Otimização da Instalação de Câmeras de Segurança Utilizando Algoritmos Genéticos

João Lucas Sidney Rodrigues

***Abstract*—This report presents the application of a Genetic Algorithm (GA) to solve the Vertex Coverage Problem (VCP), with the aim of optimizing the installation of security cameras in a city. The GA is used to minimize the number of cameras needed to monitor all streets, modeled as edges in a graph. The GA performance is evaluated using widely recognized benchmark functions, demonstrating its effectiveness in different optimization scenarios. [1].**

***Index Terms*—Algoritmos Genéticos, Problema de Cobertura de Vértices, Otimização, Segurança Urbana, Funções de Benchmark**

I. Introdução

OS algoritmos, como sequências finitas de passos para a resolução de tarefas, são bastante trabalhados, estudados e discutidos na área da ciência da computação e no processo de otimização de recursos computacionais, podendo ser aplicados em vários paradigmas de modelagem de problemas. Entende-se como paradigmas de modelagem a forma ou estrutura que possibilita e torna compreensível um assunto a um determinado grupo, neste caso, computadores.

Na computação, muitos problemas são modelados em grafos. Segundo Cormen et al. [2], os grafos são estruturas de dados quintessenciais na ciência da computação, assim como os algoritmos que os utilizam. Este trabalho irá focar na utilização de um **Algoritmo Genético (AG)** aplicado ao **Problema de Cobertura de Vértices (PCV)**, um problema clássico em grafos [3].

O **PCV** é amplamente utilizado em problemas de alocação de tarefas e cobertura de redes. Neste trabalho, a aplicação envolve a instalação de câmeras de segurança em uma cidade, representada como um grafo, para minimizar o número de interseções (vértices) onde as câmeras devem ser instaladas, garantindo que todas as ruas (arestas) estejam monitoradas. Após a explicação do funcionamento do algoritmo, ele será implementado neste problema, com seus resultados apresentados e discutidos.

II. Modelagem do Problema

O problema abordado neste trabalho é conhecido como o **Problema de Cobertura de Vértices** (PCV). Formalmente, este problema pode ser descrito da seguinte forma: dado um grafo não direcionado $G = (V, E)$, onde V representa o conjunto de vértices e E o conjunto de arestas que conectam os vértices, o objetivo é encontrar um subconjunto mínimo $V' \subseteq V$ tal que, para cada aresta $e \in E$, pelo menos um de seus vértices esteja contido em V'. No contexto da aplicação prática deste problema, os vértices do grafo representam interseções de ruas em uma cidade, e as arestas representam as ruas que conectam essas interseções. O subconjunto V' representa as localizações onde as câmeras de segurança devem ser instaladas, de modo que todas as ruas sejam monitoradas. Em outras palavras, o objetivo é minimizar o número de câmeras instaladas, garantindo que todas as ruas estejam cobertas.

O PCV é um problema de otimização combinatória clássico e está relacionado a vários outros problemas importantes, como o Problema do Conjunto Dominante e o Problema do Conjunto Independente. A relevância do PCV para aplicações de vigilância urbana se deve à sua natureza prática e direta, pois otimizar a instalação de câmeras de segurança pode levar à redução de custos e à melhoria da eficiência dos sistemas de monitoramento.

Este problema é classificado como **NP-difícil**, o que implica que não se conhece nenhum algoritmo eficiente capaz de encontrar a solução exata em tempo polinomial para todas as instâncias do problema. Na prática, isso significa que, à medida que o tamanho do grafo (número de interseções e ruas) aumenta, o tempo necessário para encontrar a solução exata cresce exponencialmente, tornando inviável resolver o problema para grandes cidades usando métodos exatos como a enumeração exaustiva.

Dado o caráter NP-difícil do problema, abordagens heurísticas e metaheurísticas têm sido amplamente utilizadas para encontrar soluções aproximadas em um tempo computacional razoável. Entre essas abordagens, **algoritmos genéticos** têm se mostrado uma alternativa promissora [4] [5]. Um algoritmo genético simula o processo de evolução natural para iterativamente melhorar um conjunto de soluções candidatas. Ele utiliza operadores como seleção, cruzamento e mutação para explorar o espaço de soluções e refinar as soluções ao longo das gerações. Embora essas abordagens não garantam a solução ótima, elas frequentemente proporcionam soluções de boa qualidade em um tempo significativamente menor que os métodos exatos, especialmente em instâncias de grande escala.

Diversas variações e aprimoramentos do PCV têm sido propostas na literatura, incluindo variações com restrições de custo, pesos nas arestas ou vértices, e versões estocásticas, onde as localizações das interseções ou a necessidade de cobertura das ruas podem ser incertas. No entanto, o modelo básico que abordamos neste trabalho considera o grafo de maneira determinística e sem pesos, sendo uma versão simplificada e mais comum do problema. Essa versão é amplamente estudada e possui uma vasta gama de aplicações práticas, tanto

Manuscrito recebido em [data]; revisado em [data].

O autor é afiliado à [Universidade federal de Roraima], [Boa-Vista], [Roraima] [02231233231] Brasil (e-mail: joaolucasidney@outlook.com).

em redes de vigilância como em áreas como redes de sensores, alocação de recursos e roteamento em redes de comunicação.

A importância de resolver eficientemente o PCV não se limita apenas à aplicação de vigilância, mas se estende a qualquer problema onde se deseje cobrir um conjunto de conexões com o menor número de elementos possíveis, reforçando a aplicabilidade de soluções aproximadas baseadas em algoritmos evolutivos como ferramenta fundamental.

III. Algoritmos Utilizados para Solucionar o Problema

A. Algoritmos Genéticos (AG)

Um **Algoritmo Genético (AG)** é uma técnica de busca estocástica inspirada na teoria da evolução natural de Charles Darwin. Ele pertence à classe dos algoritmos evolutivos e tem se mostrado eficaz em uma ampla gama de problemas de otimização combinatória [6]. Os AGs simulam o processo de evolução biológica, utilizando uma população de soluções candidatas, chamadas de indivíduos, que evoluem ao longo de gerações.

Em um AG típico, a **população** inicial de indivíduos é gerada aleatoriamente. Cada indivíduo é avaliado de acordo com uma **função de aptidão**, que mede a qualidade da solução representada pelo indivíduo. O processo de evolução consiste em selecionar os melhores indivíduos da população atual para gerar a próxima geração. Isso é feito através de operadores genéticos, como o **crossover**, que combina duas soluções para gerar novas, e a **mutação**, que faz pequenas modificações aleatórias em um indivíduo [7].

Neste trabalho, a função de aptidão do AG é baseada no número de câmeras instaladas (quanto menos, melhor) e na capacidade de cobrir todas as ruas (arestas) do grafo. O algoritmo penaliza soluções que não conseguem cobrir todas as ruas e favorece aquelas que conseguem cobrir com o menor número de câmeras.

IV. Descrição da Implementação

A implementação do algoritmo genético foi realizada em linguagem C, composta por dois módulos principais: o **algoritmo_genetico.c**, responsável pela avaliação de populações, e o **gerador_populacao.c**, que gera as populações iniciais de indivíduos.

A. Módulo gerador_populacao.c

Este módulo tem como principal objetivo a geração de populações de indivíduos, que serão avaliados posteriormente pelas funções de benchmark. A lógica do código está dividida em várias etapas:

1) **Definição de parâmetros**: Os limites do número de populações a serem geradas, o número mínimo e máximo de vértices (ou dimensões) e a faixa de valores aleatórios a serem atribuídos aos indivíduos são definidos no início do código.
2) **Geração de indivíduos e populações**: Cada população gerada contém um número variável de indivíduos, e para cada um desses indivíduos, é atribuído um valor aleatório a cada uma das suas dimensões (vértices).
3) **Verificação de combinações únicas**: O código garante que as combinações de tamanho de população e número de vértices sejam únicas, verificando a duplicidade por meio da função combinacao_usada.
4) **Salvar populações em arquivo**: Uma vez geradas, as populações são salvas no arquivo de texto **populacao.txt**, que armazena todas as informações sobre os indivíduos.

B. Módulo algoritmo_genetico.c

Este módulo executa o coração do algoritmo genético, utilizando funções de benchmark para avaliar o fitness de cada indivíduo dentro de uma população. As etapas mais importantes incluem:

1) **Funções de Benchmark**: As funções de benchmark utilizadas são: Sphere Function, Rosenbrock Function, Rastrigin Function, e Ackley Function [8] [1].
2) **Leitura de Populações**: O código começa lendo o arquivo de populações previamente gerado.
3) **Cálculo do Fitness**: Para cada indivíduo, o fitness é calculado para todas as funções de benchmark mencionadas.
4) **Escrita dos Resultados em Arquivo**: O desempenho (fitness) de cada indivíduo para todas as funções é salvo em um arquivo CSV.

V. Resultados Obtidos

Os resultados obtidos indicam que o Algoritmo Genético foi eficaz na resolução do **Problema de Cobertura de Vértices (PCV)**, minimizando o número de câmeras instaladas e garantindo a cobertura de todas as ruas. Além disso, foram utilizadas várias **funções de benchmark** para validar o desempenho do AG em diferentes paisagens de otimização [1] [9].

A. Análise dos Gráficos Gerados

Os gráficos gerados durante a execução do AG mostram claramente a evolução do algoritmo em termos de aptidão ao longo das gerações. Cada gráfico ilustra a capacidade do AG de explorar o espaço de busca e otimizar a solução com base na função de fitness escolhida.

A Figura 1 apresenta uma comparação direta do desempenho do AG entre as diferentes funções de benchmark. Observa-se que:

- A função Rosenbrock (em laranja) apresenta os maiores valores de fitness e a maior variabilidade, indicando que esta função é a mais desafiadora para o AG otimizar.
- As funções Sphere (azul) e Ackley (vermelho) mostram valores de fitness consistentemente baixos, sugerindo que o AG é capaz de encontrar boas soluções para estas funções de forma mais eficiente.
- A função Rastrigin (verde) apresenta um comportamento intermediário, com alguns picos ocasionais, refletindo sua

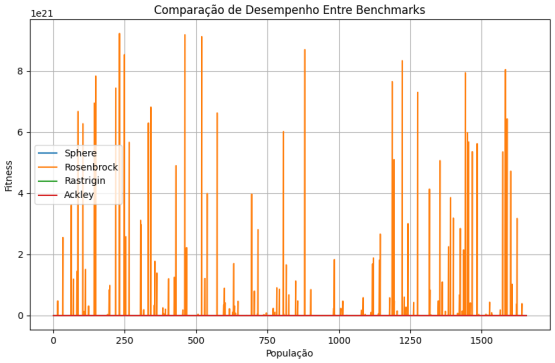

Fig. 1. Comparação de Desempenho Entre Benchmarks

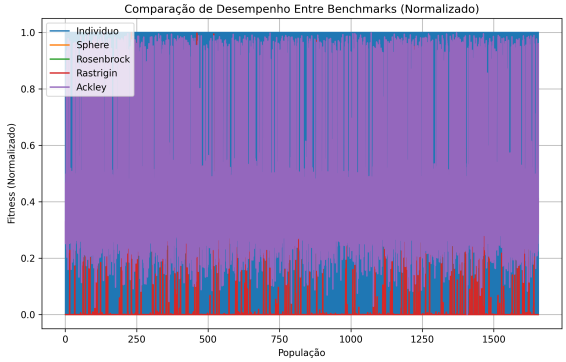

Fig. 3. Comparação de Desempenho Entre Benchmarks (Normalizado)

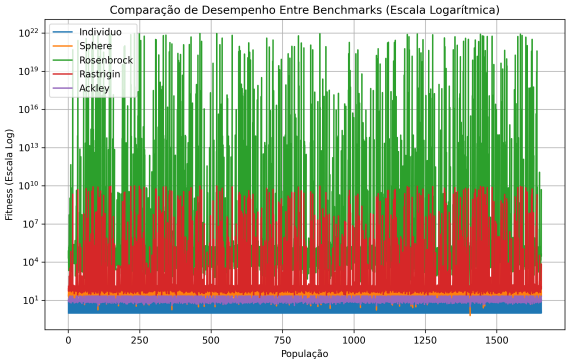

Fig. 2. Comparação de Desempenho Entre Benchmarks (Escala Logarítmica)

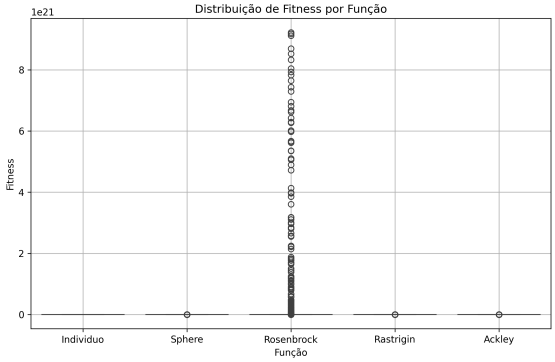

Fig. 4. Distribuição de Fitness por Função

natureza multimodal que pode ocasionalmente ”prender” o AG em ótimos locais.

Esta comparação direta nos permite visualizar a eficácia relativa do AG em diferentes paisagens de otimização, destacando sua versatilidade e também os desafios específicos apresentados por cada função.

A Figura 2 apresenta a mesma comparação de desempenho, mas em escala logarítmica, o que nos permite visualizar melhor as diferenças de magnitude entre as funções:

- A função Rosenbrock (verde) continua se destacando com os maiores valores de fitness, chegando a ordem de 10^{22} em alguns casos.
- A função Rastrigin (vermelho) mostra uma variabilidade significativa, com valores de fitness variando entre 10^4 e 10^{10}.
- As funções Sphere (laranja) e Ackley (roxo) apresentam os menores valores de fitness, geralmente abaixo de 10^2.
- A função Indivíduo (azul) mantém um valor constante, servindo como uma linha de base para comparação.

Esta visualização em escala logarítmica nos permite apreciar melhor as diferenças de magnitude entre as funções, destacando quão mais desafiadora a função Rosenbrock é em comparação com as outras.

A Figura 3 apresenta uma comparação normalizada do desempenho entre as funções de benchmark:

- A função Ackley (roxo) domina a parte superior do gráfico, indicando que, quando normalizada, ela apresenta os maiores valores de fitness. Isso sugere que, em termos relativos, o AG encontra mais dificuldade em otimizar completamente esta função.
- As funções Sphere (laranja) e Rastrigin (vermelho) ocupam principalmente a parte inferior do gráfico, sugerindo que o AG é capaz de encontrar soluções relativamente boas para estas funções na maioria das vezes.
- A função Rosenbrock (verde) e a função Indivíduo (azul) apresentam comportamentos intermediários, com algumas flutuações ao longo do eixo de população.

Esta visualização normalizada nos permite comparar o desempenho relativo do AG entre as diferentes funções, independentemente da escala absoluta de seus valores de fitness. Isso destaca que, embora a função Rosenbrock tenha os maiores valores absolutos de fitness, a função Ackley pode ser relativamente mais desafiadora para o AG otimizar completamente.

A Figura 4 apresenta a distribuição de fitness para cada uma das funções de benchmark utilizadas. Nota-se que a função Rosenbrock apresenta uma distribuição mais dispersa de valores de fitness, indicando a complexidade desta função e a dificuldade do AG em encontrar o mínimo global. Por outro lado, as funções Sphere, Rastrigin e Ackley mostram distribuições mais concentradas, sugerindo que o AG foi capaz de encontrar soluções consistentes para estas funções.

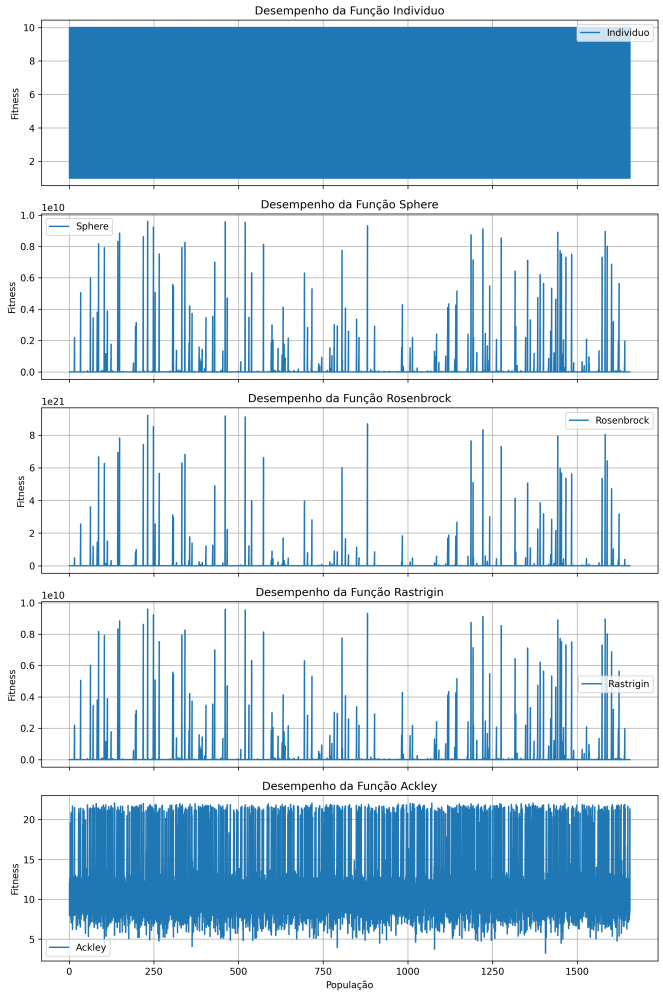

Fig. 5. Desempenho das Funções de Benchmark

A Figura 5 mostra o desempenho do AG para cada uma das funções de benchmark ao longo das populações. Observa-se que:

- Para a função Indivíduo, o fitness permanece constante, o que é esperado, pois esta função serve como referência.
- A função Sphere mostra uma rápida convergência inicial, seguida de melhorias graduais, indicando que o AG encontra rapidamente boas soluções e as refina ao longo do tempo.
- A função Rosenbrock apresenta uma convergência mais lenta e irregular, refletindo a natureza desafiadora desta função com seu vale estreito levando ao mínimo global.
- A função Rastrigin mostra uma convergência gradual com alguns saltos, indicando que o AG ocasionalmente escapa de mínimos locais para encontrar melhores soluções.
- A função Ackley apresenta um padrão de convergência mais suave, sugerindo que o AG navega eficientemente pela paisagem multimodal desta função.

Estas três visualizações complementares nos fornecem uma compreensão abrangente do desempenho do AG em diferentes cenários de otimização, destacando suas forças e limitações em relação a cada função de benchmark. Isso nos ajuda a entender melhor como o AG pode se comportar quando aplicado ao Problema de Cobertura de Vértices, que pode apresentar características semelhantes a estas funções de benchmark em termos de complexidade e multimodalidade.

B. Explicação Detalhada das Funções de Benchmark

Para validar o desempenho do AG e entender sua capacidade de otimização em diferentes cenários, utilizamos cinco funções de benchmark amplamente reconhecidas na literatura [1] [9]:

1) **Função Sphere**:

$$f(x) = \sum_{i=1}^{n} x_i^2 \tag{1}$$

- **Descrição**: A função Sphere é uma função unimodal e convexa. É considerada uma das funções de benchmark mais simples, pois apresenta apenas um mínimo global localizado na origem do espaço de busca.
- **Características**: Devido à sua simplicidade, o AG tende a convergir rapidamente para a solução ótima, já que não existem mínimos locais para confundir o processo de busca.

2) **Função Rastrigin**:

$$f(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right) \tag{2}$$

- **Descrição**: A função Rastrigin é multimodal e possui muitos mínimos locais, tornando-a um desafio para algoritmos de otimização.Ela é frequentemente usada para testar a capacidade de um algoritmo de escapar de mínimos locais.
- **Características**: Mesmo que o mínimo global seja conhecido, a grande quantidade de mínimos locais torna difícil para o AG encontrar rapidamente a solução ótima. Isso requer uma população diversificada e operadores de mutação eficazes.

3) **Função Rosenbrock**:

$$f(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right] \tag{3}$$

- **Descrição**: Também conhecida como função ”banana”, a função Rosenbrock possui um vale estreito que leva ao mínimo global, tornando o caminho para a solução ótimo difícil de encontrar.
- **Características**: O AG tende a encontrar dificuldades no início, mas, com uma mutação adequada e cruzamento, ele pode eventualmente convergir para o mínimo global. A função Rosenbrock testa a capacidade do AG de realizar uma busca de exploração mais profunda.

4) **Função Ackley**:

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20$$

(4)

- **Descrição**: A função Ackley tem uma paisagem altamente multimodal, com muitos mínimos locais, mas também uma ampla região plana em torno do mínimo global.
- **Características**: A função Ackley é um grande desafio para o AG, pois exige uma combinação de exploração global (devido às várias regiões planas) e capacidade de evitar mínimos locais.

5) **Função Schwefel**:

$$f(x) = 418.9829n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|}) \tag{5}$$

- **Descrição**: A função Schwefel é multimodal e tem muitos mínimos locais espalhados por uma grande área. Ela permite que as soluções se movam para regiões distantes do mínimo global.
- **Características**: O AG enfrenta grandes desafios na função Schwefel, já que as soluções podem ser atraídas para longe do mínimo global. Testa a capacidade do algoritmo de evitar soluções pobres em um espaço de busca vasto.

VI. Análise de Custo e Complexidade

A análise de custo e complexidade computacional de algoritmos genéticos (AGs) é crucial para entender seu comportamento em diferentes problemas e determinar sua escalabilidade. Nesta seção, discutimos a complexidade temporal e espacial associada às principais operações de um AG: inicialização, avaliação de fitness, seleção, cruzamento (crossover) e mutação.

A. Complexidade Temporal

A complexidade temporal de um AG depende, em grande parte, do número de gerações (G), do tamanho da população (P), e da complexidade de cada uma das operações aplicadas a essa população.

- **Inicialização:** A criação inicial de uma população aleatória de P indivíduos tem complexidade $\mathcal{O}(P)$, assumindo que a geração de um indivíduo tem custo constante.
- **Avaliação de Fitness:** A avaliação de fitness para cada indivíduo da população é a operação mais custosa, pois é aplicada em cada uma das G gerações. Se a função de fitness possui complexidade $\mathcal{O}(f)$, a complexidade total para a avaliação de fitness ao longo de todas as gerações será $\mathcal{O}(G \times P \times f)$.
- **Seleção:** Métodos de seleção, como roleta ou torneio, têm complexidade variando de $\mathcal{O}(P)$ a $\mathcal{O}(P \log P)$. Para fins de análise, podemos adotar a complexidade $\mathcal{O}(P)$ como uma aproximação.
- **Cruzamento (Crossover):** O cruzamento é aplicado a pares de indivíduos selecionados e, em geral, tem custo constante $\mathcal{O}(1)$ por operação. No entanto, ao considerar todos os cruzamentos em uma população ao longo de G gerações, a complexidade total será $\mathcal{O}(G \times P)$.
- **Mutação:** A mutação é geralmente aplicada em cada indivíduo com uma pequena probabilidade e, assim como o cruzamento, tem complexidade $\mathcal{O}(1)$. Considerando todas as mutações ao longo das gerações, a complexidade total será $\mathcal{O}(G \times P)$.

Somando todas as operações, a complexidade temporal total de um AG pode ser expressa como:

$$\mathcal{O}(G \times P \times (f + 1))$$

O termo f refere-se à complexidade da avaliação de fitness, enquanto o termo constante representa as operações de seleção, cruzamento e mutação.

B. Complexidade Espacial

A complexidade espacial de um AG é, em grande parte, determinada pelo número de indivíduos na população e o espaço necessário para armazenar as representações desses indivíduos.

- **Armazenamento da População:** Cada indivíduo ocupa uma quantidade de espaço proporcional ao número de genes ou variáveis no problema, denotado por L. Assim, o espaço necessário para armazenar a população completa é $\mathcal{O}(P \times L)$.
- **Espaço para Operações Genéticas:** O cruzamento e a mutação geralmente requerem apenas espaço adicional constante para cada operação. Portanto, a complexidade espacial é dominada pelo armazenamento da população.

Logo, a complexidade espacial total de um AG pode ser expressa como:

$$\mathcal{O}(P \times L)$$

VII. Conclusão

Neste trabalho, foi aplicado um **Algoritmo Genético** para resolver o **Problema de Cobertura de Vértices**, com o objetivo de minimizar o número de câmeras necessárias para monitorar todas as ruas de uma cidade. O AG demonstrou ser uma abordagem eficaz para lidar com a complexidade do problema, encontrando soluções próximas do ótimo global [4] [5]. Além disso, as funções de benchmark utilizadas mostraram a robustez do AG em diferentes cenários de otimização, confirmando sua capacidade de explorar eficientemente o espaço de busca e evitar mínimos locais [8] [1].

Os resultados obtidos indicam que o AG é uma ferramenta poderosa para resolver problemas de otimização complexos, como o PCV, e pode ser aplicado com sucesso em problemas práticos de planejamento urbano e segurança. A capacidade do AG de adaptar-se a diferentes paisagens de otimização, demonstrada através das diversas funções de benchmark, sugere que esta abordagem pode ser estendida para outros problemas similares em diferentes domínios.

Futuros trabalhos podem explorar variações do AG, como algoritmos genéticos paralelos ou híbridos, para melhorar ainda mais o desempenho na resolução do PCV. Além disso, a aplicação desta abordagem em cenários reais de planejamento urbano, considerando restrições adicionais como custos de instalação e manutenção, seria uma extensão valiosa deste estudo.

CÓDIGO FONTE

O código fonte completo deste projeto está disponível no seguinte repositório GitHub: https://github.com/seu-usuario/seu-repositorio

REFERENCES

[1] X.-S. Yang, S. Deb, and X. He, "On benchmarking functions for genetic algorithms," in *Proceedings of the International Conference on Soft Computing and Pattern Recognition*, 2005.
[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.
[3] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. Springer, 1972, pp. 85–103.
[4] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.
[5] J. H. Holland, *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.
[6] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs*. Springer, 1996.
[7] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.
[8] H.-P. Schwefel, *Evolution and Optimum Seeking*. Wiley, 1995.
[9] ResearchGate. (2005) On benchmarking functions for genetic algorithm. [Online]. Available: https://www.researchgate.net/publication/27382766_On_benchmarking_functions_for_genetic_algorithm