Enxame e Computação Evolutiva xxx (2018) 1-11

Listas de conteúdos disponíveis em ScienceDirect

Enxame e computação evolucionária

Página inicial do jornal: www.elsevier.com/locate/swevo

Uma sequência de troca baseada no algoritmo Arti fi cial Bee Colony para o Problema do Caixeiro Viajante

Indadul Khan uma , * , Manas Kumar Maiti b

uma *Departamento de Ciência da Computação, Chandrakona Vidyasagar Mahavidyalaya, Paschim-Medinipur, West Bengal, 721201, Índia*
b *Departamento de Matemática, Mahishadal Raj College, Mahishadal, Purba-Medinipur, West Bengal, 721628, Índia*

ARTICLEINFO

Palavras-chave:
Problema de vendedor ambulante
Sequência artificial de troca de algoritmo de colônia de abelhas
Operação de troca
K-opt

ABSTRATO

Neste artigo de pesquisa, o algoritmo Arti fi cial Bee Colony é modificado com várias regras de atualização e operação K-opt para resolver o Problema do Caixeiro Viajante. Aqui, os recursos de sequências de troca e operações de troca na sequência de cidades (solução / caminho) do problema são usados para criar regras de atualização de solução (caminho) diferentes do algoritmo. Oito regras diferentes são propostas para atualizar soluções no algoritmo. A atualização de uma solução por uma abelha empregada ou por uma abelha observadora é feita por uma regra selecionada aleatoriamente a partir do conjunto de regras usando o processo de seleção da Roleta. Na fase de reconhecimento do algoritmo, a técnica de perturbação, operação K-opt, é aplicada em qualquer solução estagnada por um número fixo de vezes para o possível aprimoramento dela. A operação K-opt é novamente usada no final do processo de pesquisa para melhorar a qualidade da solução final (se possível). O método proposto é testado com um conjunto de problemas de teste de benchmark do TSPLIB e observa-se que a eficiência do algoritmo é adequada no que diz respeito à precisão e à consistência para resolver TSPs padrão (simétricos e assimétricos) em comparação com os algoritmos existentes em a literatura.

1. Introdução

O problema do caixeiro viajante (TSP) é um dos problemas de otimização discreta combinatória padrão. O problema consiste em um conjunto de N vértices (nós / cidades) onde a distância entre quaisquer dois vértices é conhecida. Um vendedor parte de um vértice, visita todos os vértices exatamente uma vez e retorna ao vértice inicial de forma que a distância total percorrida seja mínima. Portanto, o objetivo do problema é encontrar um passeio mais curto possível pelo conjunto de vértices, de forma que cada vértice seja visitado exatamente uma vez, exceto pelo vértice inicial. É um problema NP-difícil bem conhecido, não pode ser resolvido exatamente usando qualquer algoritmo de tempo polinomial [1 , 2] Em um TSP, quando a distância entre quaisquer dois vértices x_{eu} e x_j é igual a distância entre x_j e x_{eu} então o problema é chamado de TSP simétrico (STSP) [3 , 4] Por outro lado, se a distância entre os vértices x_{eu} e x_j não é igual à distância entre x_j e $x_{eu,}$ para pelo menos um par de vértices, o problema pertence ao TSP assimétrico (ATSP) [5] Geralmente, existem duas abordagens para resolver um TSP: métodos exatos e métodos heurísticos. Os métodos exatos requerem um tempo enorme para maiores $N,$ assim, os métodos heurísticos são normalmente usados para resolver um TSP. Os métodos exatos incluem plano de corte [6], Relaxamento LP [7], ramificação e limite [8], ramificar e cortar [9], etc. Apenas TSPs de tamanho pequeno podem ser resolvidos por métodos exatos em uma janela de tempo razoável. Por outro lado, vários TSPs foram resolvidos usando heurísticas ou técnicas baseadas em computação suave, como Ant Colony Optimization (ACO) [3], pesquisa local [10], algoritmo híbrido [11], Algoritmo Genético (GA) [12], Otimização do Enxame de Partículas (PSO) [4], Arti fi cial Bee Colony (ABC) [13], etc. Existem várias heurísticas bem estabelecidas para STSP. Wang et al. [4] usou conceitos de operador de swap e sequência de swap, e redefiniu alguns operadores de PSO com base neles para resolver o TSP. Combinando os recursos de PSO, ACO e 3-opt, um algoritmo híbrido PSO-ACO-3-opt é apresentado por Mahi et al. [14] para resolver os TSPs padrão. Akhand et al. [15] propôs PSO com algoritmo de busca parcial para resolver TSPs. Akhand et al. [16] melhorou este algoritmo para encontrar a solução dos TSPs e o nomeou PSO de tentativa de velocidade. Geng et al. [17] propôs um algoritmo de busca local eficaz baseado em Simulated Annealing (SA) e técnicas de busca gulosa para resolver os TSPs. Jolai e Ghanbari [18] apresentou uma abordagem aprimorada de Rede Neural Arti fi cial (ANN) para resolver os TSPs. Dorigo et al. [3] propôs um Sistema Ant para resolver os TSPs. Dorigo e Gambardella [19] descreveu um ACO capaz de resolver os TSPs. Bontoux e Feillet [20] propôs um algoritmo híbrido para resolver os TSPs. O algoritmo Beam-ACO, que é um método híbrido que combina ACO com pesquisa de feixe, foi usado para resolver

* Autor correspondente.
Endereço de e-mail: indadulkhan@gmail.com (I. Khan), manasmaiti@yahoo.co.in (MK Maiti).

https://doi.org/10.1016/j.swevo.2018.05.006
Recebido em 22 de novembro de 2016; Recebido em forma revisada em 19 de maio de 2018; Aceito em 19 de maio de 2018
Disponível online XXX

os TSPs [21] Gunduz et al. [22] apresentou um novo algoritmo de inteligência de enxame baseado em método hierárquico para resolver TSPs. Khan et al. [23] apresentou um algoritmo híbrido baseado em ACO-PSO-K-otp para resolver TSP. Huang et al. [24] propôs um algoritmo baseado em PSO para resolver TSPs. Além disso, diferentes abordagens heurísticas são estabelecidas e sua eficiência na solução de outros problemas de otimização combinatória NP-difíceis como o problema de roteamento de veículos [25], problema de programação de fluxo de loja [26], etc. Recentemente, um levantamento detalhado sobre os diferentes algoritmos de colônias de abelhas desenvolvidos até agora para diferentes classes de problemas de otimização foi feito por Rajasekhar et al. [27] Dependendo das perspectivas e motivações biológicas, eles mencionam as perspectivas de diferentes algoritmos na respectiva classe de problemas e comparam os algoritmos com os outros.

A partir da discussão acima, pode-se concluir que as abordagens heurísticas são mais poderosas para resolver o TSP em uma janela de tempo viável. Desde 2005, o ABC provou ter sucesso em problemas de otimização contínua, muito trabalho foi feito de forma eficaz nesta área [28-30] Algumas versões discretas do ABC já foram introduzidas na literatura [26 , 31 , 32] Karaboga e Gorkemli [13] propôs um novo algoritmo ABC chamado Combinatorial ABC para os TSPs. Aplicando recursos de sequência de troca e operação de troca, Pan et al. [26] propôs um ABC discreto para o problema de programação de lot-streaming flow shop. Iqbal et al. [33] propôs um novo algoritmo Bee para resolver problemas de roteamento de veículos multi-objetivos com janelas de tempo suaves. Quando usado para TSP, qualquer algoritmo heurístico às vezes converge para o caminho ideal local (passeio). O K-opt é uma técnica de perturbação que pode ser aplicada em um passeio (caminho) para superar essa convergência. Na verdade, K-opt é a abordagem heurística mais amplamente usada para pesquisar o melhor vizinho de qualquer solução potencial de um TSP. Funciona como uma função de mutação de um GA. O K-opt é um algoritmo de melhoria de passeio, onde em cada etapa K links do passeio atual são substituídos por K links de tal maneira que um passeio mais curto é alcançado [34]

Neste artigo, uma nova variante do algoritmo ABC é proposta para resolver um STSP e também um ATSP. Neste algoritmo, oito regras diferentes são propostas para melhorar uma solução por uma abelha empregada ou por uma abelha observadora. As regras são definidas usando sequência de troca e operações de troca em um caminho potencial (solução) do vendedor [16 , 35] Para melhorar uma solução, uma abelha empregada ou uma abelha observadora seleciona uma regra da regra definida aleatoriamente usando o processo de seleção da Roda da Roleta (RW) [36] Para uma solução atribuída a uma abelha exploradora, a princípio a operação K-opt é aplicada para um número fixo de iterações para seu possível aprimoramento. Se não for capaz de melhorar a solução, a solução é regenerada aleatoriamente. A operação K-opt é novamente usada no final do algoritmo para melhorar a qualidade da solução final (se possível). O método proposto é testado com um conjunto de problemas de teste de benchmark do TSPLIB [37] e observa-se que a eficiência do algoritmo é adequada no que diz respeito à precisão e consistência para resolução de TSPs padrão (ATSP bem como STSP) em relação aos algoritmos existentes na literatura.

O resto do artigo está organizado da seguinte forma: na seção 2 , matemática- a formulação matemática do problema é apresentada. Na seção 3 , a sequência de troca e as operações de troca são discutidas. Na seção 4 , os recursos do ABC são discutidos. A operação K-opt é apresentada na seção 5 . O algoritmo proposto é apresentado na seção 6 . Os resultados experimentais são discutidos na seção 7 . Uma breve conclusão é tirada na seção 8 .

2. Formulação do problema

Um TSP pode ser representado por um gráfico $G = (V, E)$, Onde $V = \{1, 2, \ldots N\}$ é o conjunto de vértices ou nós e E é o conjunto de arestas. Aqui, cada nó representa uma cidade e cada borda representa o caminho entre duas cidades. Cada borda está associada a uma distância que representa a distância entre as cidades associadas a ela. Um vendedor viaja distâncias para visitar N número de cidades (ou nós) ciclicamente. Em uma excursão, ele visita cada cidade exatamente uma vez e termina onde começou com uma distância mínima de viagem. Deixar d_{jk} seja a distância entre j- a cidade e k- ª cidade. Então, o modelo pode ser formulado matematicamente como [7],

$$\left.\begin{array}{l} \text{Determinar } x_{jk}, \forall\, j, k \in V \,\&\, j \neq k \text{ para} \\ \text{Minimizar} \quad Z = \sum_{j=1}^{N} \sum_{k=1}^{N} x_{jk}\, d_{jk} \\ \text{sujeito a} \quad \sum_{j=1}^{N} x_{jk} = 1, \forall\, k \in V \\ \sum_{k=1}^{N} x_{jk} = 1, \forall\, j \in V \end{array}\right\} \quad (1)$$

Onde $x_{jk} = 1$ se o vendedor viajar da cidade- j para a cidade- k, por outro lado $x_{jk} = 0$

Deixar $(x_1, x_2 \ldots, x_N, x_1)$ ser um tour completo de um vendedor, onde $x_j \in V, \forall\, j \in V$ e tudo x_j são distintos, ou seja, $(x_1, x_2 \ldots, x_N, x_1)$ é a sequência de cidades em que o vendedor percorre as cidades. Então o o problema acima se reduz a [38],

$$\left.\begin{array}{l} \text{Determine um tour completo } (x_1, x_2, \ldots\ldots\ldots x_N, x_1) \\ \text{para minimizar } Z = \sum_{j=1}^{N-1} d_{x_j x_{j+1}} + d_{x_N x_1} \end{array}\right\} \quad (2)$$

3. Troca de sequência e troca de operadores para TSP

O conceito de operador de swap e sequência de swap foi introduzido por Wang et al. [35] em 2003 para resolver TSPs usando PSO. Pan et al. [26] usou essas operações no DABC para resolver o problema de programação do fl u shop. Breves descrições dessas operações são fornecidas abaixo.

Operador de troca: Considere uma sequência de solução normal de nós $X = (x_1, x_2, \ldots x_N, x_1)$ de um TSP com N nós, tendo o conjunto de nós $V = \{1, 2, \ldots N\}$, Onde $x_{eu} \in V$ e $x_{eu} \neq x_j, \forall\, eu \neq j$. Aqui, troca operator, $SO\ (i, j)$ é definido como a troca do nó x_{eu} e o nó x_j na sequência de solução X. Então $X' = X \Diamond SO\ (i, j)$ é definido como um novo sequência de solução após operar o operador $SO\ (i, j)$ sobre X. O símbolo $\Diamond$ representa a operação de troca de operação binária.

Pode ser um exemplo concreto: suponha que haja um TSP com seis nós, e $X = (x_1, x_2, x_3, x_4, x_5, x_6) = (1, 3, 5, 2, 4, 6)$ seja uma sequência de solução. Deixar $TÃO(2, 4)$, ser um operador de troca então $X' = X \Diamond TÃO(2, 4) = (1, 3, 5, 2, 4, 6) \Diamond TÃO(2, 4) = (1, 2, 5, 3, 4, 6)$, ou seja, nós da posição 2 e posição 4 da X são trocados para criar a nova sequência de solução X'.

Sequência de troca: Uma coleção de diferentes operadores de troca em um determinado a ordem ular é chamada de sequência de troca WL. Deixar $SS = (SO_1, TÃO_2, \ldots, TÃO_N)$, Onde $TÃO_1, TÃO_2, \ldots, TÃO_N$ são operadores de troca. A sequência de troca agindo em uma solução significa que todos os operadores de troca da sequência de troca agem na solução em ordem. Isso pode ser descrito pela seguinte fórmula:

$$X' = X \Diamond SS = X \Diamond (TÃO_1, TÃO_2, \ldots, TÃO_N) = (\ldots ((X \Diamond TÃO_1) \Diamond TÃO_2) \ldots \Diamond TÃO_N)$$

Diferentes sequências de troca agindo na mesma solução podem produzir a mesma nova solução. Todas essas sequências de troca são chamadas de conjunto equivalente de sequências de troca. No conjunto equivalente, a seqüência que tem o menor número de operadores de troca é chamada de seqüência de troca básica do conjunto ou seqüência de troca básica (BSS) resumidamente.

Várias sequências de troca podem ser mescladas em uma nova sequência de troca. Aqui, a operadora $\oplus$ é definido como a fusão de duas sequências de troca em uma nova sequência de troca. Suponha que haja duas sequências de troca, WL 1 e WL 2 agir em uma solução X em ordem, a saber WL 1 primeiro, WL 2 segundos e uma nova solução X' é obtido. Deixe que haja outra sequência de troca WL' agindo na mesma solução X e obter a solução X', então WL' é chamado de fusão de WL 1 e WL 2 e é representado como:

$WL' = WL1 \oplus WL2$

Aqui, WL' e $WL1 \oplus WL2$ estão no mesmo conjunto equivalente.

A construção da sequência de troca básica: Suponha que existam duas soluções, *UMA* e *B* e nossa tarefa é construir um BSS, *WL* que pode agir em *B* para obter solução *UMA.* Aqui, *WL* é definido como $SS = A \ominus B$. Então a operadora $\ominus$ representa a diferença entre duas sequências de cidades e obviamente essa diferença é representada por algumas operações de troca consecutivas, ou seja, trocar os nós em *B* de acordo com *UMA* da esquerda para a direita para obter *WL.* Portanto, deve haver uma equação $A = B \Diamond WL$. Por exemplo, considere duas soluções:

$A = (1, 2, 3, 4, 5)$, $B = (2, 3, 1, 5, 4)$

Aqui, $UMA(1) = B(3) = 1$. Portanto, o primeiro operador de troca é $TÃO(1, 3)$. Então operando $TÃO(1, 3)$ em *B,* a nova sequência $B1 = B \Diamond TÃO(1, 3)$ é obtido como abaixo:

$B1 = B \Diamond TÃO(1, 3) = (1, 3, 2, 5, 4)$

Novamente $UMA(2) = B1(3) = 2$, então o segundo operador é $TÃO(2, 3)$ e operacional $TÃO(2, 3)$ em $B1$, nova sequência $B2$ é obtido como

$B2 = B1 \Diamond TÃO(2, 3) = (1, 2, 3, 5, 4)$ Da mesma forma,

o terceiro operador é $TÃO(4, 5)$, e

$B3 = B2 \Diamond TÃO(4, 5) = (1, 2, 3, 4, 5) = UMA$

Conseqüentemente, o BSS, $SS = A \ominus B = (SO(1, 3), TÃO(2, 3), TÃO(4, 5))$.

4. Arti fi cial Bee Colony for TSP

Imitando o comportamento de forrageamento e dança balançante das abelhas em suas colônias, a heurística ABC foi introduzida por Karaboga em 2005 para resolver problemas de otimização numérica [39] Um algoritmo ABC normalmente começa a buscar a solução ótima do problema de otimização em consideração com um conjunto de soluções potenciais geradas aleatoriamente no espaço de busca. No algoritmo, as soluções potenciais são análogas às posições de diferentes fontes de alimentos. O objetivo das abelhas é buscar a melhor fonte de alimento que seja análoga à solução ótima do problema. Dependendo das diferentes atividades, as abelhas podem ser classificadas em três categorias: abelha empregada, abelha observadora e abelha escuteira. Assim, cada iteração do algoritmo consiste em três fases, nomeadamente a fase da abelha empregue, a fase da abelha observadora e a fase da abelha exploradora. A primeira fase é empregada na fase de abelha. Cada solução do conjunto de soluções gerado aleatoriamente está associada a uma abelha empregada. Na fase de abelha empregada, cada abelha empregada tenta melhorar sua posição (isto é, solução correspondente) para procurar melhor fonte de alimento (isto é, solução melhorada) usando uma regra específica. O algoritmo consiste em um conjunto de abelhas observadoras. Após a conclusão da fase de abelha empregada, a fase de abelha observador começa. Nesta fase, cada abelha observadora seleciona uma solução do conjunto de soluções de acordo com a adequação das soluções e tenta melhorá-la. Cada solução está associada a um contador e seu valor é zerado no momento da geração. Se uma solução não é melhorada por sua abelha empregada ou por qualquer abelha observadora, seu contador é aumentado em um. Na última fase, ou seja, na fase de escoteira, todas as soluções são verificadas. Se for descoberto que uma solução não é melhorada por sua abelha empregada ou por qualquer abelha observadora em um número predefinido de iterações, a solução é tratada como uma solução estagnada e atribuída a uma abelha escuteira. A abelha batedora o regenera aleatoriamente e seu valor de contador correspondente é definido como zero.

Na variante proposta do ABC, considera-se que na fase de abelha escoteira, uma abelha escuteira inicialmente tenta melhorar sua solução usando a operação Kopt por um número pré-definido de vezes. Se qualquer melhoria for encontrada, a solução vai para a próxima iteração com seu valor de contador zero, caso contrário a abelha escuteira o regenera aleatoriamente e seu contador valor é definido como zero. A operação K-opt é discutida na seção § 5. Além disso, semelhante a Kiran et al. [28], aqui um conjunto de regras de atualização de solução (aplicável para TSPs) é proposto para a atualização de uma solução por uma abelha empregada ou por uma abelha observadora. No momento de sua operação, uma abelha empregada ou uma abelha observadora seleciona uma regra do conjunto de regras usando um processo de seleção apropriado (que será discutido mais tarde) e tenta melhorar sua solução usando a regra selecionada.

Seguindo a discussão acima, para resolver um TSP usando o ABC proposto, suas diferentes etapas e procedimentos são brevemente discutidos aqui. Como no algoritmo ABC, cada fonte de alimento representa uma solução potencial do problema, aqui uma sequência de nós é tomada como uma solução (fonte de alimento) do considerado TSP. Então, uma solução X_{eu} do conjunto de solução inicial é gerado aleatoriamente no espaço de solução usando (3) .

$$X_{i=(x_{eu1}, x_{eu2}, \ldots x_{dentro}, x_{eu1}),\ i=1, 2, \ldots f_s} \quad (3)$$

Onde, $x_{euj} \in V, j = 1, 2, \ldots, N, V = \{1, 2, \ldots N\}$ é o nó definido, $x_{euj} \neq x_{ik}$ para $j \neq k$, f_s é o número de soluções (fontes de alimentos) no conjunto de soluções. Após a inicialização de todas as soluções por equação (3) , cada solução é atribuída a uma abelha empregada, ou seja, o número de soluções é igual ao número de abelhas empregadas. Após a geração do conjunto de solução inicial, a adequação do *eu*- a solução é calculada usando (4) .

$$caber_{i=1} / (M_c - f(X_{i)} + 1) \quad (4)$$

Onde $caber_{eu}$ é a aptidão de X_{eu}, $f(X_{eu})$ é a duração total do passeio de X_{eu} e M_c = máximo do conjunto $\{f(X_1), f(X_2), \ldots, f(X_{fs})\}$.

Na fase de abelha empregada do algoritmo ABC, cada abelha empregada tenta melhorar sua solução selecionando aleatoriamente uma das seguintes equações (5) - (12) . A técnica de seleção de equação é discutida posteriormente.

$$Y_i = X_j \Diamond (r \odot (X_{eu} \ominus X_k)), \quad i = 1, 2, \ldots f_s, eu \neq k \quad (5)$$

$$Y_i = X_{eu} \Diamond (r \odot (X_j \ominus X_k)), \quad i = 1, 2, \ldots f_s, j \neq k \quad (6)$$

$$Y_i = X_{melhor} \Diamond (r \odot (X_{eu} \ominus X_k)), \quad i = 1, 2, \ldots f_s, eu \neq j \neq k \quad (7)$$

$$Y_i = X_{eu} \Diamond (r \odot (X_{eu} \ominus X_{melhor})), \quad i = 1, 2, \ldots f_s, \quad (8)$$

$$Y_i = X_{melhor} \Diamond (r \odot (X_{melhor} \ominus X_k)), i = 1, 2, \ldots f_s, eu \neq k \quad (9)$$

$$Y_i = X_{eu} \Diamond (r \odot (X_{melhor} \ominus X_{pior})), i = 1, 2, \ldots f_s, eu \neq k \quad (10)$$

$$Y_i = X_{eu} \Diamond (r \odot (X_{melhor} \ominus X_k) \oplus r_1 \odot (X_k \ominus X_{eu})) \quad (11)$$

$$Y_i = X_j \Diamond (r \odot (X_{melhor} \ominus X_{eu})), i = 1, 2, \ldots f_s, eu \neq j \quad (12)$$

Onde Y_{eu} é uma nova solução derivada de X_{eu} por uma abelha empregada. X_j, X_k são soluções selecionadas aleatoriamente do conjunto de soluções, $k \neq j$ e $k, j \in \{1, 2, \ldots f_s\}$. X_{melhor} é a melhor solução obtida até agora e X_{pior} é a pior solução (ou seja, baixa aptidão) no conjunto de soluções, r é o aleatório número no intervalo [0, 1]. Aqui, $r \odot (X_{eu} \ominus X_k)$ significa todos os operadores de swap em BSS ($X_{eu} \ominus X_k$) deve ser mantida com a probabilidade de r, ou seja, cada operador de swap em BSS ($X_{eu} \ominus X_k$) deve ser selecionado com probabilidade r. Aqui, a operadora $\oplus$ é definido como a fusão de dois swap sequências em uma nova sequência de troca. O significado desses operadores é o mesmo para todas as outras expressões em (5) - (12) . Após a geração de a solução candidata (nova) Y_{eu}, é comparado com X_{eu}. Se Y_{eu} é melhor que X_{eu} então X_{eu} é substituído por Y_{eu} e seu contador é definido como zero, caso contrário X_{eu} permanece inalterado e seu contador é aumentado em 1.

A próxima etapa do algoritmo ABC é a fase de abelha observadora. Inicialmente, o chance p_{eu} de selecionar *eu*- a fonte de comida x_{eu} por uma abelha observadora é calculado pela seguinte equação (13) .

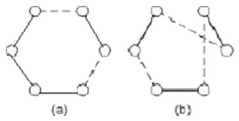

Figura 1. Todas as combinações de sub_toures para k = 2.

$$p_i = \sum_{j=1}^{caber_{eu}} \frac{Eb}{caber_j}, \quad i = 1, 2, \ldots Ob \tag{13}$$

Aqui, Eb é o número de abelhas empregadas e Ob é o número de abelhas observadoras no algoritmo ABC proposto (apresentado posteriormente). Após seleção, a abelha espectador tenta melhorar a solução usando uma das equações (5) - (12) . O processo de seleção de uma regra é apresentado posteriormente. A solução é atualizada da mesma forma que a fase de abelha empregada. As etapas restantes da seleção de RW de uma solução estão listadas abaixo:

(i) Calcule a probabilidade cumulativa cp_{eu} de eu-ª solução pelo Fórmula $cp_i = \sum_{j=1}^{eu} p_j$.
(ii) Gerar um número aleatório r no intervalo (0, 1).
(iii) Se $r < cp_1$ então selecione X_1 Caso contrário, selecione X_{eu} E se $cp_{eu-1} \le r < cp_{eu}$.

A última fase do algoritmo ABC é a fase de abelha exploradora, que já foi discutida. Portanto, os recursos do algoritmo recentemente propostos podem ser resumidos como abaixo:

- Usando a sequência de troca e a operação de troca, um conjunto de regras de atualização de solução (Eqs. (5) - (12)) apropriado para TSP é proposto e usado.
- Uma regra de avaliação de aptidão de diferentes soluções (fontes de alimentos) apropriada para TSP é proposta (Eq. (4)) Usando os valores de aptidão, o processo de seleção de RW é aplicado corretamente para selecionar soluções para as abelhas observadoras.
- Na fase de escoteira de qualquer algoritmo ABC, normalmente as soluções estagnadas são regeneradas. Aqui, a operação 3-opt é usada apropriadamente para a possível melhoria das soluções estagnadas. Essa abordagem evita a pesquisa de soluções recém-geradas e ajuda a pesquisar a solução usando menos número de iterações.
- Aqui, a operação 3-opt é novamente usada para a possível melhoria da melhor solução encontrada no final do algoritmo.

Técnica de seleção de regra: Cada uma das primeiras duas fases do algoritmo ABC seleciona uma regra do conjunto de regras (Eqns. (5) - (12)) para melhorar uma solução. Aqui, cada regra está associada a um contador com valor inicial 1. Dependendo deste valor do contador, o processo de seleção RW é usado para selecionar uma regra. Após a seleção de uma regra, ela é usada para melhorar uma solução. Se ocorrer uma melhoria, o contador da regra é aumentado em um e esse processo continua em todo o algoritmo.
Portanto, a probabilidade de selecionar eu-ª regra $\bullet_{eu}$ é dado pela seguinte equação (14) .

$$\bullet_i = \sum_{j=1}^{Neq} \frac{V_{eu}}{V_{eu}}, \quad i = 1, 2, \ldots N_{eq} \tag{14}$$

Onde $\bullet_{eu}$ é a probabilidade de seleção de eu-a regra por uma abelha empregada ou uma abelha observadora, N_{eq} é a cardinalidade do conjunto de regras e v_{eu} é o valor do contador correspondente a eu-ª regra. Etapas restantes de RW-seleção de uma regra estão listados abaixo:

(i) Calcule a probabilidade cumulativa $c\bullet_{eu}$ de eu-ª regra pelo paramula $c\bullet_i = \sum_{j=1}^{eu} \bullet_j$.
(ii) Gerar um número aleatório r no intervalo (0, 1).
(iii) Se $r < c\bullet_1$ em seguida, selecione regra-1. Caso contrário, selecione a regra- eu E se $c\bullet_{eu-1} \le r < c\bullet_{eu}$.

5. Operação K-opt para TSP

K-opt é uma técnica de perturbação [40] usado principalmente no TSP, que é baseado na troca de peças K (sub-tours) e seus reversos (sub-tours reversos) de um tour (caminho) do TSP em consideração para encontrar um tour melhor. Aqui, a operação K-opt é usada na fase de escoteira do algoritmo ABC para melhorar uma solução. Se for assumido que nós / citações estão conectados por arestas, então, enquanto quebra (remove) K arestas de uma solução em um passeio, existem $(K-1)!\,2^{K-1}$ maneiras de reconectá-lo (incluindo a sequência inicial) para formar uma sequência válida. Cada nova combinação oferece uma nova solução. Entre essas soluções, pode-se produzir um tour melhor do que o tour da solução original e pode ser considerado uma melhoria. No caso do algoritmo 2-opt, duas arestas de uma solução (sequência de cidades) são removidas, e todas as combinações possíveis de subseqüências e seus reversos são reconectadas para buscar a melhor solução (Figura 1) Este processo é continuado até que nenhuma melhoria (por 2-opt) seja encontrada. Da mesma forma, no caso de 3-opt, quebrando 3 arestas em uma sequência, há no total 8 casos de reconexão (Figura 2) [41] Se uma sequência é 3 ótima, também é 2 ótima [40] Continuando este processo de quebrar (remover) mais arestas formam uma sequência, ou seja, $K = 1, 2, 3 \ldots, n$, pode-se obter novos algoritmos, como 2-opt, 3-opt, 4-opt e assim por diante. Mas aumento de K aumenta a complexidade do tempo. Devido a isso, aqui a operação de 3 opções é usada e verifica-se que ela atua melhor do que a operação de 2 opções para TSPs de tamanho grande.

5.1. Operação K-opt em um tour X completo $_k$

Algoritmo detalhado da operação K-opt para $K = 3$ é apresentado a seguir. No algoritmo, uma matriz unidimensional de tamanho N, X_{temk}, é usado para representar uma solução temporária correspondente a X_k em iteração t. X_{ki} e X_r passeios $_k$ uma $_{eu}$, $i = 1, 2, 3$ são arrays unidimensionais usados para representar sub-tours reversos do tour original de X_k. $Maxit$ 3 é o número máximo de iterações.

6. Algoritmo ABC proposto para TSP

Presume-se que o problema envolve N nós, $d_{eu\,j}$ representa a distância entre o nó eu e nó j. X_{eu} é uma matriz unidimensional de tamanho N, costumava representar eu-ª solução. $f(X_{eu})$ representa a duração total do tour da solução X_{eu}. Aqui, Eb representa o número de abelhas empregadas e Ob é o número de abelhas observadoras no algoritmo ABC. Aqui, Ec_{eu} é um matriz unidimensional de tamanho N_{eq} isto é, número de regras), usado para representar contadores de regras diferentes. Inicialmente, os contadores de equação são definidos como 1 correspondendo a todas as regras. Outra matriz unidimensional Fc_{eu} do tamanho $f_{s\,(}$ ou seja, número de fontes de alimentos) é usado para representar contadores de fontes de alimentos de diferentes soluções. Inicialmente, todos os contadores de fontes de alimentos são zerados.
lim é um número inteiro positivo que é usado como o limite superior do contador da fonte de alimento. $MaxGen$ é um número inteiro positivo que representa o número máximo de iterações e t é o contador de iterações. Outras notações no algoritmo são as mesmas discutidas anteriormente.

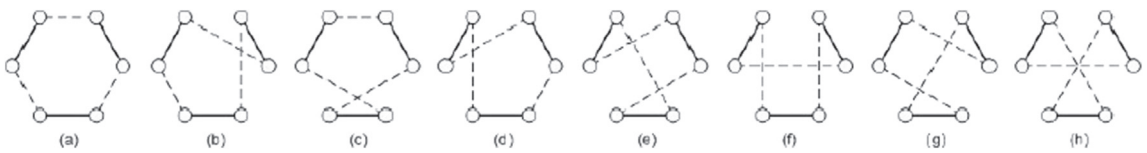

Figura 2. Todas as combinações de sub_toures para k = 3.

1: **Inicie o algoritmo.**
2: **para i = 1:** *Maxit* 3 **Faz**
3: Remova 3 arestas (selecionadas aleatoriamente) do passeio completo X_k, faz 3 sub-tour X_{ki}, $i = 1, 2, 3$.
4: Os reversos do conteúdo deste sub-tour são chamados de reverse_sub-tour, representados como X_r $_{ki}$, $i = 1, 2, 3$, ou seja, X_r $_{k1}$ = reverse_sub-tour de X_{k1}, X_r $_{k2}$ = reverse_sub-tour de X_{k2}, X_r $_{k3}$ = reverse_sub-tour de X_{k3}
5: Agora c $_k$ mbing os sub-tours { X_{k1}, X_{k2}, X_{k3}}, { X_r $_{k1}, X_r$ $_{k2}, X_r$ $_{k3}$} nova sequência de passeio pode ser formada nas seguintes 8 combinações:
6: { X_{k1}, X_{k2}, X_{k3}}.
7: { X_{k1}, X_r $_{k2}, X_{k3}$}.
8: { X_{k1}, X_{k2}, X_r $_{k3}$}.
9: { X_{k1}, X_r $_{k3}, X_{k}$
10: { X_{k1}, X_{k3}, X_{r} $_{k2}$}
11: { X_{k1}, X_r $_{k3}, X_{k2}$}
12: { X_{k1}, X_{r} $_{k2}, X_{r}$}.
13: { X_{k1}, X_{r} $_{k3}, X_{k3}$}
14: **para** cada combinação **Faz**
15: Crie uma nova sequência de solução a partir da combinação e deixe-a ser X_{temk}.
16: Determinar $f(X_{temk})$.
17: **E se** $f(X_{temk}) < f(X_k)$ então
18: $X_k = X_{temk}$
19: **fim se**
20: **fim para**
21: **fim para**
22: **Retorna** X_k.
23: **fim do algoritmo**

6.1. Implementação do algoritmo

O algoritmo é implementado na linguagem de programação Dev C ++ 5.8.3 medição em um computador com núcleo *eu* 3 CPU @ 2,10 GH $_z$, Sistema operacional Windows 8.1 e 4 GB de RAM. Para implementação do algoritmo, values de diferentes parâmetros do algoritmo são considerados como número de abelhas empregadas (fonte de alimento), E_b = 20, número máximo de iterações, *MaxGen* = 500, número de abelhas observadoras, O_b, é considerado como um função de E_b e é da forma $O_b = [Eb$ $_{2}] + 1$, limite superior de um alimento contador de fonte, lim = 5, *Maxit* 3 = 10. Aqui [Eb $_{2}]$ significa parte integrante de $\frac{Eb}{2}$

7. Resultados experimentais

O desempenho do algoritmo proposto é testado usando diferentes padrões de tamanhos de TSPs do TSPLIB. A partir de cada problema, o algoritmo é testado executando o programa 10 vezes para diferentes sementes do gerador de números aleatórios e a melhor solução obtida, o valor médio das soluções, o valor do desvio padrão (DP) das soluções e a porcentagem de erro relativo (Erro (%)) de acordo com a solução ótima são calculados. A porcentagem de relativa **Erro (%)** é calculado usando a seguinte equação.

$$\text{••••• (\%)} = \frac{\text{solução média} - \text{solução ótima}}{\text{solução ótima}} \times 100 \quad (15)$$

Os resultados obtidos pelo algoritmo proposto para dezoito problemas de teste diferentes do TSPLIB são apresentados em tabela 1 . Dentro tabela 1 , os resultados de STSPs e ATSPs são exibidos separadamente. Dentro a mesa **Melhor** coluna representa a melhor solução obtida pelo método proposto e as soluções ótimas são retiradas do TSPLIB. Os problemas cujas soluções ótimas (de acordo com o TSPLIB) são obtidas pela abordagem proposta são apresentados em negrito. É encontrado a partir de tabela 1 que o algoritmo proposto produz solução ótima para a maioria dos problemas considerados para o teste e para outros fornece soluções muito próximas das soluções ótimas. Para problemas como, *eli* 101, *kroA* 200, *cinco* 56, o algoritmo não fornece solução ideal, mas outros parâmetros como **média, Sd** e **Erro (%)** são melhores em comparação com alguns algoritmos existentes (cf. Tabela 3) Também pequenos valores de **SD** e **Erro (%)** das soluções dos problemas garantem que as soluções obtidas dos problemas estão muito próximas das soluções ótimas correspondentes, bem como o algoritmo é e fi ciente e consistente para resolver tais problemas. Pode ser concluído a partir de tabela 1 que o algoritmo precisa de uma quantidade muito pequena de tempo para encontrar soluções para os problemas de teste (ver Tabela 4)

mesa 2 representa os resultados obtidos pelo método proposto devido a diferentes problemas de teste usando operações 2-opt e 3-opt no algoritmo. No caso de problemas de tamanho pequeno, como *gr* 17 e *br* 17 ambas as abordagens fornecem a mesma solução que a solução ótima. Mas, para problemas de grande porte, 2-opt e 3-opt produzem soluções diferentes. Problemas para os quais soluções ótimas são obtidas pelo algoritmo são apresentados em negrito em mesa 2 . Em alguns problemas como *eli* 51, *kroA* 200, *cinco* 56, usando 3-opt, o algoritmo não fornece uma solução ótima, mas produz soluções melhores do que a obtida pelo algoritmo usando 2-opt. Também é claro de mesa 2 que para todos os problemas o algoritmo com 3-opt fornece um resultado melhor do que com 2-opt. Assim, no algoritmo proposto, a operação 3-opt é usada.

Tabela 3 representa uma comparação de alguns resultados computacionais do algoritmo proposto com alguns outros algoritmos existentes na literatura. Na tabela, os resultados de outros algoritmos são retirados das Refs. [14 , 46] A partir de Tabela 3 , é claro que para a maioria dos problemas considerados, o desempenho da abordagem proposta no que diz respeito à precisão é melhor em comparação com as outras abordagens existentes na literatura. Para os problemas de teste como *eli* 51, *st* 70, *eli* 76, *rato* 99, *eli* 101, *kroA* 200, o algoritmo produz melhores valores de *Média, SD, Erro* em comparação com outros algoritmos. Para o problema de teste *kroA* 100, o método proposto fornece uma solução ótima, mas outros parâmetros como *Média, SD, Erro* não são melhores em comparação com WFA com 3-opt [45] Em alguns problemas de teste, o método proposto não fornece soluções ótimas, mas as soluções obtidas estão muito próximas das soluções ótimas correspondentes e também os resultados são melhores em comparação com outros algoritmos existentes, pois o algoritmo proposto produz o desvio padrão mínimo (SD) comparado aos outros algoritmos em consideração.

7.1. Teste Estatístico de Algoritmos (Teste de Friedman)

Aqui, um procedimento estatístico não paramétrico teste de Friedman [47] é considerado para comparar o desempenho do algoritmo proposto com alguns outros algoritmos existentes na literatura. Os outros algoritmos considerados são RABNET-TSP [42], RABNET -TSP modificado [43], SA ACO PSO [44], WFA com 2 opções [45], WFA com 3 opções [45], GA-PSO- ACO [46], PSO-ACO-3opt [14] O principal objetivo do uso de Friedman

Inicie o algoritmo.
Definir valores de N, f_s, E_b, O_b, lim, *MaxGen*, *Maxit* 3, e definir $t = 0$
para $i = 1 : f_s$ **Faz**
  Gerar aleatoriamente X_{eu}.
  Determinar $f(X_{eu})$.
fim para
Defina a melhor solução inicial $X_{melhor} = X_k$, Onde $f(X_k) = \min \{ f(X_1), f(X_2), \ldots, f(X_{Eb}) \}$.
Defina a pior solução inicial $X_{pior} = X_k$, Onde $f(X_k) = \max \{ f(X_1), f(X_2), \ldots, f(X_{Eb}) \}$.
repetir até ($t > MaxGen$)
  $t = t + 1$ // Fase de abelha empregada
  para $i = 1 : E_b$ **Faz**
    Determinar Y_{eu} a partir de X_{eu} seguindo uma regra selecionada (a regra é selecionada usando eqn. (14))
    Determinar $f(Y_{eu})$.
    Se $f(Y_i) < f(X_{eu})$ então
      $X_i = Y_{eu}$.
      $Fc_i = 0$
    senão
      $Fc_i = Fc_i + 1$
    fim se
  fim para
  para $i = 1 : O_b$ **Faz** // Fase da abelha observadora
    Selecione uma solução X_k de acordo com a aptidão (usando a eqn. (13)) Determinar Y_k a partir de X_k seguindo
    uma regra selecionada (a regra é selecionada usando eqn. (14)) Determinar $f(Y_k)$.

    Se $f(Y_k) < f(X_k)$ então
      $X_k = Y_k$.
      $Fc_k = 0$
    senão
      $Fc_k = Fc_k + 1$
    fim se
  fim para
  para $i = 1 : f_s$ **Faz** // Fase de abelha batedora
    Se $Fc_{eu} >$ Limthen
      Aplicar **Operação K-opt (** § 5.1) em X_{eu} e obter a solução de vizinho Y_{eu}.
      $Fc_i = 0$
        Se $f(X_i) > f(Y_{eu})$ então
          $X_i = Y_{eu}$.
        senão
          Gerar aleatoriamente X_{eu}.
        fim se
      Determinar $f(X_{eu})$.
    fim se
  fim para
  para $i = 1 : f_s$ **Faz**
    Se $f(X_{melhor}) > f(X_{eu})$ então
      $X_{melhor} = X_{eu}$.
    fim se
  fim para
  $X_{pior} = X_1$
  para $i = 2 : f_s$ **Faz**
    Se $f(X_{pior}) < f(X_{eu})$ então
      $X_{pior} = X_{eu}$.
    fim se
  fim para
Loop final
Aplicar **Operação K-opt** sobre X_{melhor} um número pré-definido de iterações para melhorar sua qualidade (se possível).
Resultado X_{melhor}.
Fim do algoritmo.

tabela 1
O resultado obtido pelo algoritmo proposto para alguns STSP e ATSP do TSPLIB.

					Informações estatísticas de problemas de teste				
	Problema de TSPLIB	Custo ideal do problema	Melhor custo obtido	Pior custo obtido	Custo médio das soluções	Valor mediano da solução	Desvio padrão (SD) das soluções	Porcentagem de Erro (Erro (%))	Computacional tempo em segundo
STSP	gr17	2085	**2085**	2085	2.085,00	2085	0,00	0,00	1,35
	baías 29	2020	**2020**	2020	2020,00	2020	0,00	0,00	1,24
	suíço 42	1273	**1273**	1273	21273,00	1273	0,00	0,00	1,37
	eli51	426	427	428	427,01	427	0,46	0,07	4,2
	berlin52	7542	**7542**	7542	7542,00	7542	0,00	0,00	3,15
	st70	675	**675**	678	675,77	675	1,17	0,18	100,24
	eli76	538	**538**	540	538,15	538	0,60	0,02	64,34
	rat99	1211	**1211**	1213	1211,50	1211	0,67	0,041	200,21
	kroA100	21282	**21282**	21305	21287,19	21284	8,10	0,02	51,32
	eli101	629	**629**	635	630,59	629	2,37	0,25	400,32
	lin105	14379	**14379**	14385	14379,10	14379	1,30	0,00	28,42
	pr124	59030	**59030**	59090	59054,64	59042	27,30	0,04	130,41
	pr152	73682	**73682**	73705	73691,64	73686	28,26	0,12	1031,32
	kroA200	29368	29450	29500	29469,00	29464	20,03	0,146	900,12
ATSP	br17	39	**39**	39	39,00	39	0,00	0,00	1,13
	ftv33	1286	**1286**	1286	1286,00	1286	0,00	0,00	5,56
	ry48	14422	**14422**	14642	14452,79	14448	64,79	0,21	15,12
	ftv56	1608	1629	1689	1642,19	1636	18,87	0,810	25,12

mesa 2
Desempenho do algoritmo usando 2-opt e 3-opt em problemas de teste diferentes do TSPLIB.

	Problema de TSPLIB	Custo ideal de a solução	Custo do melhor caminho obtido pelo algoritmo híbrido	
			ABC + 2-opt	ABC + 3-opt
TSP	gr17	2085	**2085**	**2085**
	baías 29	2020	2028	**2020**
	suíço 42	1273	1284	**1273**
	eli51	426	447	427
	berlin52	7542	7800	**7542**
	st70	675	699	**675**
	eli76	538	550	**538**
	rat99	1211	1270	**1211**
	kroA100	21282	21910	**21282**
	eli101	629	795	**629**
	lin105	14379	15500	**14379**
	pr124	59030	62040	**59030**
	pr152	73628	73925	**73682**
	kroA200	29368	30290	29450
ATSP	br17	39	**39**	**39**
	ftv33	1286	1340	**1286**
	ry48	14422	14020	**14422**
	ftv56	1608	1648	1629

O teste aqui é detectar a existência de qualquer diferença significativa entre o comportamento de dois ou mais algoritmos. As duas suposições a seguir são feitas para o Teste de Friedman:

- Os resultados nas instâncias são mutuamente independentes (ou seja, o resultado em uma instância não influencia o resultado em outras instâncias.
- Dentro de cada instância, as observações podem ser classificadas.

Além disso, as duas hipóteses a seguir H_0 e H_1 são considerados para o Teste de Friedman, onde H_0 é a hipótese nula e H_1 é a hipótese alternativa.

H 0 Cada classificação dos algoritmos dentro de cada problema é igualmente provável.

H 1 Pelo menos um dos algoritmos tende a produzir uma função objetivo maior do que pelo menos um dos algoritmos.

Aqui número de algoritmos (m) = 8, número de teste problemas (b) = 7. Aqui, apenas sete problemas de teste são considerados para o teste estatístico porque os custos médios obtidos por diferentes algoritmos para outros problemas não são fornecidos na literatura. A tabela de classificação de Friedman é apresentada abaixo, preparada de acordo com o resultado médio de Tabela 3 . Deixar

$$UMA_2 = \sum_{i=1}^{m} \sum_{j=1}^{b} [R(ALi_j)]^2, \qquad R_i = \sum_{j=1}^{b} R(ALi_j)$$ para

$i = 1, 2, 3 \ldots, m$ e $B_2 = \frac{1}{b} \sum_{i=1}^{m} R_i^2$. Então a estatística de teste é dado é por $T_2 = (b-1)[B_2 - bm(m+1)^2/4]$. Aqui de Tabela 3 , os valores de UMA_2 e B_2 são calculados como, $UMA_2 = (448 + 330 + 132 + 100 + 72 + 223 + 78 + 45) = 1428$, $B_2 = \frac{1}{7}[56^2 + 48^2 + 30^2 + 24^2 + 20^2 + 39^2 + 20^2 + 15^2] = 1351{,}71$. A estatística de teste, T_2 é calculado como $T_2 = (7-1)[1351{,}71 - 7 \times 8(8+1)^2/4] = 17{,}12$. Usando a mesa do F distribuição com um nível de signi fi cância $\bullet = 0{,}01$, verifica-se que $F(1-\bullet),(m-1),(b-1)(m-1) = F_{0{,}99,7,42} \simeq 3{,}12$. Desde a $T_2 > F_{0{,}99,7,42}$, a hipótese nula H_0 é rejeitado e então H_1 é aceito. Portanto, existem alguns algoritmos cujos desempenhos são significativamente diferentes de outras.

Tabela 3
Comparação dos resultados obtidos pela abordagem proposta com algumas outras abordagens da literatura.

Método diferente (nome simbólico)	Problema de teste	eli51	berlin52	st70	Eil76	rat99	kroA100	eil101	lin105	kroA200
RABNET-TSP [42] (AL1)	Média	438,70	8073,97	-	556,10	-	21868,47	654,83	14702,17	30257,53
	SD	3,52	270,14	-	8,03	-	245,76	6,57	328,37	342,98
	Erro (%)	2,98	7,05	-	3,36	-	2,76	4,11	2,25	3,03
RABNET -TSP modificado [43] (AL2)	Média	437,47	7932,50	-	556,33	-	21522,73	648,64	14400,7	30190,27
	SD	4,20	277,25	-	5,30	-	93,34	3,85	44,03	273,38
	Erro (%)	2,69	5,18	-	3,41	-	1,13	3,12	0,15	2,80
SA ACO PSO [44] (AL3)	Média	427,27	7542,00	-	540,20	-	21370,30	635,23	14406,37	29738
	SD	0,45	0,00	-	2,94	-	123,36	3,59	37,28	356,07
	Erro (%)	0,30	0,00	-	0,41	-	0,41	0,99	0,19	1,26
WFA com 2 opções [45] (AL4)	Média	426,65	7542,00	-	541,22	-	**21282,00**	639,87	14379,2500	29654,03
	SD	0,66	0,00	-	0,66	-	0,00	2,88	0,00	151,42
	Erro (%)	0,15	0,00	-	0,60	-	0,00	1,73	0,00	0,97
WFA com 3 opções [45] (AL5)	Média	426,60	7542	-	539,44	-	21282,80	633,50	14459,40	29646,50
	SD	0,52	0,00	-	1,51	-	**0,00**	3,47	1,38	110,91
	Erro (%)	0,14	0,00	-	0,27	-	**0,00**	0,72	0,56	0,95
GA-PSO-ACO [46] (AL6)	Média	431,84	7544,37	694,6	550,16	1275	21305	637,93	14521	31015
	SD	-	-	-	-	-	-	-	-	-
	Erro (%)	1,37	0,03	2,90	2,26	5,28	3,5	1,41	0,98	5,60
PSO-ACO-3opt [14] (AL7)	Média	426,45	7543,20	678,20	538,30	1227,40	21445,10	632,70	14379,15	29646,05
	SD	0,61	2,37	1,47	0,47	1,98	78,24	2,12	**0,48**	114,71
	Erro (%)	0,11	0,02	0,47	0,06	0,28	0,77	0,59	**0,00**	0,95
Método proposto (AL8)	**Média**	427,01	**7543,00**	**675,77**	**538,15**	**1211,50**	21287,19	**630,59**	**14379,10**	**29469,00**
	SD	**0,46**	**0,00**	**0,00**	**0,60**	**0,67**	8.01	**2,37**	1,30	**20,03**
	Erro (%)	**0,07**	**0,00**	**0,18**	**0,02**	**0,04**	0,02	**0,25**	**0,00**	**0,146**

Tabela 4
Classificação do Teste de Friedman.

b	Método (m) classificação(*R)*	RABNET-TSP (AL1) *R (AL* 1 b)	Mo.RABNET-TSP (AL2) *R (AL* 2 b)	SA-ACO-PSO (AL3) *R (AL* 3 b)	WFA com-2-opt (AL4) *R (AL* 4 b)	WFA com-3-opt (AL5) *R (AL* 5 b)	GA-PSO-ACO (AL6) *R (AL* 6 b)	PSO-ACO-3Opt (AL7) *R (AL* 7 b)	Proposto-Método (AL8) *R (AL* 8 b)
1	eli51	8	7	5	3	2	6	1	4
2	berlin52	8	7	3	2	1	6	5	4
3	Eil76	8	7	4	5	3	6	2	1
4	kroA100	8	7	5	1	2	4	6	3
5	eil101	8	7	4	6	3	5	2	1
6	lin105	8	7	4	3	6	5	2	1
7	kroA200	8	6	5	4	3	7	2	1
Classificação média		7,0	6,8	4,2	3,4	2,8	5,5	2,8	2,1
$R_i = \sum_{j=1}^{b} R(ALi_j)$ *fori* = 1, 2, 3 ..., *m* 56			48	30	24	20	39	20	15

7.2. (Post Hoc) comparação emparelhada

Este método é usado para saber se os algoritmos *Todos* e *ALj* são considerados diferentes após a rejeição da hipótese nula com o teste de Friedman. Seguindo esta técnica [47] calcular a diferença absoluta da soma das classificações dos algoritmos *Todos* e *ALj* e declarar *Todos* e *ALj* diferente se

$$|R_{eu} - R_{j}| > t_{1-\bullet/2} \sqrt{\frac{2BA_2 - B_2)}{2(b-1)(m-1)}\left]_1\right.}$$

Onde $t_{1-\bullet}$ é o $1-\frac{\bullet}{2}$ quantil do t - distribuição com $(b-1)(m-1)$ graus de liberdade. Aqui, valor de $t_{1-\bullet}$ para $\bullet = 0{,}01$ e 42 graus de liberdade são ≃ 2.7 e o valor crítico da diferença

ence é 2,7 $\left[2 \times 7(1428 - 1351{,}71)\right]_1 = 13.6$. A tabela a seguir resume a compressão emparelhada; valores sublinhados indicam que os algoritmos são diferentes.

A partir de Tabela 5 , pode-se concluir que, o algoritmo proposto tem não superou o desempenho de todos os outros algoritmos, mas tem desempenho ≃ 60% melhor em comparação com outros algoritmos. Todos os algoritmos resolveram

Tabela 5
Comparação emparelhada do Teste de Friedman.

$\lvert R_{eu} - R_{j}\rvert$	Mo.RABNET-TSP (AL2)	SA-ACO-PSO (AL3)	WFA com-2-opt (AL4)	WFA com-3-opt (AL5)	GA-PSO-ACO (AL6)	PSO-ACO-3-opt (AL7)	Proposto-Método (AL8)
RABNET-TSP (AL1)	8	26	32	36	17	36	41
Mo.RABNET-TSP (AL2)	-	18	24	28	9	28	33
SA-ACO-PSO (AL3)	-	-	6	10	9	10	15
WFA com 2 opções (AL4)	-	-	-	4	15	4	9
WFA com 3 opções (AL5)	-	-	-	-	19	0	5
GA-PSO-ACO (AL6)	-	-	-	-	-	19	24
PSO-ACO-3-opt (AL7)	-	-	-	-	-	-	5

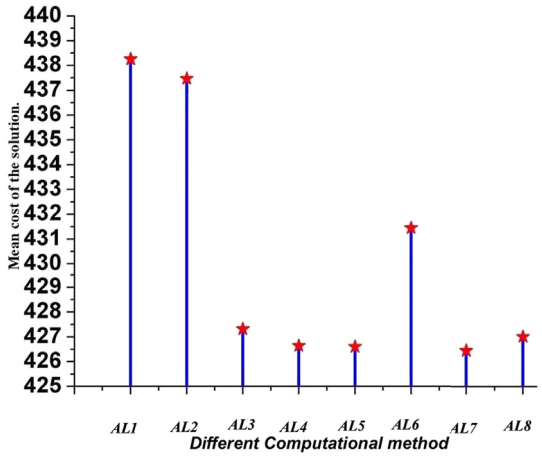

Fig. 3. Comparação dos custos médios das soluções obtidas por diferentes abordagens para as instâncias eil51.

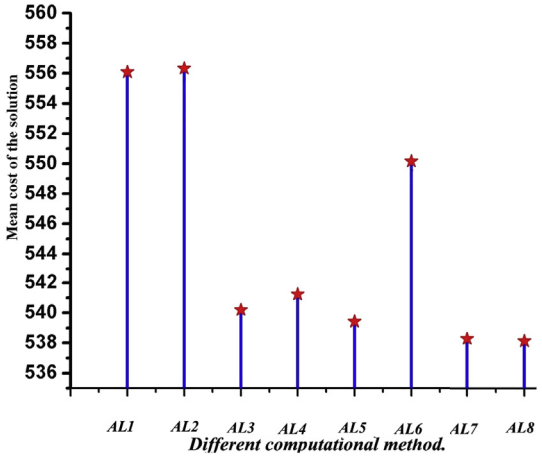

Fig. 6. Comparação dos custos médios das soluções obtidas por diferentes abordagens, por exemplo eil76.

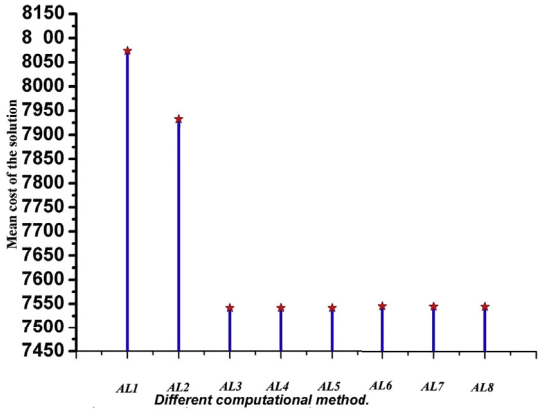

Fig. 4. Comparação dos custos médios das soluções obtidas por diferentes abordagens, por exemplo, berlim52.

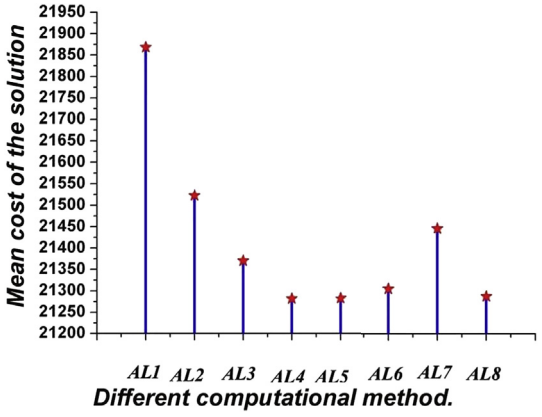

Fig. 7. Comparação dos custos médios das soluções obtidas por diferentes abordagens por exemplo KroA100.

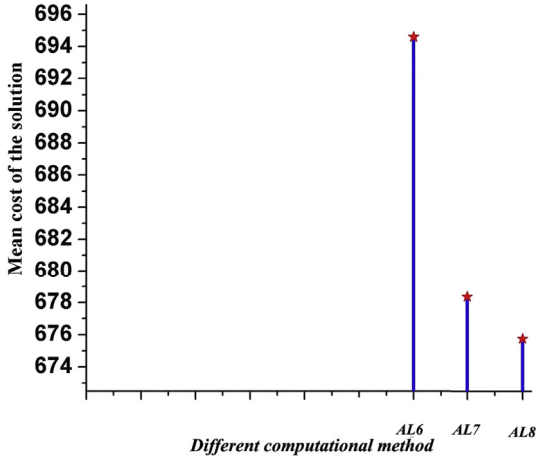

Fig. 5. Comparação dos custos médios das soluções obtidas por diferentes abordagens, por exemplo, st70.

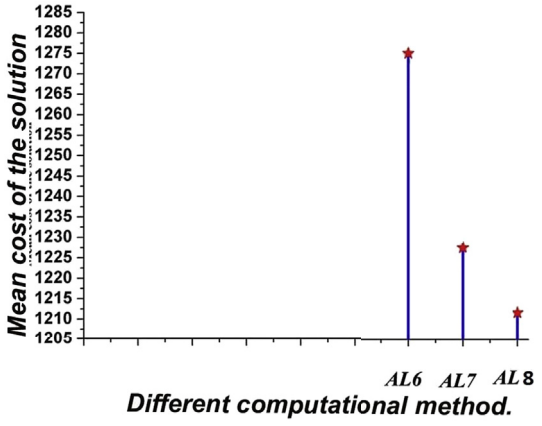

Fig. 8. Comparação dos custos médios das soluções obtidas por diferentes abordagens, por exemplo, rat99.

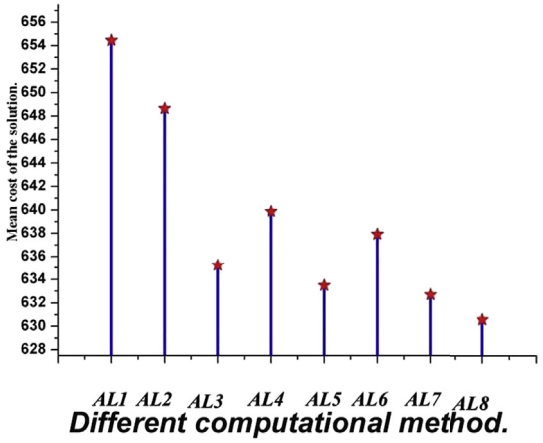

Fig. 9. Comparação dos custos médios das soluções obtidas por diferentes abordagens, por exemplo eil101.

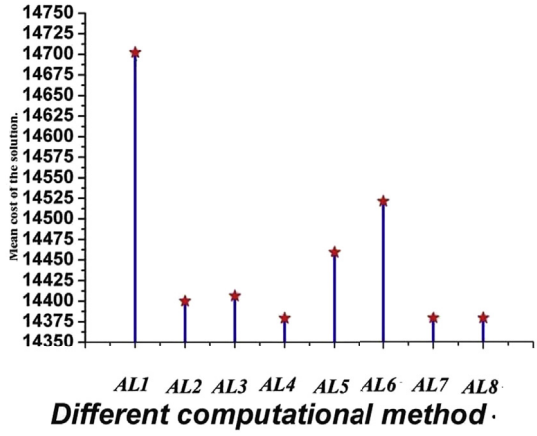

Fig. 10. Comparação dos custos médios das soluções obtidas por diferentes abordagens, por exemplo lin105.

o mesmo problema, mas usando abordagens diferentes. O algoritmo proposto introduz um novo conceito de hibridização de ABC com as características de sequência de troca, operação de troca e 3-opt para resolver TSP. ABC geralmente usado para problemas de otimização contínua; aqui, ele é usado para resolver problemas de otimização discreta.

Os custos médios das soluções de cada problema obtido em diferentes execuções por diferentes algoritmos são plotados em uma figura separada. Figs. 3-11

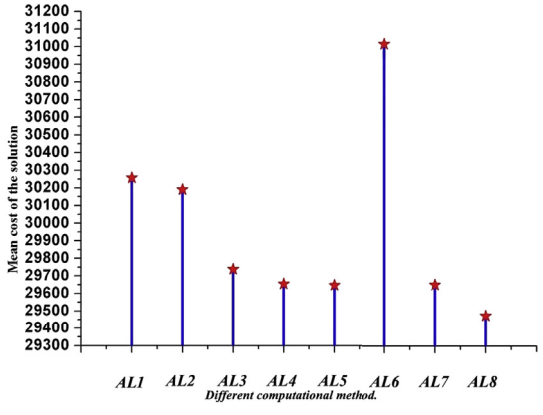

Fig. 11. Comparação dos custos médios das soluções obtidas por diferentes abordagens, por exemplo KroA200.

representam tais plotagens de nove problemas diferentes do TSPLIB. Em cada figura, os símbolos - AL1, AL2, AL3, AL4, AL5, AL6, AL7, AL8 representam nomes simbólicos de métodos diferentes (cf., Tabela 3) usado para resolver o TSP correspondente. A partir dessas figuras, uma comparação dos desempenhos médios de diferentes algoritmos para resolver diferentes problemas de teste de benchmark do TSPLIB pode ser facilmente feita. Pelas figuras fica claro que o desempenho do algoritmo proposto é melhor do que a maioria dos algoritmos existentes e para outros também seu desempenho é adequado.

8. Conclusão

Aqui, pela primeira vez, os recursos de sequência de troca e operação de troca em uma permutação do conjunto de nós de um TSP são usados para definir um conjunto de regras de atualização de solução do algoritmo ABC aplicáveis ao TSP e usá-los como uma nova variante de ABC é apresentado para resolver STSP, bem como ATSP. No método proposto, o número de abelhas observadoras é uma função linear do número de abelhas empregadas. A fase de abelha escoteira de qualquer algoritmo ABC geralmente regenera soluções estagnadas. Aqui, o 3-opt é usado para a possível melhoria de qualquer solução estagnada nesta fase. Não só isso, no final do processo de busca pelo algoritmo, aqui 3-opt é novamente imposto à melhor solução encontrada um número pré-definido de vezes para sua melhoria (se possível). Todos os resultados experimentais implicam que o desempenho da abordagem proposta é adequado em comparação com as outras abordagens existentes na literatura no que diz respeito à precisão, eficiência e consistência. Portanto, pode-se concluir que o ABC pode ser usado para resolver problemas de otimização discreta com sucesso por meio da modificação adequada. O algoritmo pode ser facilmente usado para resolver TSPs em ambiente fuzzy e TSPs sólidos com pequena modificação.

Referências

[1] EL Lawler, JK Lenstra, AHG Rinnooy Kan, DB Shmoys, The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization, John Wiley & Sons, Nova York, 1985 .

[2] S. Lin, BW Kernighan, Um algoritmo heurístico eficaz para o problema do caixeiro viajante, Oper. Res. 21 (2) (1973) 498-516 .

[3] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Trans. Syst. Man Cybern. B Cybern. 26 (1) (1996) 29-41 .

[4] XH Shi, YC Liang, HP Lee, C. Lu, QX Wang, Enxame de partículas algoritmos baseados em otimização para TSP e TSP generalizado, Inf. Processar. Lett. 103 (5) (2007) 169-176 .

[5] J. Majumdar, AK Bhunia, algoritmo genético para o problema do caixeiro viajante assimétrico com tempos de viagem imprecisos, J. Comput. Appl. Matemática. 235 (9) (2011) 3063–3078 .

[6] P. Miliotis, Usando planos de corte para resolver o problema do caixeiro viajante simétrico, North-Holland Publishing Company, Math. Programa. 15 (1978) 177-188 .

[7] GB Dantzig, DR Fulkerson, SM Johnson, Solução do problema do caixeiro viajante em grande escala, J. Oper. Res. Soc. Sou. 2 (4) (1954) 393–410 .

[8] M. Padberg, G. Rinaldi, Otimização de um problema de caixeiro viajante simétrico de 532 cidades por ramo e corte, Oper. Res. Lett. 6 (1) (1987) 1-7 .

[9] MW Petberg, S. Homg, Sobre os problemas simétricos do caixeiro viajante: um estudo computacional, Math. Programa. Estudo 12 (1980) 61-77 .

[10] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno, M. Yagiura, algoritmo de busca local eficaz para problemas de roteamento e programação com restrições gerais da janela de tempo, Transporte. Sci. 39 (2) (2005) 206–232 .

[11] F. Focacci, A. Lodi, M. Milano, Um algoritmo exato híbrido para o TSPTW, INFORMS, J. Comput. (2002) 403-417 .

[12] HD Nguyen, I. Yoshihara, K. Yamamori, M. Yasunaga, Implementação de um GA híbrido eficaz para o problema do caixeiro viajante em grande escala, IEEE Trans. Syst. Man Cybern. B Cybern. 37 (1) (2007) 92-99 .

[13] D. Karaboga, B. Gorkemli, Um algoritmo de colônia de abelhas arti fi cial combinatória para o problema do caixeiro viajante, em: INISTA 2011: Simpósio Internacional de Inovações em Sistemas Inteligentes e Aplicações, 2011. Istambul, Turquia .

[14] M. Mahi, OK Baykan, H. Kodaz, Um novo método híbrido baseado na otimização do enxame de partículas, otimização da colônia de formigas e algoritmos de 3 opções para o problema do caixeiro viajante, Appl. Soft Comput. 30 (2015) 484-490 .

[15] MAH Akhand, S. Akter, SS Rahman, MMH Rahman, Otimização de enxame de partículas com pesquisa parcial para resolver o problema do caixeiro viajante, em: Computer and Communication Engineering (ICCCE), 2012 International Conference on, IEEE, 2012 .

[16] MAH Akhand, Shahina Akter, MA Rashid, Velocity tentative particle swarm optimization to solve TSP, em: 2013 Conferência Internacional sobre Tecnologia de Informação e Comunicação Elétrica (EICT), Publicações da Conferência IEEE, 2014, pp. 1-6 .

[17] XT Geng, ZH Chen, W. Yang, DQ Shi, K. Zhao, Resolvendo o problema do caixeiro viajante com base em um algoritmo de recozimento simulado adaptativo com busca gananciosa, Appl. Soft Comput. 11 (4) (2011) 3680-3689 .
[18] F. Jolai, A. Ghanbari, Integração de técnicas de transformação de dados com redes neurais Hopfield para resolver o problema do caixeiro viajante, Expert Syst. Appl. 37 (2010) 5331–5335 .
[19] M. Dorigo, LM Gambardella, Colônias de formigas para o problema do caixeiro viajante, Biosystems 43 (1997) 73-81 .
[20] B. Bontoux, D. Feillet, Otimização da colônia de formigas para o problema do comprador viajante, Comput. Oper. Res. 35 (2) (2008) 628-637 .
[21] M. Lopez-Ibanez, C. Blum, Beam-ACO para o problema do caixeiro viajante com janelas de tempo, Comput. Oper. Res. 37 (9) (2010) 1570–1583 .
[22] M. Gunduz, MS Kiran, E. Ozceylan, Uma abordagem hierárquica baseada na inteligência de enxame para resolver o problema do caixeiro viajante, Turk. J. Electr. Eng. Comput. Sci. 23 (2015) 103-117 .
[23] I. Khan, MK Maiti, M. Maiti, Otimização de enxame de partículas de coordenação otimização de colônia de formigas e algoritmo K-opt para o problema do caixeiro viajante, em: Conferência Internacional sobre Matemática e Computação, Comunicação em Ciência da Computação e da Informação, vol. 655, Springer, 2017, pp. 103-119 .
[24] L. Huang, K. Wang, C. Zhou, W. Pang, L. Dong, L. Peng, Otimização do enxame de partículas para problemas do caixeiro viajante, Acta Sei. Nat. Univ. Jilinebsis 4 (2003) .
[25] Sumaiya Iqbal, M. Sohel Rahman, Vehicle Routing problems with soft time windows, in: Electrical & Computer Engineering (ICECE), 2012 7th International Conference on, IEEE, 2012 .
[26] QK Pan, M. Fatih Tasgetiren, PN Suganthan, TJ Chua, Um algoritmo de colônia de abelhas arti fi cial discreto para o problema de programação de loja de fluxo de lote, Inf. Sci. 181 (2011) 2455–2468 .
[27] A. Rajasekhar, N. Lynnb, S. Das, PN Suganthan, Computing with the coletivo intelligence of honey Bees - a survey, Swarm Evolut. Comput. 32 (2017) 25–48 .
[28] MS Kiran, H. Hakli, M. Gunduz, H. Uguz, Algoritmo de colônia de abelha artificial com estratégia de busca variável para otimização contínua, Inf. Sci. 300 (2015) 140-157 .
[29] D. Karaboga, B. Akay, A comparative study of arti fi cial bee colony algorithm, Appl. Matemática. Comput. Sci. 214 (2009) 108-132 .
[30] D. Karaboga, B. Basturk, Um algoritmo poderoso e eficiente para otimização de função numérica: algoritmo arti fi cial bee colony (ABC), J. Global Optim. 39 (2007) 459-471 .
[31] A. Singh, An arti fi cial bee colony algoritmo for the leaf-constrained minimum spanning tree problem, Appl. Soft Comput. 9 (2) (março de 2009) 625-631 .
[32] Li-Pei Wong, MYH Low, CS Chong, Otimização de colônias de abelhas com busca local por problema do caixeiro viajante, Int. J. Artif. Intell. Ferramenta. 19 (03) (2010) 305-334 .
[33] Sumaiya Iqbal, M. Kaykobad, M. Sohel Rahman, Resolvendo o problema de roteamento de veículos multi-objetivos com janelas de tempo suaves com a ajuda de abelhas, Swarm Evolut. Comput. 24 (2015) 50-64 .
[34] K. Helsgaun, General k-opt submoves para a heurística TSP de Lin Kernighan, Mathem. Programa. Comput. 1 (2009), 199-163 .
[35] KP Wang, L. Huang, CG Zhou, W. Pang, Otimização do enxame de partículas para o problema do caixeiro viajante, em: International Conference on Machine Learning and Cybernetics (2003), IEEE Xplore, 2004, ISBN: 0-7803-7865-2, pp. 1583-1585 .
[36] Z. Michalewicz, Genetic Algorithms + Data Structure = Evolution Programs, 27. Z. Michalewicz, Springer-Verlag, Berlin Heidelderg, 1992 .
[37] G. Reinelt, TSPLIB - uma biblioteca de problemas do caixeiro viajante, Oper. Res. Soc. Sou. J. Comput. 3 (1991) 376-384 .
[38] A. Khanra, MK Maiti, M. Maiti, Maximização de lucro de TSP através de um algoritmo híbrido, Comput. Ind. Eng. 88 (2015) 229-236 .
[39] D. Karaboga, An Idea Based on Honey Bee Swarm for Numerical Optimization, Technical Report-TR06, Erciyes University, Kayseri, Turquia, 2005 .
[40] G. Sierksma, Hamiltonicty and the 3-Opt Procedure para o problema do caixeiro viajante, Appl. Matemática. 22 (2) (2014) 351-358 .
[41] I. Khan, MK Maiti, um novo algoritmo híbrido para problemas generalizados do caixeiro viajante em ambientes diferentes, Vietnã J. Comput. Sci. (2018), https: // doi. org / 10.1007 / s40595-017-0099-z , Springer.
[42] R. Pasti, LN De Castro, Uma rede neuroimune para resolver o problema do caixeiro viajante, em: International Joint Conference on Neural Networks, 2006. IJCNN'06, IEEE, 2006, pp. 3760–3766 .
[43] TAS Masutti, LN de Castro, Uma rede neural auto-organizada usando idéias do sistema imunológico para resolver o problema do caixeiro viajante, Inf. Sci. 179 (10) (2009) 1454–1468 .
[44] SM Chen, CY Chien, Resolvendo o problema do caixeiro viajante com base no sistema de colônia de formigas de recozimento genético simulado com técnicas de otimização de enxame de partículas, Expert Syst. Appl. 38 (2011) 14439–14450 .
[45] ZA Othman, AI Srour, AR Hamdan, PY Ling, algoritmo de desempenho semelhante ao fluxo de água para TSP, melhorando sua busca local, Int. J. Adv. Comput. Technol. 5 (14) (2013) 126 .
[46] W. Deng, R. Chen, B. He, Y. Liu, L. Yin, J. Guo, Um romance de algoritmo e aplicação de otimização de inteligência de enxame híbrido de dois estágios, Soft Comput. 16 (10) (2012) 1707-1722 .
[47] J. Derrac, S. Garcia, D. Molina, F. Herrera, Um tutorial prático sobre o uso de testes estatísticos não paramétricos como uma metodologia para comparar algoritmos evolucionários e de inteligência de enxame, Swarm Evolut. Comput. 1 (2011) 3-18 .