LVI SBPO SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
DE 04 ATÉ 07 NOVEMBRO 2024
FORTALEZA | CEARÁ

BRKGA com Aprendizado por Reforço para o Problema de p-Centros com α-Vizinhos

Eduardo Machado Silva
Universidade Federal de São Paulo (UNIFESP-ICT)
Av. Cesare Monsueto Giulio Lattes, 1201 - Eugênio de Melo, São José dos Campos - SP, 12247-014
machado.eduardo@unifesp.br

Antônio Augusto Chaves
Universidade Federal de São Paulo (UNIFESP-ICT)
Av. Cesare Monsueto Giulio Lattes, 1201 - Eugênio de Melo, São José dos Campos - SP, 12247-014
antonio.chaves@unifesp.br

RESUMO

Este trabalho propõe uma aplicação do *Biased Random-Key Genetic Algorithm* (BRKGA) combinado com Aprendizado por Reforço para resolver o Problema de p-Centros com α-Vizinhos. Neste problema, dado um grafo, busca-se selecionar p vértices como centros de forma que a distância entre os demais vértices e seus α centros mais próximos seja minimizada. Para isso, é apresentado um procedimento de decodificação, juntamente com três estratégias de busca local, comparando seus resultados com aqueles presentes na literatura utilizando instâncias conhecidas. O conjunto de instâncias da *OR-library* e *TSPLIB* foram considerados nos testes, e os resultados mostram que o método proposto é robusto e competitivo, frequentemente encontrando soluções próximas ou iguais às melhores soluções conhecidas.

PALAVRAS CHAVE. Meta-heurística, Aprendizado por Reforço, p-centros.

Tópicos: Metaheurísticas

ABSTRACT

This work presents a Biased Random-Key Genetic Algorithm with Reinforcement Learning to solve the $\alpha-$neighbor p-center problem. Given a graph, the $\alpha-$neighbor p-center problem objective is to select p vertices, also known as centers, so that the maximum distance between each other vertices and its $\alpha-$neighbor centers is minimized. A decoder process and three local search procedures are proposed. The results are compared using benchmark instances. The instance sets from the *OR-library* and *TSPLIB* were used for computational evaluation, and the results indicate that the proposed method is both robust and competitive, frequently achieving solutions that are close to or match the best-known solutions.

KEYWORDS. Metaheuristic. Reinforcement Learning. α-neighbor p-centers.

Paper topics: Metaheuristics

DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br

1. Introdução

Um dos problemas de localização de facilidades mais estudados na literatura é o problema dos p-Centros (pCP). O pCP foi proposto por [Hakimi, 1964, 1965] e consiste em selecionar, em um dado grafo, p vértices, também chamados de centros, de forma que a distância dos demais vértices para seus centros mais próximos seja minimizada. O pCP possui diversas aplicações como a determinação de pontos de serviços emergenciais como, postos de polícia, ambulâncias e brigadas de incêndio [Çalık et al., 2019; Mousavi, 2023]. O trabalho de [Callaghan et al., 2019] discute alguns exemplos de aplicações do problema envolvendo instâncias reais.

No pCP, cada vértice é atribuído a uma única instalação. No entanto, em algumas aplicações, as instalações podem estar sujeitas a falhas e se tornarem indisponíveis devido a razões imprevisíveis, como problemas meteorológicos e elétricos [Panteli et al., 2021]. Em tais casos, é importante garantir a continuidade do serviço aos clientes atribuídos à instalação com falha. Isso é comum em serviços críticos, como hospitais, estações de bombeiros. Um outro exemplo pode ser considerado durante cenários de pandemias, onde hospitais em áreas urbanas altamente densas ficam sobrecarregados em dias normais. Nessa situação, hospitais em regiões menos densas, mas com capacidade adicional de atendimento, poderiam servir como *backups* para esses grandes centros. Assim, realocar pacientes para esses postos de saúde mais próximos se tornaria mais viável do que abrir postos temporários [Chagas et al., 2024].

Para lidar com situações como a descrita acima, [Krumke, 1995] generalizou o pCP e propôs o pCP com α-vizinho (αNpCP, do inglês *α-neighbor p-Center Problem*), nos quais os vértices não são atribuídos apenas para o centro mais próximo, mas sim para seus α centros mais próximos. Assim como o pCP, o αNpCP continua sendo um problema NP-difícil. Métodos exatos e aproximações para o problema podem ser encontrados em [Chaudhuri et al., 1998; Khuller et al., 2000; Chen e Chen, 2013; Callaghan et al., 2019].

Há apenas três trabalhos na literatura que propuseram métodos heurísticos para o αNpCP. Em [Sánchez-Oro et al., 2022], foi proposta uma metaheurística baseada no GRASP (do inglês *Greedy Randomized Adaptive Search Procedure*), busca tabu e uma estratégia de oscilação, chamada de GRASP-SO. Os autores testaram o método em 77 instâncias derivadas da *TSPLIB* [Reinelt, 1991] e compararam seus resultados com o método exato de [Chen e Chen, 2013], onde o GRASP-SO obteve os melhores resultados em todas as instâncias testadas. Em [Mousavi, 2023], um algoritmo de busca local foi desenvolvido para o problema. O autor executou o algoritmo em 40 instâncias da *OR-library* [Beasley, 1990] por 10.000 segundos, demonstrando que a heurística pode encontrar soluções ótimas em um curto período de tempo. Por fim, [Chagas et al., 2024] propuseram um método VNS (do inglês *Variable Neighborhood Search*) com paralelismo para resolver problemas de localização de facilidades, incluindo o αNpCP. Estruturas de vizinhança baseadas na abertura e fechamento aleatório de facilidades foram propostas. Também foi introduzida uma nova função objetivo que adiciona mais informações ao custo da solução e ajuda o VNS a escapar de ótimos locais. O método foi avaliado através das instâncias da *TSPLIB* e da *OR-library* considerando um tempo total de 30 minutos, mostrando que o VNS supera os algoritmos do estado da arte.

Na configuração de uma metaheurística, determinar a ordem e os valores na qual seus diferentes componentes são usados é uma tarefa difícil. [Eiben et al., 2007] discute abordagens para a definição dos parâmetros que são classificadas de duas maneiras: ajuste de parâmetros (configuração *offline*) e controle de parâmetros (configuração *online*). O ajuste de parâmetros define os valores antes do início da execução da metaheurística, enquanto o controle de parâmetros modifica dinamicamente os parâmetros da metaheurística durante a execução. [Eiben et al., 2007] chamam a atenção para o potencial dos algoritmos evolutivos ajustarem seus próprios parâmetros de forma *online*. Ao longo do projeto, será usado o termo "heurística/metaheurística adaptativa" para denotar

DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br

métodos que utilizam técnicas de controle de parâmetros.

O Trabalho de [Li et al., 2022] propõe uma técnica de Aprendizado Profundo para o Problema do Caixeiro Viajante com Cobertura. Chaves et al. [2018] propuseram uma metaheurística com controle de parâmetros *online* chamada de *Adapted* BRKGA (A-BRKGA). O A-BRKGA é uma variação do BRKGA de [Gonçalves e Resende, 2011]. Mais tarde, [Chaves et al., 2022] projetaram um *framework* de controle de parâmetros melhorado para o A-BRKGA. Os autores utilizaram conceitos de Aprendizado por Reforço [Sutton e Barto, 2018], mais especificamente, o algoritmo *Q-learning* [Watkins, 1989] para controlar os parâmetros do BRKGA. Uma revisão da literatura sobre heurísticas adaptativas pode ser encontrada em [Alabas-Uslu e Dengiz, 2014; Sevaux et al., 2015; Bengio et al., 2021].

Este trabalho contribui para o estado da arte do αNpCP desenvolvendo uma metaheurística baseada no BRKGA com *Q-learning* (BRKGA-QL), proposto por [Chaves et al., 2022]. Além de um procedimento de decodificação para o problema em termos de chaves aleatórias, também são apresentadas três heurísticas de busca local para intensificar a busca no espaço de soluções.

O restante do artigo se organiza da seguinte forma: o modelo e as definições matemáticas do problema são descritos na Seção 2. O método BRKGA-QL e as buscas locais são apresentados na Seção 3. Os resultados e análises computacionais são apresentados na Seção 4, por fim, a Seção 5 conclui o trabalho e apresenta as propostas de pesquisa futura.

2. Modelo matemático

Seja $G = (V, E)$ um grafo não-direcional, ponderado e conectado, no qual V é o conjunto de vértices e E o conjunto de arestas, tais que $|V| = n$ e $|E| = m$, e para cada arco $(i, j) \in E$ tem-se o valor $d_{ij} \in \mathbb{R}^+$ associado. Para problemas de localização de facilidades no geral, o valor d_{ij} é usado como a distância euclidiana entre os vértices i e j, porém, outros valores podem ser considerados. Em todos os casos, a propriedade da desigualdade triangular não é violada por esses valores. Mesmo que o valor d_{ij} não exista no contexto de um determinado grafo para o par de vértices (i, j), pode-se definir d_{ij} como a distância do menor caminho entre i e j de modo que a propriedade da desigualdade triangular não seja violada. Portanto, $D = (d_{ij})$ é uma matriz quadrada de dimensão n^2 com coordenadas não negativas.

Assumimos que $S \subset V$ seja o conjunto de centros a serem selecionados dentre as facilidades e que cada vértice $i \in V \backslash S$ é atribuído para seus α centros mais próximos. A distância entre um vértice i e seu α-ésimo centro mais próximo é denotado por $d_\alpha(i, S) = \min_{S' \subset S, |S'|=\alpha}\{\max_{j \in S'} d_{ij}\}$. Desta forma, define-se o objetivo do αNpCP como sendo minimizar a maior distância dentre os vértices e seus centros, *i.e.*, encontrar o conjunto $S \subset V$, $|S| = p$, de forma que $\max_{i \in V \backslash S} d_\alpha(i, S)$ seja mínimo. Nota-se que quando um vértice é selecionado como um centro, ele não é atribuído a outras facilidades.

A formulação matemática proposta em [Chagas et al., 2024] permite que cada vértice seja atribuído a mais de um centro. Nesse modelo matemático as variáveis de decisão x_{ij} determinam quando uma facilidade i é atribuída a um centro j. Desta forma:

$$x_{ij} = \begin{cases} 1 & \text{se o vértice } i \in V \text{ é atribuído para a o centro } j \in V \\ 0 & \text{caso contrário} \end{cases}$$

Note que quando um vértice i é escolhido como centro, tem-se $x_{jj} = 1$. Desta forma, um modelo matemático do αNpCP é dado por:

$$\textbf{Min } z \tag{1}$$

DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br

Sujeito a:

$$\sum_{i \in V, i \neq j} x_{ij} = \alpha(1 - x_{jj}) \qquad j \in V, \tag{2}$$

$$\sum_{j \in V} x_{jj} = p, \tag{3}$$

$$x_{ij} \leq x_{jj} \qquad i \in V,\ j \in V,\ i \neq j, \tag{4}$$

$$d_{ij} x_{ij} \leq z \qquad i \in V,\ j \in V,\ i \neq j, \tag{5}$$

$$z \in \mathbb{R}^+, x_{ij} \in \{0, 1\} \qquad i \in V,\ j \in V. \tag{6}$$

A função objetivo (1) corresponde a minimização da variáveis z, cujo limitante inferior é dado pelas restrições (5). Em outras palavras, a função objetivo (1) minimiza a maior distância entre um vértice e o seu α centro mais distante. As restrições (2) garantem que cada vértice $i \in V \backslash S$ seja atribuído para α centros. Note que se $x_{jj} = 1$, então o vértice j não é atribuído a nenhuma outra facilidade (lado direito da restrição (2) será zero). As restrições (3) garantem que exatamente p centros sejam abertos. Uma facilidade i pode ser atribuída ao centro j se, e somente se, $x_{jj} = 1$, essa condição é garantida pelas restrições (4). As restrições (6) determinam o domínio das variáveis.

3. BRKGA-QL

O BRKGA é uma metaheurística desenvolvida por [Gonçalves e Resende, 2011] e baseia-se no *Random Key Genetic Algorithm* (RKGA) de [Bean, 1994]. No RKGA, os indivíduos são codificados usando números aleatórios (também chamadas de chaves aleatórias), no intervalo $[0, 1)$. Portanto, um indivíduo χ é visto como um elemento de $[0, 1)^r$, onde r é o tamanho do cromossomo. Um algoritmo determinístico, chamado *decoder*, mapeia os vetores de chaves aleatórias para soluções do problema de otimização em questão.

O processo evolutivo do RKGA começa após a geração aleatória dos primeiros k indivíduos. O processo de reprodução é realizado copiando os melhores indivíduos de uma geração para a próxima, o que é chamado de estratégia elitista. Suponha que $p_e, p_e \leq k$, seja o número de soluções da população elite (com os melhores valores de *fitness*) mantidas de uma geração para outra. A população é então dividida em soluções de elite e não-elite. Uma vantagem observada ao se adotar estratégias elitistas é que os melhores indivíduos são melhorados monotonicamente de uma geração para outra. A desvantagem é o risco de convergência prematura em ótimos locais. No entanto, esse risco pode ser controlado por altas taxas de mutação. Um total de p_m novos indivíduos são gerados aleatoriamente em vez da mutação usual de baixa probabilidade gene por gene nos indivíduos. Esse processo ajuda a evitar a convergência prematura da população. Os $k - p_e - p_m$ indivíduos restantes para a nova geração são obtidos através do processo de *crossover*. O RKGA utiliza o *Parametric Uniform Crossover* (PUX) [Spears e De Jong, 1991], no qual dois indivíduos são selecionados para passar seus genes para gerar uma nova solução, e um parâmetro ρ_e é usado para decidir se os genes do filho são obtidos através do primeiro ou do segundo pai. Os critérios de parada do método são baseados no número de gerações, max_{gen}, ou no tempo de execução total. O BRKGA difere do RKGA na seleção dos pais para conduzir o operador de cruzamento. No BRKGA, cada filho é gerado combinando um indivíduo selecionado aleatoriamente da partição de elite e um da partição não-elite.

No BRKGA-QL de [Chaves et al., 2022], o algoritmo *Q-Learning* [Watkins, 1989] foi utilizado para controlar os parâmetros do BRKGA durante o processo evolutivo. O *Q-Learning* é um algoritmo de aprendizado por reforço independente de modelos que utiliza um *framework* de processo de decisão baseado em cadeias de Markov para modelar a interação entre um agente de aprendizado e seu ambiente em termos de estados, ações e recompensas. Através da exploração

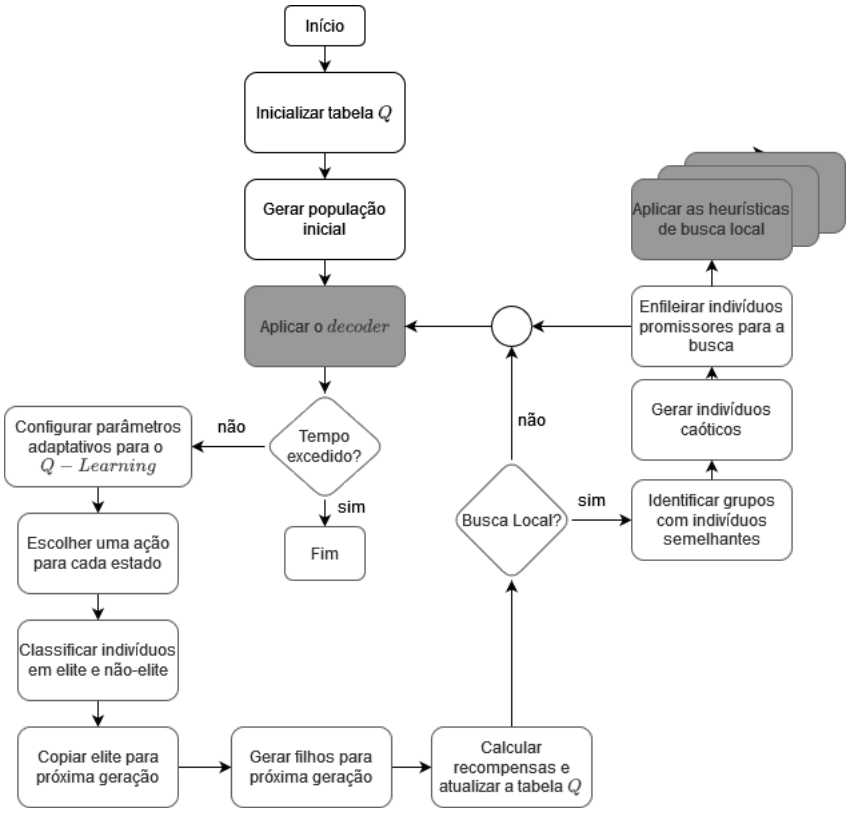
DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br

iterativa e avaliação de todas as ações possíveis em vários estados, o agente adquire *insights* sobre as escolhas ótimas para um problema específico. Tais avaliações são armazenadas em forma de tabela, chamada de tabela Q, que é inicializada com valores zero.

O BRKGA-QL possui um conjunto de estados S (parâmetros) e ações H que representam os valores dos parâmetros. Há apenas um estado para cada parâmetro do BRKGA, e o valor $Q(s_e, h_l)$ representa o valor de usar $[s_e, h_l]$. Os parâmetros são selecionados da tabela Q em cada geração usando uma política ϵ-gulosa. O agente tem como objetivo maximizar sua recompensa total, com a política ϵ-gulosa selecionando a ação com a maior recompensa da tabela Q no estado s_e com uma probabilidade de $1 - \epsilon$, ou qualquer outra ação caso contrário.

A Figura 1 ilustra o fluxograma do algoritmo BRKGA-QL. O *decoder* e a busca local são os únicos componentes dependentes do problema dentro deste *framework*. Todos os outros elementos podem ser encapsulados dentro de uma estrutura geral e independente do problema. Em comparação com os demais métodos baseados no BRKGA, novas características foram introduzidas em Chaves et al. [2022]. Estas incluem o ajuste adaptativo de parâmetros do *Q-Learning*, projetado para equilibrar a exploração e a intensificação durante o processo evolutivo no agente. Além disso, a introdução de indivíduos caóticos gerados a partir de comunidades e uma modificação na forma do operador de mutação usando o *PUX*, substituindo o uso de mutantes.

Figura 1: *Framework* do BRKGA-QL. Baseado em Chaves et al. [2022].



3.1. Decoder

Primeiramente, definimos o vetor de chaves aleatórias, X, como sendo de tamanho p. Cada chave aleatória será usada para determinar uma facilidade da lista de vértices para ser aberta. O *decoder* produz uma lista S contendo p facilidades selecionadas. Inicialmente, o algoritmo cria uma lista de candidatos (C) contendo todos os possíveis vértices do grafo. Em seguida, itera sobre

DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br

o número de facilidades desejadas (p), selecionando um índice i da lista de candidatos com base nos valores fornecidos em X através da fórmula $\lfloor X(i) \times |C| \rfloor$. A facilidade correspondente a esse índice é adicionada à lista S, e o vértice correspondente é removido da lista de candidatos. Esse processo é repetido até que todas as facilidades necessárias sejam selecionadas. O Algoritmo 1 apresenta o pseudocódigo do *decoder*.

Algorithm 1 *Decoder* para o αNpCP

Dados de entrada: X, n, p
Dados de saída: Lista S com p facilidades
Inicializa a lista de candidatos: $C \leftarrow \{1, 2, \ldots, n\}$
for $i = 1, \ldots, p$ **do**
    $indice \leftarrow \lfloor X(i) \times |C| \rfloor$
    $S \leftarrow C(indice)$
    $C \leftarrow C \setminus \{C(indice)\}$
end for

Após obtida a lista de centros a serem abertas a partir de um cromossomo X, o valor de aptidão (*fitness*, $f(X)$) desse indivíduo é calculado. Para cada $i \in V \backslash S$, são atribuídas as α facilidades de S mais próximas do vértice i. Desta forma, no geral, $f(X) = \max_{i \in V \backslash S} d_\alpha^X(i, S)$.

Em [Chagas et al., 2024], uma função objetivo alternativa é proposta. Para seu cálculo são necessários os vetores v e d, cujas dimensões são iguais à quantidade de valores de distâncias distintas da matriz D. Isto é, se o grafo G contém 200 arestas nas quais cada uma tem uma distância diferente da outra, temos que v e d terão dimensão igual a 200. O vetor v contabiliza, para cada distância do grafo inicial, a quantidade de arestas que têm aquela distância específica. O vetor d realiza a mesma contagem, porém, considerando o grafo obtido por uma dada solução do problema. O Algoritmo 2 ilustra o processo para obter a função objetivo alternativa.

Algorithm 2 Função Objetivo Alternativa: Fonte: [Chagas et al., 2024]

Dados de entrada: X, v, d
Dados de saída: Função objetivo alternativa
$\delta \leftarrow 0$
for $i = 0$ **to** $f(X)$ **do**
    if $d_i > 0$ **then**
        $\delta \leftarrow \frac{\delta + d_i}{v_i}$
    end if
end for
Retorne: $f(X) \leftarrow f(X) + \delta$

Neste trabalho, a *fitness* de um indivíduo X será considerada usando o Algoritmo 2. Isso se deve ao fato de que se dois indivíduos possuem mesmo valor *fitness*, então o δ fornece um critério de desempate interessante baseado no número de arestas com a maior distância cada solução tem. Dados dois indivíduos X e X', se $f(X) = f(X')$ enquanto $\delta_X > \delta_{X'}$, então tem-se que X' possui menos arestas com valor $f(X')$ (arestas extremas). Desta forma, está mais "próxima" de ser melhorada em relação ao indivíduo X.

A seguir, discutimos as buscas locais implementadas. Notamos que as buscas locais são realizadas no domínio do problema e, portanto, uma função de *encoding* é necessária para transformar a solução do problema novamente em chaves aleatórias. Dado um indivíduo X, o *encoding*

DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br

determina a posição dos centros $i \in S$ no vetor de índices C. Suponha que $p = 3$ e que tenhamos abertos os centros 30, 18 e 20 (nesta ordem) de uma lista de 50 facilidades. Para encontrar as chaves aleatórias X_1, X_2 e X_3 que seriam usadas pelo Algoritmo 1, iteramos sobre o número de centros p. Para $p = 1$, temos que a facilidade escolhida é a de índice 30. Como nenhuma facilidade foi removida da lista de vértices, o índice que X_1 gera deve levar novamente para 30 via fórmula da linha 5 do *decoder*, daí $X_1 = 30/50$. Para $p = 2$, deve-se atentar à quantidade de centros que foram retirados da lista de candidatos e realizar uma realocação de índices. Por exemplo, se algum centro retirado antes do 18 for referente a um índice maior que 18, sua posição na lista de vértices não alteraria. Porém, caso fosse menor, o índice referente à facilidade 18 passaria a ser o 17. Desta forma, $X_2 = \frac{18-k_{18}}{n-p_{18}}$, em que k_{18} se refere à quantidade de centros retirados da lista de candidatos antes do 18 e p_{18} à quantidade total de centros retirados. Neste caso, $k_{18} = 0$ e $p_{18} = 1$. Para $p = 3$, temos $X_3 = \frac{20-k_{20}}{n-p_{20}}$, onde $k_{20} = 1$ ($18 < 20$) e $p_{20} = 2$ (foram retirados dois centros da lista).

3.2. Busca Local

A primeira busca local consiste em abrir e fechar um centro por vez. Inicialmente, a lista de facilidades fechadas é definida como $C = V \backslash S$, onde V é o conjunto total de facilidades e S é o conjunto de facilidades abertas. Enquanto a lista C não estiver vazia, uma facilidade j é selecionada aleatoriamente da lista C. Para cada facilidade aberta i no conjunto S, a função $TrocarFacilidade(X, i, j)$ é chamada para gerar uma nova solução X' onde a facilidade i é substituída por j. Se a nova solução X' tem um valor de função objetivo $f(X')$ menor que o valor da solução atual $f(X)$, ou se as soluções têm o mesmo valor objetivo (desconsiderando o δ de cada uma) mas a nova solução X' tem uma penalidade $\delta_{X'}$ menor que a penalidade δ_X da solução atual, a solução X é atualizada para X' e a busca é finalizada. A solução X otimizada é retornada.

A segunda busca local consiste em identificar as arestas cuja distância corresponde à maior distância na solução (arestas extremas) e, em seguida, selecionar facilidades na tentativa de remover tais arestas. O algoritmo começa inicializando um contador $k = 0$ e continua enquanto $k \leq d(f(X))$, sendo que $d(f(X)$ é quantidade de arestas extremas da solução X. A busca tenta melhorar a solução iterativamente, removendo uma aresta extrema por vez. Primeiramente, a lista de facilidades fechadas é inicializada. Em seguida, para cada facilidade j em $V \backslash S$, verifica-se se a distância $d_\alpha(j, S)$ é menor que o valor da função objetivo $f(X)$. Se a condição for satisfeita, j é adicionada à lista C, que contém as facilidades cuja distância a qualquer centro aberto é menor que $f(X)$. Enquanto a lista C não estiver vazia, uma facilidade j é selecionada aleatoriamente. Para cada facilidade aberta i em S, a função $TrocarFacilidade(X, i, j)$ cria uma nova solução X' substituindo a facilidade i por j. Se a nova solução X' apresentar um valor de função objetivo menor que X, a solução atual é atualizada para X' e a busca é interrompida. Se o valor de $f(X')$ for igual a $f(X)$ (desconsiderando o valor de δ), mas a penalidade $\delta_{X'}$ for menor que δ_X, a solução X é atualizada para X', e o contador k é incrementado para tentar remover a próxima aresta extrema. O processo continua até que todas as facilidades tenham sido verificadas ou uma solução melhor seja encontrada. O algoritmo retorna a solução X.

A terceira busca local consiste em abrir e fechar aleatoriamente c centros como uma estratégia de perturbação das soluções do problema, similar ao método utilizado por [Chagas et al., 2024]. Após essa perturbação, a Busca Local 2 é aplicada na solução encontrada. O processo começa iterando sobre a quantidade de facilidades c a serem abertas e fechadas simultaneamente. Para cada c, a solução X' é inicializada como uma cópia de X. Em seguida, ocorre outra iteração com c passos, onde, em cada passo, uma facilidade i em $V \backslash S$ e uma facilidade j em S são selecionadas, e a função $TrocarFacilidade(X', i, j)$ é aplicada para criar uma nova solução X'. Após essas trocas, a busca local 2 é aplicada para refinar X'. Se a nova solução X' resultar em um valor de função objetivo ($f(X') - \delta_{X'}$) menor que o da solução atual, X é atualizado para X' e a busca é

DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br

interrompida. Caso contrário, se os valores das funções objetivo forem iguais, mas a penalidade $\delta_{X'}$ for menor que δ_X, X ainda é atualizado para X' e a busca também é interrompida. O algoritmo retorna a solução X.

4. Resultados experimentais

Os testes computacionais do BRKGA-QL foram realizados em uma máquina com as seguintes características: processador Intel i9-12900K com 128 GB de memória RAM. Para os testes foram utilizadas 77 instâncias derivadas da *TSPLIB* [Reinelt, 1991] e 40 instâncias da *OR-library* [Beasley, 1990]. Os valores para α foram considerados no conjunto $\{1, 2, 3\}$. Usamos 24 *threads* e um tempo limite de 1800 segundos para cada instância. A qualidade das soluções do BRKGA-QL foram comparadas com as soluções do modelo (1)-(6), do VNS de [Chagas et al., 2024] e do GRASP-SO de [Sánchez-Oro et al., 2022]. Para uma comparação justa em termos de tempo computacional para obter as melhores soluções de cada método, os tempos apresentados em [Chagas et al., 2024] foram multiplicados por 1.28 [PassMark Software Pty Ltd, 1998].

A Tabela 1 resume os resultados dos métodos comparados nesta seção. Na primeira coluna, são listados os conjuntos de instâncias utilizadas, enquanto na segunda coluna são apresentados os valores de α adotados nos testes. O restante da tabela detalha, para cada método examinado nesta seção, a média das soluções obtidas, a quantidade de soluções iguais às melhores soluções conhecidas (denominadas como BKS, do inglês "*Best Known Solutions*") e o tempo médio (em segundos) necessário para alcançar as melhores soluções.

Tabela 1: Resumo dos resultados.

Instância	α	Modelo			GRASP-SO			VNS			BRKGA-QL		
		média	#BKS	t(s)	média	#BKS	t(s)	média	#BKS	t(s)	média	#BKS	t(s)
OR-library	1	37,33	40	468,48	–	–	–	37,33	40	44,21	37,33	40	78,84
OR-library	2	54,65	22	5506,17	–	–	–	45,55	38	5,27	45.6	36	87,00
OR-library	3	60,98	18	6370,26	–	–	–	51,10	40	9,66	51.1	40	81,00
TSPLIB	1	2153.49	55	3738,23	505,43	11	836	481,70	60	508,71	485,99	31	329,55
TSPLIB	2	4515,61	36	5743,25	773,35	5	989,89	732,60	76	518,35	739,92	32	425,50
TSPLIB	3	4881,12	26	6000	997,18	2	1469,16	945,06	74	570,44	954,21	27	565,60

Para as instâncias da *OR-library*, as soluções do BRKGA-QL foram comparadas com as soluções do modelo matemático e do VNS de [Chagas et al., 2024]. Para $\alpha = 1$, todos os métodos conseguiram alcançar a solução ótima de cada instância. O BRKGA-QL obteve um tempo médio de $78,84$ segundos para encontrar suas melhores soluções. O tempo médio do modelo foi de $468,48$ segundos e do VNS foi de $44,21$ segundos. Para $\alpha = 3$, o BRKGA-QL e o VNS obtiveram a solução ótima para todas as instâncias. O tempo médio do BRKGA-QL foi de 81 segundos. O tempo médio do modelo foi de $6370,26$ segundos e do VNS foi de $9,66$ segundos.

A Figura 2 apresenta os $box\ plots$ para as instâncias da *OR-library* com $\alpha = 2$. Como esperado, o modelo matemático apresenta maior dispersão dos resultados para instâncias grandes. Das 40 instâncias, o BRKGA-QL não conseguiu encontrar a mesma solução que o VNS apenas em duas. O tempo médio do BRKGA-QL foi de 87 segundos. O tempo médio do modelo foi de $5506,17$ segundos e do VNS foi de $5,27$ segundos.

As Figuras 3a e 3b exibem os $box\ plots$ para as instâncias derivadas da *TSPLIB* com $\alpha = 1$. Na Figura 3a são apresentados os resultados de desempenho do BRKGA-QL em comparação com a função objetivo do modelo, o VNS e o GRASP-SO. Já na Figura 3b, o BRKGA-QL é comparado apenas com o VNS e o GRASP-SO. Observa-se que o modelo apresentou alguns valores discrepantes em relação à distribuição de resultados, enquanto os demais métodos obtiveram resultados semelhantes. Em termos gerais, o BRKGA-QL teve um *gap* médio de $2,98\%$, alcançado em um tempo médio de $329,55$ segundos. Por outro lado, o modelo apresentou um *gap* de $17,72\%$,

DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br

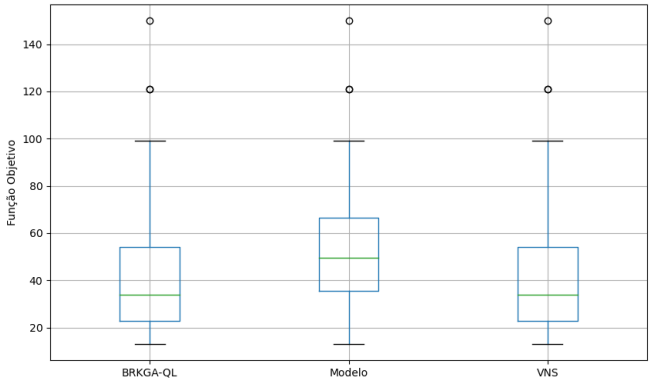
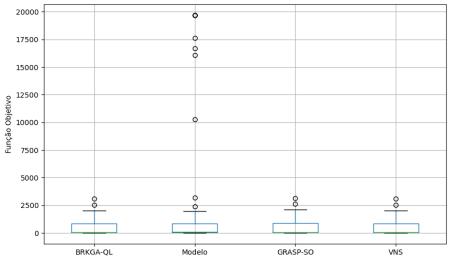
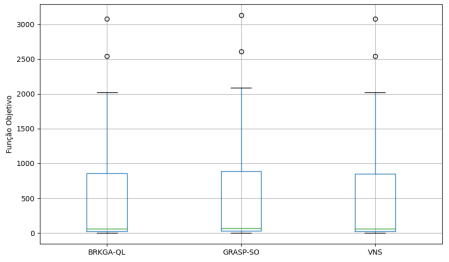


Figura 2: Resultados das instâncias *OR-library* com $\alpha = 2$.

com um tempo médio de execução de $3738,23$ segundos. Quanto ao GRASP-SO e ao VNS, seus *gaps* médios foram de $5,63\%$ e $0,29\%$, respectivamente, com tempos médios de execução de 836 e $508,71$ segundos, respectivamente.



(a) *Box plots* dos resultados para o modelo e os métodos BRKGA-QL, GRASP-SO e VNS.



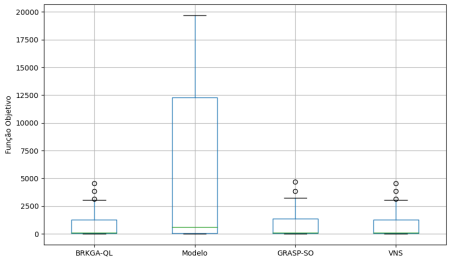
(b) *Box plots* dos resultados para os métodos BRKGA-QL, GRASP-SO e VNS.

Figura 3: Comparação para as instâncias da *TSPLIB* com $\alpha = 1$.

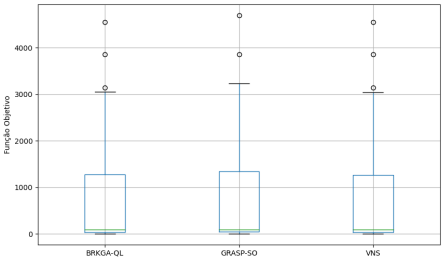
As Figuras 4a e 4b mostram os $box\ plots$ para as instâncias derivadas da *TSPLIB* com $\alpha = 2$. Na Figura 4a, são exibidos os resultados de desempenho do BRKGA-QL em comparação com a função objetivo do modelo, o VNS e o GRASP-SO. Por outro lado, a Figura 4b compara o BRKGA-QL apenas com o VNS e o GRASP-SO. Observa-se que o modelo apresentou os piores resultados para várias instâncias, conforme evidenciado pelos valores discrepantes no diagrama. Uma análise mais detalhada da Figura 4b revela que o GRASP-SO obteve resultados inferiores, como indicado pela maior discrepância no diagrama. Em termos gerais, o BRKGA-QL teve um *gap* médio de $1,46\%$, alcançado em um tempo médio de $425,5$ segundos. Por outro lado, o modelo apresentou um *gap* de $52,63\%$, com um tempo médio de execução de $5743,25$ segundos. Quanto ao GRASP-SO e ao VNS, seus *gaps* médios foram de $8,36\%$ e 0%, respectivamente, com tempos médios de execução de $989,89$ e $518,35$ segundos, respectivamente.

As Figuras 5a e 5b mostram os $box\ plots$ para as instâncias derivadas da *TSPLIB* com $\alpha = 3$. o Padrão de exibição das figuras é semelhante à exibição das figuras 3a, 3b, 4a e 4b. De

DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br



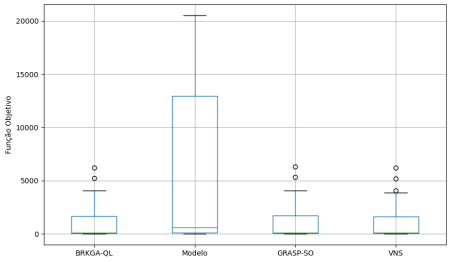
(a) *Box plots* dos resultados para o modelo e os métodos BRKGA-QL, GRASP-SO e VNS.



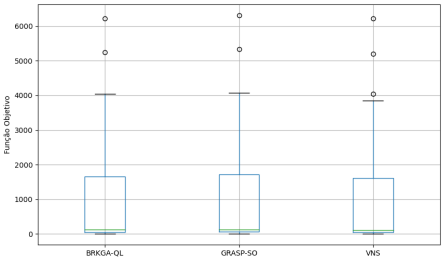
(b) *Box plots* dos resultados para os métodos BRKGA-QL, GRASP-SO e VNS.

Figura 4: Comparação para as instâncias da *TSPLIB* com $\alpha = 2$.

forma análoga a análise para o caso com $\alpha = 2$, observa-se que o modelo apresentou os piores resultados para várias instâncias e que o GRASP-SO obteve resultados inferiores ao BRKGA-QL e o VNS. Em termos gerais, o BRKGA-QL teve um *gap* médio de $1,12\%$, alcançado em um tempo médio de $565,60$ segundos. Por outro lado, o modelo apresentou um *gap* de $60,99\%$, com um tempo médio de execução de 6000 segundos. Quanto ao GRASP-SO e ao VNS, seus *gaps* médios foram de $7,46\%$ e 0%, respectivamente, com tempos médios de execução de $1469,16$ e $570,44$ segundos, respectivamente.



(a) *Box plots* dos resultados para o modelo e os métodos BRKGA-QL, GRASP-SO e VNS.



(b) *Box plots* dos resultados para os métodos BRKGA-QL, GRASP-SO e VNS.

Figura 5: Comparação para as instâncias da *TSPLIB* com $\alpha = 3$.

5. Conclusão

Este trabalho apresentou uma aplicação do BRKGA com Q-*learning* para o controle de parâmetros, visando resolver o Problema de p-Centros com α-Vizinhos. Além disso, foram apresentados um procedimento de decodificação e três métodos de busca local. A avaliação dos resultados foi realizada utilizando instâncias derivadas das bases de dados da *OR-library* e da *TSPLIB*. O desempenho do BRKGA-QL foi comparado com métodos da literatura que se baseiam em modelos matemáticos, bem como com as metaheurísticas GRASP e VNS. Os resultados demonstraram que o BRKGA-QL é robusto na resolução das instâncias e competitivo em termos de tempo médio computacional para encontrar as melhores soluções.

DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br

Como sugestões para trabalhos futuros, considera-se a investigação de novas estratégias de busca local para o BRKGA-QL, ou a extensão do procedimento de decodificação para abordar o Problema de p-Centros com α-Vizinhos considerando a capacidade dos centros.

Agradecimentos

Eduardo M. Silva agradece a Fundação de Amparo à Pesquisa do Estado de São Paulo, (FAPESP) (processo 23/04588-4) e Antonio A. Chaves agradece a FAPESP (processos 2018/15417-8 e 2022/05803-3), e também ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) (processos 423694/2018-9 e 303736/2018-6).

Referências

Alabas-Uslu, C. e Dengiz, B. (2014). A self-adaptive heuristic algorithm for combinatorial optimization problems. *International Journal of Computational Intelligence Systems*, 7:827–852.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160.

Beasley, J. E. (1990). Or-library: distributing test problems by electronic mail. *J. Oper. Res. Soc.*, 41:1069–1072.

Bengio, Y., Lodi, A., e Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421. ISSN 0377-2217. URL https://www.sciencedirect.com/science/article/pii/S0377221720306895.

Callaghan, B., Salhi, S., e Brimberg, J. (2019). Optimal solutions for the continuous p-centre problem and related-neighbour and conditional problems: a relaxation-based algorithm. *Journal of the Operational Research Society*, 70:192–211.

Chagas, G. O., Lorena, L. A., dos Santos, R. D., Renaud, J., e Coelho, L. C. (2024). A parallel variable neighborhood search for $alpha$-neighbor facility location problems. *Computers & Operations Research*, 165:106589.

Chaudhuri, S., Garg, N., e Ravi, R. (1998). The p-neighbor k-center problem. *Inform. Process. Lett.*, 65:131–134.

Chaves, A., Gonçalves, J. F., e Lorena, L. (2018). Adaptive biased random-key genetic algorithm with local search for the capacitated centered clustering problem. *Computers & Industrial Engineering*, 124:331–346.

Chaves et al. (2022). Hybrid metaheuristic for an integrated dial-a-ride problem with common carrier and public transportation. Technical report, University of Montreal (CIRRELT).

Chen, D. e Chen, R. (2013). Optimal algorithms for the alpha-neighbor p-center problem. *European Journal of Operational Research*, 225:36–43.

Eiben, A., Michalewicz, Z., Schoenauer, M., e E., S. J. (2007). Parameter control in evolutionary algorithms. *Lobo F.G., Lima C.F., Michalewicz Z. (eds) Parameter Setting in Evolutionary Algorithms. Studies in Computational Intelligence*, 30:19–46.

DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br

Gonçalves, J. F. e Resende, M. (2011). Biased random-key genetic algorithms for combinatorial optimization. *J Heuristics*, 17:487–525.

Hakimi, S. (1964). Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12:450–459.

Hakimi, S. (1965). Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13:462–475.

Khuller, S., Pless, R., e Sussmann, Y. J. (2000). Fault tolerant k-center problems. *Theoretical Computer Science*, 242:237–245.

Krumke, S. (1995). On a generalization of the p-center problem. *Inform. Process. Lett.*, 56:67–71.

Li, K., Zhang, T., Wang, R., Wang, Y., Han, Y., e Wang, L. (2022). Deep reinforcement learning for combinatorial optimization: Covering salesman problems. *IEEE Transactions on Cybernetics*, 52(12):13142–13155. ISSN 2168-2275. URL `http://dx.doi.org/10.1109/TCYB.2021.3103811`.

Mousavi, S. R. (2023). Exploiting flat subspaces in local search for p-center problem and two fault-tolerant variants. *Computers & Operations Research*, 149:106023.

Panteli, A., Basilis, B., e Giannikos, I. (2021). On solving the multiple p-median problem based on biclustering. *Oper. Res.*, 21:775–799.

PassMark Software Pty Ltd (1998). Cpu benchmarks. `https://www.cpubenchmark.net/`. Acessado: 2024-10-06.

Reinelt, G. (1991). Tsplib - a traveling salesman problem library. *ORSA J. Comput.*, 3(4):267–384.

Sánchez-Oro, J., López-Sánchez, A. D., Herná, A. G., e Duarte, A. (2022). Grasp with strategic oscillation for the α-neighbor p-center problem. *European Journal of Operational Research*, 303:143–158.

Sevaux, M., Sörensen, K., e Pillay, N. (2015). Adaptive and multilevel metaheuristics.

Spears, W. M. e De Jong, K. A. (1991). On the virtues of parametrized uniform crossover. *Proc. 4th Int. Conf. Genetic Algorithms, R. Belew and L. Booker, Eds. San Mateo, CA: Morgan Kaufmann*, p. 230–236.

Sutton, R. S. e Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford.

Çalık, H., Labbé, M., e Yaman, H. (2019). p-center problems. In *Location Science*, p. 51–65. Springer, Cham.

DOI: 10.59254/sbpo-2024-193541
https://proceedings.science/p/193541?lang=pt-br