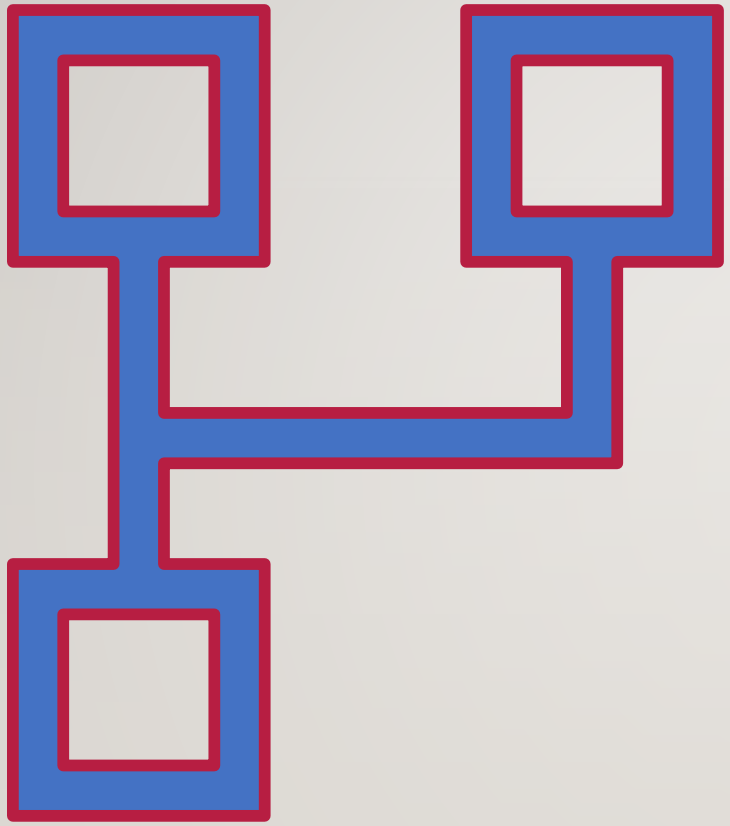

METAHEURÍSTICA TABU PARA O PROBLEMA DA COLORAÇÃO DE VÉRTICES

DAIVES KAWON CHU

GABRIEL BARROS DE PAULA

MARCOS SAMUEL WINKEL LANDI

O PROBLEMA A SER RESOLVIDO

- Uma solução para o problema da Coloração de Vértices é definido da seguinte maneira:

Atribuir cores para os vértices de forma que quaisquer dois vértices ligados por uma aresta não possuam a mesma cor, e a versão de otimização é encontrar uma solução válida com o menor número de cores possível.

FORMULAÇÃO MATEMÁTICA

2.1 Modelo

$$\text{Minimiza} \sum_{h=1}^{|V|} y_h \tag{1}$$

Sujeito a:

$$\sum_{c=1}^{|V|} x_{uc} = 1 \qquad \forall u \in V \tag{2}$$

$$x_{uc} + x_{vc} \leq y_c \qquad (u,v) \in E, 1 \leq c \leq |V| \tag{3}$$

$$x_{uc} \in \{0,1\} \qquad \forall u \in V, 1 \leq c \leq |V| \tag{4}$$

$$y_c \in \{0,1\} \qquad 1 \leq c \leq |V| \tag{5}$$

2.2 Variáveis

- y_c são variáveis binárias que representam se a cor c está na solução
- x_{uc} são variáveis binárias que representam se um vertice i recebeu a cor c

2.3 Objetivo e Restrições

- (1) – O objetivo é minimizar a quantidade de cores usadas
- (2) – Cada vértice é pintado com apenas uma única cor
- (3) – No máximo um dos vértices de cada aresta é pintado com uma cor que é usada. Esta restrição também assegura que caso a cor não seja usada, nenhum vértice é pintado com ela.
- (4) e (5) – Assegura que as variáveis são binárias

IMPLEMENTAÇÃO GLPK

vertexcolouringproblem.mod

```
/* Número de vértices */
param n, integer, >= 2;

/* Conjunto de vértices */
set VERTICES := {1..n};

/* Conjunto de cores */
set CORES := {1..n};

/* Conjunto dos arcos */
set ARCOS within (VERTICES cross VERTICES);

/* Variável se o vértice u é colorido com a cor c: 1 se sim; 0 caso contrário */
var x{u in VERTICES, c in CORES} binary;

/* Variável se a cor c é usada: 1 se sim; 0 caso contrário */
var y{c in CORES} binary;

/* Função Objetivo */
minimize qtdCores: sum{c in CORES} y[c];

/* Restrições */
s.t. verticeTemCor {u in VERTICES}: sum{c in CORES} x[u,c] = 1;
s.t. arcoSemDoisVerticesMesmaCorQueEhUtilizada {(u,v) in ARCOS, c in CORES}: x[u,c] + x[v,c] <= y[c];
s.t. coresDeIndiceMenorEhUsadoPrimeiro {c in CORES, d in CORES: d == c + 1} : sum{u in VERTICES} x[u,c] >= sum{u in VERTICES} x[u,d];
```

REPRESENTAÇÃO DO PROBLEMA

- Cada nó do grafo a ser colorido é representado por um n úmero inteiro e as possíveis cores são representadas por um número de 0 ao número de nodos menos um.
- O resultado de cada iteração é uma coloração diferente, a qual possui um valor da solução, e uma combinação de nodo e cor alterado a ser inserido na lista TABU.

ESTRUTURA DE DADOS

- Para armazenar o grafo, a biblioteca utilizada foi a networkx. O grafo é armazenado em dicionários de dicionários de dicionários (é isso mesmo), cada chave representa o nome/número do nodo e seu valor é outro dicionário com seus atributos, por exemplo, sua cor e seus vizinhos.
- Na parte da geração de vizinhos, é calculado um domínio de cores possíveis para determinado nodo. Este é guardado em sets, uma estrutura de dados que armazena conjuntos de itens de forma não ordenada e não repetida. Queremos saber somente se uma cor está presente ou não no conjunto, não é ideal que uma mesma cor seja guardada duas vezes e não é necessário que elas estejam em uma dada ordem, por isso o uso de sets.

GERAÇÃO DA SOLUÇÃO INICIAL

- A solução inicial é uma coloração onde cada nodo possui uma cor diferente – a cor correspondente ao número do nodo – e portanto sempre com o número de cores igual ao número de nodos.

VIZINHANÇA

- Existem duas fases de geração de vizinhança, uma mais completa porém mais demorada nas primeiras iterações em que o número de cores usadas na coloração é alto, e uma vizinhança menor, construída visando a eficiência nas primeiras iterações.

VIZINHANÇA COMPLETA

- A vizinhança 'completa' é composta por uma lista com a combinação de todos os nodos e cores possíveis, essas sendo as cores já utilizadas na coloração menos as cores dos seus vizinhos e as cores tabu para o determinado nodo. Todos as possíveis combinações são testadas e seus resultados são ordenados. O melhor valor é escolhido como vizinho, caso haja empate no melhor valor, o escolhido é sorteado entre eles. Essa geração de vizinhos de todas as combinações de vértices e cores cria uma vizinhança de tamanho |V| * cores usadas. Antes de gerar a vizinhança, este valor é calculado, e caso exceda um valor limite pré definido empiricamente como 17000, o outro método de geração de vizinhanças é utilizado.

VIZINHANÇA MENOR

- A vizinhança menor é gerada sorteando um vértice e uma cor de um domínio de cores possíveis. Esse domínio é calculado como sendo o conjunto de todas cores menos a atual cor do vértice sorteado, as cores dos vértices vizinhos a ele e as cores associadas a este vértice na lista tabu. Este sorteio é feito até a vizinhança atingir um tamanho pré determinado como 450.

PARÂMETROS

- Número de Iterações máximo = 13520000/ $(|V| + 20)^2$

O número de iterações máximo é um valor alto escalado pelo número de vértices do grafo pois quanto maior o número de vértices maior o tempo de execução de cada iteração, por conta do cálculo da vizinhança com base no número de vértices.

- Número de Vizinhos da Vizinhança Reduzida = 450

Um número menor de vizinhos na vizinhança reduzida implica em iterações mais rápidas, e no começo das iterações isso significa um avanço mais rápido da melhora da melhor solução, já que é mais fácil reduzir o número de cores, pois existem menos conflitos de cores entre vizinhos.

PARÂMETROS

- Tamanho da Lista Tabu = 10 * |V|

10 vezes o tamanho do conjunto de vértices significa que, quando a lista encher, em média, 10 cores indesejadas serão evitadas nas próximas iterações para cada vértice, o que parece uma quantidade razoável, observando os melhores valores encontrados para cada uma das 10 instâncias disponíveis para teste.

- Tamanho Máximo da Vizinhança Completa = 17000

Realizando testes, se verificou que tamanhos máximos da vizinhança completa maiores que 170000 implicam em iterações desnecessariamente lentas no começo com o uso da vizinhança completa, enquanto que valores menores implicam em um maior número de iterações mais à frente com o uso das vizinhanças reduzidas.

CRITÉRIO DE PARADA

- O algoritmo para quando o número de iterações ultrapassa o limite de iterações máximo.

RESULTADOS DAS INTÂNCIAS

	valor relaxação linear GLPK	melhor solução inteira GLPK	tempo execução GLPK	valor médio solução inicial TABU	valor médio melhor solução TABU	desvio padrão melhores soluções TABU	tempo execução médio (min) TABU	desvio médio TABU
2-FullIns_3	3	6	1:07:33	52	6.6	0.489	0:20	65%
2-FullIns_4	-	-	6:40:00	212	58.33	2.210	3:16	872.22%
4-FullIns_3	3	5	1:26:43	114	17.8	1.469	2:23	154.28%
5-FullIns_3	-	-	1:50:00	154	30.8	2.039	3:42	926.66%
queen5_5	4	5	0:17:41	25	5	0	0:06	0%
queen6_6	3	16	1:00:12	36	7.6	0.489	0:10	26.66%
queen7_7	2	10	1:00:52	49	9.8	0.2	0:14	39.99%
queen9_9	3	-	1:03:00	81	15.6	0.489	0:35	73.33%
queen10_10	2	-	1:01:50	100	20	0.632	0:59	81.82%
queen11_11	-	-	2:51:44	121	25	0.894	1:33	129.09%

Os resultados indicados por '-' não foram encontrados pelo GLPK dentro do tempo indicado.

ANÁLISE DOS RESULTADOS

- O algoritmo desenvolvido foi capaz de encontrar uma solução melhor ou igual à encontrada pelo solver do GLPK em 8 de 10 das instâncias testadas, sendo que em 5 das instâncias o GLPK não encontrou solução em 1 hora ou mais.
- A solução do algoritmo TABU é encontrada, com o parâmetro de critério de parada escolhido, em todos os casos, em menos de 3 minutos e 49 segundos. Com o GLPK, soluções piores, com excessão de uma instância, são obtidos em tempos que passam de 1 hora.

CONCLUSÃO

- Os resultados contrastantes entre GLPK e a solução aqui proposta tornam a solução TABU uma alternativa atraente a um solver padrão e universal como o do GLPK.

REFERÊNCIAS

- Malaguti, Enrico; Monaci, Michele; Toth, Paolo. An exact approach for the Vertex Coloring Problem. https://www.sciencedirect.com/science/article/pii/S157252861000054X
- A. Hertz and D. de Werra. Using Tabu Search Techniques for Graph Coloring.

MUITO OBRIGADO