INF05010 - Otimização Combinatória - Turma B
2022/1

Trabalho Final

Henrique Zanotto Vazatta - 00303532
Thiago Haab dos Santos - 00301345

Porto Alegre - RS **06/10/2022**

1. INTRODUÇÃO

O objetivo deste trabalho foi implementar a formulação matemática e uma meta-heurística para resolução de um problema de agrupamento de vértices em um grafo (definido na Seção 2). A meta-heurística escolhida foi algoritmos genéticos (definido na Seção 5).

2. PROBLEMA

Dado um grafo não-orientado $G = (V, E)$ com custo $c_a \in \mathbb{Z}$ para cada aresta $a \in E$. A partir desses custos, para qualquer par de vértices v_i, v_j podemos calcular o custo total do menor caminho entre eles denotado por d_{ij}. Considerando $A \subseteq V$ com um tamanho máximo T, tal que $d_{ij} \leq D$ para quaisquer dois vértices $v_i, v_j \in A$, e D fixo encontrando a partição $P = \{A_1, A_2, ..., A_k\}$ do conjunto V que minimiza o número k de subconjuntos.

3. FORMULAÇÃO

3.1 Variáveis:

Sendo B o conjunto dos Booleanos:
$x_{i,j} \in B$: Vértice $i \in [n]$ pertence $j \in [n]$.
$y_j \in B$: No conjunto j.
$z_{ijk} \in B$: Vértice i,j pertence ao conjunto k.

3.2 Função Objetivo:

$$min \sum_{j=i}^{n} y_j$$

3.3 Restrições:

$$x_{ij} \leq y_j \qquad \forall i, j \in [n]$$

$$\sum_{j=i}^{n} x_{ij} = 1 \qquad \forall i \in [n]$$

$$\sum_{i=1}^{n} x_{ij} \leq T \qquad \forall j \in [n]$$

$$z_{ijk} \leq \frac{x_{ik}+x_{jk}}{2} \qquad \forall i,j,k \in [n]$$

$$z_{ijk} \geq x_{ik} + x_{jk} - 1 \qquad \forall i,j,k \in [n]$$

$$d_{ij} \leq D + M(1 - \sum_{k=1}^{n} z_{ijk}) \qquad \forall i,j,k \in [n]$$

$$x_{ij} \in B \qquad \forall i,j \in [n]$$

$$y_{j} \in B \qquad \forall j \in [n]$$

$$z_{ijk} \in B \qquad \forall i,j,k \in [n]$$

4. FORMATO E INSTÂNCIAS

As instâncias foram fornecidas no formato de um arquivo instance_n_m_D_T.dat onde n e m são respectivamente o número de vértices e o número de arestas, D a distância permitida entre dois vértices de um mesmo subconjunto e T o tamanho máximo do subconjunto.

5. IMPLEMENTAÇÃO

Para realizar a implementação da meta-heurística algoritmo genético, utilizamos a linguagem Python 3 com o módulo de grafos i-graph.

Os parâmetros do algoritmo genético são:

- n_gen: Número máximo de gerações
- n_ind: Número de indivíduos
- selection_ratio: Porcentagem de indivíduos selecionados para o torneio
- mutation_chance: Chance de ocorrer mutação
- elitism: Se vai haver elitismo

O algoritmo possui cinco etapas:

1. Geração da população
2. Seleção de indivíduos para participar do torneio
3. Torneio (um para cada pai)
4. Cross-over
5. Mutação

Um indivíduo no nosso algoritmo é uma lista de subconjuntos, em que cada subconjunto consiste de uma lista de nós do grafo. Consequentemente, a população é o conjunto de várias listas de subconjuntos diferentes.

Na função de geração da população, criamos o grafo da instância e atribuímos valores de peso aleatórios para suas arestas, de maneira a gerar indivíduos diferentes. Isso faz com que nem sempre o número de indivíduos a serem gerados passado para essa função irá gerar de fato esta quantidade de indivíduos, visto que há um limite superior estabelecido pelo número de vértices e arestas do grafo para o número de partições diferentes que podem existir. Por fim, a segmentação é feita utilizando o algoritmo Walktrap para detecção de comunidades, que consiste basicamente de realizar caminhos aleatórios pelo grafo, calcular a distância entre os vértices desse caminho e "agrupar" caminhos de tamanhos parecidos na mesma comunidade. A fim de não gerar uma população inicial que contenha uma solução ótima, efetuamos a segmentação do grafo com no mínimo $min\ cluster\ amount\ *\ n$ e no máximo n nodos.

Caso haja elitismo, o melhor indivíduo da população é escolhido para ser "resguardado" das etapas do algoritmo genético, que poderiam infactibilizá-lo ou piorá-lo em relação ao seu fitness.

Em seguida, a função de seleção irá selecionar os melhores indivíduos da população gerada inicialmente baseando-se no critério de menor número de subconjuntos por indivíduo. Esse valor é computado por meio do somatório das distâncias do menor caminho entre os vértices de um subconjunto. Caso a distância seja menor ou igual ao valor máximo D e o número de itens (nós) em cada subconjunto do indivíduo seja menor ou igual a T, o fitness (valor da função objetivo) será o tamanho do indivíduo (seu número de subconjuntos). Caso contrário, será infinito, indicando que em uma comparação com outro indivíduo que não tenha valor infinito, o primeiro sempre "perderá". Sempre serão escolhidos k indivíduos da população, logo nem sempre todos os indivíduos selecionados terão valor finito.

A função de torneio irá escolher os dois melhores indivíduos da população, ou seja, os que tiverem o menor número de subconjuntos. O torneio é responsável por escolher os indivíduos que realizarão o cross-over (ou seja, os "pais").

A função de recombinação (one-point cross-over) irá escolher um nó de um subconjunto aleatório do primeiro pai e irá colocá-lo no primeiro subconjunto do segundo pai em que o nó escolhido tenha vizinhos, a fim de manter a factibilidade da solução. E o mesmo ocorre para o segundo pai. Assim são gerados dois filhos.

A função de mutação recebe um indivíduo e caso a chance de mutação seja sorteada, clusters que possuem nós vizinhos serão agrupados (é possível perder a factibilidade nesta etapa). O parâmetro de entrada da mutação são os filhos gerados na etapa anterior.

Por fim, uma nova população é atualizada com os dois novos filhos gerados, e uma nova iteração do loop é iniciada caso não cumpra o critério de parada, que ocorre quando o tamanho da população atualizada for maior que o número de indivíduos ou se o valor da função objetivo se repete a cada x iterações (escolhemos $x = 3$). Quando a condição de parada for cumprida, a população é atualizada com a nova população composta pelos filhos dos indivíduos da geração anterior, e inicia-se uma nova geração.

6. ANÁLISE DOS RESULTADOS

Arquivo instance_n_m_D_T.dat

n: Número de arestas do grafo.

m: Número de vértices do grafo.

D: Maior distância permitida entre dois vértices em um mesmo conjunto.

T: Tamanho máximo de um subconjunto.

Gerações: Número de gerações que levou para executar o melhor resultado obtido.

Indivíduos: Número de indivíduos da população.

Tempo: Tempo, em minutos, decorrido para obter o melhor resultado.

BKV: Melhor valor conhecido para a instância.

Fitness: Menor e maior fitness que obtivemos como resultado.

Com o decorrer do trabalho descobrimos que nossa implementação do algoritmo não era eficaz para todas as instâncias disponíveis para teste, por exemplo quando o número das arestas ou dos vértices era muito elevado, porém para instâncias de números menores o programa funciona e a seguir uma tabela com alguns resultados que obtivemos:

n	m	D	T	Gerações (Menor)	Indivíduos	Tempo (min.)	BKV	Fitness (Menor)	Fitness (Maior)
6	6	4	3	3	300	0.019	4	3	3
20	30	20	3	3	300	0.276	11	10	11
20	100	10	5	4	300	0.554	9	8	11
50	75	50	5	8	300	7.006	21	20	24
50	750	10	5	14	300	7.941	20	23	26
100	350	50	10	35	300	168.53	19	33	37
100	1000	25	15	35	300	141.93	12	39	40

É possível notar que no caso das instâncias com um número menor de arestas e vértices o algoritmo resultou em soluções melhores que as soluções fornecidas representadas na tabela pela cor de fundo verde, porém como é possível notar quando muito elevado o número de arestas o fitness representado com a coloração amarela já demonstra uma diferença de três unidades a mais que a resposta fornecida pelo professor, e por fim com a cor vermelha os resultados que obtivemos a maior discrepância em relação aos dados do professor.

Com isso concluímos que o algoritmo, a partir de um certo número de arestas, se torna inviável. Acreditamos que há algumas razões para a ineficiência da implementação:

- **Escolha ruim dos parâmetros**: deveríamos ter aumentado o número de indivíduos para instâncias maiores.
- **Uso de estruturas de dados ruins**: utilizamos listas para representar os indivíduos, porém não há necessidade de ordenação na solução, visto que os subconjuntos [1, 2, 3] e [3, 2, 1], por exemplo, são idênticos, logo, poderíamos ter utilizado sets ao invés de listas; poderíamos também ter utilizado uma estrutura de árvore binária durante a etapa de seleção.