2019) 1950004 (28 páginas)
World Scienti fi c Publishing Company DOI:
1142 / S0218213019500040

Resolvendo o problema do caixeiro viajante usando o Combinatorial
Algoritmos artificiais de colônia de abelhas

Dervis Karaboga * e Beyza Gorkemli †

Departamento de Engenharia da Computação, Universidade Erciyes
Kayseri, Kayseri 38700 / Melikgazi, Turquia
* *karaboga@erciyes.edu.tr*
† *bgorkemli@erciyes.edu.tr*

Recebido em 9 de março de 2016
Aceito em 11 de fevereiro de 2019
Publicado em 28 de fevereiro de 2019

Arti fi cial bee colony (ABC) é uma abordagem de otimização bastante popular que tem sido usada em muitos campos, com sua forma não apenas padrão, mas também com versões aprimoradas. Neste artigo, novas versões do algoritmo ABC para resolver o TSP são apresentadas e descritas em detalhes. Uma delas é a versão combinatória do ABC padrão, chamada de algoritmo ABC combinatório (CABC). O outro é uma versão aprimorada do algoritmo CABC, chamado algoritmo CABC rápido (qCABC). Para ver a eficiência das novas versões, 15 diferentes benchmarks de TSP são considerados e os resultados gerados são comparados com diferentes métodos de otimização bem conhecidos. Os resultados da simulação mostram que os algoritmos CABC e qCABC demonstram bom desempenho para TSP e também a nova definição em Quick ABC (qABC) melhora o desempenho de convergência do CABC no TSP.

Palavras-chave: Otimização combinatória; TSP; algoritmo de colônia de abelhas arti fi cial; algoritmo combinatório de colônia de abelhas arti fi cial.

Introdução

otimização combinatória tenta encontrar as soluções ótimas para os problemas no espaço discreto. n grande conjunto de problemas práticos pode ser definido como a forma de problemas de imização combinatória, como o problema do caixeiro viajante (TSP), árvore geradora mínima, oblema de roteamento de veículos, problema de atribuição quadrática, árvore de caminho mais rto, problema de balanceamento de linha de montagem, problema de programação, Entre eles, o P é geralmente usado para testar o desempenho de abordagens recentemente desenvolvidas para oblemas de otimização combinatória e relacionados a muitos campos importantes, como logística, odução, transporte e indústrias de semicondutores.

utor correspondente

O TSP é um problema NP difícil sobre como encontrar um caminho hamiltoniano com custo ínimo.[1] De acordo com o princípio básico do TSP, o vendedor parte de alguma cidade, visita das as outras exatamente uma vez e retorna à cidade de partida tentando obter um roteiro chado com custo mínimo. O custo do passeio depende diretamente da duração do passeio. ra calcular a distância entre a cidade *eu* e a cidade seguinte, *(T [i], T [i +* 1]), a distância euclidiana é usada neste estudo e definida pela Eq. (1).

$$d\,(T[i], T[i+1]) = \sqrt{(x_{eu} - x_{i+1})^2 + (y_{eu} - y_{i+1})^2}. \quad (1)$$

ração total do passeio *f* pode ser dada usando as distâncias euclidianas entre as cidades como na . (2)

$$\sum_{i=1}^{n-1} d\,(T[i], T[i+1]) + d\,(T[n], T[1]) \quad (2)$$

nde *n* representa o número total de cidades.

Muitos pesquisadores propuseram resolver o TSP, que é um problema interessante com a fácil definição e solução muito difícil. Em primeiro lugar, alguns métodos tradicionais nsistindo em métodos exatos e heurísticos foram introduzidos para resolver o TSP. Por emplo, planos de corte[2] e ramificar e cortar[3] os métodos são capazes de resolver pequenos Ps de maneira otimizada. Por outro lado, métodos heurísticos como a cadeia de Markov[4] e opt[5] são mais preferíveis para grandes TSPs. Além disso, existem alguns algoritmos baseados n princípios gananciosos que podem ser usados para resolver o TSP, como o algoritmo do zinho mais próximo.[6] No entanto, os métodos tradicionais geralmente resultam em tempos de mputação exponenciais ou resolvem os problemas com qualidades insatisfatórias. A fim de perar essas deficiências, ou seja, devido à grande importância de se obter melhores soluções n um tempo computacional aceitável, diversos algoritmos metaheurísticos foram senvolvidos para o TSP na literatura. Eles podem ser agrupados como algoritmos baseados n solução única e baseados em população. Recozimento simulado (SA)[7] e pesquisa tabu (TS)[8] são goritmos baseados em solução única bem conhecidos que pegam uma solução inicial e ntam melhorá-la produzindo soluções vizinhas através do processo de otimização. Por outro do, o algoritmo genético (GA),[9] sistema de colônia de formigas (ACS),[10] otimização de enxame partículas (PSO),[11] evolução diferencial (DE)[12] e algoritmo arti fi cial bee colony (ABC)[13] são as etaheurísticas baseadas em população mais populares que tentam melhorar um conjunto de luções chamadas de população em paralelo. Alguns desses métodos metaheurísticos foram iginalmente propostos para busca em espaços discretos, como ACS, otimização de colônias abelhas (BCO), algoritmo GA, SA e TS. Dorigo e Gambardella resolveram o TSP usando o ACS.[10] ang e Feng usaram um sistema de formiga máximo-mínimo para resolver o TSP.[14] Um goritmo de otimização de colônia de formigas (ACO) foi proposto para TSP por Gan *et al* ..[15] Na f. 15, para resolver o comportamento de estagnação e o problema de convergência ematura do algoritmo ACO básico no TSP, uma característica de scout foi introduzida. Ilie e dica propuseram uma nova abordagem distribuída para o algoritmo ACO em uma quitetura distribuída.[16] A fim de melhorar a qualidade das soluções da ACO, Puris *et al.* usou na abordagem de dois estágios.[17] Tuba e Jovanovic apresentaram um algoritmo ACO rimorado com uma nova estratégia de correção de feromônio ra TSP.[18] Wong *et al.* algoritmo BCO descrito para TSP[19] e então, a fim de melhorar as luções anteriores geradas pelo modelo BCO, eles integraram a heurística 2-opt no BCO.[20] ém disso, existem muitas versões diferentes de TS,[21,22] SA,[23] GA[24,9] para TSP. Além dessas etaheurísticas, existem algumas outras metaheurísticas que foram originalmente troduzidas para problemas numéricos, em vez de problemas combinatórios. No tanto, tendo inspirado seu sucesso em problemas numéricos, muitos pesquisadores ntaram descrever as variantes combinatórias dessas abordagens de otimização umérica para resolver o TSP, como o PSO.[25,26] e DE.[27-29]

ABC também foi originalmente proposto para resolver problemas numéricos por Karaboga.[13] m algoritmo metaheurístico baseado em inteligência de enxame que modela o comportamento de forrageamento abelhas.[13,30] Desde 2005, o ABC tem sido usado em muitos campos como uma técnica de otimização eficaz. stem vários estudos de levantamento relacionados ao ABC na literatura.[31-35] Alguns dos pesquisadores puseram versões discretas do ABC para o TSP. Li *et al.* desenvolveu um algoritmo ABC discreto que usa o conceito operador de troca para resolver o TSP.[36] Ao combinar a abordagem modificada do vizinho mais próximo e a eração interna aprimorada, Li *et al.* descreveu uma variante do ABC para TSP.[37]

spirado no sucesso da otimização do ABC em problemas numéricos,[38] e um operador de utação bem-sucedido de GA introduzido para TSP,[39] os autores definiram um novo ecanismo de busca de vizinhança e propuseram uma nova versão combinatória do ABC ra TSP e nomearam esse algoritmo como Combinatorial ABC (CABC).[40] Dentro 2012, os autores introduziram uma nova definição para o comportamento de forrageamento das abelhas observadoras e chamaram a nova variante de ABC rápido (qABC).[41,42] Esses estudos mostraram que a nova definição melhora significativamente o desempenho de vergência do ABC padrão em problemas de otimização numérica. Portanto, eles integraram esta nova definição no algoritmo CABC, é uma das versões combinatórias do ABC para TSP.[43] Esta foi a primeira vez que o modelo qABC é aplicado a um problema de nização combinatória. No entanto, esses estudos sobre CABC e qCABC fornecem apenas as informações básicas sobre os algoritmos e estudos experimentais foram realizados apenas em dois problemas de teste: 150 e 200 cidades. Apenas os melhores e os erros médios apresentados como resultados e são comparados com os resultados das variantes GA. Além disso, gráficos de convergência são necidos. Uma vez que esses estudos limitados não são suficientes para examinar o desempenho desses algoritmos, neste artigo, os oritmos CABC e qCABC são descritos de uma maneira mais detalhada e seus desempenhos são examinados profundamente em um junto maior de benchmarks de TSP usando testes estatísticos. Seus resultados não são apenas comparados com os resultados das antes GA, mas também com os resultados de alguns outros algoritmos de otimização de última geração, como ACS e BCO, que bém são algoritmos de otimização baseados em inteligência de enxame. E também alguns estudos de ajuste no limite do parâmetro de trole são realizados para os algoritmos CABC e qCABC. Além disso, algumas análises sobre os tempos de CPU que os algoritmos uerem são fornecidas neste estudo. O resto do artigo está organizado da seguinte forma; na Seç˜ao 2, o algoritmo CABC é apresentado, a Seç˜ao 3 descreve o algoritmo qCABC. Os resultados da simulação são apresentados na Seção 4 e na Seção 5, a conclusão é dada. E bém alguns estudos de ajuste no limite do parâmetro de controle são realizados para os algoritmos CABC e qCABC. Além disso, umas análises sobre os tempos de CPU que os algoritmos requerem são fornecidas neste estudo. O resto do artigo está organizado da uinte forma; na Seç˜ao 2, o algoritmo CABC é apresentado, e na Seç˜ao 3 descreve o algoritmo qCABC. Os resultados da simulação são esentados na Seção 4 e na Seção 5, a conclusão é dada. E também alguns estudos de ajuste no limite do parâmetro de controle são lizados para os algoritmos CABC e qCABC. Além disso, algumas análises sobre os tempos de CPU que os algoritmos requerem são necidas neste estudo. O resto do artigo está organizado da seguinte forma; na Seç˜ao 2, o algoritmo CABC é apresentado, e na Seç˜ao 3 creve o algoritmo qCABC. Os resultados da simulação são apresentados na Seção 4 e na Seção 5, a conclusão é fornecida.

Algoritmo Combinatório Arti fi cial de Colônia de Abelhas CABC

algoritmo CABC é uma variante do ABC que funciona em espaço discreto e foi troduzido na literatura em 2011.[40] No algoritmo CABC, as fases básicas do algoritmo 3C padrão, que também são adequadas para o espaço de pesquisa discreto, são talmente protegidas. Apenas o mecanismo de busca de vizinhança é adaptado para imização combinatória para TSP. É fato conhecido que o mecanismo de busca ~~na~~ zinhança tem um efeito significativo no desempenho de um algoritmo de otimização. Às zes é difícil determinar se um bom ou mau desempenho se deve ao próprio algoritmo u ao mecanismo de busca na vizinhança usado dentro dele. Um dos operadores de utação de sucesso do GA para TSP, que foi proposto por Albayrak e Allahverdi,[39] é usado ra esta adaptação. Este operador de mutação é denominado Greedy Sub Tour Mutation STM). Assim, embora as soluções vizinhas sejam produzidas usando este poderoso ecanismo para aumentar o sucesso geral do processo de otimização proposto, ao esmo tempo também é possível comparar os CABC's *desempenho algorítmico puro* com speito ao GA para problemas de otimização combinatória.

O fluxograma do algoritmo CABC é dado na Fig. 1. Deve ser declarado que as etapas sicas do CABC são semelhantes ao ABC padrão. A adaptação do ABC ao espaço discreto é seada principalmente no mecanismo de produção de abelhas vizinhas, utilizado nas etapas s abelhas empregadas e observadoras para o TSP. As etapas da produção vizinha ecanismo são dados na Fig. 2 para uma solução x_{eu}. Para uma compreensão clara deste ecanismo, os detalhes do operador GTSM podem ser examinados na Ref. 39

Neste mecanismo, T_{eu} refere-se ao passeio original que representa a solução x_{eu} m pesquisas de vizinhança. *aleatória* é usado como um número gerado aleatoriamente r em (0,1). Fontes de alimentos (soluções) são cadeias de cidades que representam soluções áveis. Os parâmetros do operador GSTM são apresentados em mais detalhes na Ref. 39 ses parâmetros podem ser listados como: P_{RC} (probabilidade de reconexão), P_{CP} (probabilidade correção e perturbação), P_{EU} (probabilidade de linearidade), eu_{MIN} (duração mínima do sub ur), eu_{MAX} (duração máxima do sub tour), NL_{MAX} (tamanho da lista de vizinhanças). Ganho de onto *R,* G_R é calculado usando a Eq. (3) da mesma forma que na Ref. 39

$$G_R = d(T[R], T[R-1]) + d(T[NL_R], T[NL_R - 1]) - (d(T[R], T[NL_R]) + d(T[R-1], T[NL_R - 1])) \quad (3)$$

nde $d(T[R], T[R-1])$ representa a distância entre o ponto R e o ponto que visitou pouco tes do R na turnê T.

As etapas detalhadas do algoritmo CABC são fornecidas na Fig. 3.

A otimização do TSP visa minimizar a duração total do circuito fechado. Então, a qualidade cada turnê $caber_{eu}$ é calculado usando a Eq. (4)

$$caber_i = \frac{1}{1 + f_{eu}} \quad (4)$$

nde f_{eu} refere-se ao valor objetivo, em outras palavras, a duração do passeio da solução x

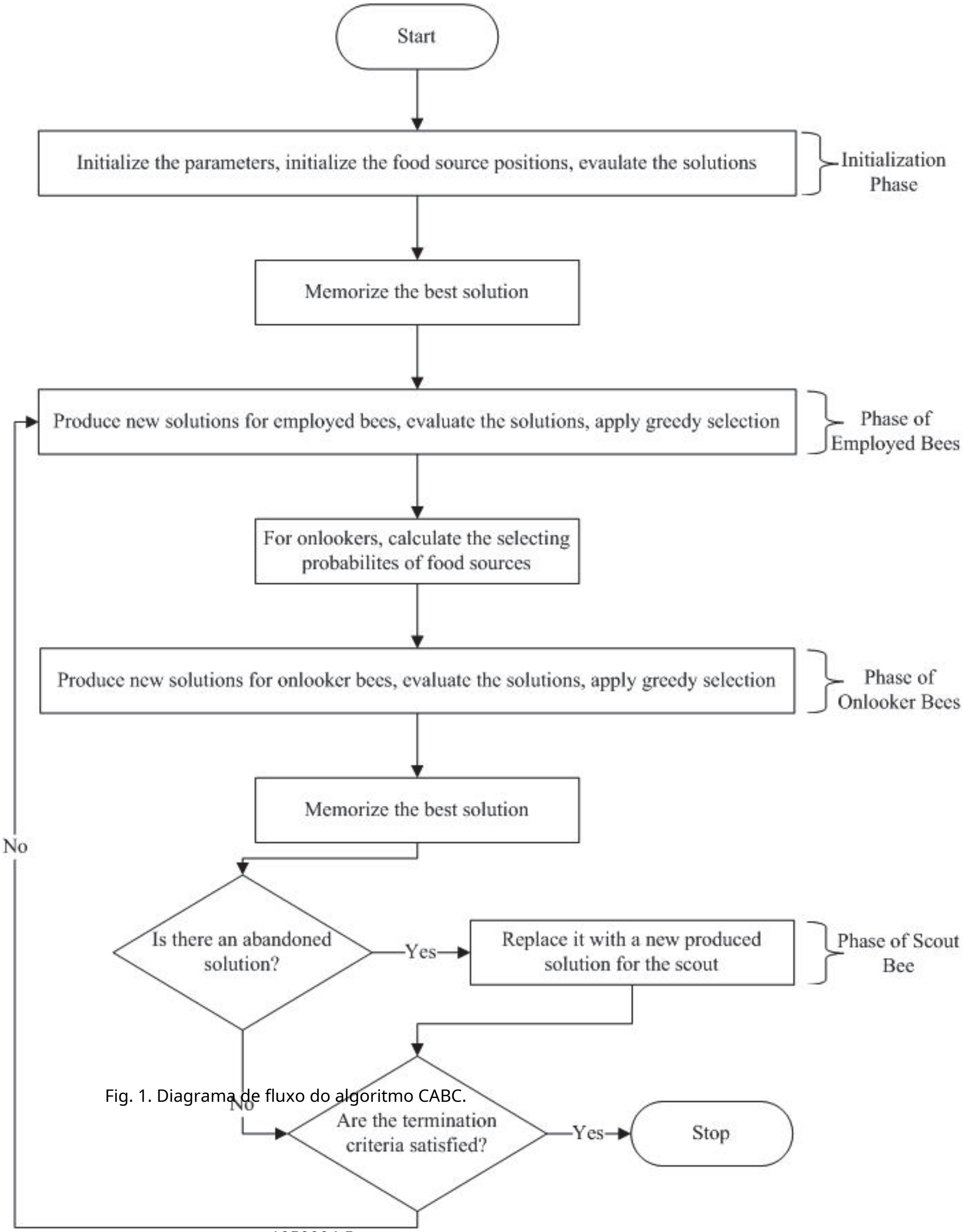


Fig. 1. Diagrama de fluxo do algoritmo CABC.

Selecione uma solução aleatoriamente x_k na população. ($k = i$)
Selecione uma cidade aleatoriamente *j.*
Determine o valor da direção do parâmetro de pesquisa φ aleatoriamente. (φ = - 1, 1)
E se (φ = - 1) **então**
A cidade visitada pouco antes da cidade *j* em turnê T_{eu} é definida como a cidade visitada apenas antes da cidade *j* em turnê T_k.
senão
A cidade visitada imediatamente após a cidade *j* em turnê T_{eu} é definida como a cidade visitada imediatamente após a cidade *j* em turnê T_k.
fim se// Um novo tour fechado T_{eu} é produzido com esta operação.
Como resultado desta nova conexão, haverá um sub tour aberto T^*_{eu}. Este sub a primeira cidade do passeio é atribuída como R_1 e a última cidade é atribuída como R_2
E se (*aleatória* $\leq P_{RC})$ **então**
Adicionar T^*_{eu} para T_{eu} para que uma extensão mínima seja gerada.
senão
E se (*aleatória* $\leq P_{CP})$ **então**
Comece da posição de R_1 dentro T_{eu} e adicione cada cidade de T^*_{eu} para o T_{eu} de rolando ou misturando-se com P_{eu} probabilidade. (Se *aleatória* $\leq P_{eu}$ em seguida, adicione a cidade misturando, do contrário adicione a cidade rolando)
senão
Selecione aleatoriamente um vizinho das listas de vizinhos para os pontos R_1 e R_2 (NL_{R1} e NL_{R2}). (Para garantir a inversão, esses pontos selecionados não devem ser as cidades imediatamente anteriores de R_1 e R_2 em turnê T_{eu}.)
Inverta os pontos NL_{R1} ou NL_{R2} que fornece ganho máximo organizando esses pontos para serem os vizinhos (cidades imediatamente anteriores) dos pontos R_1 ou R_2
fim se
fim se

Fig. 2. Mecanismo de produção vizinho.

Os valores de probabilidade de ser selecionado por uma abelha observadora são formulados com a Eq. (5) para passeios.

$$p_i = \frac{0.9 \times caber_i}{em\ forma_{melhor}} + 0{,}1. \tag{5}$$

O valor limite, *eu,* é calculado pela Eq. (6)

$$l = \frac{cs \times D}{eu} \tag{6}$$

Fase de inicialização:

Defina os valores iniciais dos parâmetros de controle: tamanho da colônia *cs,* número máximo de iterações *MaxNumber.*

Inicialize as posições das fontes de alimentos (passeios iniciais) x_{eu}. $i = 1, 2, \ldots, SN$.

Avalie a duração do passeio.

Memorize o melhor passeio.

$c = 0$

repetir

Fase de abelhas empregadas: para cada abelha empregada;

Gerar uma nova solução candidata (tour) υ_{eu} na vizinhança de x_{eu} (Fig. 2) e avalie este passeio.

Selecione o melhor entre x_{eu} e υ_{eu}.

Usando seus valores de aptidão calculados pela Eq. (4), calcule os valores de probabilidade p_{eu} para os passeios x_{eu} com Eq. (5).

Fase das abelhas espectadoras: para cada abelha observadora;

Dependendo p_{eu} valores, selecione um tour x_{eu}.

Gerar uma nova solução candidata (tour) υ_{eu} na vizinhança de x_{eu} (Fig. 2) e avalie esta solução.

Selecione o melhor entre x_{eu} e υ_{eu}.

Memorize o melhor passeio realizado até o momento.

Fase de abelha batedora: Se houver um roteiro abandonado (o roteiro que não pôde ser melhorado por meio de um número predeterminado de tentativas. Esse número é chamado de "limite"), substitua-o por um novo roteiro para o batedor (esse roteiro será gerado usando a mesma estratégia como na fase de inicialização).

$c = c + 1$

até ($c = MaxNumber$)

Fig. 3. Etapas detalhadas do algoritmo CABC.

nde *cs* representa o tamanho da colônia e *D* é a dimensão do problema ($D = n$). *eu* é um eficiente de divisão que tem um papel importante ao determinar o valor do parâmetro nite.

Algoritmo de Colônia de Abelhas Arti fi cial Combinatório Rápido qCABC

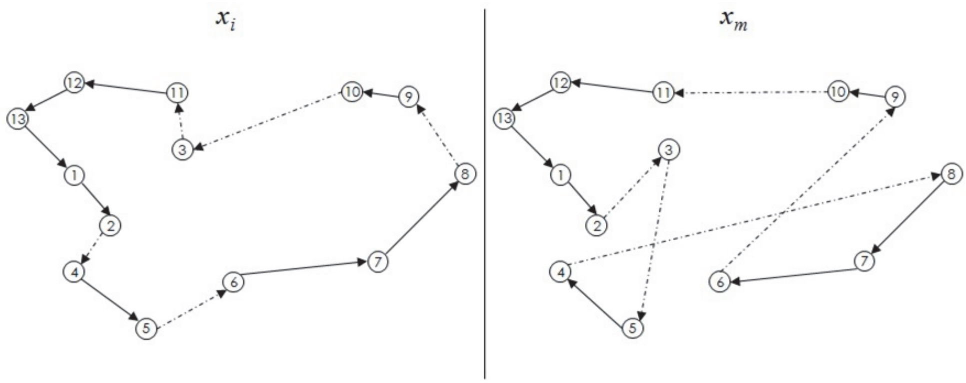
mparando com o algoritmo ABC padrão, o qABC simula mais de perto o comportamento de rrageadoras de abelhas reais.[41,42] Em colônias de abelhas melíferas reais, as abelhas empregadas e abelhas observadoras usam diferentes formas de busca enquanto determinam suas fontes de mento. As abelhas empregadas exploram as fontes de alimento que já visitaram antes. Contudo, comportamento de forrageamento das abelhas observadoras não se baseia em seus óprios conhecimentos sobre as fontes de alimento. Um espectador decide uma região de ntes de alimento observando as danças das abelhas empregadas. Quando ela chega à região colhida, onde não visitou e viu antes, ela examina visualmente as fontes de cima e escolhe a ais adequada. Ou seja, antes de tomar uma decisão sobre uma fonte de alimento, a abelha servadora avalia as informações de fontes vizinhas que são semelhantes em termos de sição, já que ela visita aquela região pela primeira vez. No entanto, no algoritmo ABC básico, na vez que se assume que tanto as abelhas empregadas quanto as observadoras determinam as fontes de alimento da mesma maneira, a mesma fórmula é definida e usada para ambas forrageadoras. Para o comportamento de forrageamento das abelhas observadoras, uma va definição foi introduzida e descreveu uma nova variante do ABC.[41,42] e usado para resolver oblemas de otimização numérica. Na literatura, alguns pesquisadores propuseram superar a aqueza do desempenho de convergência do ABC, introduzindo diferentes estratégias como odificação, hibridização, etc.[31] Usando a ideia qABC, a capacidade de busca local do ABC é elhorada, uma vez que mais testes são realizados pelos espectadores para melhorar suas luções, que são as soluções mais promissoras. Detalhes do algoritmo qABC proposto para oblemas numéricos podem ser encontrados nas Refs. 41 e 42. É muito claro que, se uma edida de similaridade apropriada for definida para as soluções para o TSP, essa idéia útil pode r facilmente incorporada no CABC proposto para problemas combinatórios também.

A fim de usar a definição em qABC para as abelhas observadoras no algoritmo CABC, na medida de similaridade é definida para as soluções de TSP na Ref. 43. Essa nova riante é chamada de ABC-qCABC combinatória rápida. No qABC, a distância euclidiana i proposta para determinar a diferença entre duas soluções de otimização numérica. sto que se pretende obter a combinação ótima das cidades a serem visitadas, a milaridade das soluções pode ser definida com base nas mesmas conexões das cidades os roteiros fechados no TSP. Portanto, a Expressão 7 e a Eq. (8) pode ser usado para terminar a distância entre duas soluções $_u$ e $x_{m,}$

$$dv_{Eu\ estou(}\ j) = \begin{cases} 0, \ E\ se\ T_{eu[}\ j+1] = T_{m\,[}\ j+1]\ ou\ T_{eu[}\ j+1] = T_{m\,[}\ j-1]) \\ 1, \ senão \end{cases} \tag{7}$$

nde $T_{eu[}\ j+1]$ e $T_{m\,[}\ j+1]$ referem-se às cidades visitadas imediatamente após a cidade j n turnê T_{eu} e $T_{m,}$ respectivamente. De forma similar, $T_{m\,[}\ j-1]$ refere-se à cidade que visitou pouco tes da cidade j em turnê T_m que está relacionado com a solução x_m e $dv_{Eu\ estou}$ presenta um vetor de distância entre dois passeios fechados x_{eu} e $x_{m.}$ Conforme explicado ima, a definição de similaridade é baseada nas mesmas conexões das cidades no sseios fechados. Para calcular a distância entre duas soluções, $d\ (i,\ m),$ a seguinte quação é usada.

$$d\ (i,\ m) = \sum_{j=1}^{n} dv_{Eu\ estou(}\ j). \tag{8}$$



j	1	2	3	4	5	6	7	8	9	10	11	12	13
$dv_{im}(j)$	0	1	1	0	1	0	0	1	0	1	0	0	0

$d(i,m)=5$

Fig. 4. Um exemplo de medida de similaridade entre dois passeios.

Um exemplo simples para a medida de similaridade do qCABC é dado na Fig. 4. Neste emplo, um TSP de 13 cidades é usado. Usando a expressão 7, vetor de distância $dv_{Eu\ estou}$ produzido para duas soluções diferentes deste problema, x_{eu} e x_{m}. Ao obter este vetor, verdade, verifica-se se existe ligação entre as mesmas cidades. as soluções, x_{eu} e x_{m}. Se houver, a célula relacionada do vetor é definida com o valor “0”, so contrário, o valor “1” é atribuído a esta célula. Depois de produzir este tor de distância, a distância total $d\,(i, m)$ é calculado usando a Eq. (8).

A vizinhança de x_{eu} é descrito dependendo do coeficiente do raio da vizinhança r e a stância média calculada para x_{eu}, md_{eu}. md_{eu} é a distância média entre x_{eu} e o resto das luções na população e definidas pela Eq. (9).

$$md_i = \frac{\sum_{m=1}^{SN} d\,(i, m)}{SN - 1} \tag{9}$$

nde, SN representa o número de soluções na população.

Finalmente, a seguinte regra é usada para determinar as soluções vizinhas de x_{eu}.

$$\begin{aligned} &E\ se\ d\,(i, m) \leq r \times md_{eu}\ então\ x_m\ é\ um\ vizinho\ de\ x_{eu}, \\ &senão\ não \end{aligned} \tag{10}$$

As etapas detalhadas do algoritmo qCABC são fornecidas na Fig. 5.

Observe que apenas a diferença entre as variantes CABC e qCABC é com a fase de abelhas servadoras. Embora as abelhas empregadas e observadoras determinem seus novos passeios da esma maneira no CABC, eles o fazem de forma diferente no qCABC.

Fase de inicialização:

Defina os valores iniciais dos parâmetros de controle: tamanho da colônia *cs,* número máximo de iterações *MaxNumber.*

Inicialize as posições das fontes de alimentos (passeios iniciais) x_{eu}. $i = 1, 2, . . . , SN$.

Avalie a duração do passeio.

Memorize o melhor passeio.

$c = 0$

repetir

Fase de abelhas empregadas: para cada abelha empregada;

Gerar uma nova solução candidata (tour) v_{eu} na vizinhança de x_{eu} (Fig. 2) e avalie este passeio.

Selecione o melhor entre x_{eu} e v_{eu}.

Usando seus valores de aptidão calculados pela Eq. (4), calcule os valores de probabilidade p_{eu} para os passeios x_{eu} com Eq. (5).

Fase das abelhas espectadoras: para cada abelha observadora;

Dependendo p_{eu} valores, selecione um tour x_{eu}.

Determine o melhor passeio $x^{melhor}_{N_{eu}}$ entre os vizinhos determinados por Expressão 10 do passeio x_{eu} e ele mesmo.

Gerar uma nova solução candidata (tour) $v^{melhor}_{N_{eu}}$ na vizinhança de $x^{melhor}_{N_{eu}}$ (Fig. 2) e avalie esta solução.

Selecione o melhor entre $x^{melhor}_{N_{eu}}$ e $v^{melhor}_{N_{eu}}$.

Memorize o melhor passeio realizado até o momento.

Fase de abelha batedora: Se houver um roteiro abandonado (o roteiro que não pôde ser melhorado por meio de um número predeterminado de tentativas. Esse número é chamado de "limite"), substitua-o por um novo roteiro para o batedor (esse roteiro será gerado usando a mesma estratégia como na fase de inicialização).

$c = c + 1$

até ($c = MaxNumber$)

Fig. 5. Etapas detalhadas do algoritmo qCABC.

Estudo Experimental e Resultados da Simulação

este estudo, primeiro um conjunto de experimentos é conduzido sobre o valor do râmetro limite para os algoritmos CABC e qCABC, uma vez que o parâmetro limite é um râmetro de controle significativo do ABC.[38] E, alguns estudos de comparação são alizados entre dez métodos de otimização diferentes. Esses algoritmos são CABC, CABC e oito variantes diferentes de GA com diferentes operadores de mutação xchange Mutation (EXC), Displacement Mutation (DISP), Inversion Mutation (INV), sertion Mutation (INS), Simple Inversion Mutation (SIM), Scramble utation (SCM), Greedy Swap Mutation (GSM) e GTSM). Nestes estudos de comparação, resultados obtidos por GAs são retirados do estudo de Ref. 39 que fornece o operador TSM que é adaptado ao mecanismo de busca de vizinhança do ABC no CABC e no CABC. Portanto, neste estudo, os mesmos problemas de teste são considerados na Ref. . Além disso, os algoritmos CABC e qCABC são executados para o problema FL1577 a m de ver o desempenho dos algoritmos baseados em ABC em um problema com muito ais cidades do que outros 14 benchmarks com vários números de cidades. Esses oblemas de teste são benchmarks de TSP do tipo simétrico. Os problemas de teste, os úmeros das cidades dos problemas e os comprimentos de passeio ideais são fornecidos Tabela 1. Eles também podem ser encontrados no TSPLIB: tp://www.iwr.uniheidelberg.de/groups/comopt/software/TSPLIB95/. Em nosso estudo,

Tabela 1. Comprimentos de passeio ideais dos benchmarks TSP.

Problema	Número de cidades	Comprimento ideal do passeio
Berlin52	52	7542
KroA100	100	21282
Pr144	144	58537
Ch150	150	6528
KroB150	150	26130
Pr152	152	73682
Rat195	195	2323
D198	198	15780
KroA200	200	29368
Ts225	225	126643
Pr226	226	80369
Pr299	299	48191
Lin318	318	42029
Pcb442	442	50778
Fl1577	1577	22249

Para todos os experimentos, os algoritmos foram executados dez vezes com sementes aleatórias mo na Ref. 39. Para uma comparação justa, a heurística de construção do tour vizinho mais próximo usada para produzir as soluções iniciais e os mesmos valores de parâmetro são tomados como na f. 39. Considerando os mecanismos de busca dos algoritmos CABC e qCABC, enas o parâmetro eu_{MAX} está definido como $n/2$, não $Int(\sqrt{(n)})$ como na Ref. 39. As configurações dos râmetros dos algoritmos CABC e qCABC são fornecidas na Tabela 2.

Em primeiro lugar, uma vez que o limite do parâmetro tem um efeito significativo no sempenho do algoritmo ABC,[38] realizamos alguns experimentos sobre o parâmetro limite s algoritmos CABC e qCABC para os benchmarks de TSP apresentados na Tabela 1. Nesses perimentos, para cada problema, procuramos os valores apropriados do limite *eu* alterando coeficiente de divisão *EU.* O valor médio, o desvio padrão e o melhor valor dos comprimentos percurso obtidos fi nalmente dessas dez execuções são dados nas Tabelas 3-6 para o goritmo CABC e nas Tabelas 7-10 para o algoritmo qCABC. Quando

Tabela 2. Configurações de parâmetro.

Parâmetro	CABC	qCABC
cs	40	40
Número máximo de avaliações	800 000	800 000
P_{RC}	0,5	0,5
P_{PC}	0,8	0,8
P_{eu}	0,2	0,2
eu_{MIN}	2	2
eu_{MAX}	$n/2$	$n/2$
NL_{MAX}	5	5
r	-	1
eu	1, 2, 3, 4	1, 2, 3, 4

tabelas 3–10 são examinadas e pode ser visto a partir dos resultados que o parâmetro limite m um efeito importante no desempenho dos algoritmos ABC. Os melhores valores das lunas da tabela são escritos em negrito e os valores ideais são escritos em itálico.

Se compararmos o valor objetivo médio de dez execuções independentes, o algoritmo BC dá melhores resultados para Pr144 quando $L = 1$. Para problemas KroA100, Pr152, D198, oA200 e Pcb442, esta abordagem obtém melhores resultados quando $L = 2$. Problemas de 150, Krob150, Rat195 e Lin 318 resolvidos com menos erros quando $L =$ Os problemas 3 e 226, Pr299, Fl1577 são resolvidos com menos erros quando *eu* está definido como = 4. O algoritmo CABC atinge o comprimento ideal do passeio e o valor do desvio padrão é zero para problema Berlin52 para todos os valores-limite experimentados e para Ts225 quando = 4. Os comprimentos de tour ideais dos problemas KroA100, Ts225 e Pr152 também são encontrados pelo BC para quatro diferentes *eu* valores. Além disso, quando o $L = 3$, o comprimento ideal do tour de Pr144 mbém é alcançado.

Quando os resultados de qCABC são avaliados, considerando o valor objetivo médio de dez ecuções independentes, pode-se observar que este algoritmo obtém melhores resultados ra Rat195, KroA200 e Lin318 se *eu* está definido como $L = 1$. O problema Pr144 é resolvido m menos erros quando $L = 2$. Quando $L = 3$, o algoritmo produz melhores resultados para oA100, Krob150, Pr152, Pcb442. No entanto, quando o valor limite do algoritmo qCABC é ustado com $L =$ Problemas 4, Ch150, D198, Pr226, Pr299 e Fl1577 resolvidos com menos erros. melhante ao algoritmo CABC, o algoritmo qCABC também atinge o comprimento de percurso eal e o valor do desvio padrão é zero para o problema Berlin52 para todos os valores limite perimentados e para Ts225 quando $L = 2$ e $L = 4$. Comprimentos de turnos ótimos de oblemas KroA100, Pr152 e Ts225 também são obtidos por qCABC para quatro diferentes *eu* valores. comprimento ideal do passeio é encontrado para Pr144 quando $L = 3$ por qCABC.

Considerando os valores médios da função objetivo, o mais adequado *eu* os valores o determinados para cada problema. E usando esses resultados, os desempenhos de BC, qCABC e oito variantes diferentes de GA são comparados nas Tabelas 11 e 12. esta comparação, os melhores valores de erro percentual e médio são usados

Tabela 3. Limite de experimentos do algoritmo CABC.

	Berlin52			KroA100			Pr144			Ch150		
eu	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor
1	***7542***	***0***	***7542***	21297,4	29,4184	***21282***	**58631**	**35,3072**	58590	6565	14,7037	6549
2	***7542***	***0***	***7542***	**21291**	**8,5790**	***21282***	58641,6	41,0638	58590	6561,2	**7,7175**	6549
3	***7542***	***0***	***7542***	21291,4	11,4123	***21282***	58649,4	66.6126	***58537***	**6556,6**	8,8566	**6542**
4	***7542***	***0***	***7542***	21292,6	13,6250	***21282***	58632,9	38,2582	58590	6566,4	8,9353	6549

Tabela 4. Continuação da Tabela 3.

	KroB150			Pr152			Rat195			D198		
eu	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor
1	26350,1	52,2904	26265	73855,7	142,6464	***73682***	2358,2	**3,3704**	2353	15866,5	28,3275	**15825**
2	26338	59.6406	**26186**	**73792**	142,2385	***73682***	2355,4	6,0033	2343	**15854,4**	25.8503	**15825**
3	**26311,6**	49.6955	26211	73797,1	109.6654	***73682***	**2350**	7,2250	**2337**	15862,4	**18,6022**	15838
4	26334,6	**44,2384**	26217	73842,2	**93.5252**	***73682***	2352,3	4,7550	2345	15863,9	24,3740	15826

Tabela 5. Tabela 3 continuação.

	KroA200			Ts225			Pr226			Pr299		
eu	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor
1	29551,5	35,3674	29487	126706,8	127,6	***126643***	81152,3	144,8199	80929	48775	**53,9870**	48693
2	**29517,5**	46,5623	**29467**	126751,7	169,4007	***126643***	81133	123,4480	80872	48743,1	98,0617	48640
3	29545,2	**34,6087**	29503	126706,8	127,6	***126643***	81005,9	**89,3705**	80908	48657,2	138,3523	**48470**
4	29532,6	37,7603	29503	***126643***	***0***	***126643***	**80999,6**	98,1745	**80860**	**48649,2**	86.0451	48512

Tabela 6. Tabela 5 continuação.

	Lin318			Pcb442			Fl1577		
eu	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor
1	43155	123,1073	42947	51607,3	156,9159	**51285**	22852	105,5036	22737
2	43037,2	186.2089	**42756**	**51539,1**	108,3138	51309	22813,7	92,8031	**22665**
3	**43018,1**	127,4523	42810	51555,3	**76.0369**	51424	22814,3	80.5122	22702
4	43126,7	**118,0432**	42898	51556,3	89,3611	51389	**22791,1**	**45,7918**	22703

Tabela 7. Experimentos de limite do algoritmo qCABC.

	Berlin52			KroA100			Pr144			Ch150		
eu	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor
1	***7542***	***0***	***7542***	21291,4	10.0817	***21282***	58656,9	65,7076	58590	6566,3	9,9	6549
2	***7542***	***0***	***7542***	21288,9	10,6626	***21282***	**58624,2**	**39,7437**	58590	6563,6	7,2966	6553
3	***7542***	***0***	***7542***	**21284,4**	**4,8**	***21282***	58626,1	49,1558	***58537***	6563,3	9,9995	**6544**
4	***7542***	***0***	***7542***	21289	9,2520	***21282***	58632,6	41,4782	58570	**6561,9**	**4,9285**	6549

Tabela 8. Continuação da Tabela 7.

	KroB150			Pr152			Rat195			D198		
eu	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor
1	26384,2	72.8146	26257	73872,3	147,1292	***73682***	**2350,4**	5,3889	**2340**	15872,6	21.9691	**15812**
2	26344	43,9500	26251	73903,5	116,2327	***73682***	2353,2	**5.0160**	2347	15875,7	**20.6109**	15845
3	**26317,1**	68,4872	**26193**	**73843,2**	**107,3665**	***73682***	2353,4	6,5146	**2340**	15862,1	31,7945	15819
4	26358,3	**33,0395**	26284	73850,3	116,7810	***73682***	2350,5	6,8739	**2340**	**15856,1**	24,8775	15820

Tabela 9. Continuação da Tabela 7.

	KroA200			Ts225			Pr226			Pr299		
eu	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor
1	**29508,4**	43,3917	**29420**	126674,9	95,7	***126643***	81119,6	160.5143	80829	48754,6	134,6820	48524
2	29526,6	43,2856	29461	***126643***	***0***	***126643***	81057,7	118,4399	80876	48723	**114,1622**	48513
3	29512,5	**28,4824**	29477	126651,3	24,9	***126643***	81108,3	**57.0159**	81040	48690,7	154,7043	48452
4	29534,3	35.5276	29477	***126643***	***0***	***126643***	**81051,6**	131,8523	**80804**	**48617,2**	133,5049	**48347**

1950004-15

Tabela 10. Tabela 7 continuação.

	Lin318			Pcb442			Fl1577		
eu	Mau	Std	Melhor	Mau	Std	Melhor	Mau	Std	Melhor
1	**42986,4**	146,4494	42809	51537	116,2188	51387	22841,3	95,4694	22707
2	43037,1	110.8958	42829	51542,3	**71,0972**	51403	22793,7	83,6935	**22677**
3	43069,5	129,1551	42837	**51524,2**	151,7859	**51240**	22758,7	**40,7015**	22688
4	43052,8	**99,2510**	**42774**	51543,8	84,6638	51419	**22803,9**	50,4251	22742

Tabela 11. Comparação de desempenho de algoritmos.

todo		BERLIN52	KROA100	PR144	CH150	KROB150	PR152	RAT195
C	Best Err. (%)	**0**	0,1692	0,2426	1.9761	2,9277	2.0358	4,477
	Ave. Err. (%)	2.0353	2,8714	0,533	2,8232	5.0919	3,99	5,9836
P	Best Err. (%)	0,0663	4.2947	1,4845	3.4161	7,8263	4,7216	5.0366
	Ave. Err. (%)	1.424	8.0552	1,7628	3,9859	10,4635	5,5932	6,9393
/	Best Err. (%)	**0**	2.6125	1.0284	2,9412	7,1183	3.1202	3,7021
	Ave. Err. (%)	1,1854	4,7965	1,9755	3,6596	9.0521	5.0318	6,3108
S	Best Err. (%)	**0**	0,4652	0,1879	0,6434	1.8178	1,3938	4,2617
	Ave. Err. (%)	0,1525	3.0726	1,3802	2,3943	4.5488	2,8468	5,7426
I	Best Err. (%)	**0**	0,1692	0,2255	0,8425	2.8397	1.5282	2,3676
	Ave. Err. (%)	0,6099	1,3114	1,1717	1.5809	3,8596	2,7976	4.1886
M	Best Err. (%)	**0**	0,8176	0,1589	0,9344	3,2836	1,5146	4.6492
	Ave. Err. (%)	**0**	3,6966	1,3016	2,1078	7.0609	2.694	5,9923
M	Best Err. (%)	**0**	0,3007	0,2426	1,6544	3.452	2,2977	4,3048
	Ave. Err. (%)	0,9931	3,698	0,6891	2,4908	4,9541	2,9112	6,5476
TM	Best Err. (%)	**0**	**0**	**0**	0,4596	0,9644	0,7695	**0,6027**
	Ave. Err. (%)	**0**	1,1836	1.0809	0,6357	1,7616	1.6202	1,8425
3C	Best Err. (%)	**0**	**0**	0,0905	**0,2145**	0,3100	**0**	**0,6027**
	Ave. Err. (%)	**0**	0,0423	0,1606	**0,4381**	**0,6950**	**0,1493**	**1,1623**
ABC	Best Err. (%)	**0**	**0**	0,0905	0,3217	**0,2411**	**0**	0,7318
	Ave. Err. (%)	**0**	**0,0113**	**0,1490**	0,5193	0,7160	0,2188	1,1795

fim de considerar os resultados apresentados na Ref. 39. Os erros percentuais são lculados usando a Eq. (11)

$$\text{Erro (\%)} = \frac{eu_F - eu_O \times 100}{eu_O} \qquad (11)$$

nde eu_F é a duração do passeio encontrada pelo algoritmo e eu_O é o comprimento de passeio ideal ra o problema.

Nas Tabelas 11 e 12, podemos ver que os resultados dos algoritmos baseados em ABC são ralmente próximos uns dos outros, mas muito melhores do que outras variantes do GA. Apenas ra problemas Lin318, Rat195 e Pr144, o GSTM fornece melhores resultados em termos da melhor dez execuções independentes. Observe que, quando $L = 3$, tanto CABC como qCABC atingem 0% de ro para Pr144, no entanto, nesta comparação, são escolhidos os valores limite que fornecem elhores valores médios de dez execuções.

A fim de mostrar a robustez do ABCs, nas Tabelas 13 e 14, os resultados do CABC e CABC também são comparados aos resultados do GA quando os valores limites de CABC qCABC são calculados definindo o valor de *eu* igual para todos os problemas ($L = 2$) que a equação original definida na literatura para ABC,[44] e também usado para qABC.[42] A bela mostra que, mesmo que não seja realizado um estudo de ajuste de parâmetros ra cada problema, os algoritmos CABC e qCABC apresentam melhor desempenho nos oblemas considerados e as soluções geralmente são melhores do que as encontradas las variantes do GA.

Ambas as tabelas de comparação mostram que os algoritmos GSTM, CABC e qCABC peram os outros algoritmos claramente. Considerando os erros percentuais médios

Tabela 12. Tabela 11 continuação.

Método		D198	KROA200	TS225	PR226	PR299	LIN318	PCB442	FL1577
EXC	Best Err. (%)	1.9709	2,5061	3.0424	4,9186	10,2198	10,1026	7,3437	-
	Ave. Err. (%)	4,9728	4,9877	3,8174	7,2295	15,1775	11,7195	10,4774	-
DISP	Best Err. (%)	3.891	6,7284	3.3693	4,1322	12,2616	10,9996	11,3435	-
	Ave. Err. (%)	5,9835	8,8276	3,8916	7,993	16,849	12,9013	12,3053	-
INV	Best Err. (%)	2.6743	6,2245	3,1640	8,1512	10,9543	9,541	10,5124	-
	Ave. Err. (%)	4,9823	8,3594	3,7355	8,8249	15,706	13,0386	11.6322	-
INS	Best Err. (%)	2.0279	2,1554	3.1751	3,8423	3,8804	6,3099	8,9685	-
	Ave. Err. (%)	3,41	4,2819	3,8767	6,484	11,4187	10,3935	11,0585	-
SIM	Best Err. (%)	1.6984	1,5766	1,6930	2,5769	6,439	6,3932	6,3728	-
	Ave. Err. (%)	2.5241	3.2457	2,5818	4.1795	9,1264	8,714	8,124	-
SCM	Best Err. (%)	3,1179	3,9873	2,7945	3,7875	10,8776	9,1651	7.8597	-
	Ave. Err. (%)	4.5818	5,4502	3,8029	5.0694	13,8686	10,4456	10,6251	-
GSM	Best Err. (%)	3.0482	4,4675	3.0424	5.0517	8,8191	9,4126	5,7643	-
	Ave. Err. (%)	4,2421	5,7018	4,4254	6,4485	13,6295	11,0036	8.0389	-
GSTM	Best Err. (%)	0,3866	0,8683	0,2527	0,7242	1,2326	**0,9827**	2.0501	-
	Ave. Err. (%)	1,2193	1,5432	0,4994	1.5287	2,9169	3,3099	2.7758	-
CABC	Best Err. (%)	0,2852	0,3371	**0**	0,6109	0,6661	1.8582	1.0457	**2.0405**
	Ave. Err. (%)	**0,4715**	0,5090	**0**	**0,7846**	0,9508	2,3534	1,4989	**2,4365**
qCABC	Best Err. (%)	**0,2535**	**0,1771**	**0**	**0,5412**	**0,3237**	1.8557	**0,9098**	2,2158
	Ave. Err. (%)	0,4822	**0,4781**	**0**	0,8493	**0,8844**	**2,2780**	**1,4695**	2,4940

1950004-17

Tabela 13. Comparação de desempenho de algoritmos quando $L = 2$

Método		BERLIN52	KROA100	PR144	CH150	KROB150	PR152	RAT195
EXC	Best Err. (%)	**0**	0,1692	0,2426	1.9761	2,9277	2.0358	4,477
	Ave. Err. (%)	2.0353	2,8714	0,533	2,8232	5.0919	3,99	5,9836
DISP	Best Err. (%)	0,0663	4.2947	1,4845	3.4161	7,8263	4,7216	5.0366
	Ave. Err. (%)	1.424	8.0552	1,7628	3,9859	10,4635	5,5932	6,9393
INV	Best Err. (%)	**0**	2.6125	1.0284	2,9412	7,1183	3.1202	3,7021
	Ave. Err. (%)	1,1854	4,7965	1,9755	3,6596	9.0521	5.0318	6,3108
INS	Best Err. (%)	**0**	0,4652	0,1879	0,6434	1.8178	1,3938	4,2617
	Ave. Err. (%)	0,1525	3.0726	1,3802	2,3943	4.5488	2,8468	5,7426
SIM	Best Err. (%)	**0**	0,1692	0,2255	0,8425	2.8397	1.5282	2,3676
	Ave. Err. (%)	0,6099	1,3114	1,1717	1.5809	3,8596	2,7976	4.1886
SCM	Best Err. (%)	**0**	0,8176	0,1589	0,9344	3,2836	1,5146	4.6492
	Ave. Err. (%)	**0**	3,6966	1,3016	2,1078	7.0609	2.694	5,9923
GSM	Best Err. (%)	**0**	0,3007	0,2426	1,6544	3.452	2,2977	4,3048
	Ave. Err. (%)	0,9931	3,698	0,6891	2,4908	4,9541	2,9112	6,5476
GSTM	Best Err. (%)	**0**	**0**	**0**	0,4596	0,9644	0,7695	**0,6027**
	Ave. Err. (%)	**0**	1,1836	1.0809	0,6357	1,7616	1.6202	1,8425
CABC	Best Err. (%)	**0**	**0**	0,0905	**0,3217**	**0,2143**	**0**	0,8610
	Ave. Err. (%)	**0**	0,0423	0,1787	**0,5086**	**0,7960**	**0,1493**	1,3947
qCABC	Best Err. (%)	**0**	**0**	0,0905	0,3830	0,4631	**0**	1.0331
	Ave. Err. (%)	**0**	**0,0324**	**0,1490**	0,5453	0,8190	0,3006	**1,3000**

Tabela 14. Tabela 13 continuação.

Método		D198	KROA200	TS225	PR226	PR299	LIN318	PCB442	FL1577
EXC	Best Err. (%)	1.9709	2,5061	3.0424	4,9186	10,2198	10,1026	7,3437	-
	Ave. Err. (%)	4,9728	4,9877	3,8174	7,2295	15,1775	11,7195	10,4774	-
DISP	Best Err. (%)	3.891	6,7284	3.3693	4,1322	12,2616	10,9996	11,3435	-
	Ave. Err. (%)	5,9835	8,8276	3,8916	7,993	16,849	12,9013	12,3053	-
INV	Best Err. (%)	2,6743	6,2245	3,1640	8,1512	10,9543	9,541	10,5124	-
	Ave. Err. (%)	4,9823	8,3594	3,7355	8,8249	15,706	13,0386	11.6322	-
INS	Best Err. (%)	2.0279	2,1554	3.1751	3,8423	3,8804	6,3099	8,9685	-
	Ave. Err. (%)	3,41	4,2819	3,8767	6,484	11,4187	10,3935	11,0585	-
SIM	Best Err. (%)	1.6984	1,5766	1,6930	2,5769	6,439	6,3932	6,3728	-
	Ave. Err. (%)	2.5241	3.2457	2,5818	4.1795	9,1264	8,714	8,124	-
SCM	Best Err. (%)	3,1179	3,9873	2,7945	3,7875	10,8776	9,1651	7.8597	-
	Ave. Err. (%)	4.5818	5,4502	3,8029	5.0694	13,8686	10,4456	10,6251	-
GSM	Best Err. (%)	3.0482	4,4675	3.0424	5.0517	8,8191	9,4126	5,7643	-
	Ave. Err. (%)	4,2421	5,7018	4,4254	6,4485	13,6295	11,0036	8.0389	-
GSTM	Best Err. (%)	0,3866	0,8683	0,2527	0,7242	1,2326	**0,9827**	2.0501	-
	Ave. Err. (%)	1,2193	1,5432	0,4994	1.5287	2,9169	3,3099	2.7758	-
CABC	Best Err. (%)	**0,2852**	0,3371	**0**	**0,6259**	0,9317	1.7298	**1.0457**	**1.8697**
	Ave. Err. (%)	**0,4715**	**0,5090**	0,0858	0,9506	1,1456	2,3988	**1,4989**	2,5381
qCABC	Best Err. (%)	0,4119	**0,3167**	**0**	0,6308	**0,6682**	1,9034	1,2308	1.9237
	Ave. Err. (%)	0,6065	0,5400	**0**	**0,8569**	**1,1039**	**2.3986**	1,5052	**2,4482**

Tabela 15. Resultados dos testes estatísticos.

	Comparado Pares	Mau Diferença	*N*	*p* Valor de Teste t pareado	*p* valor de Teste Wilcoxon
aso 1	GSTM-CABC	0,907	14	0,000	0,000
	GSTM-qCABC	0,906	14	0,000	0,000
	CABC-qCABC	- 0,005	15	0,685	0,839
aso 2	GSTM-CABC	0,842	14	0,000	0,000
	GSTM-qCABC	0,840	14	0,000	0,000
	CABC-qCABC	0,004	15	0,834	0,715

s Tabelas 11–14, algumas análises estatísticas são realizadas para os algoritmos GSTM, CABC qCABC para ver se há uma diferença significativa entre os desempenhos desses algoritmos. ma vez que os dados de teste têm distribuição aproximadamente normal, o teste t pareado, n dos testes paramétricos, é usado para comparar as médias de duas amostras. A hipótese la é que a diferença média das amostras é igual a 0. No entanto, o tamanho da amostra pode r pequeno para o teste t pareado. Assim, o teste de postos sinalizados de Wilcoxon, que é um s testes não paramétricos, também é aplicado. A hipótese nula deste teste é que a diferença ediana entre pares de observações é igual a 0. Os pares comparados, diferença média dos ros percentuais, tamanhos de amostra (*N)* e
os valores são dados na Tabela 15. Nesta tabela, o caso 1 apresenta os resultados do teste ra algoritmos ABC tendo os mais apropriados *eu* valores, enquanto o caso 2 apresenta os sultados para algoritmos ABC com o mesmo *eu* valores ($L = 2$). Quando os resultados são aminados, ambos os testes estatísticos afirmam que os algoritmos CABC e qCABC têm sempenho significativamente melhor do que o GSTM em ambos os casos. Quando a ferença média dos erros percentuais são considerados, CABC produz melhores resultados do e qCABC no caso 1 e qCABC produz resultados melhores do que CABC no caso 2. No entanto, sas diferenças não são significativas de acordo com o teste t pareado com $p = 0{,}685$ e $= 0{,}834$ e teste de classificação sinalizada de Wilcoxon com $p = 0{,}839$ e $p = 0{,}715$. Observe que esses stes são realizados para os resultados finais de 800.000 avaliações.

Para apresentar o desempenho de convergência do CABC e do qCABC, quatro problemas teste são selecionados como tendo diferentes e vários números de cidades. Os gráficos de nvergência dos algoritmos são mostrados nas Figs. 6–9 para problemas KroA100, KroA200, b442 e Fl1577, respectivamente. Esses números demonstram que o qCABC é mais rápido do e o CABC para todos os problemas considerados com 100, 200, 442 e 1577 cidades.

Estudos mostram que ABC fornece melhor desempenho do que muitos algoritmos de imização baseados em computação evolutiva, especialmente no significado de exploração, na vez que tem um mecanismo eficaz na fase de abelha escoteira. Além disso, na fase de elhas empregadas, todas as soluções na população são tentadas a melhorar uma vez, alizando uma pesquisa na vizinhança ao redor delas, não procurando se essa solução é boa não. No entanto, todas as fontes de alimento podem não ser consideradas da mesma forma fase das abelhas espectadoras. Mais pesquisas de vizinhança são realizadas para as elhores soluções nesta fase. Na verdade, a exploração de melhores soluções é realizada

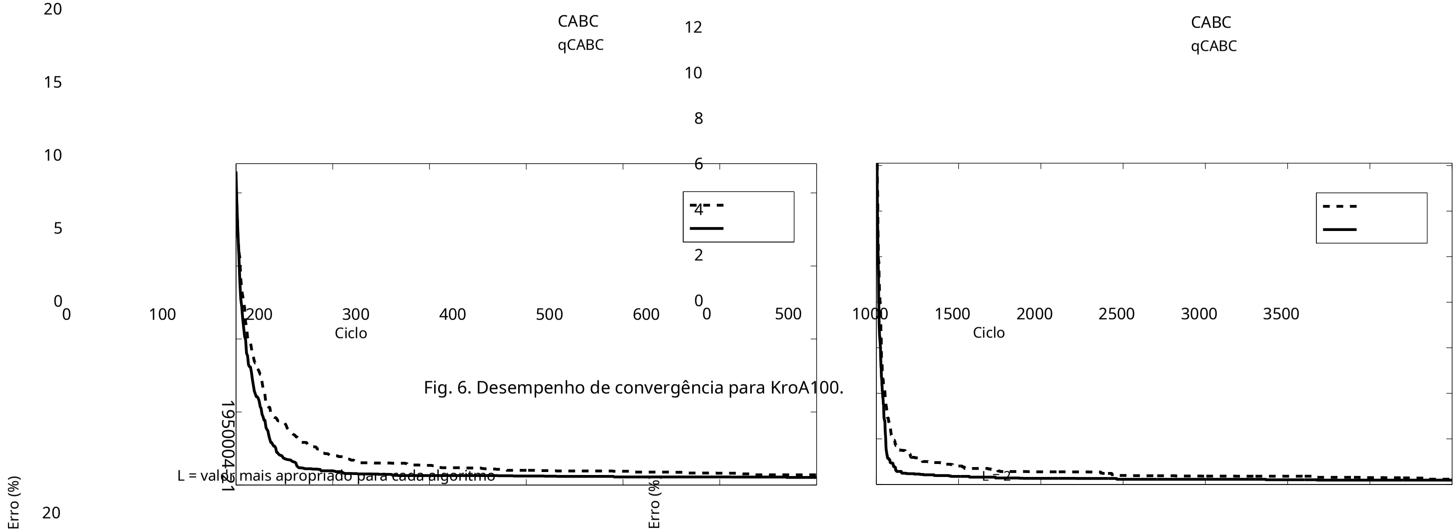


Fig. 6. Desempenho de convergência para KroA100.

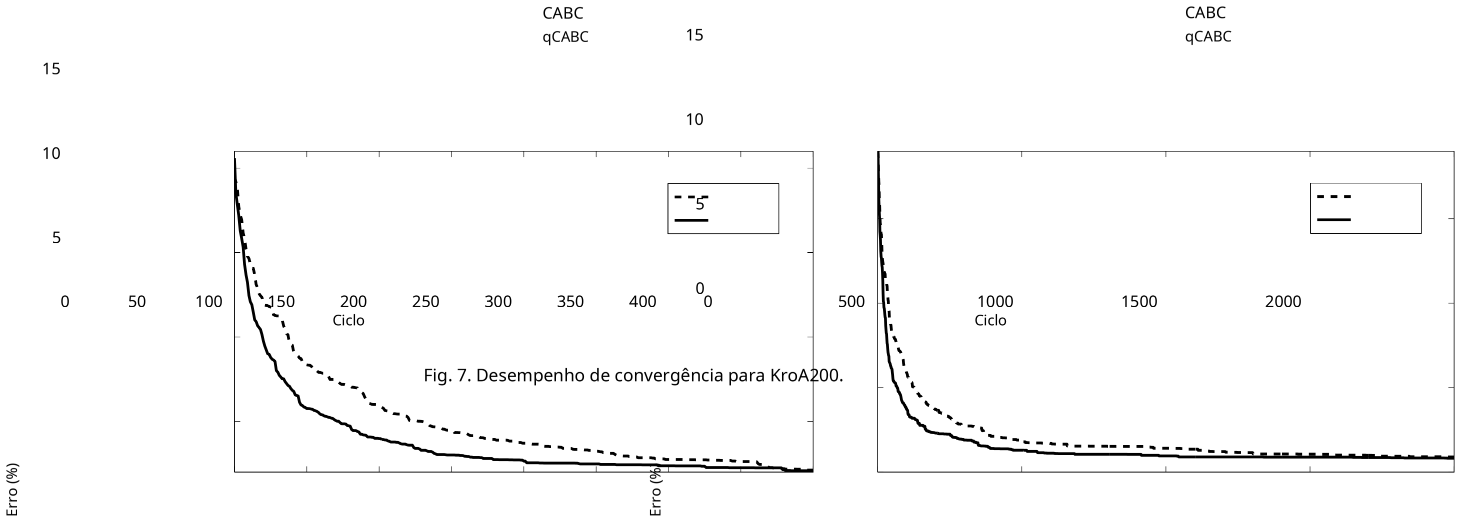


Fig. 7. Desempenho de convergência para KroA200.

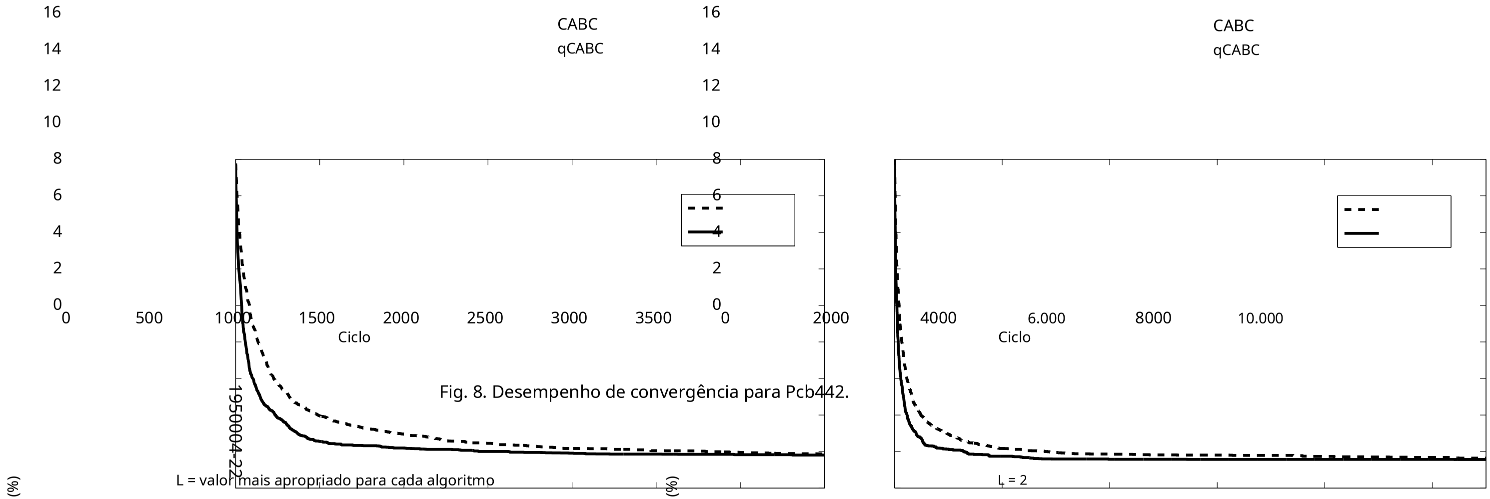


Fig. 8. Desempenho de convergência para Pcb442.

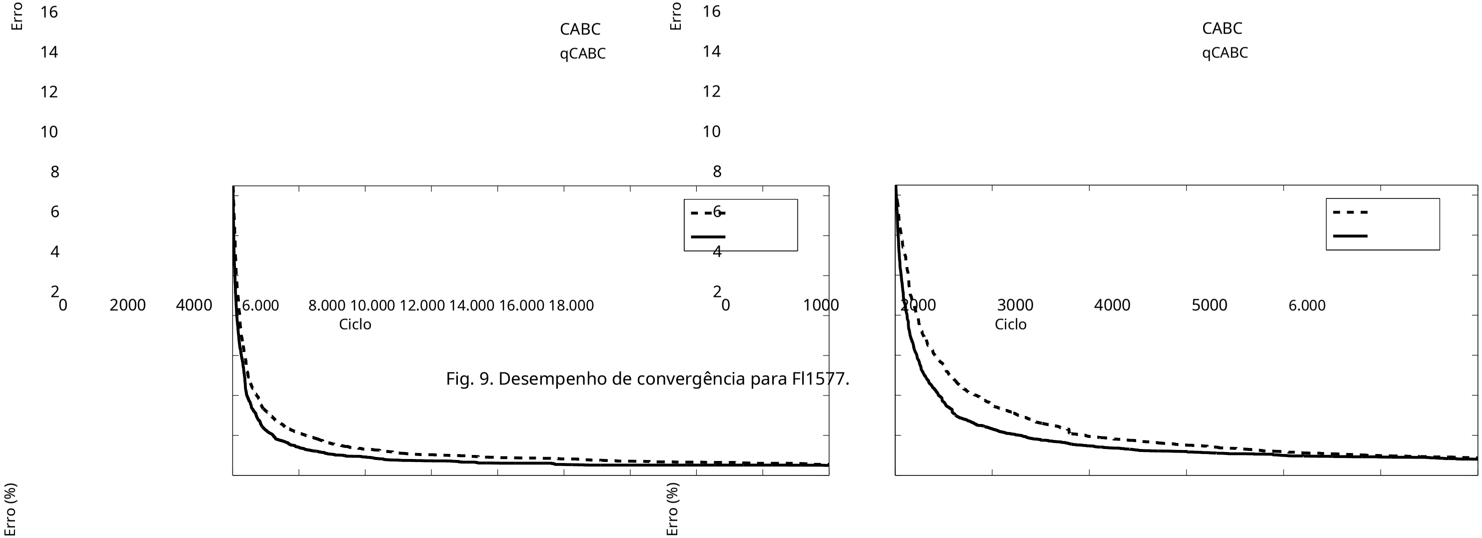


Fig. 9. Desempenho de convergência para Fl1577.

algoritmo de uma forma poderosa como a exploração é declarada. Assim, o sucesso ABC não está apenas relacionado com o tipo de problema de otimização (por exemplo, umérico ou combinatório), o poder do algoritmo está na sua estrutura de busca que se seia no bom equilíbrio entre os mecanismos de exploração e aproveitamento. Embora ABC padrão seja introduzido para problemas numéricos, se pudermos proteger seus ecanismos balanceados de exploração-exploração enquanto o adaptamos a problemas scretos, o sucesso do algoritmo continuará. Visto que o CABC é aprimorado com o nprego de um operador de mutação eficaz, considerando o mecanismo de busca de zinhança do ABC original, ele mantém sua superioridade. E também o modelo qABC opõe uma ideia para aumentar a exploração de melhores soluções e isso acarreta uma usca local mais rápida através de ciclos.

1. *Tempos de CPU dos algoritmos CABC e qCABC*

esta seção, os tempos de CPU dos algoritmos CABC e qCABC são fornecidos para 50, 100, 0, 500 e 1000 problemas de cidade quando o tamanho da colônia é 40, *limite = (tamanho da colônia * mensão)/* 2 e o critério de parada é atingir um total de 4000 avaliações. 30 execuções dependentes são realizadas para cada caso e os valores médios e desvios padrão de 30 ecuções são apresentados na Tabela 16. Os tempos de CPU dos algoritmos podem ser aliados considerando a mudança de dimensão. Essa alteração é mostrada em um gráfico monstrando os valores médios dos tempos de CPU na Fig. 10 para os algoritmos CABC e ABC. Essas análises foram realizadas no Windows 10 Education em um processador Intel (R) re (TM) M-5Y51 de 1,20 GHz com 8 GB de RAM e os algoritmos foram codificados usando a guagem de programação C Sharp e o perfil do cliente .net framework 4 foi usado.

A Tabela 16 mostra que, o tempo de CPU necessário de qCABC é maior do que o algoritmo ABC para completar 4000 avaliações de função, pois há uma parte adicional na fase de elhas observadoras. No entanto, a taxa de incremento no tempo de computação completo qCABC é menor do que a taxa de incremento da dimensão, como o algoritmo CABC para a xa de dimensão 100/50. Para valores de dimensão maiores, as taxas de incremento em nbos os algoritmos são maiores do que a taxa de incremento da dimensão.

Tabela 16. Tempos de CPU para problemas com diferentes números de cidades.

	CABC		qCABC	
Número de cidades	Mau	Std	Mau	Std
50	0,2787	0,0952	0,5521	0,1398
100	0,3115	0,0188	1.0844	0,1822
200	1.0100	0,2325	2,2947	0,1855
500	3,5468	0,11118	5,9587	0,7467
1000	8.4452	1,6377	18,5469	0,9318

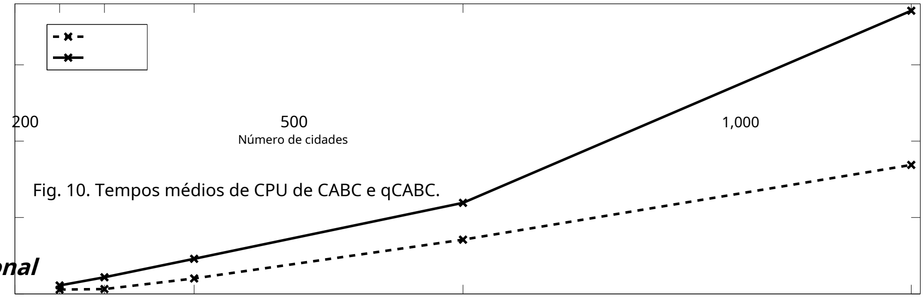


Fig. 10. Tempos médios de CPU de CABC e qCABC.

2. *Estudo adicional*

fim de comparar o desempenho de CABC e qCABC com outros algoritmos baseados em teligência de enxame, apresentamos os resultados de ABCs e ACS na Tabela 17 e BCO Tabela 18. Observe que, ACS e BCO que foram originalmente introduzidos para esquisa em espaços discretos foram considerados nessas comparações. Usamos os esmos resultados da Tabela 11 para CABC e qCABC.

A Tabela 17 relata as melhores durações e o número de passeios necessários para contrar o melhor passeio (entre parênteses) para ACS, CABC e qCABC. Os resultados são tirados da Ref. 10 para ACS. Como visto na Tabela 17, qCABC produz melhores soluções

Tabela 17. Comparação de ACS, CABC e qCABC.

Problemas	ACS	CABC	qCABC
KROA100	***21282***	***21282***	***21282***
	(4820)	(61688)	**(1680)**
D198	15888	15825	**15820**
	(585000)	(408388)	**(269680)**
PCB442	51268	51309	**51240**
	(595000)	**(145440)**	(339055)
FL1577	22977	**22703**	22742
	(942000)	(651242)	**(328763)**

Tabela 18. Comparação de BCO, CABC e qCABC.

Problemas		BCO	CABC	qCABC
BERLIN52	Best Err. (%)	**0**	**0**	**0**
	Ave. Err. (%)	**0**	**0**	**0**
KROA100	Best Err. (%)	0,23	**0**	**0**
	Ave. Err. (%)	0,65	0,0423	**0,0113**
KROB150	Best Err. (%)	1,66	0,3100	**0,2411**
	Ave. Err. (%)	2,22	**0,6950**	0,7160

m menor número de construções turísticas do que ACS para todos os problemas considerados. Também mparando os resultados do CABC, melhores soluções com menor número de construções turísticas são produzidas o qCABC para a maioria dos problemas, exceto Fl1577. Para o problema KroA100, embora todos os algoritmos neçam o comprimento de passeio ideal, o algoritmo CABC o encontra construindo mais passeios. Examinando a bela, pode-se dizer que o CABC é mais bem-sucedido do que o ACS para problemas D198 e Fl1577. No entanto, para o442, ACS tem melhor desempenho do que CABC.

A Tabela 18 apresenta os melhores e médios (%) erros dos algoritmos BCO, CABC e CABC para problemas Berlin52, KroA100 e KroB150. Usamos os resultados de BCO rnecidos na Ref. 20. Na Ref. 20, o número de ciclos é 10.000 e o tamanho da colônia é ual ao número total de cidades para o algoritmo BCO (520.000 avaliações para Berlin52, 000.000 avaliações para Kroa100 e 1.500.000 avaliações para Krob150). Como os râmetros dos algoritmos CABC e qCABC são definidos como na Tabela 2, o número de aliação de qCABC e CABC é 800.000 para todos os problemas. Embora ABCs sejam ecutados para mais avaliações (800.000), eles encontraram a duração ideal do tour para das as execuções muito antes de 520.000 avaliações para o problema Berlin52. Para oblemas KroA100 e KroB150, os algoritmos CABC e qCABC demonstram desempenhos uito melhores do que o algoritmo BCO.

O desempenho do ABCs é geralmente melhor do que algoritmos bem conhecidos, GA e suas riantes, ACS e BCO para problemas de teste considerados. Conseqüentemente, pode-se concluir e CABC e qCABC podem ser empregados para resolver problemas do tipo TSP, de forma eficaz.

Conclusões

este estudo, duas variantes diferentes da abordagem ABC são explicadas para o TSP. Uma rsão combinatória do ABC padrão, denominado algoritmo CABC e uma versão aprimorada do goritmo CABC, denominado qCABC. Os estudos experimentais foram realizados em um njunto de 15 problemas de teste de TSP. Em seguida, o desempenho desses dois algoritmos e to variantes de GA diferentes são comparados. Além disso, o desempenho do ABCs também comparado com algoritmos de sistema de colônia de formigas (ACS) e otimização de colônia abelhas (BCO) que foram originalmente propostos para problemas do tipo combinatório. ém disso, algumas análises sobre os tempos de CPU dos algoritmos CABC e qCABC e alguns tudos de ajuste de parâmetro de *limite* são fornecidos. Quando os resultados nas tabelas de mparação foram examinados, ficou claro que os algoritmos CABC e qCABC podem resolver problemas com bastante sucesso. O desempenho de convergência do CABC é melhorado licando a ideia qABC para TSP.

econhecimento

te trabalho foi financiado pelo Fundo de Pesquisa da Universidade Erciyes, Número do ojeto: FBA-11-3536.

eferências

G. Laporte, O problema do caixeiro viajante: uma visão geral dos algoritmos exatos e aproximados, *European Journal of Operational Research* **59 (** 2) (1992) 231–247.
P. Miliotis, Usando planos de corte para resolver o problema do caixeiro viajante simétrico, *Programação Matemática* **15 (** 1) (1978) 177–188.
M. Padherg e R. Rinaldi, Otimização de um problema de caixeiro viajante simétrico de 532 cidades por ramo e corte, *Cartas de pesquisa operacional* **6 (** 1) (1987) 1-7.
O. Martin, S. Otto e E. Felten, grandes cadeias de Markov para o problema do caixeiro viajante, *Sistemas Complexos* **5 (** 3) (1991) 299-326.
GA Croes, um método para resolver problemas de caixeiros-viajantes, *Pesquisa Operacional* **6 (** 6) (1958) 791–812.
G. Gutin, A. Yeo e A. Zverovich, caixeiro viajante não deve ser ganancioso: Análise de dominação de heurísticas do tipo ganancioso para o TSP, *Matemática Aplicada Discreta* **117 (** 1-3) (2002) 81-86.
S. Kirkpatrick, CD Gelatt e MP Vecchi, Otimização por recozimento simulado, *Ciência* **220 (** 4598) (1983) 671–680.
F. Glover, Tabu search - Um tutorial, *Interfaces* **20 (** 4) (1990) 74–94.
Y. Nagata e S. Kobayashi, um poderoso algoritmo genético usando crossover de montagem de borda para o problema do caixeiro viajante, *Informa Journal on Computing* **25 (** 4) (2013) 346–363.

M. Dorigo e LM Gambardella, sistema de colônia de formigas: uma abordagem de aprendizagem cooperativa para o problema do caixeiro viajante, *Transações IEEE em computação evolutiva* **1 (** 1) (1997) 53–66.
J. Kennedy e R. Eberhart, um romance max-min ant system algoritmo para o problema do caixeiro viajante, em *Proc. do IEEE Int. Conf. na rede neural,* Vol. 4 (1995), pp. 1942-1948.

R. Storn e K. Price, evolução diferencial - Uma heurística simples e eficiente para otimização global em espaços contínuos, *Journal of Global Optimization* **11 (** 4) (1997) 341–359.

D. Karaboga, Uma ideia baseada em enxame de abelhas melíferas para otimização numérica, Tech. Rep. TR06, Universidade Erciyes, Faculdade de Engenharia, Departamento de Engenharia da Computação (2005).
Z. Zhang e Z. Feng, um novo algoritmo de sistema de formiga máximo-mínimo para o problema do caixeiro viajante, em *Proc. da Computação Inteligente e Sistemas Inteligentes (ICIS 2009)* (2009), pp. 508-511.
R. Gan, Q. Guo, H. Chang e Y. Yi, algoritmo de otimização de colônia de formigas aprimorado para os problemas do caixeiro viajante, *Jornal de Engenharia de Sistemas e Eletrônica* **21 (** 2) (2010) 329–333.
S. Ilie e C. Badica, E ff ectiveness of resolver o problema do caixeiro viajante usando a otimização de colônia de formigas em middleware multiagente distribuído, em *Proc. do Int. Multiconferência em Ciência da Computação e Tecnologia da Informação (* 2010), pp. 197–203.
A. Puris, R. Bello e F. Herrera, Análise da eficácia de uma metodologia de dois estágios para otimização de colônias de formigas: Caso de estudo com TSP e QAP, *Sistemas especialistas com aplicativos* **37 (** 7) (2010) 5443–5453.
M. Tuba e R. Jovanovic, algoritmo ACO aprimorado com estratégia de correção de feromônio para o problema do caixeiro viajante, *Jornal Internacional de Comunicações e Controle de Computadores* **8 (** 3) (2013) 477–485.
LP Wong, MYH Low e CS Chong, Um algoritmo de otimização de colônia de abelhas para o problema do caixeiro viajante, em *Proc. da Segunda Ásia Int. Conf. em Modelagem e Simulação (AMS 2008) (* 2008), pp. 818–823.

LP Wong, MYH Low e CS Chong, Otimização de colônias de abelhas com busca local por problema do caixeiro viajante, em *Proc. da Informática Industrial (INDIN 2008)* (2008), pp. 1019–1025.
Y. He, Y. Qiu, G. Liu e K. Lei, Uma abordagem de pesquisa de tabu adaptativa paralela para problemas de caixeiro-viajante, em *Proc. do IEEE Int. Conf. em Processamento de Linguagem Natural e Engenharia do Conhecimento (* 2005), pp. 796–801.
N. Yang, P. Li e B. Mei, uma busca tabu cruzada baseada em ângulo para o problema do caixeiro viajante, em *Proc. do Int. Conf. na computação natural (* 2007), pp. 512–516.
JW Pepper, BL Golden e EA Wasil, Resolvendo o problema do caixeiro viajante com heurísticas baseadas em recozimento: um estudo computacional, *Sistemas, Homem e Cibernética, Parte A: Sistemas e Humanos* **32 (** 1) (2002) 72–77.
JD Wei e DT Lee, Uma nova abordagem para o problema do caixeiro viajante usando algoritmos genéticos com codificação prioritária, em *Proc. do Congresso de Computação Evolutiva (* 2004), pp. 1457–1464.
WL Zhong, J. Zhang e WN Chen, uma nova otimização de enxame de partículas discretas para resolver o problema do caixeiro viajante, em *Proc. do Congresso IEEE sobre Computação Evolutiva (* 2007), pp. 3283–3287.
Z. Yuan, L. Yang, Y. Wu, L. Liao e G. Li, algoritmo de otimização de enxame de partículas caóticas para o problema do caixeiro viajante, em *Proc. do IEEE Int. Conf. em Automação e Logística (* 2007), pp. 1121–1124.
RS Prado, RCP Silva, FG Guimarães e OM Neto, Usando a evolução diferencial para otimização combinatória: Uma abordagem geral, em *Proc. do IEEE Int. Conf. em Systems, Man and Cybernetics (SMC 2010) (* 2010).
JG Sauer e L. dos S. Coelho, Evolução diferencial discreta com busca local para resolver o problema do caixeiro viajante: Fundamentos e estudos de caso, em *Proc. do 7º IEEE Int. Conf. em Sistemas Inteligentes Cibernéticos,* eds. R. Comley, B. Amavasai,
X. Cheng, M. O'Grady, C. Huyck e N. Siddique (2008), pp. 299-304.
X. Wang e G. Xu, algoritmo de evolução diferencial híbrido para o problema do caixeiro viajante, em *Proc. do CEIS 2011,* eds. C. Ran e G. Yang, **15 (** 2011).
D. Karaboga, algoritmo Arti fi cial bee colony (ABC), *Scholarpedia* **5 (** 3) (2010) 6915.
D. Karaboga, B. Gorkemli, C. Ozturk e N. Karaboga, uma pesquisa abrangente: algoritmo e aplicações de colônia de abelhas artificiais (ABC), *Revisão de Inteligência Arti fi cial* (2012).
JC Bansal, H. Sharma e SS Jadon, Arti fi cial bee colony algoritmo: A survey, *International Journal of Advanced Intelligence Paradigms* **5 (** 1/2) (2013) 123–159.
AL Bolaji, AT Khader, MA Al-Betar e MA Awadallah, Algoritmo de colônia de abelha artificial, suas variantes e aplicações: um levantamento, *Journal of Theoretical & Applied Information Technology* **47 (** 2) (2013) 434–459.
B. Kumar e D. Kumar, Uma revisão sobre algoritmo de colônia de abelhas arti fi cial, *Jornal Internacional de Engenharia e Tecnologia* **2 (** 3) (2013) 175–186.
K. Balasubramani e K. Marcus, uma revisão abrangente do algoritmo de colônia de abelhas arti fi cial, *International Journal of Computers & Technology* **5 (** 1) (2013) 15–28.
L. Li, Y. Chong, L. Tan e B. Niu, um algoritmo discreto de colônia de abelha artificial para o problema de TSP, em *Proc. da 7ª Int. Conf. em computação inteligente (ICIC) (* 2011), pp. 566–573.

W. Li, W. Li, Y. Yang, H. Liao, JL Li e X. Zheng, Arti fi cial bee colony algoritmo para o problema do caixeiro viajante, em *Proc. do Int. Conf. em Design Avançado e Engenharia de Manufatura (ADME 2011) (* 2011), pp. 2191–2196.

D. Karaboga e B. Basturk, No desempenho de algoritmo de colônia de abelhas arti fi cial (ABC), *Applied Soft Computing* **8 (** 1) (2008) 687–697.
M. Albayrak e N. Allahverdi, desenvolveram um novo operador de mutação para resolver o problema do caixeiro viajante com o auxílio de algoritmos genéticos, *Sistemas especialistas com aplicativos* **38 (** 3) (2011) 1313–1320.
D. Karaboga e B. Gorkemli, um algoritmo de colônia de abelhas arti fi cial combinatória para o problema do caixeiro viajante, em *Proc. do INISTA 2011: Int. Symp. em inovações em sistemas e aplicativos inteligentes (* 2011), pp. 50–53.
D. Karaboga e B. Gorkemli, Uma colônia de abelhas arti fi cial rápida -qABC- algoritmo para problemas de otimização, em *Proc. do INISTA 2012: Int. Symp. em inovações em sistemas e aplicativos inteligentes (* 2012).
D. Karaboga e B. Gorkemli, um algoritmo de colônia de abelhas arti fi cial rápida (qABC) e seu desempenho em problemas de otimização, *Applied Soft Computing* **23 (** 2014) 227–238.
B. Gorkemli e D. Karaboga, Quick combinatorial arti fi cial bee colony -qCABCoptimization algoritmo for TSP, in *Proc. do Segundo Int. Symp. em Computação em Informática e Matemática (ISCIM 2013) (* 2013).
D. Karaboga e B. Akay, Um estudo comparativo do algoritmo de colônia de abelhas arti fi cial, *Matemática Aplicada e Computação* **214 (** 1) (2009) 108-132.