os produtos ofertados.
- Média: objetiva adquirir metade dos produtos da lista, selecionados aleatoriamente da lista de produtos disponíveis.
- Pequena: propõe a compra de 25% dos produtos disponíveis da lista de produtos.

5.1 – Instâncias Geradas

Nome	Quantidade de Mercados	Quantidade de Produtos
Instancia1	50	50

Instancia2	50	100
Instancia3	100	100
Instancia4	100	500
Instancia5	500	500

6. Implementação e Testes Computacionais

O programa foi desenvolvido na linguagem C#, pelo Microsoft Visual Studio Community 2017. Foi escolhida a linguagem C# por ser uma linguagem muito utilizada no mercado de programação e também por promover suporte para que a aplicação criada se torne, futuramente, uma *API* para um site ou aplicativo. Ademais, a linguagem traz muitos recursos de interação com usuário e plug-ins com integração ao Google Maps entre outros.

Os testes foram realizados em um notebook *Dell* com 4GB de *RAM* e processador Intel I3 1.70GHz e sistema operacional Windows 10 de 64 bits.

A primeira geração do AG proposto foi constituída por meio da criação aleatória de 100 cromossomos, onde não há garantias de que as soluções criadas sejam viáveis. A partir dessa geração, as próximas foram constituídas a partir dos melhores cromossomos da geração anterior, tal princípio básico do AG evita que cromossomos não viáveis sejam elencados, pois quando um cromossomo fere a viabilidade da solução, tem seu custo elevado, associado à penalidade de não aquisição de um produto associado à lista de compras.

O critério de parada adotado na etapa construtiva foi identificar a estabilidade do método após 100 gerações sem nenhuma melhoria na melhor solução da geração, podendo interromper o AG e passar para a próxima fase. Na fase de Refinamento o critério de parada foi reduzido para 5 gerações sem melhoria, pois identificou-se que esse número seria suficiente para identificar a estabilização das soluções.

A análise de desempenho das combinações algorítmicas propostas deve ser realizada por meio da comparação entre os desempenhos de cada uma durante a realização dos testes computacionais. Idealmente, seria interessante comparar os resultados dos testes com os apresentados em bibliotecas públicas de problemas testes, que oferecem dados a serem testados e a solução ideal, matematicamente estabelecida. Entretanto, não foram encontradas bibliotecas públicas de problemas testes que se aplicassem ao TPP. Visto isso, para a realização dos testes, algumas instâncias foram artificialmente geradas, conforme explicitado na seção anterior.

As tabelas a seguir apresentam os resultados das execuções dos algoritmos para cada instância. Em cada uma das tabelas foram listadas as informações referentes à identificação da combinação de algoritmo adotada, o tamanho da lista de compras, o número de gerações necessárias para finalizar o algoritmo, o custo da melhor solução encontrada, tempo em milissegundos necessário para o término do algoritmo, e se ao passar pela fase de refinamento foram obtidas melhorias.

Tabela 1. Resultados da Instancia 1
Fonte: Elaboração própria

Combinação	Lista	Gerações	Custo	Tempo	Refinamento
AddDrop	Grande	153	930	1302	Não
Full	Grande	106	930	2452	Não
Refinamento	Grande	7	1369	45	Sim
Single	Grande	113	930	1309	Não
AddDrop	Media	111	600	860	Não
Full	Media	116	600	3573	Não
Refinamento	Media	9	871	329	Sim
Single	Media	134	600	1660	Não
AddDrop	Pequena	114	427	874	Não
Full	Pequena	111	427	2004	Não
Refinamento	Pequena	7	515	126	Sim
Single	Pequena	113	427	1201	Não

Tabela 2. Resultados da Instancia 2
Fonte: Elaboração própria

Combinação	Lista	Gerações	Custo	Tempo	Refinamento
AddDrop	Grande	141	1154	2117	Não
Full	Grande	144	1154	6315	Não
Refinamento	Grande	10	1544	447	Sim
Single	Grande	151	1377	2944	Não
AddDrop	Media	124	920	1732	Não
Full	Media	111	772	3693	Não
Refinamento	Media	7	10461	92	Sim
Single	Media	131	920	2661	Não
AddDrop	Pequena	136	414	1915	Não
Full	Pequena	148	564	5171	Não
Refinamento	Pequena	8	956	145	Sim
Single	Pequena	111	414	2245	Não

Tabela 3. Resultados da Instancia 3
Fonte: Elaboração própria

Combinação	Lista	Gerações	Custo	Tempo	Refinamento
AddDrop	Grande	307	1534	7964	Não
Full	Grande	191	1424	18566	Não
Refinamento	Grande	10	2078	259	Sim
Single	Grande	165	1636	5429	Não
AddDrop	Media	280	813	6354	Não
Full	Media	123	1121	9826	Não
Refinamento	Media	10	1412	604	Sim
Single	Media	199	911	6383	Não
AddDrop	Pequena	183	485	4571	Não
Full	Pequena	133	485	11310	Não
Refinamento	Pequena	11	1098	745	Sim
Single	Pequena	130	485	3964	Não

Tabela 4. Resultados da Instancia 4
Fonte: Elaboração própria

Combinação	Lista	Gerações	Custo	Tempo	Refinamento
AddDrop	Grande	122	1764	9792	Não
Full	Grande	161	1535	29698	Não
Refinamento	Grande	9	2028	1892	Sim
Single	Grande	136	1889	13436	Não
AddDrop	Media	205	1087	14718	Não
Full	Media	283	1012	48788	Não
Refinamento	Media	10	1460	2148	Sim
Single	Media	149	1007	13559	Não
AddDrop	Pequena	250	628	17477	Não
Full	Pequena	151	699	24863	Não
Refinamento	Pequena	12	740	3586	Sim
Single	Pequena	174	800	14852	Não

Tabela 5. Resultados da Instancia 5
Fonte: Elaboração própria

Combinação	Lista	Gerações	Custo	Tempo	Refinamento
AddDrop	Grande	820	6229	274962	Não
Full	Grande	553	6035	422520	Sim
Refinamento	Grande	17	7424	61978	Sim
Single	Grande	375	5931	168565	Sim
AddDrop	Media	307	3740	102922	Não
Full	Media	529	3096	457082	Não
Refinamento	Media	13	4550	52808	Sim
Single	Media	407	3089	173798	Não
AddDrop	Pequena	714	1894	234785	Não
Full	Pequena	207	1916	185944	Não
Refinamento	Pequena	11	2240	39754	Sim
Single	Pequena	755	1682	312220	Não

Os testes iniciais foram realizados para instâncias pequenas e médias (10, 20, 30, 40 e 50 mercados). Os resultados mostraram o mesmo comportamento para todas as instâncias com até 50 mercados. Por isso, para fins comparativos, optou-se por apresentar somente a instância com 50 mercados.

Para cada combinação de algoritmo, os testes nas diferentes instâncias foram repetidos no mínimo 10 vezes. Nas primeiras execuções feitas pela combinação Full foi constatado que em 80% dos testes alcançava-se o mesmo valor de custo nas execuções. Nos testes realizados sobre a instância 4 foram apresentadas variações nos dados obtidos. Nesse caso, o percentual de assertividade caiu para 60%, porém nas vezes que não foi possível alcançar o melhor resultado conhecido, o total do custo final obtido se manteve, no pior caso, em torno de 25% acima da melhor solução já encontrada.

Nas combinações *AddDrop+TourReduction* e *Single*, a assertividade também se manteve por volta de 60% para instâncias menores e chegou a 40% para as instâncias 4 e 5, apresentando nos piores casos custos finais em torno de 50% do menor custo já obtido.

A combinação de Refinamento foi a que obteve menor assertividade. Nela, o melhor custo foi atingido em apenas em 20% das vezes, chegando a se obter uma diferença de quase 100% entre o custo obtido e o melhor já obtido.

Como é perceptível nas tabelas, a combinação com pior aproveitamento foi aquela que usou apenas o Refinamento. Tal resultado pode ser justificado pelo fato de que, como as heurísticas de *Tour Reduction* e *Path Relinking* partem de soluções existentes e a partir delas buscam encontrar soluções vizinhas melhores, é admissível que as soluções

vizinhas circulem em torno de mínimos locais e não consigam alcançar mínimos globais.

Para instâncias pequenas e médias, instâncias de 1 a 4, constatou-se que a fase de refinamento não foi necessária. Apenas no caso de instâncias maiores, com listas de compras grandes, a fase de refinamento se mostrou efetiva. Com isso, pode-se concluir que, para instâncias de menor porte não se mostrou necessária uma segunda fase de refinamentos, pois o poder de convergência do AG combinado à busca local se mostrou suficiente para atingir resultados eficientes, demonstrando assim que, nesses casos, apenas o algoritmo genético com operadores de busca local tem robustez para convergir para boas soluções para o TPP. Cabe ressaltar que, para instâncias de maior porte, a fase de refinamento mostra-se capaz de potencializar a qualidade das soluções obtidas.

No caso do algoritmo que combinou as três versões propostas, *Full*, os resultados alcançaram, na quase totalidade dos testes realizados, o mesmo poder de convergência que os procedimentos combinados *AddDrop+TourReduction*. Nesses casos, a etapa de refinamento não se mostrou efetiva para melhorar a qualidade das soluções, em se tratando de instâncias de pequeno porte. Por outro lado, no que tange ao tempo necessário para a convergência, a versão *AddDrop+TourReduction*, se mostrou mais eficiente, encontrando soluções de qualidade em menor tempo computacional. Cabe ressaltar que, em um pequeno número de ocasiões, onde instâncias de maior porte demandaram maior processamento, a combinação *Full* conseguiu encontrar resultados melhores, entretanto, o tempo total de necessário para a convergência se tornou praticamente três vezes maior que o necessário para a execução do algoritmo *AddDrop+TourReduction*, que obteve o segundo melhor resultado.

Confirmando as observações anteriores, quando se observa os resultados apresentados para a instância 5 e lista de compras grandes, o algoritmo *Single* apresenta o melhor resultado em temos de custo, obtendo custos ainda mais baixos após a realização da etapa de Refinamento.

Quando falamos de instâncias pequenas e médias, os dados que temos não nos permitem tirar muitas conclusões já que em alguns casos o *Single* apresenta os melhores resultados, em outros o *Full* demostra capacidade de obter custos que os demais não obtêm. A única coisa que podemos concluir é que o tempo de execução para o *Full* é o maior de todos.

7. Conclusão

Nesse trabalho foram pesquisadas heurísticas computacionais para o Problema do Comprador Viajante, também chamado *Travelling Purchaser Problem*. Foi implementado um Algoritmo Genético com métodos de Busca Local inspirados na literatura e um Gerador de Instâncias para realizar os testes. Foram desenvolvidas 4 combinações diferentes do algoritmo e realizados os testes.

Pôde-se concluir que, para instâncias menores que 500 mercados por 500 produtos, não se faz necessário o uso dos métodos de Refinamento, posto que o Algoritmo Genético já contempla métodos de Busca Local.

Quanto à combinação mais simples, que gera soluções aleatórias e posteriormente aplica os métodos refinamento de *Tour Reduction* e *Path Relinking*, se provou ineficaz para a solução do problema.

As 3 versões de Algoritmos Genéticos geradas se mostraram muito competitivas, fazendo que a *AddDrop+TourReduction* fosse a mais indicada para instâncias pequenas por encontrar o mesmo resultado nas instâncias com a metade do tempo.

Entretanto, para instâncias médias os resultados se contradizem no que tange a custo e tempo de processamento, pois a versão *Full* é a mais indicada para garantir a assertividade e diminuir a possibilidade de ficar preso em um mínimo local, mas em contrapartida ela pode chegar a demorar o dobro do tempo das outras implementações.

Ao buscar velocidade de execução em instâncias médias é indicado o uso tanto da versão *AddDrop+TourReduction* como a versão *Single,* que apresentam resultados iguais ao do *Full,* obtendo um gap de 20% do ótimo conhecido.

Para instâncias grandes foi constatado que a melhor versão foi a *Single,* alcançando resultados melhores em todas as listas de compras.

Devido a limitações do tempo disponível para a conclusão do trabalho, não foi viável realizar testes em experimentos reais, com parâmetros elevados de total de mercados e lista de compras. Visto isso, na sequência do presente trabalho, cabe realizar tais procedimentos de forma mais detalhada buscando futuramente investigar o efeito dos algoritmos implementados frente a tais situações.

9. Referências

Angelelli, Enrico, Renata Mansini, and Michele Vindigni. "Exploring greedy criteria for the dynamic traveling purchaser problem." Central European Journal of Operations Research 17.2 (2009): 141.

Boctor, Fayez F., Gilbert Laporte, and Jacques Renaud. "Heuristics for the traveling purchaser problem." Computers & Operations Research 30.4 (2003): 491-504.

Bontoux, Boris, and Dominique Feillet. "Ant colony optimization for the traveling purchaser problem." Computers & Operations Research 35.2 (2008): 628-637.

Bontoux, Boris, Christian Artigues, and Dominique Feillet. "A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem." Computers & Operations Research 37.11 (2010): 1844-1852.

Fernandes, Diogo RM, et al. "A simple and effective genetic algorithm for the two-stage capacitated facility location problem." Computers & Industrial Engineering 75 (2014): 200-208.

Glover, Fred, and Rafael Marti. "Fundamentals of scatter search and path relinking 681 scheduling rules." Carnegie Mellon University. 1963.

Goldbarg, Marco César, Ligia Bariani Bagi, and Elizabeth Ferreira Gouvêa Goldbarg. "Algoritmo transgenético aplicado ao problema do caixeiro comprador capacitado simétrico." Pesquisa Operacional 28.1 (2008): 93-121.

Golden B. L., Levy L, Dahl R. Two generalizations of the traveling Salesman Problem. OMEGA 1981;9:439–45.

Lima, L.R.. "Operadores de Recombinação Baseados em Permutação para Representações de Grafos." Dissertação de Mestrado. UFGO, 2017.

Ochi, Luiz S., Lucia Drummond, and Rosa Figueiredo. "Design and implementation of a parallel genetic algorithm for the travelling purchaser problem." Proceedings of the 1997 ACM symposium on Applied computing. ACM, 1997.

Ong H. L. Approximate algorithms for the traveling purchaser problem. Operations Research Letters 1982;1(5): 201–5.

Ramesh, t. " Traveling purchaser problem." Opsearch 18.1-3 (1981): 78-91.

Reeves, Colin. "Genetic algorithms." Handbook of metaheuristics. Springer, Boston, MA, 2003. 55-82.

Riera-Ledesma, Jorge, and Juan José Salazar-González. "The biobjective travelling purchaser problem." European Journal of Operational Research 160.3 (2005): 599-613.

Riera-Ledesma, Jorge, and Juan José Salazar-González. "A heuristic approach for the travelling purchaser problem." European Journal of Operational Research 162.1 (2005): 142-152.

Teeninga, A., and A. Volgenant. "Improved heuristics for the traveling purchaser problem." Computers & Operations Research 31.1 (2004): 139-150.

Voß, Stefan. "Dynamic tabu search strategies for the traveling purchaser problem." Annals of Operations Research 63.2 (1996): 253-275.

Wen, Jianfeng, et al. "Systematic literature review of machine learning based software development effort estimation models." Information and Software Technology 54.1 (2012): 41-59.