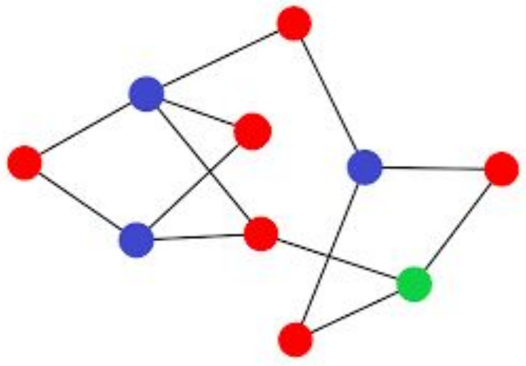
Heurística Construtiva e Problema de Coloração de Grafos

Ênio Filipe e Lorena Avelino

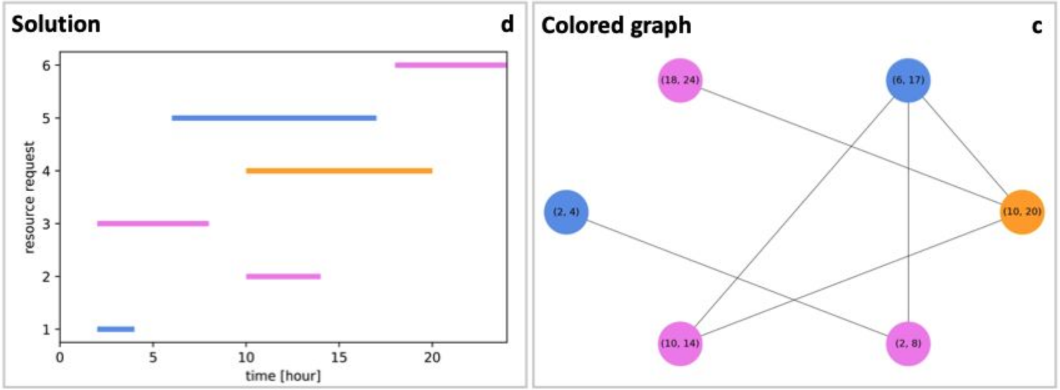
Coloração de grafos

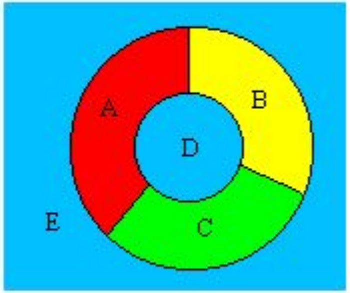
- Atribuição de cores aos vértices para que vértices adjacentes tenham cores distintas.
- Minimização do número de cores (número cromático).
- NP-difícil.

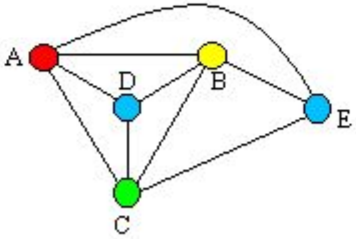


Coloração de grafos - Relevância prática

- Aplicações em programação de horários, planejamento de redes, coloração de mapas.







Coloração de grafos - Motivação

- Exploração teórica
- Impacto prático
- interdisciplinaridade

Revisão bibliográfica

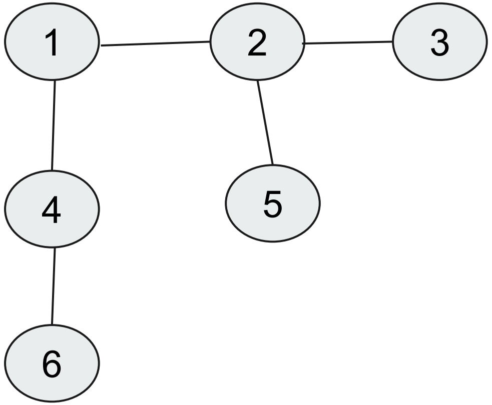
- Hertz & Werra (1987): Introdução da Busca Tabu (Tabu Search)
- Chams et al. (1987): Recozimento Simulado (Simulated Annealing).
- Galinier & Hao (1999): Algoritmo Evolutivo Híbrido (HEA).
- Laguna & Martí (2001): GRASP para grafos esparsos.
- Blöchliger & Zufferey (2008): Tabu Reativo e soluções parciais.

Artigo	Contribuições principais
Hertz & Werra (1987)	Busca Tabu; Evita ciclos; Testes em grafos densos; Resultados melhores que heurísticas gananciosas.
Galinier & Hao (1999)	Algoritmo Evolutivo Híbrido; Operadores genéticos; Soluções para instâncias complexas; Superação de heurísticas.
Laguna & Martí (2001)	GRASP para grafos esparsos; Construção adaptativa; Busca local eficiente; Foco em flexibilidade.
Blöchliger & Zufferey (2008)	Tabu Reativo; Ajustes automáticos; Soluções parciais; Desempenho consistente.
Chams et al. (1987)	Recozimento Simulado; Escapar de mínimos locais; Parâmetros bem calibrados; Bons resultados em grafos pequenos.

Heurística Construtiva

- Baseada no DSATUR (Saturação)
- Inicialização de estruturas: cores, saturação e grau.
- Seleção do vértice com maior saturação e maior grau (em caso de empate).
- Atribuição da menor cor disponível.
- Atualização de saturações dos vizinhos.

Inicialização

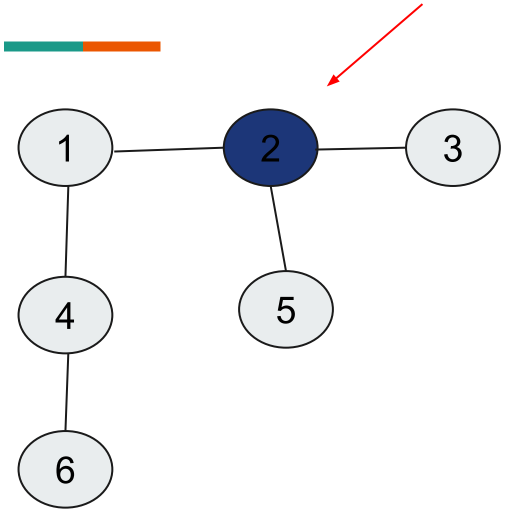


	1	2	3	4	5	6
cores:	-1	-1	-1	-1	-1	-1
grau:	2	3	1	2	1	1
saturação:	0	0	0	0	0	0

COR 1

COR 2

Iteração 1

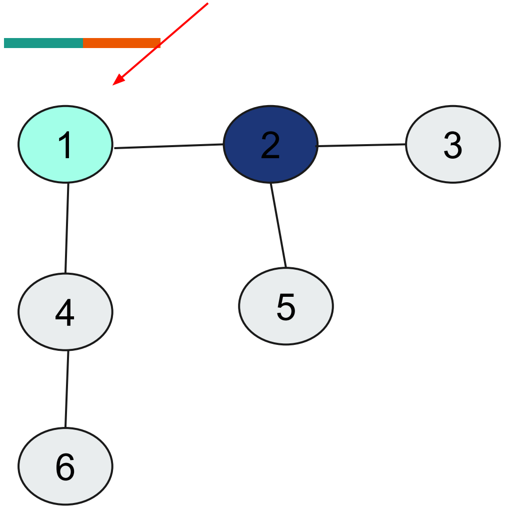


	1	2	3	4	5	6
cores:	-1	1	-1	-1	-1	-1
grau:	2	3	1	2	1	1
saturação:	1	0	1	0	1	0

COR 1

COR 2

Iteração 2

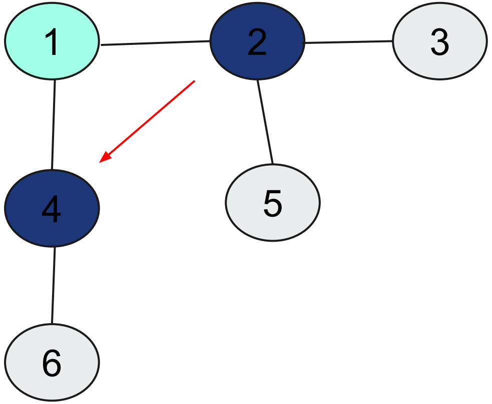


	1	2	3	4	5	6
cores:	2	1	-1	-1	-1	-1
grau:	2	3	1	2	1	1
saturação:	1	1	1	1	1	0

COR 1

COR 2

Iteração 3

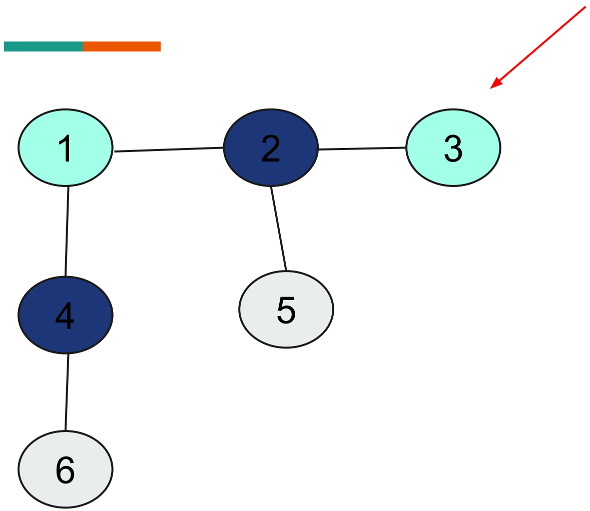


	1	2	3	4	5	6
cores:	2	1	-1	1	-1	-1
grau:	2	3	1	2	1	1
saturação:	2	1	1	1	1	1

COR 1

COR 2

Iteração 4

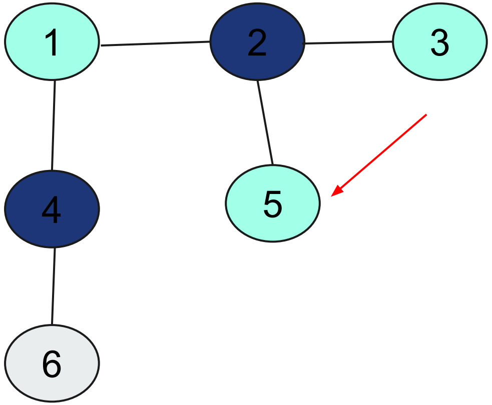


	1	2	3	4	5	6
cores:	2	1	2	1	-1	-1
grau:	2	3	1	2	1	1
saturação:	2	2	1	1	1	1

COR 1

COR 2

Iteração 5

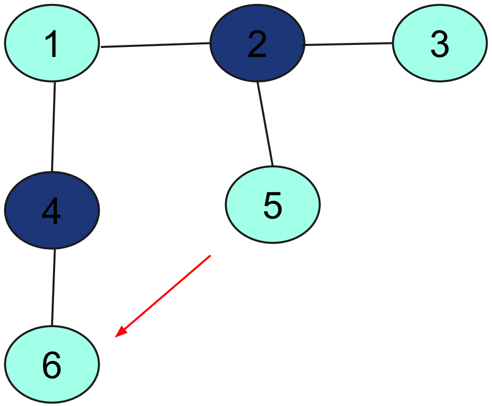


	1	2	3	4	5	6
cores:	2	1	2	1	2	-1
grau:	2	3	1	2	1	1
saturação:	2	3	1	1	1	1

COR 1

COR 2

Iteração 6



	1	2	3	4	5	6
cores:	2	1	2	1	2	2
grau:	2	3	1	2	1	1
saturação:	2	3	1	2	1	1

COR 1

COR 2

```cpp
int Problem::getMaxColor(Graph &graph, int num_vertices, int &grauMaximo)
{
    vector<int> cores(num_vertices, -1);
    vector<int> saturacao(num_vertices, 0);
    vector<int> graus(num_vertices, 0);

    for (int i = 0; i < num_vertices; i++)
    {
        graus[i] = graph.get_neighbors(i).size();
        if (graus[i] > grauMaximo)
        {
            grauMaximo = graus[i];
        }
    }

    for (int passada = 0; passada < num_vertices; passada++)
    {
        int melhorVertice = -1;
        int maiorSaturacao = -1;
        int maiorGrau = -1;

        for (int vertice = 0; vertice < num_vertices; vertice++)
        {
            if (cores[vertice] == -1)
            {
                if (saturacao[vertice] > maiorSaturacao ||
                (saturacao[vertice] == maiorSaturacao && graus[vertice] > maiorGrau))
                {
                    melhorVertice = vertice;
                    maiorSaturacao = saturacao[vertice];
                    maiorGrau = graus[vertice];
                }
            }
        }
```

Inicializa

Escolhe melhor candidato

```cpp
        int menorCor = 1;
        set<int> coresIndisponiveis;
        for (CoupleVertice vizinho : graph.get_neighbors(melhorVertice))
        {
            if (cores[vizinho.vertice2] != -1)
            {
                coresIndisponiveis.insert(cores[vizinho.vertice2]);
            }
        }

        while (coresIndisponiveis.count(menorCor) > 0)
        {
            menorCor++;
        }

        cores[melhorVertice] = menorCor;
        for (CoupleVertice vizinho : graph.get_neighbors(melhorVertice))
        {
            saturacao[vizinho.vertice2]++;
        }
    }

    int maxCor = 0;
    for (int i = 0; i < num_vertices; i++)
    {
        if (cores[i] > maxCor)
        {
            maxCor = cores[i];
        }
    }

    return maxCor;
}
```

Escolhe menor cor

Atualiza saturação

Reconhece a maior cor utilizada e retorna

Algoritmo de identificação de clique máximo - Limite Inferior

- Algoritmo guloso
- Simplificação e velocidade
- Ordenação por Grau
- Inicializa-se um conjunto vazio para armazenar os vértices do clique.
    - Cada vértice, começando pelos de maior grau, é avaliado:
        - Se o vértice é adjacente a todos os vértices já presentes no clique, ele é adicionado ao conjunto.
    - Após a avaliação de todos os vértices, o conjunto resultante é retornado como o clique encontrado.

```cpp
vector<int> Problem::calcClique(Graph &graph)
{
    int n = graph.get_num_vertices();
    vector<int> vertices(n);
    for (int i = 0; i < n; ++i)
        vertices[i] = i;

    sort(vertices.begin(), vertices.end(), [&](int a, int b)
         { return graph.get_neighbors(a).size() > graph.get_neighbors(b).size(); });

    vector<int> clique;
    for (int v : vertices)
    {
        bool canAdd = true;
        for (int u : clique)
        {
            if (!graph.ehVizinho(u, v))
            {
                canAdd = false;
                break;
            }
        }
        if (canAdd)
            clique.push_back(v);
    }
    return clique;
}
```

Instâncias utilizadas

- Grafos aleatórios com densidade 0.5 e número de vértices (100,300,500,1000)
- Instâncias DIMACS de problemas já conhecidos

Resultados

SA X Tabu Search X Solução Heurística

GRUPO	I	N	MEDIA CLIQUE	SA		TABU SEARCH		SOLUÇÃO HEURÍSTICA	
				S	D	S	D	S	D
100 vertices	20	100	8	16	110,5	16	110,5	20	161,8
300 vertices	10	300	9	36	283,0	35	272,3	47	404,3
500 vertices	5	500	9	54	487,0	51	454,3	71	673,9
1000 vertices	1	1000	11	98	790,9	85	672,7	124	1027,3

GRASP X Solução Heurística

GRUPO	I	N	M	MEDIA CLIQUE	GRASP		SOLUÇAO HEURÍSTICA	
					S	D	S	D
LEI	13	450	10787	1	16,5	1550,0	19,5	1846,2
MYC	5	73,4	688,4	1	6	500,0	4,2	320,0
REG	14	362,1	7608,1	1	37,4	3640,0	10,9	985,7
SGB	24	135,9	3745,25	9,7	22,7	134,8	19,5	101,3

HCA X Solução Heurística

GRUPO	I	N	M	MC	HCA		SH	
					S	D	S	D
DSJC250.5	1	250	15668	1	28	2700,0	38,0	3700,0
DSJC500.5	1	500	62624	1	48	4700,0	67	6600,0
DSJC1000.5	1	1000,0	249826	1	83	8200,0	118,0	11700,0
le450_15c	1	450,0	16680	1,0	15	1400,0	25,0	2400,0
le450_25c	1	450	17343	1	26	2500,0	29	2800,0
flat300_28_0.col	1	300	21695	1	31	3000,0	42	4100,0
flat1000_76_0.col	1	1000	246708	1	83	8200,0	119	11800,0
DSJC1000.1	1	1000	49629	1	20	1900,0	29	2800,0
DSJC1000.9	1	1000	449449	1	224	22300,0	276	27500,0

Parcial x Tabu x Solução heurística

GRUPO	N	M	MC	PARTIAL				TABU				SH	
				FOO		DYN		FOO		DYN			
				S	D	S	D	S	D	S	D	S	D
DSJC500.1	500	12458	1	12	1100,0	12	1100	12	1100	12	1100	17	1600,0
DSJC500.9	500	112437	1	128	12700,0	127	12600	127	12600	127	12600	150	14900,0
DSJR500.1c	500	121275	1	85	8400,0	85	8400	85	8400	85	8400	119	11800,0
DSJR500.5	500	58862	1	128	12700,0	126	12500	128	12700	126	12500	83	8200,0
R1000.1c	1000	485090	1	98	9700,0	99	9800	98	9700	98	9700	167	16600,0
R1000.5	1000	238267	1	252	25100,0	249	24800	254	25300	249	24800	160	15900,0
R250.1c	250	30227	1	64	6300,0	64	6300	64	6300	66	6500	65	6400,0
R250.5	250	14849	1	67	6600,0	66	6500	67	6600	67	6600	38	3700,0
flat1000_50_0	1000	245000	1	50	4900,0	50	4900	73	7200	50	4900	121	12000,0
flat1000_60_0	1000	245830	1	60	5900,0	60	5900	79	7800	60	5900	118	11700,0
le450_15d	450	16750	1	15	1400,0	15	1400	15	1400	15	1400	26	2500,0
le450_25d	450	17425	1	27	2600,0	27	2600	27	2600	27	2600	28	2700,0

Conclusões

- **Pontos fortes da heurística**:
  - Simplicidade e eficiência computacional.
  - Boa aproximação para instâncias específicas.
- **Pontos fracos e limitações**:
  - Maior número de cores utilizado em relação a métodos mais sofisticados.
  - Alta variabilidade nos resultados.
- **Sugestão de melhorias**
  - Integração com métodos híbridos (e.g., Tabu Search ou GRASP).
  - Ajustes nos critérios de seleção de vértices para reduzir desvios.

Obrigado ;)