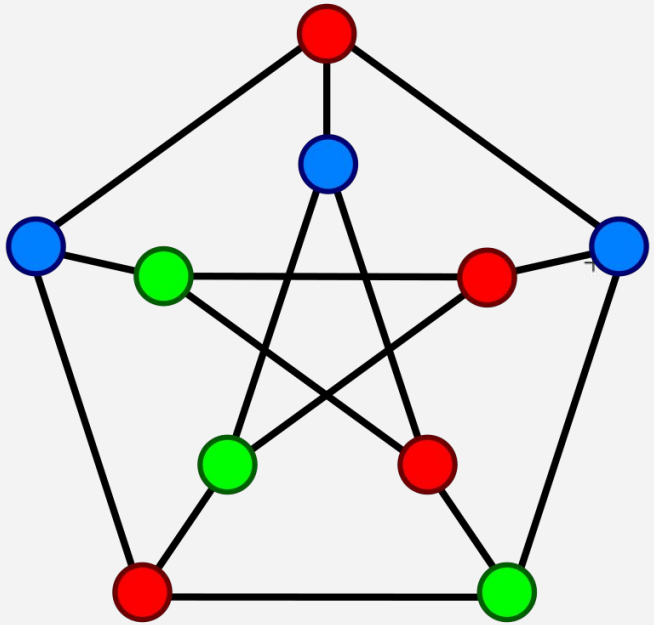
TRABALHO PRÁTICO

ALGORITMO EM GRAFOS



Olivia Campos

CONTEXTO

Considere uma extensa rede em uma empresa de grande porte, composta por diversos dispositivos interconectados, como computadores, servidores e roteadores. Imagina que você seja responsável por garantir que dois dispositivos não possam compartilhar o mesmo canal, ou seja, mesma cor pois isso causaria interferência na rede.

Como poderíamos resolver esse problema?

CONTEXTO

Uma solução seria de utilizar o conceito de coloração em vértices em grafos. A partir disso, poderíamos representar essa rede por meio de um grafo. Onde os vértices seriam os dispositivos e as arestas as conexões.

Para evitar que dois dispositivos compartilhassem de um mesmo canal, adicionariamos cores a cada vértice. Com isso, dois dispositivos não poderiam ter a mesma cor.

PROBLEMA

Mas com isso teríamos um problema, uma vez que estamos tratando de uma rede de uma empresa gigante. Então o desafio em questão é determinar a coloração mínima necessária para essa rede considerando que ela será composta por muitos vértices.

O problema de determinar a coloração mínima necessária para redes com muitos vértices é complexo e ainda não possui uma solução polinomial eficiente, dessa forma se torna um problema NP-Difícil.

PROBLEMA NP-DIFÍCIL

Uma vez que a quantidade de dispositivos aumenta, a complexidade do problema cresce exponencialmente. A coloração mínima requer a atribuição de cores distintas a cada dispositivo conectado diretamente a outro, o que se torna impraticável em tempo polinomial conforme a rede se expande.

Exemplo: um grafo de 10 vértices teria 2^{10} = 1.024 combinações possíveis de cores para os vértices.

HEURÍSTICA GULOSA

Diante da complexidade do problema, empregamos uma heurística para encontrar soluções próximas do ótimo em um tempo viável.

A heurística escolhida foi a Heurística Gulosa para Coloração de Vértices.

HEURÍSTICA GULOSA

A heurística gulosa busca resolver o problema fazendo escolhas locais em cada passo, a fim de alcançar uma solução eficiente.

Nesse contexto, essa heurística atribui cores aos vértices em ordem decrescente de grau. Essa abordagem visa reduzir o número de conflitos de cor e, consequentemente, encontrar uma coloração eficiente para a rede.

HEURÍSTICA GULOSA

O método LF(largest-first) foi apresentado pela primeira vez por (WELSH; POWELL, 1967). É um dos métodos sequenciais mais antigos e simples. O método é baseado na observação de que vértices com menor grau são mais flexíveis na escolha de cores se comparado a vértices com maior grau, logo é natural colorir primeiro os outros vértices de maior grau. Apesar de sua simplicidade o método LF mostra ser eficiente, na quantidade de cores necessárias para colorir um grafo.

Força bruta

Recursivo

Complexidade exponencial

Eficiência menor em resoluções subótimas

Solução ótima quando possível

Mais útil em grafos menores

Heurística Gulosa

Não garante a melhor solução

Eficiência no tempo de execução

Utilidade maior para grafos grandes

Maior flexibilidade e adaptabilidade

Maior facilidade de modelagem para problemas reais

NA PRÁTICA

E como fica no código?

```java
// Função recursiva para colorir os vértices
private static boolean colorirRecursivo(int[][] matriz, int nVertices, int[] cores, int vertice) {
    if (vertice == nVertices) {
        return true;
    }

    for (int cor = 0; cor < nVertices; cor++) {
        if (podeAtribuirCor(matriz, cores, vertice, cor)) {
            cores[vertice] = cor;

            if (colorirRecursivo(matriz, nVertices, cores, vertice + 1)) {
                return true; // Solução encontrada
            }
            cores[vertice] = -1;
        }
    }

    return false;
}
```

Força bruta

```java
// Função para verificar se é possível atribuir uma cor a um vértice
private static boolean podeAtribuirCor(int[][] matriz, int[] cores, int vertice, int cor) {

    for (int verticeAdj = 0; verticeAdj < vertice; verticeAdj++) {
        if (matriz[vertice][verticeAdj] == 1 && cores[verticeAdj] == cor) {
            return false;
        }
    }
    return true;
}
```

Força bruta

```java
// Cria um array de índices dos vértices em ordem decrescente de grau
Integer[] indices = new Integer[nVertices];
for (int i = 0; i < nVertices; i++) {
    indices[i] = i;
}
Arrays.sort(indices, (a, b) -> Integer.compare(grau[b], grau[a]));
```

Heuristica gulosa

```java
public static int[] colorirHeuristica(int[][] matriz, int nVertices, Integer[] ordemVertices) {
    int[] cores = new int[nVertices];
    for (int i = 0; i < nVertices; i++) {
        cores[i] = -1;
    }

    for (int i : ordemVertices) {
        boolean[] coresVizinhas = new boolean[nVertices];

        for (int vizinho = 0; vizinho < nVertices; vizinho++) {
            if (matriz[i][vizinho] == 1 && cores[vizinho] != -1) {
                coresVizinhas[cores[vizinho]] = true;
            }
        }

        for (int cor = 0; cor < nVertices; cor++) {
            if (!coresVizinhas[cor]) {
                cores[i] = cor;
                break;
            }
        }
    }
    return cores;
}
```

Heuristica gulosa

Obrigada!

REFERÊNCIAS

On the recursive largest first algorithm for graph colouring, tandfonline. Disponível em: <https://www.tandfonline.com/doi/abs/10.1080/00207160701419114>. Acesso em: 12 dez. 2023.

UMA NOVA HEURÍSTICA DE COLORAÇÃO GULOSA SEQUENCIAL. Disponível em: <https://repositorio.ufc.br/bitstream/riufc/39615/1/2018_tcc_lscosta.pdf>. Acesso em: 12 dez. 2023.