Trabalho Prático de Heurísticas e Meta-heurísticas

Lucas Rômulo[1], Paulo Gabriel[1]

[1]Universidade Federal de São João del-Rei (UFSJ)

1. Introdução

Com o crescente aumento da criminalidade em nosso país, a utilização de câmeras de segurança tornou-se um utensílio indispensável para grandes empresas e instituições públicas. No entanto, deve-se ressaltar que existe um custo relativamente alto no processo de instalação e manutenção das mesmas. Dessa forma, é essencial que haja um planejamento para determinar quais são os melhores pontos para instalação de câmeras, garantindo a segurança do perímetro.

Um dos problemas que podem surgir durante o processo de instalação de câmeras é determinar quais serão os locais de instalação. Um bom posicionamento pode reduzir os custos de implementação e garantir cobertura total sobre o perímetro. Esse problema citado é conhecido na literatura como *Optimal Camera Placement problem (OCP)*, tendo como objetivo identificar o menor conjunto de locais de instalação de câmeras que garantem a cobertura total do local.

Em busca de uma otimização no manejamento dessas câmeras, uma solução inicial será gerada utilizando uma heurística construtiva aleatória com validação. Futuramente, as soluções geradas serão aprimoradas por meio de um algoritmo genético, que é uma técnica de busca local interessante para problemas np-completo e np-difícil. O código desenvolvido está disponível no github.

2. Modelagem

2.1. Função Objetivo

O objetivo do algoritmo proposto é determinar o menor conjunto de locais de instalação de câmeras de segurança que asseguram a cobertura de todos os pontos de vigilância da instância do problema. A qualidade das soluções encontradas serão mensuradas utilizando a equação 1,

$$\min f(s) = (\sum_{i=i}^{n} C[i]) \tag{1}$$

em que C é o cromossomo binário e n é a quantidade de locais de instalação de câmeras.

Todas as soluções inválidas encontradas pelo algoritmo são reparadas. Assim, não é necessário utilizar penalizações na função objetivo .

2.2. Estrutura de Dados

De forma geral, o algoritmo proposto utiliza apenas duas estruturas de dados, que foram denominadas de *spot_t* e *solution_t*.

A estrutura *spot_t* é um tipo abstrato de dados (TAD) que será responsável por armazenar todas as informações pertinentes aos pontos que devem ser cobertos

pela solução. Em suma, ela armazena o identificador do ponto e a lista de câmeras que cobrem esse respectivo ponto. Assim, cada um dos pontos de cobertura possuirá uma estrutura de armazenamento nesse formato.

```
typedef struct spot_t {
  int id;
  int num_cams;
  int* cams;
} spot_t;
```

As soluções geradas pela heurística construtiva serão armazenadas na estrutura *solution_t*. A solução será representada no formato de vetor binário onde as posições (índice no vetor) que contém "1" indicam quais serão as câmeras instaladas. Também temos um vetor binário que indica quais os pontos que foram cobertos pela solução encontrada. Se todas as posições desse vetor conterem "1", então a solução é válida. Por último, temos também uma variável que armazena a quantidade de câmeras utilizadas na solução. Como a solução inicial será feita por meio da heurística aleatória com garantia de viabilidade, o número de câmeras utilizadas será o próprio *profit* da solução.

```
typedef struct solution_t {
  int cost;
  int fitness;
  int num_cams;
  int num_spots;
  int* binary_solution;
  int* coverage_spots;
} solution_t;
```

2.3. Algoritmo Construtivo

A utilização de heurísticas construtivas auxilia na criação rápida de soluções iniciais para serem utilizadas nas meta-heurísticas. A princípio, é utilizada uma heurística que cria soluções viáveis de forma aleatória. Seu funcionamento é simples: enquanto a solução gerada não é uma solução viável, adiciona-se aleatoriamente uma câmera para fazer parte da solução. A lógica da heurística é descrita abaixo.

```
solution_t* random_valid_solution(int num_spots, int num_cams, spot_t**
    spot_list){

  solution_t* solution;
  solution = new_solution(num_spots, num_cams);

  do {

    int cam_id = rand() % num_cams;

    add_cam_to_solution(cam_id, solution, spot_list);

  } while(!validate_solution(solution));

  calc_cost(solution);

  return solution;
}
```

2.4. Algoritmo Genético

O algoritmo genético foi a heurística utilizada para resolução do problema abordado neste trabalho. Os cromossomos, que são as soluções para o problema, utilizam uma representação binária. Dessa forma, ele será um vetor de tamanho igual ao número de locais de instalação de câmeras preenchidos por 0's e 1's. Cada valor 1 no vetor indica que a câmera cujo ID é igual aquele índice do vetor fará parte da solução.

Nos AG's, a aptidão de cada indivíduo deve ser mensurado. Isto será feito por meio da equação 2. Por ser uma função quadrática, as soluções que utilizam grandes quantidades de câmeras terão seu *fitness* penalizado, propiciando que o algoritmo genético opte por escolher as soluções de menor quantidade de câmeras.

$$\max f(s) = \frac{1}{Custo^2} \tag{2}$$

A população inicial é criada utilizando o algoritmo construtivo apresentado anteriormente. Assim, temos a garantia que a solução cobre todos os pontos designados.

Em cada geração, os indivíduos da população atual são selecionados aos pares para dar origem a uma nova população, filhos da geração atual. A escolha desses pais é feita utilizando a técnica de roleta viciada, em que a probabilidade de escolha do indivíduo é proporcional a sua nota.

De posse desse par de pais, a nova população será concebida combinando os seus cromossomos. A estratégia de recombinação utilizada foi o *crossover* em um único ponto. Imediatamente, aplica-se o processo de mutação nos cromossomos dos indivíduos gerados, com o intuito de explorar novas soluções do espaço de busca e fugir de ótimos locais.

O processo de mutação ocorre um pouco diferente do usual. Como o objetivo é minimizar a quantidade de câmeras, a inversão do bit será feito apenas nos elementos que fazem parte da solução, ou seja, somente os bits 1 do indivíduo tem a possibilidade de sofrer essa mutação.

Deve-se ressaltar que tanto o processo de *crossover* quanto o de mutação podem invalidar a solução. Assim, foi implementado uma função de reparação que busca os pontos descobertos da solução e adiciona aleatoriamente uma câmera que o cubra.

A estratégia de substituição utilizada foi o elitismo. É gerada uma nova população em que as soluções participantes são os melhores indivíduos na união da população atual com a população gerada por meio da reprodução.

Por último, o critério de parada é feito por meio da quantidade de gerações estabelecidas a *priori*. Abaixo é apresentado um pseudocódigo do fluxo de execução do AG.

```
1 inicio
2     p_atual = GeraPopulacaoInicial()
3     para n geracoes faca
```

```
        p_filha = Crossover(p1)
        Mutacao(p_filha)
        ReparoDaPopulacao(p_filha)
        p_elite = Elitismo(p_atual, p_filha)
        p_atual = AtualizaPopulacaoAtual(p_elite)
    fim_para
fim
```

Listing 1. Fluxo de Execução

3. Resultados

Os resultados de execução do algoritmo genético foram satisfatórios. Por ter sido implementado na linguagem C, seu tempo de execução é relativamente baixo. As execuções de 1000 gerações com 100 indivíduos por população gastaram 1 minuto de tempo de execução.

A melhor solução encontrada utilizou apenas 13 câmeras para cobrir todos os locais alvo. A melhor solução da população inicial normalmente utilizava entre 25 e 40 câmeras. O algoritmo foi capaz de reduzir consideravelmente a quantidade de soluções encontradas.

Tabela 1. Resultados de Execução do AG

Execução	Tamanho População	Gerações	Prob. Mutação	Câmeras	Geração da Solução	Tempo (s)
01	10	100	10%	17	15	≈ 0
02	10	100	10%	15	21	≈ 0
03	10	100	10%	18	45	≈ 0
04	20	250	10%	16	18	≈ 3
05	20	250	10%	18	83	≈ 3
06	20	250	10%	18	45	≈ 2
07	30	500	10%	19	92	≈ 8
08	30	500	10%	18	401	≈ 8
09	30	500	10%	15	81	≈ 8
10	40	600	10%	16	133	≈ 14
11	40	600	10%	19	98	≈ 14
12	40	600	10%	18	115	≈ 14
13	100	1000	10%	19	17	≈ 59
14	100	1000	10%	17	669	≈ 60
15	100	1000	10%	20	17	≈ 60

Além disso, ao analisar a tabela 1 percebe-se o comportamento aleatório e ao mesmo tempo direcionado do algoritmo genético. Em algumas iterações, o algoritmo encontrou a solução corrente nas primeiras gerações. Em outras, a solução corrente foi encontrada tardiamente.

4. Considerações Finais

Neste trabalho foi implementado um algoritmo genético para resolução do problema conhecido na literatura como *Optimal Camera Placement problem (OCP)*.

Alguns procedimentos do AG sofreram ajustes para que essa metaheurística fosse capaz de resolver o problema abordado. Usualmente, a mutação por probabilidade inverte os valores do cromossomo binário. Por se tratar de um problema

de minimização, em que as soluções iniciais são válidas, não faz muito sentido inverter os valores de $0 \rightarrow 1$ quando ocorrer a mutação. Isso apenas piora a solução. Dessa forma, optou-se por utilizar a mutação $1 \rightarrow 0$ e implementar uma função de reparação.

Outro ponto importante é a linguagem utilizada. A linguagem C é muito eficiente, porém, oferece poucos recursos para o programador. Consequentemente, o processo de implementação é mais lento pois todas as funcionalidades utilizadas devem ser implementadas.

Por último, os resultados foram satisfatórios tendo em vista que o algoritmo foi capaz de encontrar boas soluções gastando um baixo tempo de execução. O custo de memória é proporcional à quantidade de indivíduos por população, já que estes devem estar na memória naquele período.