HEURÍSTICA HÍBRIDA APLICADA AO PROBLEMA DO CAIXEIRO VIAJANTE COM SELEÇÃO DE HOTÉIS

Augusto Pizano Vieira Beltrão
Instituto de Computacão - Universidade Federal Fluminense (UFF)
Niterói, RJ, Brasil
augusto_pvb@hotmail.com

Luiz Satoru Ochi
Instituto de Computacão - Universidade Federal Fluminense (UFF)
Avenida General Milton Tavares de Souza, s/n - Boa Viagem - CEP 24210-330, Niteroi (RJ), Brasil
satoru@ic.uff.br

José André de Moura Brito
Escola Nacional de Ciências Estatísticas – ENCE/IBGE
Rua André Cavalcanti, 106, sala 503-A
jambrito@gmail.com

RESUMO

O Problema do Caixeiro Viajante com Seleção de Hotéis (TSPHS) é uma variante do conhecido Problema do Caixeiro Viajante. A resolução do TSPHS corresponde a determinar uma rota na qual um vendedor visita todos os clientes e retorna para localização inicial. Ao final de cada jornada de trabalho (dada por um limite de tempo), o vendedor deve visitar algum hotel para descansar. O objetivo é minimizar o número de viagens necessárias para completar a rota. Neste trabalho, é proposto um algoritmo heurístico, inspirado no BRKGA, que foi combinado com métodos construtivos e procedimentos de busca local. Experimentos conduzidos mostram que o algoritmo proposto é capaz de produzir resultados competitivos frente aos da literatura. Além disso, foram encontradas soluções melhores que as da literatura em 7 das 131 instâncias da literatura.
Palavra-chave: Problema do Caixeiro Viajante; BRKGA; Metaheurísticas.

ABSTRACT

The Traveling Salesman Problem with Hotel Selection (TSPHS) is a variation of the well-known Traveling Salesman Problem. The resolution of TSPHS corresponds to a route in which the salesman visits all clients and returns to the initial location. At the end of each working day (given by a time limit), the salesman must visit some hotel to rest. The goal is to minimize the number of trips required to complete the route. This work proposes a heuristic algorithm, inspired by BRKGA, which was combined with constructive methods and local search procedures. Experiments conducted show that the proposed algorithm is capable of producing competitive results compared to the literature. In addition, better solutions than those found in the literature were found in 7 of the 131 instances of the literature.
Keywords: Traveling Salesman Problem; BRKGA; Metaheuristics.

Como Citar:
BELTRÃO, Augusto Pizano Vieira; DE MOURA BRITO, José André; OCHI, Luiz Satoru. Heurística Híbrida Aplicada ao Problema do Caixeiro Viajante com Seleção de Hotéis. In: SIMPÓSIO DE PESQUISA OPERACIONAL E LOGÍSTICA DA MARINHA, 19., 2019, Rio de Janeiro, RJ. Anais […]. Rio de Janeiro: Centro de Análises de Sistemas Navais, 2019.

1. INTRODUÇÃO

Ao longo das últimas décadas, o Problema do Caixeiro Viajante (do termo em inglês, *The Traveling Salesman Problem* – TSP) tem sido um dos problemas de otimização combinatória mais estudados, com diversas variantes conhecidas na literatura [2]. O Problema do Caixeiro Viajante com Seleção de Hotéis (do termo em inglês, *The Traveling Salesperson Problem with Hotel Selection* - TSPHS), proposto inicialmente por [26], corresponde a uma variante do TSP na qual a viagem feita pelo caixeiro possui um limite de tempo/distância percorrida, e ao final do dia o caixeiro deve repousar em algum hotel. Por ter sido proposto recentemente, o TSPHS ainda não foi tão explorado quanto outras variantes do TSP. Dito isso, o objetivo neste artigo é propor uma nova abordagem para este problema**,** ainda não utilizada na literatura**,** e avaliar a qualidade das soluções produzidas.

O diferencial do TSPHS é que, neste problema, uma rota (solução para o TSP) é dividida em viagens por meio de hotéis intermediários. A rota é composta por uma sequência de viagens (*trips*) ligadas por hotéis, enquanto uma viagem é composta por uma sequência de clientes. A visita a um hotel representa o fim de uma jornada de trabalho, ou seja, o caixeiro deve descansar em algum hotel ao terminar o dia de trabalho. O hotel visitado ao final de uma viagem i deve ser o mesmo hotel que inicia a viagem i+1. A rota começa e termina no ponto de partida, dado no problema, que pode ser utilizado como hotel intermediário.

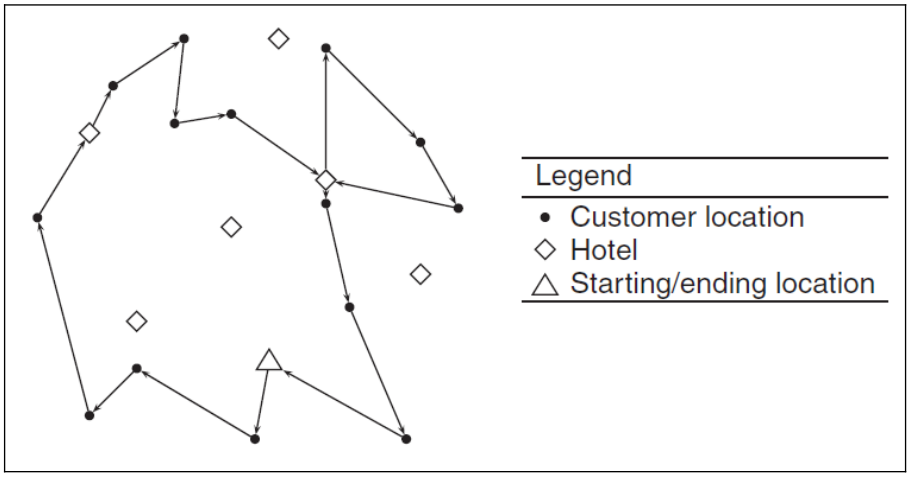

Figura 1 – Um exemplo de solução para o TSPHS. Os círculos representam clientes, os quadrados hotéis e o triângulo o ponto de partida e chegada. Imagem retirada de [26].

Seja $H = \{0, 1, 2, ..., s\}$ um conjunto de s localizações de hotéis e $C = \{s+1, ..., s+n\}$ um conjunto à localização dos n clientes. Cada cliente está associado a um tempo de atendimento ou visita t_i. O Problema do Caixeiro Viajante com Seleção de Hotéis (TSPHS) pode ser modelado através de um grafo completo $G = (V, E)$, sendo $V = C \cup H$ o conjunto de vértices. No conjunto $A = \{(i,j) \mid i,j \in V, i \neq j\}$, cada aresta (i, j) está associada a um peso c_{ij}, conhecido no problema, que corresponde ao tempo necessário de deslocamento da localidade i para j (cada localidade pode ser um cliente ou um hotel). Sendo assim, o tempo

gasto com o deslocamento e atendimento do cliente j, vindo do cliente (ou hotel) i, é igual à $c_{ij}+t_j$. O tempo de visita t_i para qualquer hotel é zero.

A solução para este problema é dada por uma rota que visita todos os clientes, sendo tal rota composta por um conjunto de viagens conectadas por hotéis intermediários. Uma viagem T_i corresponde a um caminho no grafo G, que passa por clientes (localizações de C), começa e termina em hotéis intermediários (localizações de H); sendo que a viagem pode começar e terminar no mesmo hotel ou em hotéis diferentes. A viagem T_{i+1} começa no mesmo hotel que a viagem T_i terminou. A rota começa e termina no ponto de partida, dado no problema, que pode ser utilizado como hotel intermediário. Ou seja, o ponto de partida faz parte do conjunto de hotéis.

O tempo limite de uma viagem é dado no problema por t_L, ou seja, intuitivamente interpretado como um dia de trabalho, de modo que, ao final de cada dia em sua jornada de trabalho, o caixeiro visite um hotel para descansar. Se o tempo de qualquer viagem exceder t_L a solução se torna inviável. Neste problema, o objetivo principal é minimizar o número de viagens em uma rota (número necessário de dias de trabalho) e o objetivo secundário é minimizar o total de tempo gasto da rota (dado pela soma dos tempos de cada viagem).

Conforme já comentado, o TSPHS que corresponde a uma variante do Problema do Caixeiro Viajante, sendo também classificado como um problema NP-difícil [4]. Face à complexidade, propõe-se neste artigo um algoritmo baseado em uma heurística híbrida para resolver o TSPHS. A heurística foi aplicada a 131 instâncias da literatura, de diferentes tamanhos (10 a 1002 clientes) e com diferentes quantidades de hotéis (2 a 11 hotéis). O algoritmo proposto é capaz de produzir soluções competitivas frente às soluções da literatura e, em alguns casos, produzir soluções de qualidade superior àquelas produzidas pelos melhores algoritmos da literatura.

O restante do artigo está organizado da seguinte forma: A segunda seção traz a revisão da literatura, citando os mais recentes trabalhos. Na seção 3, faz-se a descrição do algoritmo proposto. Os resultados dos testes computacionais e a descrição das instâncias utilizadas estão na seção 4. Finalmente, na última seção, são apresentadas as conclusões, análises e sugestões para trabalhos futuros.

2. REVISÃO DA LITERATURA

Por ser um problema amplamente explorado, o TSP possui diversas variantes semelhantes ao TSPHS. Algumas dessas variantes estão associadas a problemas de roteamento, que lidam com o planejamento e construção de múltiplas rotas, como por exemplo, o Problema do Caixeiro Viajante Múltiplo (mTSP) [3], o Problema de Roteamento de Veículos (VRP) [25] e o Problema de Roteamento de Veículos com Múltiplos Depósitos (MDVRP) [6].

Assim como no mTSP, o objetivo no VRP é usualmente minimizar o custo total da viagem, mas, neste caso, existe uma restrição que impede que um único veículo seja capaz de visitar e atender todos os clientes. A restrição pode ser um limite para a distância que um veículo pode percorrer em uma rota, ou um limite quanto à quantidade de demanda de clientes a ser atendida pelo veículo (capacidade do veículo).

O MDVRP é similar ao VRP, considerando uma pequena modificação: existem diversos depósitos (pontos de partida); cada um com sua própria frota de veículos. Existem, na literatura, diversos algoritmos utilizados para resolver os problemas citados e que podem ser adaptados para o problema aqui abordado.

O TSPHS foi inicialmente abordado em 2011 por [26], sob a motivação de que um caixeiro pode não ser capaz de realizar todas as visitas e atendimentos em uma única jornada de trabalho. O autor fez uma revisão da literatura, apresentando as diferenças entre o TSPHS

e outras variantes do TSP, pois diversas aplicações do VRP possuem características em comum com o TSPHS. Um modelo matemático é apresentado, assim como diversas instâncias para o TSPHS, adaptadas de outros problemas da literatura. Para resolver o problema é proposta uma heurística que faz uso de dois procedimentos construtivos, e aplica procedimentos de busca local para aprimorar as soluções produzidas. As soluções produzidas pela heurística são comparadas às produzidas a partir da aplicação da formulação matemática para o primeiro e segundo grupo de instâncias (SET1 e SET2). No caso do terceiro grupo de instâncias (SET3), as soluções ótimas já são conhecidas. As soluções obtidas pela heurística para o quarto grupo de instâncias (SET4) não foram comparadas, pois as soluções ótimas para estas instâncias são desconhecidas e a formulação não foi utilizada.

Em [5] apresenta-se um Algoritmo Memético (*Memetic Algorithm* - MA) que combina operadores genéticos (do Algoritmo Genético) [14] com procedimentos de Busca Tabu [10,11]. Os testes realizados mostraram que o algoritmo é capaz de encontrar soluções de alta qualidade. Em 2015, estes mesmos autores propuseram um novo algoritmo, que constrói a solução inicial por meio da heurística de Lin-Kernighan [16] (gera uma sequência de clientes) e particiona o tour (insere os hotéis) em viagens viáveis aplicando um procedimento inspirado em [20]. A solução é então aprimorada por meio de um procedimento *Variable Neighborhood Descent* (VND) [13,18]. A vantagem deste novo algoritmo é que ele é capaz de produzir resultados competitivos em pouco tempo em comparação aos tempos de processamento do MA [4].

Em [21] são apresentados dois novos Algoritmos Meméticos: um deles é combinado a uma Busca Tabu e o outro ao *Random Variable Neighborhood Descent* (RVND). Em [22] foi proposto um ILS denominado *Client Perturbation Variable Neigborhood Search* (CP-VNS), que também produz soluções competitivas consumindo pouco tempo de processamento, em comparação aos apresentados na literatura.

Lu et al. (2018) [17] propõe um algoritmo híbrido, combinando Programação Dinâmica e busca local (para soluções viáveis e não viáveis) a um Algoritmo Memético que possui três tipos de cruzamento. Os resultados encontrados são de alta qualidade e o tempo de processamento é semelhante ao do MA [4].

Sousa et al. (2019) [23] apresentam um algoritmo ILS competitivo, capaz de encontrar soluções de alta qualidade para quase todas as instâncias e, em alguns casos, melhorar a qualidade das melhores soluções encontradas na literatura.

3. HEURÍSTICA PROPOSTA

O algoritmo proposto neste trabalho corresponde a uma heurística híbrida, inspirada na meta-heurística Algoritmo Genético de Chaves Aleatórias Viciadas (do termo em inglês, *Biased Random-Key Genetic Algorithm* - BRKGA) [12]. Diversos procedimentos foram incorporados neste algoritmo, como, por exemplo, métodos construtivos, de busca local e um algoritmo para imputação de hotéis.

3.1. ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS VICIADAS - BRKGA

O BRKGA é uma metaheurística evolutiva que pode ser aplicada a diversos problemas de otimização representando suas soluções através de um vetor de chaves aleatórias. Na versão original da metaheurística, cada chave aleatória é um número real, gerado aleatoriamente, no intervalo contínuo [0,1). Um vetor de chaves aleatórias pode então ser transformado em uma solução para o problema de otimização abordado por meio de um "decodificador" para o problema. Um decodificador é uma função que recebe um vetor de chaves aleatórias, e converte este vetor em uma solução viável para o problema de otimização. Cada solução decodificada é avaliada a partir do cálculo de uma função objetivo.

O funcionamento do BRKGA se dá por meio da aplicação de três operadores

genéticos (Seleção, Cruzamento e Mutação) na população. Uma população de indivíduos (cromossomos) pode ser representada por uma matriz, onde as linhas correspondem aos indivíduos e as colunas correspondem às características de cada indivíduo. A população de p indivíduos e n características é representada por uma matriz de dimensões p x n.

Os três operadores genéticos são aplicados na população para gerar uma nova população. O funcionamento dos operadores genéticos é descrito abaixo.

Seleção: ordena-se os cromossomos do melhor para o pior de acordo com o valor obtido da função objetivo. Em seguida, um percentual dos melhores cromossomos é copiado para a população da próxima geração. Os melhores cromossomos formam o Conjunto Elite desta geração (de tamanho p_e), enquanto o resto forma o Conjunto Não Elite (de tamanho p - p_e).

Mutação: gera uma determinada quantidade p_m de cromossomos mutantes, que são novos vetores de chaves aleatórias. Estes cromossomos formam a população da próxima geração.

Cruzamento: dois cromossomos da população são selecionados de maneira aleatória e combinados para gerar um novo cromossomo filho. Mais especificamente, um cromossomo do Conjunto Elite e outro do Conjunto Não Elite. Este processo ocorre com reposição, sendo assim, um cromossomo pode ter mais de um filho por geração. Os cruzamentos são realizados sucessivamente até que determinado número (p - p_e - p_m) de cromossomos filho seja gerado. O cromossomo filho deve possuir mais características do cromossomo do conjunto elite, sendo assim, existe uma probabilidade p_e (maior que 0,5) associada ao cruzamento que indica a probabilidade de determinada chave aleatória do cromossomo filho pertencer ao cromossomo elite ou não elite.

Depois de aplicar os três operadores genéticos à primeira população obtém-se, na segunda geração, uma nova população formada pelos cromossomos: conjunto elite, filhos e mutantes da geração anterior. Os operadores genéticos são aplicados a cada geração até que uma condição de parada seja satisfeita.

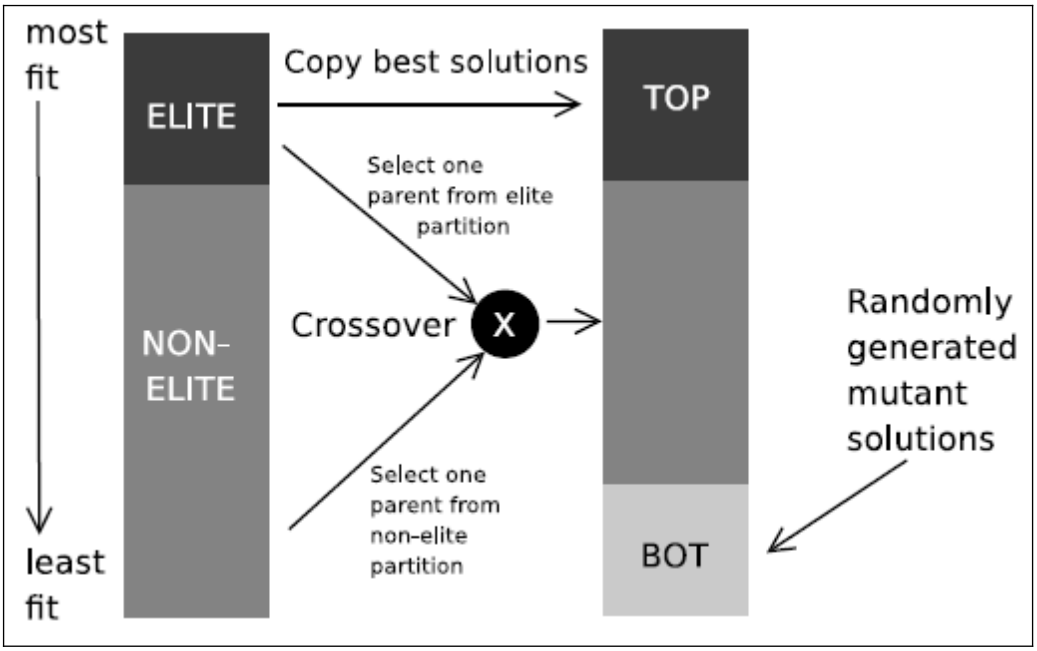

Figura 2 – Transição da geração k para geração k+1 em um BRKGA. Imagem retirada de [12].

3.2. HEURÍSTICA PROPOSTA: BRKGA-RVND

Até o momento, de acordo com a pesquisa bibliográfica realizada durante o desenvolvimento deste artigo, não existem trabalhos na literatura que utilizem o BRKGA para resolver o TSPHS.

O BRKGA adaptado para este problema recebe, em sua estrutura de entrada, a instância (contendo as localizações dos clientes e hotéis, número *n* de clientes, número *s* de hotéis e tempo limite de viagem t_L), tamanho *p* da população *P*, tamanho p_e do conjunto elite P_E, tamanho p_m do conjunto de mutantes P_M, probabilidade ρ_e associada ao cruzamento, tamanho K_{max} da Lista de Candidatos (associado aos métodos construtivos), e o limite *TP* para o tempo de processamento (critério de parada).

Neste trabalho, não foram utilizados vetores de chaves aleatórias, ao invés disso, uma solução é representada por um vetor solução X de tamanho *n*, que indica a sequência de clientes (tour) a serem visitados no problema. Cada entrada *i* do vetor solução é um número inteiro *j*, no intervalo [1,*n*], indicando que o i-ésimo cliente a ser visitado é o cliente *j*. O vetor solução não é uma solução para o problema, pois não indica quais hotéis foram visitados.

Um vetor solução (indivíduo) preserva apenas as informações relativas à ordem de visita dos clientes. Os hotéis a serem visitados são adicionados à solução apenas no momento do cálculo do valor da função objetivo. Assim, um indivíduo pode ser representado na população por uma sequência de clientes (vetor solução), sem levar os hotéis em consideração durante o cruzamento e a busca local.

O vetor solução pode ser transformado em uma solução para o TSPHS por meio de um procedimento que particiona a rota (sequencia de clientes dada pelo vetor solução) em viagens (que respeitam a restrição de tempo) separadas por hotéis. Este procedimento é inspirado no algoritmo de *Dijkstra* [1] para caminho de menor custo. O procedimento particiona a rota em viagens viáveis de modo ótimo e foi detalhadamente descrito em [4].

Neste procedimento, um grafo auxiliar é construído utilizando os clientes e hotéis. As arestas do grafo representam viagens viáveis enquanto as arestas que representam viagens não viáveis recebem um peso arbitrariamente grande, deste modo, estas arestas são evitadas. Os nós são os pontos de chegada e partida de cada viagem. A primeira viagem começa no ponto de partida *H0*. Ao fim do procedimento, o algoritmo encontrará o caminho que minimiza a distância percorrida ao longo do trajeto, sem violar a restrição de tempo das viagens [4].

No BRKGA desenvolvido para o TSPHS a avaliação de cada solução (função objetivo) está associada ao número de viagens (objetivo principal) e ao custo de percorrer todo o percurso (objetivo secundário). Caso a solução não seja viável, é adicionado um termo de penalidade à função objetivo. Deste modo, durante a ordenação da população de acordo com a função objetivo, as soluções que apresentarem menor número necessário de viagens são consideradas as melhores e as soluções não viáveis são consideradas as piores. O pseudocódigo do algoritmo proposto encontra-se no Apêndice.

3.2.1. Heurísticas Construtivas

A população inicial é constituída por diversas soluções geradas por meio da aplicação de três heurísticas construtivas, descritas a seguir:

HC1 – baseada na fase construtiva do GRASP [9]. Constrói uma solução adicionando (linha 8), a cada iteração, o próximo cliente a ser visitado (denominado v), selecionado aleatoriamente de uma lista (*Restricted Candidate List* - RCL) de potenciais clientes. A variável v é atualizada para ser o último cliente adicionado à solução a cada iteração (linha 7). A RCL (linha 6) é composta pelos *K* clientes mais próximos de v (pertencentes a *C*) que ainda não foram visitados. O tamanho da RCL é um número aleatório

entre 2 e K_{max}, escolhido a cada nova solução gerada (linha 2). O ponto de partida para a construção de uma nova solução é algum cliente escolhido aleatoriamente de *C* (linha 4).

```
Procedimento 1: HC1(K_max)
1.  Início
2.    K_rng = randInt(2, K_max);
3.    v = selecionarElementoAleatoriamente(C);
4.    X = {v};
5.    Enquanto (algum cliente não foi visitado) Faça
6.      criarRCL(v, RCL, K_rng);
7.      v = selecionarElementoAleatoriamente(RCL);
8.      X = X ∪ {v};
9.    Fim-Enquanto
10. Fim-Início
11. Retorna X;
```

HC2 – semelhante à HC1, no entanto, a RCL é composta pelos *K* clientes mais distantes de v que ainda não foram visitados.

HC3 – Constrói uma solução adicionando, a cada iteração, o próximo cliente (denominado v) a ser visitado aleatoriamente da RCL. A variável v é atualizada para ser o último cliente adicionado à solução a cada iteração. Neste caso, a RCL é composta pelos *K* clientes mais próximos de v, podendo incluir clientes que já foram visitados. O ponto de partida para a construção de uma nova solução é algum cliente escolhido aleatoriamente de *C*.

Após n iterações, aplica-se um procedimento de correção, trocando os clientes que foram visitados mais de uma vez por clientes que não foram visitados. As trocas são feitas visando à minimização do custo em distância, testando a cada iteração todas as possibilidades de troca de clientes não visitados para cada posição, escolhendo a troca de menor custo. A posição escolhida i, para inserir o cliente j não visitado, no vetor solução a cada iteração é aquela que minimiza o custo em distância da troca, dado por: $c_{(i-1)j} + c_{(i+1)j}$.

A seguir, uma figura ilustrativa apresenta o funcionamento da HC3 passo a passo (com $n = 7$ clientes).

Passo	Construção do Vetor Solução						
1	**2**	-	-	-	-	-	-
2	2	**5**	-	-	-	-	-
3	2	5	**2**	-	-	-	-
4	2	5	2	**3**	-	-	-
5	2	5	2	3	**6**	-	-
6	2	5	2	3	6	**5**	-
7	2	5	2	3	6	5	**6**
8	2	5	**1**	3	6	5	6
9	2	5	1	3	6	5	**7**
10	2	5	1	3	6	**4**	7

Figura 3 – Construção de um vetor solução utilizando a HC3.

O cliente 2 é selecionado como primeiro cliente a ser visitado. Então, os clientes (pertencentes a *C*) são iterativamente adicionados até o passo 7. Depois disso, verifica-se que os clientes 1, 7 e 4 não foram visitados, enquanto os clientes 2, 5 e 6 foram visitados mais de uma vez. A segunda etapa da HC3 se baseia em substituir os clientes 2, 5 e 6 pelos clientes 1, 7 e 4. No passo 8, a terceira posição do vetor solução é escolhida aleatoriamente. Dentre os clientes não visitados (1, 7 e 4), o cliente 1 na posição 3 apresenta menor custo de troca.

3.2.2. Operador Genético: Seleção

As soluções de maior qualidade são copiadas, sem alteração, para a próxima geração e formam o Conjunto Elite (Algoritmo 1, linhas 6-8).

3.2.3. Operador Genético: Mutação

A cada geração são geradas $|P_M|$ soluções mutantes através das três heurísticas construtivas HC1, HC2 e HC3. As heurísticas HC2 e HC3 são executados uma única vez por geração, enquanto HC1 é utilizado para gerar o resto das soluções mutantes.

3.2.4. Operador Genético: Cruzamento

O operador genético de Cruzamento é aplicado para gerar soluções filho geradas a partir da combinação de dois vetores solução, considerando indivíduos do Conjunto Elite ($|P_E| = p_e$) e do Conjunto Não-Elite ($|P_{NE}| = p - p_e$), selecionadas aleatoriamente (e com reposição).

Foram utilizados dois procedimentos de cruzamento, metade dos filhos é gerado por um cruzamento e metade por outro. Os dois procedimentos de cruzamento aqui utilizados são iterativos. Ambos começam com um vetor sem elementos e a cada iteração adiciona-se algum cliente a este vetor. O próximo cliente a ser visitado é vizinho do último cliente visitado. Neste caso, considera-se vizinho do cliente *v*, aquele cliente que está à direita ou esquerda de v no vetor solução. Ou seja, os clientes que foram visitados imediatamente antes ou depois de v na rota. Para isso, são utilizadas duas funções, descritas a seguir.

selecionarProximoElemento – recebe como parâmetro de entrada o cliente v e o vetor solução X. Retorna o vizinho à direita de *v*, segundo a sua posição em X. Caso o vizinho já tenha sido visitado, retorna o cliente não visitado de menor distância de *v*.

vetor solução (X)	1	5	3	7	6	4	2
selecionarProximoElemento	5	3	7	6	4	2	1

Figura 4 – Exemplo de aplicação da função selecionarProximoElemento. Neste caso, o vizinho à direita do cliente 5 é o cliente 3, ou seja, selecionarProximoElemento(5, X) retorna 3.

selecionarProximoElemento2 – recebe como parâmetro de entrada o cliente v e o vetor solução X. Retorna o vizinho à direita ou à esquerda de v (segundo a sua posição em X), que apresenta menor distância de v. Caso ambos os vizinhos já tenham sido visitados, retorna o cliente não visitado de menor distância de *v*. No exemplo dado na figura 1, os clientes à direita e à esquerda de 5, são respectivamente, 3 e 1.

O primeiro procedimento de cruzamento (denominado cruzamento1) constrói uma solução filho (sequência de clientes) adicionando, a cada iteração, o próximo cliente a ser visitado, sendo este, vizinho do último cliente selecionado. O cliente selecionado na iteração i+1 é o vizinho não visitado do cliente da iteração i, segundo a solução Elite com probabilidade ρ_e (linha 6), ou segundo a solução Não-Elite com probabilidade 1- ρ_e (linha 8). Caso o vizinho já tenha sido visitado, o cliente mais próximo não visitado é selecionado para iteração i+1. O ponto de partida para a construção da solução filho é algum cliente escolhido aleatoriamente de *C*.

```
Procedimento 2: cruzamento1(X1, X2, ρe);
1.  Início
2.    v = selecionarElementoAleatoriamente(X1);
3.    Xfilho = {v};
4.    Enquanto (algum cliente não foi visitado) Faça
5.      rng = rand(0,1);
6.      Se (rng < ρe) Então
7.        v = selecionarProximoElemento(v, X1);
8.      Senão
9.        v = selecionarProximoElemento2(v, X2);
10.     Xfilho = X ∪ {v};
11.   Fim-Enquanto
12. Fim-Início
13. Retorna Xfilho;
```

O segundo algoritmo de cruzamento é semelhante ao primeiro, no entanto, as funções selecionarProximoElemento e selecionarProximoElemento2 são alteradas para que possam retornar o vizinho selecionado ainda que ele já tenha sido visitado. A consequência desta modificação é que, ao final das *n* iterações, o vetor solução contém alguns clientes visitados múltiplas vezes e alguns clientes que não foram visitados. Para resolver este problema, aplica-se um procedimento iterativo de correção, trocando clientes que foram visitados mais de uma vez por clientes que não foram visitados. As trocas são feitas visando à minimização do custo, testando a cada iteração todas as possibilidades de trocas de clientes e escolhendo a troca de menor custo.

3.2.5. Refinamento da Solução

Após o cruzamento, um procedimento de busca local é aplicado nas soluções geradas, antes da imputação dos hotéis e do cálculo da função objetivo, sem levar em consideração o limite de tempo t_L.

O procedimento utilizado é baseado no RVND (*Random Variable Neighborhood Descent*), no qual diversas estruturas de vizinhança são exploradas a partir de uma solução. A ordem de aplicação das vizinhanças é aleatória.

```
Procedimento 3: RVND(X)
1.  Início
2.  Defina aleatoriamente a ordem de aplicação de vizinhanças
3.    k = 1;
4.    Enquanto (k <= 2) faça
5.      X' = buscaLocal(k, X);
6.      Se (f(X') < f(X)) então
7.        X ← X';
8.        k ← 1;
9.      Senão
10.       k = k + 1;
11.   Fim(enquanto)
12. Fim(início)
13. Retorna X;
```

A estratégia utilizada na busca local (linha 5, Procedimento 3) é de analisar todos os movimentos possíveis em determinada vizinhança de uma solução, e aplicar aquele que melhor contribui para a redução do custo da distância (estratégia *best improvement*). Com o objetivo de reduzir o tempo de processamento, este procedimento é aplicado em um espaço de busca reduzido. Utiliza-se uma estrutura de dados auxiliar contendo, para cada cliente i, uma lista indicando até 20 clientes mais próximos de i. A busca é realizada levando em consideração, apenas os movimentos que colocariam o cliente movimentado ao lado de um dos seus 20 vizinhos mais próximos.

As estruturas de vizinhança utilizadas são descritas a seguir.

2-opt [8] - duas arestas são removidas e reconectadas de forma diferente, de modo que a ordem de visita dos clientes internos as arestas é invertida. Esta estrutura de vizinhança é útil por remover cruzamentos da rota.

X (Original)	1	2	3	4	5	6	7
X (2-opt)	1	**6**	5	4	3	**2**	7

Figura 5 – Exemplo de aplicação do movimento 2-opt. Neste caso, os clientes escolhidos para o movimento são o 2 e o 6. Para que esta alteração seja levada em consideração, é necessário que o cliente 1 esteja na lista dos 20 clientes mais próximos do cliente 6, ou que o cliente 7 esteja na lista dos 20 clientes mais próximos do cliente 2.

Or-opt [19] - Realoca uma quantidade c de clientes consecutivos, sendo c um número que varia de 1 a 3.

X (Original)	1	2	3	4	5	6	7
X (Or-opt)	1	**4**	5	**2**	3	6	7

Figura 6 – Exemplo de aplicação do movimento Or-opt, para c = 2. Neste caso, os clientes escolhidos para o movimento são o 2 e o 4. Para que esta alteração seja levada em consideração, é necessário que o cliente 1 esteja na lista dos 20 clientes mais próximos do cliente 4, ou que que o cliente 2 esteja na lista dos 20 clientes mais próximos do cliente 5.

3-opt [15] - três arestas são removidas, então são reconectadas de todas as formas possíveis e a melhor solução gerada dentre todas é selecionada. Neste caso, existem 8 formas possíveis de alocação das arestas, no entanto, apenas 4 são levadas em consideração além da alocação original. As 3 opções de alocação restantes são semelhantes a movimentos 2-opt e portanto, desconsideradas da avaliação. A busca local 3-opt é aplicada em apenas uma solução filho selecionada aleatoriamente por geração.

No momento do cálculo da função objetivo, a rota é dividida em viagens separadas por hotéis por meio do algoritmo de *Dijkstra*. Ao final do procedimento, o valor da função objetivo associado a cada solução é calculado. A sequência de clientes encontrada para cada solução é adicionada à população.

4. EXPERIMENTOS COMPUTACIONAIS

O algoritmo proposto foi implementado em linguagem C e compilado utilizando o MinGW, todos os experimentos foram realizados em um computador dotado de um processador i5-6400 (2.70GHz 16GB RAM) no sistema operacional Windows 7.

Os parâmetros utilizados no algoritmo são apresentados a seguir: p = 100, p_e = 20, p_m = 20, ρ_e = 0.8, K_{max} = 5. O limite *TP* para o tempo de processamento para cada instância foi semelhante ao do MA [5]. Este critério de parada é o mesmo que foi utilizado para o algoritmo HDP [17].

4.1. INSTÂNCIAS

Para testar o algoritmo proposto neste artigo, foram utilizadas 131 instâncias da literatura (de *benchmark*). Criadas por [26], tais instâncias foram utilizadas em diversos outros algoritmos propostos para este problema. As instâncias estão divididas em 4 grupos que são brevemente descritos a seguir. Todas as instâncias podem ser obtidas em http://antor.uantwerpen.be/instances-in-the-paper-a-memetic-algorithm-for-the-travelling-salesperson-problem-with-hotel-selection/.

SET1 - é constituído por 6 instâncias conhecidas do VRPTW (*Vehicle Routing Problem with Time Windows*) [24] e 10 MDVRPTW (*Multi-Depot Vehicle Routing Problem with Time Windows*) [7]. As instâncias do VRPTW (c101, c201, r101, r201, rc101 e rc201) têm 100 clientes cada, enquanto as do MDVRPTW (pr1–pr10) têm entre 48 e 288 clientes. A janela de tempo dos clientes é descartada, mas o tempo de fechamento dos hotéis é utilizado para o tempo limite t_L.

SET2 - é criado a partir das instâncias do primeiro grupo, incluindo apenas os K primeiros clientes (K = 10, 15, 30, 40), adicionando um hotel extra no primeiro cliente. Instâncias triviais não foram incluídas (quando é possível resolver o problema sem visitar hotéis). As instâncias c201, r201 e rc201 não foram utilizadas por apresentarem soluções que necessitam de apenas uma viagem. Este grupo contém 13 x 4 = 52 instâncias.

SET3 - contém instâncias derivadas do TSP, conhecidas da literatura, contendo entre 51 e 1002 clientes e 3, 5, ou 10 hotéis extras (além do ponto de partida). Essas instâncias são criadas de modo que a solução ótima é conhecida, utilizando a solução ótima do TSP para definir a localização dos hotéis e valor de t_L no TSPHS. Este grupo contém 16 x 3 = 48 instâncias.

SET4 - é definido a partir das mesmas instâncias do terceiro grupo, no entanto, 10 hotéis são adicionados nas localizações dos clientes 1, 6, 11, 16, 21, 25, 31, 35, 41 e 45. O valor de t_L, é definido como o custo total da solução ótima do TSP divido por 5. Este grupo contém 15 instâncias.

4.2. RESULTADOS

Os resultados obtidos a partir do algoritmo proposto, denotado por BRKGA-RVND, foram comparados com os resultados dos melhores algoritmos da literatura. Para isto, foram considerados dois trabalhos recentes: HDM (*Hybrid Dynamic Programming Memetic Algorithm*) [17] e EA-ILS (*Efficient Adaptive Iterated Local Search*) [23]. Todos os resultados foram obtidos por meio de uma única execução do algoritmo. A medida de comparação aqui utilizada foi o gap. A expressão que permite calcular o valor do gap para comparação entre o resultado *R1* e *R2* é apresentada a seguir:

$$\text{gap} = 100 \times (custo(R1) - custo(R2)) / custo(R2)$$

Quando o valor do gap é negativo, o custo de tempo total do tour (incluindo tempo de visita aos clientes) da solução *R1* é menor que o de *R2*. No entanto, a solução de menor custo de tempo não necessariamente possui maior qualidade. O objetivo primário deste problema é reduzir o número de viagens, e apenas quando as soluções possuem o mesmo número de viagens, o gap indica qual é a melhor. Para maioria das instâncias, os algoritmos aqui comparados apresentaram o mesmo número de viagens, ou seja, o valor do gap pode ser usado, em geral, como uma ferramenta de comparação.

Foi considerada como a melhor solução conhecida da literatura (do termo em inglês, *Best Known Solution* - BKS), segundo a pesquisa feita, a melhor solução encontrada dentre os seguintes algoritmos: modelo matemático de [26], MA [5], P-LS [4], HDP [17], EA-ILS [23], BRKGA-RVND (algoritmo proposto).

Para cada instância da literatura o algoritmo BRKGA-RVND foi executado uma única vez durante um tempo máximo de processamento semelhante ao de [5], e o resultado foi comparado com a BKS da literatura. As figuras a seguir apresentam o percentual de instâncias em que cada algoritmo obteve o valor da BKS.

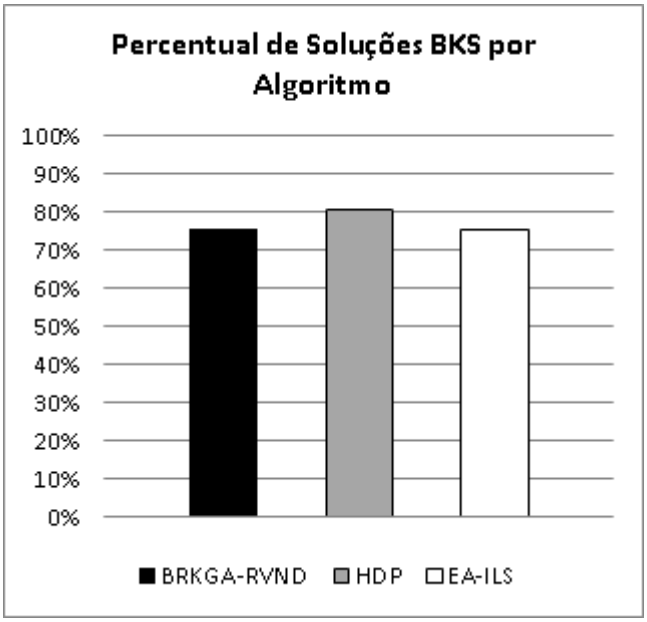

Figura 7 – Resultados comparativos para todas as 131 instâncias.

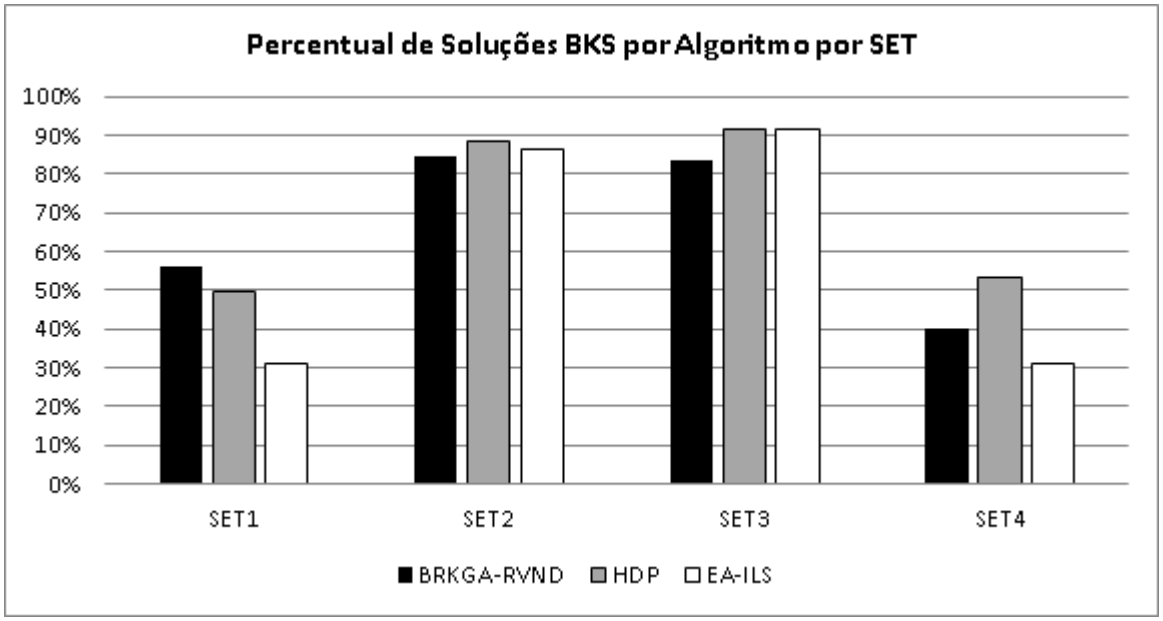

Figura 8 – Resultados comparativos separados por conjunto de instâncias (SET1, SET2, SET3, SET4).

O BRKGA-RVND foi capaz de produzir soluções melhores que a BKS em 6 das 16 instâncias que compõe o SET1. No caso particular da instância pr05 deste conjunto, o BRKGA_RVND encontrou uma solução de menor custo quanto ao tempo de deslocamento, no entanto, o número de viagens é maior tornando a solução de baixa qualidade se comparada à solução do HDP ou EA-ILS. Este foi o grupo de instâncias no qual o BRKGA-RVND obteve melhor desempenho comparativo.

O SET2 é o conjunto composto por instâncias de menor tamanho. Para as instâncias do tipo pr1-10 (desenvolvidas a partir do MDVRPTW [7]) o BRKGA-RVND, assim como o HDP, encontra todas as soluções ótimas independente do número de clientes. A instância rc101 aparenta ser a mais difícil visto que nenhum dos algoritmos foi capaz de encontrar o ótimo para as versões com 30 e 40 clientes. Esta instância também foi responsável pelos piores desempenhos (em gap) dos algoritmos. No caso do HDP e o EA-ILS, o gap para esta instância é de 30% enquanto no BRKGA-RVND o gap é de 45%.

O valor da solução ótima das instâncias do SET3 é conhecido. O HDP e o EA-ILS não produziram a solução ótima para a instância a280 com 5 ou 10 hotéis extras, enquanto o BRKGA-RVND foi capaz de encontrar o ótimo nos dois casos. Em 2015, o CP-VNS [22] também encontrou o ótimo para estas instâncias.

O último grupo de instâncias é o SET4. Na instância eil51, pertencente a este conjunto, o BRKGA-RVND encontrou uma solução melhor que o HDP e o EA-ILS, no entanto, o MA [4] já havia encontrado uma solução semelhante para esta instância. O BRKGA-RVND apresentou uma solução melhor que a BKS para a instância ch150.

Os resultados agrupados por grupo (SET's) de instâncias são apresentados a seguir na tabela 1. A primeira linha apresenta a fração do número de vezes em que cada algoritmo produziu a solução ótima, para o SET2 e SET3, onde as soluções ótimas são conhecidas. A segunda linha apresenta a fração do número de vezes em que cada algoritmo produziu a BKS, para o SET1 e SET4, onde as soluções ótimas são desconhecidas. Na terceira linha, são apresentadas as médias para os valores do gap. Na quarta linha, são apresentadas as médias para os valores do tempo de processamento em segundos.

Tabela 1 – Resumo dos resultados.

	SET1			SET2			SET3			SET4		
	BRKGA-RVND	HDP	EA-ILS	BRKGA-RVND	HDP	EA-ILS	BRKGA-RVND	HDP	EA-ILS	BRKGA-RVND	HDP	EA-ILS
Solução Ótima	-	-	-	44 / 52	46 / 52	45 / 52	40 / 48	44 / 48	44 / 48	-	-	-
BKS	9 / 16	8 / 16	5 / 16	-	-	-	-	-	-	6 / 15	8 / 15	5 / 15
gap médio	0,21%	0,03%	0,37%	1,45%	0,98%	0,98%	0,81%	0,15%	0,12%	3,02%	0,17%	0,84%
Tempo médio (s)	110,2	110,6	1,0	1,0	1,2	0,0	267,9	277,6	4,0	326,7	326,7	33,7

O BRKGA-RVND, em comparação com o HDP e o EA-ILS, apresentou melhores resultados no SET1. O BRKGA-RVND apresenta dificuldades para encontrar o ótimo em duas instâncias (pcb442 e pr1002) do SET 3 não importando o número de hotéis. O quarto grupo de instâncias (SET4) é composto pelas instâncias mais desafiadoras. O BRKGA-RVND obteve seu pior gap médio neste grupo.

Nas tabelas a seguir são apresentados os valores para as instâncias em que o BRKGA-RVND encontrou soluções melhores do que as disponíveis na literatura. A primeira e segunda coluna indicam o nome e o número de clientes da instância. As colunas 3 e 4 indicam o número de viagens e o custo total (tempo ou distância) associados a BKS. Para cada algoritmo, são utilizadas quatro colunas indicando o número de viagens, o custo total, o tempo de processamento (em segundos) e o gap (comparando com a BKS).

Tabela 2 – Novas BKS encontradas pelo BRKGA-RVND no SET1.

Instância	n	BKS		BRKGA - RVND				HDP				EA-ILS			
		V	T. Total	V	T. Total	Tempo (s)	gap (%)	V	T. Total	Tempo (s)	gap (%)	V	T. Total	Tempo (s)	gap (%)
c201	100	3	9559,9	3	**9559,7**	17,0	0,00%	3	9559,9	16,3	0,00%	3	9561,8	0,2	0,02%
r201	100	2	1642,8	2	**1637,7**	12,0	-0,31%	2	1642,8	12,4	0,00%	2	1643,7	0,3	0,05%
pr04	192	5	4215,3	5	**4213,4**	162,0	-0,05%	5	4215,3	166,2	0,00%	5	4217,4	0,5	0,05%
pr08	144	4	3367,7	4	**3362,8**	66,0	-0,15%	4	3367,7	66,9	0,00%	4	3389,3	0,4	0,64%
pr09	216	5	4414,9	5	**4408,4**	234,1	-0,15%	5	4414,9	234,7	0,00%	5	4459,6	1,4	1,01%
pr10	288	7	5932,0	7	**5923,8**	422,3	-0,14%	7	5932	422,8	0,00%	7	5956,2	7,9	0,41%
Média							-0,13%				0,00%				0,36%

Tabela 3 – Novas BKS encontradas pelo BRKGA-RVND no SET4.

Instância	n	BKS		BRKGA - RVND				HDP				EA-ILS			
		V	T. Total	V	T. Total	Tempo (s)	gap (%)	V	T. Total	Tempo (s)	gap (%)	V	T. Total	Tempo (s)	gap (%)
ch150	150	6	6578	6	**6565,0**	56,1	-0,20%	6	6578	56,1	0,00%	6	6619	0,3	0,62%

5. COMENTÁRIOS FINAIS

Neste artigo foi proposta uma nova heurística para resolver o TSPHS. Tal heurística foi capaz de encontrar, em 7 das 131 instâncias, soluções melhores que as encontradas por algoritmos da literatura. A heurística é inspirada no BRKGA com dois tipos de cruzamento, três heurísticas construtivas e procedimentos de busca local. Diferentemente do BRKGA original, não foi feito uso das chaves aleatórias, ao invés disso, a solução foi representada por meio da sequência de clientes a serem visitados e foi utilizado um algoritmo para inserção de hotéis no momento do cálculo da qualidade da solução. A heurística proposta foi comparada a duas heurísticas recentemente propostas da literatura e que apresentam resultados de alta qualidade. A comparação demonstrou que o BRKGA-RVND é competitivo, capaz de produzir soluções de alta qualidade em tempo de processamento semelhante aos Algoritmos Meméticos da literatura (que assim como o BRKGA, trabalham com conjuntos de soluções). O terceiro quartil dos gap's dos três algoritmos aqui comparados é zero (logo, o gap mediano também é zero), significando que um grande percentual das soluções encontradas pelos algoritmos são ótimas ou semelhantes às melhores soluções encontradas na literatura. No entanto, o gap médio (1,24%) das soluções produzidas pelo BRKGA-RVND é maior que o gap médio das soluções do HDP (0,47%) e EA-ILS (0,57%).

Concluindo, o BRKGA-RVND apresenta vantagens com relação a outros algoritmos da literatura, encontrando diversas soluções melhores que as apresentadas na literatura para o SET1, no entanto, apresenta dificuldades para encontrar o ótimo em algumas instâncias. As instâncias pcb442, pr1002 e rc101 (K = 40 clientes) devem ser investigadas para implementação de melhorias na heurística proposta para trabalhos futuros. A implementação de novos procedimentos de cruzamento e heurísticas construtivas pode melhorar os resultados para estas instâncias.

6. AGRADECIMENTOS

Os autores agradecem o apoio dado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

7. REFERÊNCIAS BIBLIOGRÁFICAS

AHUJA, Ravindra K.; MAGNANTI, Thomas L.; ORLIN, James B. Network flows. 1988.

APPLEGATE, David L. et al. The traveling salesman problem: a computational study. Princeton university press, 2006.

BEKTAS, Tolga. The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega, v. 34, n. 3, p. 209-219, 2006.

CASTRO, Marco et al. A fast metaheuristic for the travelling salesperson problem with hotel selection. 4OR, v. 13, n. 1, p. 15-34, 2015.

CASTRO, Marco et al. A memetic algorithm for the travelling salesperson problem with hotel selection. Computers & Operations Research, v. 40, n. 7, p. 1716-1728, 2013.

CORDEAU, Jean-François; GENDREAU, Michel; LAPORTE, Gilbert. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks: An International Journal, v. 30, n. 2, p. 105-119, 1997.

CORDEAU, Jean-François; LAPORTE, Gilbert; MERCIER, Anne. A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational research society, v. 52, n. 8, p. 928-936, 2001.

CROES, Georges A. A method for solving traveling-salesman problems. Operations research, v. 6, n. 6, p. 791-812, 1958.

FEO, Thomas A.; RESENDE, Mauricio GC. Greedy randomized adaptive search procedures. Journal of global optimization, v. 6, n. 2, p. 109-133, 1995.

GLOVER, Fred. Tabu search—part I. ORSA Journal on computing, v. 1, n. 3, p. 190-206, 1989.

GLOVER, Fred. Tabu search—part II. ORSA Journal on computing, v. 2, n. 1, p. 4-32, 1990.

GONÇALVES, José Fernando; RESENDE, Mauricio GC. Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics, v. 17, n. 5, p. 487-525, 2011.

HANSEN, Pierre; MLADENOVIĆ, Nenad; PÉREZ, José A. Moreno. Variable neighbourhood search: methods and applications. Annals of Operations Research, v. 175, n. 1, p. 367-407, 2010.

HOLLAND, John H. Genetic algorithms. Scientific american, v. 267, n. 1, p. 66-73, 1992.

LIN, Shen. Computer solutions of the traveling salesman problem. Bell System Technical Journal, v. 44, n. 10, p. 2245-2269, 1965.

LIN, Shen; KERNIGHAN, Brian W. An effective heuristic algorithm for the traveling-salesman problem. Operations research, v. 21, n. 2, p. 498-516, 1973.

LU, Yongliang; BENLIC, Una; WU, Qinghua. A hybrid dynamic programming and memetic algorithm to the traveling salesman problem with hotel selection. Computers & Operations Research, v. 90, p. 193-207, 2018.

MLADENOVIĆ, Nenad; HANSEN, Pierre. Variable neighborhood search. Computers & operations research, v. 24, n. 11, p. 1097-1100, 1997.

OR, I. Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking. PhD thesis (Department of Industrial Engineering and Management Science, Northwestern University), 1976.

PRINS, Christian. A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research, v. 31, n. 12, p. 1985-2002, 2004.

SOUSA, Marques M.; GONÇALVES, Luciana Brugiolo. Comparação de abordagens heurísticas baseadas em algoritmo memético para o problema do caixeiro viajante com seleção de hotéis. In: Proc. XLVI Simpósio Brasileiro de Pesquisa Operacional. 2014. p. 1543-1554.

SOUSA, Marques M. et al. A variable neighborhood search heuristic for the traveling salesman problem with hotel selection. In: 2015 Latin American Computing Conference (CLEI). IEEE, 2015. p. 1-12.

SOUSA, Marques Moreira; OCHI, Luiz Satoru; DE LIMA MARTINS, Simone. An Efficient Heuristic to the Traveling Salesperson Problem with Hotel Selection. *In*: International Workshop on Hybrid Metaheuristics. Springer, Cham, 2019. p. 31-45.

SOLOMON, Marius M. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research, v. 35, n. 2, p. 254-265, 1987.

TOTH, Paolo; VIGO, Daniele (Ed.). The vehicle routing problem. Society for Industrial and Applied Mathematics, 2002.

VANSTEENWEGEN, Pieter; SOUFFRIAU, Wouter; SÖRENSEN, Kenneth. The travelling salesperson problem with hotel selection. Journal of the Operational Research Society, v. 63, n. 2, p. 207-217, 2012.

APÊNDICE

Pseudocódigo do BRKGA-RVND.

Algoritmo 1: BRKGA-RVND(*instância*, p, p_e, p_m, ρ_e, K_{max}, TP)

```
1.  Início
2.    f* ← +∞;
3.    Inicialização das matrizes de distâncias e custos;
4.    Gerar a população inicial P, com as heurísticas construtivas;
5.    Enquanto (critério de parada não satisfeito) Faça
6.      Avalie o custo f(Xi) de cada solução em P com o alg. de Dijkstra;
7.      Particione P em dois conjuntos: P_E e P_NE;
8.      Inicialize a população da nova geração: P_nova ← P_E;
9.      Gere um conjunto de mutantes P_M, com as heurísticas construtivas;
10.      Adicione P_M à população da próxima geração: P_nova ← P_nova ∪ P_M;
11.     Para i ← 1 Até p - p_e - p_m Faça
12.       Escolha a solução X1 aleatoriamente de P_E;
13.       Escolha a solução X2 aleatoriamente de P_NE;
14.       Se (i MOD 2 = 0) Então
15.         X_filho = cruzamento1(X1, X2, ρ_e);
16        Senão
17.         X_filho = cruzamento2(X1, X2, ρ_e);
18.       Aplique o procedimento de busca local ao filho gerado: RVND(X_filho);
19.       Adicione X_filho à população da próxima geração: P_nova ← P_nova ∪ {X_filho};
20.     Fim-Para
21.     Aplique o procedimento 3-opt em um dos filhos gerados: LS3opt(X_filho);
22.     Atualize a população: P ← P_nova;
23.     Ache a melhor solução X_nova em P;
24.     Se (f(X_nova) < f*) Então
25.       X* ← X_nova;
26.       f* ← f(X_nova);
27.     Fim-Se
28.   Fim-Enquanto
29. Fim-Início
30. Retorna X*;
```