

Relatório do Projecto de Algoritmos e Complexidade

João Melo n.º 47053

8 de Janeiro de 2009

Resumo

Este projecto trata do desenvolvimento de uma aplicação que resolva o problema da alocação de alunos às Unidades Curriculares de Especialização (UCE) bem como da alocação das UCes a horários de forma a maximizar as preferências dos alunos, isto é, o maior número possível de alunos alocados às suas UCE's preferidas. A base deste problema reside num problema bastante conhecido no mundo da informática pela sua complexidade - Coloração de Grafos.

Conteúdo

1	Introdução	3
2	Análise	4
2.1	Entidades e Propriedades	4
2.1.1	Alunos	4
2.1.2	UCEs	4
2.2	Análise da Problemática	5
2.2.1	Dimensão do Problema	6
2.2.2	Melhor Combinação	7
2.3	Coloração de grafos	8
2.3.1	Alocação de UCE's	8
2.3.2	Alocação de Alunos & UCE's vs Coloração de Grafos	8
3	Decisões de Implementação	10
3.1	Comparações dos Algoritmos	11
3.2	Estruturas de dados	11
3.2.1	Listas	11
3.2.2	Hash Tables	12
3.2.3	Aluno	13
3.2.4	UCE	13
3.2.5	Grafo	14
3.3	Implementação	15
3.3.1	DSATUR	15
4	Tempo de Execução	17
4.1	Mínimo de Horários sem remover arestas	17
4.2	Tempo de execução para alocação	18
5	Extras	19
6	Conclusão	19

Lista de Figuras

1	Exemplo da alocação de 3 UCE's a 3 Horários.	6
2	Exemplo de um grafo.	14
3	Grafo por processar.	15
4	1 ^a proposta para maior clique.	16
5	2 ^a proposta para maior clique.	16
6	Última proposta para maior clique.	16

1 Introdução

Este projecto tem como objectivos implementar um programa que irá receber como input uma sequência de Alunos, um conjunto de UCE's¹, o número de horários diferentes e um limite superior para o número de UCE's por horário. O programa deverá maximizar o número de entradas como primeira escolha. Problemas deste género são bastante conhecidos, este em particular ganhou maior dimensão após a entrada do modelo de bolonha, mais precisamente na entrada para o segundo ciclo (Mestrado ou Especialização).

Este projecto está implicitamente ligado à coloração de grafos. A coloração de grafos consiste na coloração dos nós do grafo que se interligam sem que se nenhuma cor seja repetida. O número de cores a utilizar, depende do número de recursos disponíveis no espaço ou tempo. Cada nó do grafo representa um evento. Se a dois nós interligados for atribuída a mesma cor, estes estão em conflito, pois ocupam um recurso ao mesmo tempo.

Formalmente Seja $G(V, E)$ um grafo e $C = \{c_i, i \in N\}$ um conjunto de cores. Uma coloração de $G(V, E)$ é uma atribuição de uma cor de C para cada vértice de V , de tal modo que a dois vértices adjacentes sejam atribuídas cores diferentes, i.e., uma coloração de G é uma função $f : V \rightarrow C$ tal que para cada par de vértices $v, w \in V$ tem-se $(v, w) \in A \Rightarrow f(v) \neq f(w)$. [1]

Uma k -coloração de G é uma coloração que utiliza um total de k cores. Diz-se então que G é k -colorível. Denomina-se número cromático, $\lambda(G)$, de um grafo G , ao menor número de cores k , para o qual existe uma k -coloração de G .

Os conceitos de coloração, clique e conjunto independente de vértices estão naturalmente relacionados: De fato, como são necessárias k cores para colorir os k vértices de uma clique de tamanho k , isto implica que $\lambda(G)$ é maior ou igual ao tamanho da maior clique de G .

Contudo, esta problemática dos horários já existe à bastante tempo. Em 1964, Broder e Cole, ambos apresentaram, separadamente, métodos heurísticos na abordagem do processo de geração de horários. Broder usou um algoritmo sequencial, no qual atribuía aulas a períodos. Cada aula era atribuída no primeiro período livre. Esta abordagem foi adaptada para a alocação de cursos seleccionando o período com o menor número de conflitos entre estudantes considerando que o número de períodos disponíveis era limitado. Broder sugeriu que o algoritmo fosse executado várias vezes permitindo obter um conjunto de soluções, de entre as quais o utilizador escolheria a melhor. Cole utilizou um algoritmo semelhante aplicado ao escalonamento de exames. A diferença residia no facto de este apenas seleccionar os exames que não tinham conflitos entre eles no período de alocação corrente, repetindo o procedimento para cada período. Uma conclusão importante observada por ambos, foi que a ordem pela qual eram atribuídos os exames afectava, directamente, o número de conflitos a resolver. Desde 1967, a abordagem de Welsh e Powel levou a que muitos outros algoritmos baseados na coloração de grafos fossem desenvolvidos.

¹Unidades Curriculares de Especialização

2 Análise

Partiu-se para uma análise detalhada ao problema. Começando-se por distinguir as diferentes entidades do problema e as suas propriedades.

2.1 Entidades e Propriedades

2.1.1 Alunos

Os Alunos são a entidade fundamental de todo o processo, ou seja, é em função dos Alunos que deve desenvolver este programa. Relativamente aos alunos, estes são caracterizados por terem uma ordem de preferência sobre as UCE's a que pode candidatar-se. Existem três tipos de alunos distintos:

- Mestrado²
- Especialização
- Extra-curricular

Os Alunos podem candidatar-se a uma UCE apenas, ou então, candidatar-se a duas. É neste aspecto que reside o maior problema da alocação de alunos às UCE's e UCE's aos horários, isto porque, um aluno quando inscrito a duas UCE's não pode de forma alguma ser inscrito numa outra UCE com o mesmo horário.

2.1.2 UCEs

As Unidades Curriculares de Especialização são a outra entidade do problema. As UCEs serão alocadas a horários, sendo que existe um número de horários disponíveis. Para além disso existe uma limitação física de laboratórios, sendo que, existe um número máximo de UCEs num mesmo horário³.

²Percurso normal, após tirar licenciatura avança-se para o 2º ciclo, isto é, para mestrado

³Entenda-se por horário um intervalo de tempo, por exemplo: um dia, uma manhã, etc.

2.2 Análise da Problemática

Antes de se avançar com uma análise detalhada da problemática, axou-se importante esclarecer à cerca dos conceitos usados. Realce-se a abstracção relativa aos valores utilizados nas definições, permitindo assim a uma generalização desta análise.

UCE Unidade Curricular de Especialização, sendo que, existem U UCE's diferentes.

Par de UCE's Um termo bastante aplicado, no sentido que, a maioria dos alunos frequenta duas UCE's, que representam, o 1º ano de mestrado (de acordo com o modelo de Bolonha) – existindo ${}^U C_2$ pares de UCE's possíveis.

Horário Um horário pode ser considerado como um intervalo de tempo bem definido, sendo que, a existir H horários, isto significa que nenhum dos H horários se sobrepõe, num horário poderá ser leccionada uma ou várias (num máximo de M) UCE's.

Combinação de Horários Como o próprio nome indica, uma combinação de horários é uma sequência de K horários.

Alocação de UCE a um Horário Permite estabelecer em que horário será leccionada determinada UCE.

Conjunto de Alocações Resultado de atribuir a todas as UCE's um único horário.

Conflito ou Sobreposição Aplica-se quando, duas UCE's são leccionadas no mesmo horário.

Suponha-se, então, que se quer definir em que horários serão leccionadas as várias UCE's, sob as seguintes condições:

- Existem U UCE's distintas;
- São H os diferentes horários para as várias UCE's;
- Está em análise a candidatura de N alunos ao 1º ano de mestrado (cada aluno escolhe um par de UCE's);

O problema maior verifica-se na impossibilidade de um aluno assistir a duas UCE's leccionadas no mesmo horário, assim:

1. Para $U \leq H$, o problema tem solução trivial, uma vez que cada UCE poderá ter o seu horário não existindo qualquer sobreposição;
2. Para $U > H$, o problema complica-se à medida que a diferença $H - U$ vai aumentando, é essencialmente olhando para esta situação que o enunciado deste trabalho prático ganha forma, pelo que, daqui em diante, irá assumir-se como ponto de discussão apenas e só esta situação em concreto;

A partir do momento que o número de UCE's é maior que o número de horários irão ocorrer sobreposições (ou conflitos) de UCE's numa média de,

$\frac{H}{U}$ UCE's por horário.

Realce-se que é referido no enunciado um limite de UCE's por horário por questões laboratoriais, isto quer dizer que, para um limite de M UCE's em colisão por horário deduz-se que o problema não tem solução possível se,

$$\frac{H}{U} > M$$

uma vez que, não é possível encontrar uma combinação de horários que respeite o limite de não ter mais de M UCE's em colisão.

2.2.1 Dimensão do Problema

Pretende-se agora determinar a dimensão do problema⁴, isto é, determinar uma forma de cálculo do número de combinações de alocações. Para isso irá usar-se exemplos práticos com valores pequenos e tentar assim perceber que cálculos são necessários para obter uma fórmula para valores genéricos.

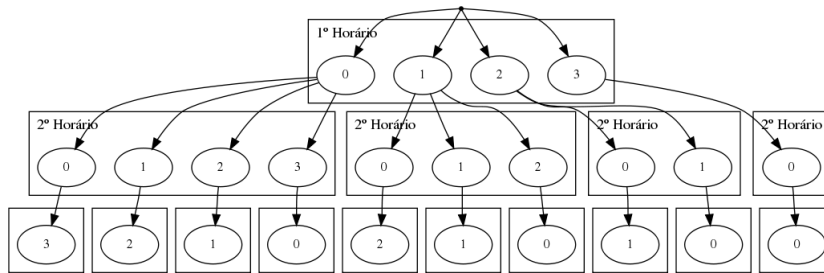


Figura 1: Exemplo da alocação de 3 UCE's a 3 Horários.

Na imagem 1, o valor no interior dos nodos corresponde ao número de UCE's alocadas ao horário em questão, sendo que, esse valor varia entre zero e M (máximo de UCE's num horário). A imagem 1 denota também um pormenor a ter em atenção, isto é, o facto de se alocar u UCE's num determinado horário irá limitar as alocações posteriores, o que se torna óbvio, uma vez que, uma UCE só é leccionada em apenas um horário. Tal facto implica que a fórmula de cálculo genérica seja definida por recorrência, pois é a única forma de matemática de formular problemas deste género. Realce-se ainda um facto importante, a imagem 1 não representa o número de alternativas possíveis, isto porque, imagine-se a situação de dividir uma UCE por horário, só nessa situação iriam existir ${}^3C_1 \times {}^2C_1$ combinações possíveis:

1. $UCE1 \rightarrow UCE2 \rightarrow UCE3$
2. $UCE1 \rightarrow UCE3 \rightarrow UCE2$
3. $UCE2 \rightarrow UCE1 \rightarrow UCE3$
4. $UCE2 \rightarrow UCE3 \rightarrow UCE1$

⁴Sem as limitações causadas pelo facto de um aluno não poder ser alocado a duas UCE's no mesmo horário.

5. $UCE3 \rightarrow UCE1 \rightarrow UCE2$

6. $UCE3 \rightarrow UCE2 \rightarrow UCE1$

Assim sendo, o tamanho do problema para valores genéricos é da forma:

$$size(U, H, M) = \begin{cases} 0 & U < M \times H \\ 1 & H = 0 \\ \sum_{i=0}^{\min(U, M)} U C_i \times size(U - i, H - 1, M) & \text{otherwise} \end{cases}$$

Só para se fornecer ao leitor uma ideia dos valores, apresentam-se alguns exemplos e respectivo valor calculado.

- 2 UCE's
• 2 Horários
• 2 UCE's por Horário no máximo

Resultado 4 Combinações

- 4 UCE's
• 2 Horários
• 2 UCE's por Horário no máximo

Resultado 6 Combinações

- 14 UCE's
• 5 Horários
• 4 UCE's por Horário no máximo

Resultado 29.925.400 Combinações

2.2.2 Melhor Combinação

A alocação (de UCE's aos horários) deverá ser feita em função das escolhas dos alunos. Surge, no entanto, a questão:

Qual é a melhor alocação? Contudo, a resposta não é trivial. Este é um tema bastante discutido, no sentido que, é subjectivo, intimamente ligado à opinião de cada um. Convém no entanto realçar que, a melhor alocação terá de lidar com o seguinte trade-off:

“Dar preferência a determinados alunos poderá ser injusto para a maioria (e vice-versa).”

2.3 Coloração de grafos

2.3.1 Alocação de UCE's

A base deste problema, isto é, a alocação de UCE's aos horários é interpretada, facilmente, segundo um problema de coloração de grafos, sendo que, se faz o seguinte mapeamento:

Vertices Os vertices do grafo são as UCE's.

Arestas As arestas⁵ desse grafo significam que as duas UCE's (vertices) têm, pelo menos, um aluno em comum.

Cor As cores dos vertices representam os distintos horários existentes.

Limitações Como é sabido, a atribuição de uma cor a um vertice em específico está limitada segundo várias condições. Essas condições podem ser distinguidas em **condições intrinsecas ao problema de coloração de grafos e condições extrinsecas**.

As condições intrinsecas, derivam da propria definição do problema de coloração de grafos, sendo que, vertices vizinhos, isto é, que têm uma aresta que os liga, têm cores diferentes. Relativamente às condições extrinsecas, estas derivam da definição do problema original, isto é, alocação de UCE's aos horários, pelo que, dada a limitação física de haver apenas M UCE's num mesmo horário, tal se traduz no problema de coloração de grafos como um limite máximo para o número de vertices da mesma cor.

2.3.2 Alocação de Alunos & UCE's vs Coloração de Grafos

Apesar de existir um mapeamento evidente entre este problema de alocação de UCE's aos horários disponiveis com o problema de coloração de grafos, o problema original apresenta ainda mais algumas complicações, isto porque, existem outras limitações⁶ e que acrescentam um carácter subjectivo ao programa conforme foi dito anteriormente.

Á parte disso, existem alterações ao algoritmo de coloração de grafos que devem ser realizadas de forma a permitir a resolução destes problemas em certos casos. Imagine-se, a titulo de exemplo, a seguinte situação, existem H horários e número cromático⁷ $\lambda(G) = H + 1$. Como é óbvio, o problema original não tem solução nestas condições, mas apesar disso é necessário proceder à alocação de UCE's aos horários, pelo que, existe apenas uma hipótese viável - **remover arestas até que, $\lambda(G) \leq H$** .

Contudo ao proceder-se à remoção de uma aresta é necessário perceber de que forma vai afectar o problema original. Ao remover uma aresta está-se implicitamente a definir que nenhum aluno poderá estar alocado às duas UCE's

⁵Sendo que uma aresta liga, como é óbvio, dois vertices.

⁶Não abordadas anteriormente.

⁷ $\lambda(G)$ número mínimo de cores de forma a garantir a condição de vizinhança.

ao mesmo tempo, para não existirem sobreposições, isto terá implicações no algoritmo, sendo que, uma das mais importantes no contexto deste trabalho é o aumento considerável do tempo de execução⁸. Ao efectuar essa operação será necessário proceder à realocação dos alunos afectados a outros pares de UCE's, esta operação vai implicar uma descida dos níveis de satisfação dos alunos, sendo que é ainda assunto de discussão dado que é necessário escolher qual aresta se vai remover, podendo nesta situação beneficiar a maioria em detrimento dos melhores, ou vice-versa.

⁸O tempo de execução será discutido em maior detalhe mais à frente.

3 Decisões de Implementação

Na realização deste projecto foram tomadas várias decisões que influenciaram o resultado final. Foi tomada uma posição concreta à cerca da remoção de uma aresta, **decidiu-se beneficiar a maioria**, isto é, o peso de uma aresta é definido pelo número de alunos que formam esta sobreposição de horários. Esta decisão foi tomada tendo em conta os seguintes critérios:

Tempo de execução Na outra situação, seria necessário calcular o peso de cada aresta, sendo uma operação mais pesada incutir a noção de viciação do peso para os melhores;

Subjectividade do processo Os melhores poderiam nem ser realmente bons;

Injustiças Se a comunidade de alunos fosse muito homogénea, em termos de notas, beneficiar os melhores pode nem sempre ser justo para todas as partes;

Decidiu-se perder mais tempo na implementação do algoritmo de coloração de grafos, isto é, no problema base do problema original. Mas foi também necessário tomar uma decisão sobre qual algoritmo utilizar, sendo que, houve uma inércia inicial, dado que, existe na rede bastantes algoritmos que resolvem o problema. Tomou-se então a iniciativa de comparar as várias soluções existentes, escolhendo-se a que melhor cumpria os requisitos.

Reduções Apesar de se terem encontrado várias implementações de algoritmos para resolver o problema de coloração de grafos, existe uma alternativa cada vez mais em uso no mundo da informática. A coloração de grafos, como outros problemas bastante conhecidos da informática, é um problema **NP-Completo**⁹, é também facto que, encontrada uma solução para um problema **NP-Completo** essa solução pode ser aplicada todos os outros problemas da mesma classe, tal é conseguido através de uma Redução.

Imagine-se que existe um algoritmo π que resolve um problema A (pertencente à classe NP-Completo), sendo B um problema também da classe NP-Completo, então é possível aplicar o algoritmo π para resolver B , usando para isso uma redução, isto é, transformando o problema B de forma a que o output do algoritmo π seja solução do problema. Ou seja, $A \propto B$ (A reduz-se a B) se e só se,

$$\exists T : \pi \text{ resolve } T(B).$$

A razão pela qual não se optou por esta via, de onde se retirariam vantagens como algoritmos e heurísticas já maduras, uma maior documentação sobre outros problemas NP-Completo, mais concretamente o problema conhecido como SAT¹⁰, deve-se ao facto de ser um assunto desconhecido, pelo que, exigia um estudo detalhado e cuidadoso, e portanto, mais tempo. Dada a falta de tempo não se optou por esta via, preferiu-se uma implementação de um algoritmo já existente e directamente vocacionado para resolver este tipo de problemas.

⁹É o subconjunto dos problemas de decisão em NP.

¹⁰Cálculo de uma valoração de símbolos de uma fórmula proposicional de forma a que esta seja verdadeira

3.1 Comparações dos Algoritmos

Aproveitando o estudo levado a cabo por *Walter Klotz*, restringiu-se o estudo a três algoritmos conhecidos:

DSATUR Degree of Saturation

RLF Recursive-Largest-First

IBSC Incomplete Backtracking Sequential Coloring

Resumindo o estudo de *Walter Klotz*, este implementou os vários algoritmos e executou-os sobre os mesmos grafos, usando o tempo de execução (real) como comparação entre os algoritmos. As experiências foram realizadas sobre grafos com 60 vértices com arestas geradas aleatoriamente. O gerador de grafos permitiu controlar a densidade de arestas, isto é, o número médio de saídas de cada vértice. Cada algoritmo foi aplicado ao grafo gerado, sendo que, foram gerados 100 grafos por cada nível de densidade de arestas[2].

Arestas	DSATUR	RLF	IBSC
1.1	4.18 (1.84)	4.17 (10.13)	4.00 (2.99)
1.3	8.21 (2.89)	7.96 (11.95)	7.57 (27.38)
1.5	12.50 (4.62)	11.94 (12.06)	11.52 (45.68)
1.7	18.08 (5.82)	17.18 (13.05)	16.83 (53.87)
1.9	27.49 (8.25)	27.26 (13.26)	26.23 (43.32)

Na tabela estão representados os números de cores e dentro de parêntesis os tempos de execução. Como se pode observar, o melhor algoritmo em termos de custo benefício é o **DSATUR**. Sendo portanto este o algoritmo escolhido para ser implementado.

3.2 Estruturas de dados

Relativamente às estruturas de dados foi efectuado um estudo bastante apurado a este nível. Foram equacionadas todas as hipóteses, tomando-se as seguintes decisões quanto à representação dos dados. Para facilitar a compreensão do leitor, vai-se proceder à apresentação das estruturas de dados auxiliares e de seguida às estruturas de dados principais.

3.2.1 Listas

Realça-se que se procedeu à implementação destas estruturas de dados auxiliares, de forma a, poderem albergar qualquer tipo de dados.

LinkedList Para a listas ligadas definiu-se uma struct, onde o único campo de dados é um apontador para `void`, permitindo assim apontar para uma posição de memória que preserva qualquer tipo de dados.

```
typedef struct Element {
    void *data;
    struct Element *next;
} LinkedList;
```

ArrayList Por forma a libertar de tarefas futuras o peso de dimensionar os arrays, decidiu-se implementar uma estrutura de dados que, com a ajuda de algumas funções uteis tirasse o peso da gestão de memória em fases mais avançadas do projecto.

```
typedef struct {
    void **list;
    int size;
    int elems;
} ArrayList;
```

Queue As queues são estruturas baseadas no princípio FIFO (first in, first out), em que os elementos que foram inseridos no início são os primeiros a serem removidos. Uma fila possui duas funções básicas: ENQUEUE, que adiciona um elemento ao final da fila, e DEQUEUE, que remove o elemento no início da fila.

A operação DEQUEUE só pode ser aplicado se a fila não estiver vazia, causando um erro de underflow ou fila vazia se esta operação for realizada nesta situação.

```
typedef struct {
    LinkedList *in;
    LinkedList *out;
} Queue;
```

3.2.2 Hash Tables

Uma hash table (tabela de dispersão) é uma estrutura de dados especial, que associa chaves de pesquisa a valores. Seu objetivo é, a partir de uma chave simples, fazer uma busca rápida e obter o valor desejado. É algumas vezes traduzida como tabela de escrutínio.

HashMapElem Esta estrutura foi criada para se estabelecer um padrão para todas as hash maps.

```
typedef int (*hashFunction)(void *,int);
```

```
typedef struct {
    void *key;
    void *data;
} HashMapElem;
```

O tipo **hashFunction**, recebe uma chave (apontador para **void**) e um inteiro que é o tamanho da tabela de hash, dando como resultado um inteiro que é o valor de hash da chave.

ChainingHashMap Implementação de uma hash map que usa **chaining** (encadeamento) como metodologia para resolver colisões. O encadeamento é a solução mais simples, em que normalmente um índice da tabela aponta para uma lista onde são armazenados os elementos em conflito. A inserção na tabela requer uma busca e inserção dentro da lista.

```
typedef int (*equalFunction)(void *,void *);
```

```
typedef struct {  
    LinkedList **list;  
    int size;  
    hashFunction hash;  
    equalFunction eq;  
} ChainingHashMap;
```

O tipo **equalFunction** define uma função que recebe dois apontadores para **void** e retorna um inteiro, zero se os dados armazenados em memória forem diferentes, ou diferente de zero se forem iguais.

3.2.3 Aluno

O aluno, para além dos dados óbvios (nome, ano de candidatura, média, etc.) irá ter associado a si a informações das preferências sobre as várias UCE's disponíveis.

```
typedef enum { NORMAL, PROFESSIONAL, EXTRA } StudentType;
```

```
typedef struct {  
    char *name;  
    int number;  
    StudentType type;  
    int average;  
    LinkedList *selections;  
    int year;  
    int ucesWanted;  
} Student;
```

Tais informações estão armazenadas numa lista ligada (*LinkedList*) sob a forma de pares (UCE, Preferência), para isso foi criado uma estrutura de dados.

```
typedef struct {  
    char *uceName;  
    int preference;  
} Selection;
```

3.2.4 UCE

Uma UCE irá guardar apenas informações simples.

```
typedef struct uce {  
    char *name;  
    int maxStudents;  
} UCE;
```

Básicamente, uma UCE vai guardar o seu nome (único) e o número máximo de estudantes que suporta.

3.2.5 Grafo

Tipicamente, um grafo é representado como um conjunto de pontos (vértices) ligados por retas (as arestas). Dependendo da aplicação, as arestas podem ser direcionadas, e são representadas por setas.

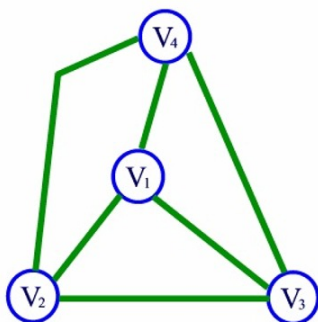


Figura 2: Exemplo de um grafo.

As estruturas de dados lecionadas para representar grafos são todas elas vocacionadas para o armazenamento de grafos direccionados. Decidiu-se então implementar uma estrutura de dados especial para representação de grafos não direccionados.

```
typedef struct {  
    int nodes;  
    int *edge;  
} Graph;
```

Como se pode ver, esta estrutura tem a informação à cerca do número de nodos, isto é, vertices e um arrays de inteiros.

Relativamente a esta estrutura pode-se estabelecer as seguintes propriedades.

Número máximo de arestas Também, o tamanho do array *edge*. Sendo $|V|$ o número de vertices,

$$\text{maxArestas}(|V|) = \frac{|V| \times (|V| + 1)}{2}$$

Posição de uma Aresta Para saber em que posição do array *edge* está a informação à cerca da aresta é necessário saber o indice de cada vertice. Sendo que,

$$\text{posicao}(vA, vB) = \frac{vA \times (vA - 1)}{2} + vB$$

Essencialmente, é preservado em memória apenas o triângulo inferior da matriz de arestas.

3.3 Implementação

3.3.1 DSATUR

A implementação do DSATUR parte da resolução de um outro problema bastante semelhante à coloração de grafos - Clique.

Clique Dado um grafo $G = (V, E)$, um clique do gráfico é um conjunto de vértices mutuamente adjacentes. Um clique máximo é um clique cujo número de vértices é maior ou igual ao de outro qualquer clique do grafo. Se se tratar de um grafo pesado, então o maior clique pesado é o clique cuja a soma dos pesos dos seus arcos seja máxima. Clique e coloração de grafos são problemas muito relacionados. É simples ver que o tamanho do maior clique é um limite inferior para o mínimo de etiquetas necessárias para colorir um grafo. muitos problemas de interesse prático podem ser modelados como problemas de clique e coloração de grafos. Ambos os problemas pertencem à classe NP, pelo que, prevê-se difícil a tarefa de encontrar um algoritmo em tempo polinomial.

Algoritmo O algoritmo para encontrar um clique que se aproxima ao maior clique parte de encontrar o vertice com mais saídas e encontrar o clique cujos vértices também incluem este vertice¹¹. Depois o problema de otimização é resolvido percorrendo os vértices que não estavam incluídos nos cliques anteriores continuando até que não haja vértices por percorrer nestas condições, guardando como resultado o maior clique encontrado [3].

Exemplo De seguida apresenta-se um exemplo do funcionamento deste algoritmo.

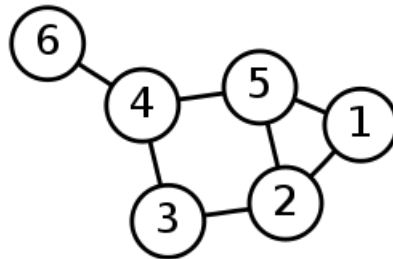


Figura 3: Grafo por processar.

Inicialmente começa-se por identificar o vertice com mais saídas e posteriormente calcular o clique que o inclui. No exemplo acima existem várias alternativas idênticas, imagine-se que o vertice escolhido seria o vertice 2.

¹¹Resolução do problema de decisão.

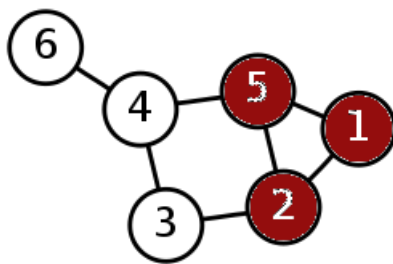


Figura 4: 1ª proposta para maior clique.

De seguida é calculado o próximo clique que não envolve os vertices dos cliques anteriores.

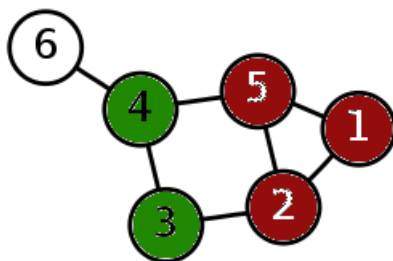


Figura 5: 2ª proposta para maior clique.

O mesmo acontece até que não haja vertices por incluir em cliques.

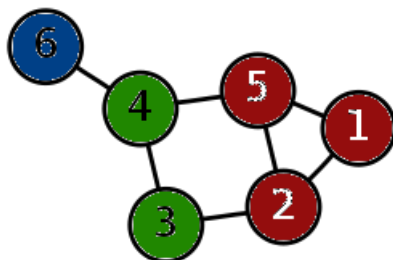


Figura 6: Última proposta para maior clique.

Após descobrir o maior clique é então atribuída uma cor diferente a cada vertice pertencente a este clique. Posteriormente são preenchidos os outros vertices com as cores usadas anteriormente.

No contexto da alocação de alunos às UCE's e das UCE's aos horários, é ainda efectuada a verificação do número de cores máximo e testadas novas alternativas de coloração após passar estes teste, este algoritmo é executado ciclicamente até que se tenha como resultado um número de cores menor ou igual ao número de horários, sendo que, entre execuções deste algoritmo são removidas arestas.

4 Tempo de Execução

A análise do tempo de execução permitiu, para além de apresentar uma estimativa assintótica do tempo de execução concluir que existe uma falha do programa no contexto do problema original, que anteriormente não tinha sido detectada.

4.1 Mínimo de Horários sem remover arestas

Pretende-se saber qual o tempo de execução teórico do programa quando se calcula o número mínimo de horários para que não haja alunos a entrar em segundas escolhas. Para simplificar a análise do tempo de execução avaliou-se uma função de cada vez.

O algoritmo implementado baseia-se na chamada de três funções fundamentais:

- $clique(validNodes)$
- $maxClique()$
- $color(clique)$

$clique(validNodes)$ é uma função que calcula um clique num grafo. Como é uma função percorre sempre todas as arestas então o seu tempo de execução é:

$$T_{exec}(clique) = \Theta(|V|^2).$$

$maxClique()$ invoca ciclicamente a função clique, sendo que, fornece como input os vertices que ainda não fizeram parte de nenhum clique anteriormente calculado. Uma análise ponderada permite saber que o seu tempo de execução depende do número de cliques. No melhor caso existe apenas 1 clique, no pior caso existe $|V|$ cliques. Ou seja, o tempo de execução da função $maxClique$ é de,

$$T_{exec}(maxClique) = \Omega(|V|^2) \wedge T_{exec}(maxClique) = O(|V|^3).$$

A coloração do grafo depende também ela do número de cliques, mais precisamente, do número de vertices pertencentes ao maior clique. Este valor (número de vertices) é um limite inferior para o número de cores atribuídas. A atribuição de cores aos vertices tem então um tempo de execução dependente desse valor, sendo $|C|$ o número de cores utilizadas na coloração do grafo, sem olhar ao facto da função $color$ invocar a função $maxClique$ uma vez, temos que,

$$T_{exec}(color) = \Theta(|V|^2 - |V| \times |C|).$$

Contudo, a função $color$ invoca a função $maxClique$ o que em termos assintóticos significa que,

$$T_{exec}(color) = \Omega(|V|^2) \wedge T_{exec}(color) = O(|V|^3).$$

Tempo de Execução O tempo de execução deste algoritmo depende de apenas uma grandeza - $|V|$ - isto é, o número de nodos, sendo que, no pior caso,

$$T_{exec} = \Theta(|V|^3).$$

4.2 Tempo de execução para alocação

Para esta operação o tempo de execução aumenta, dado que, é executada a função *color* ciclicamente, até que se cumpram todos os requisitos, isto é, não haja mais de M cores (limite de laboratórios) e também não haja mais que H cores distintas, isto é, número de horários atribuídos. Sendo que, entre cada chamada ciclica é removida uma aresta (forward-only¹²). A remoção é uma operação que identifica qual a aresta a remover (linear no número de arestas) posteriormente remove a aresta, isto é, percorre os alunos em comum das duas UCE's e coloca-os nas suas segundas preferências, ao colocar nas suas segundas preferências é necessário perceber que podem nem poder entrar em segundas preferências se essas arestas já tiverem sido removidas. O tempo de execução de uma realocação, pode ser, no melhor caso, de $\Theta(1)$ e no pior caso $\Theta(|E|)$. Como se pretende a realocação de todos os alunos com sobreposição, temos também dois casos, todos os N alunos estarem em sobreposição, ou então não estar nenhum. Uma avaliação de pior e melhor caso permitem estabelecer,

$$T_{exec}(removerAresta) = O(N \times |E|) \wedge T_{exec}(removerAresta) = \Omega(1).$$

Tempo de Execução A execução deste algoritmo depende de três grandezas,

1. N - Número de alunos inscritos;
2. $|E|$ - Cardinalidade das arestas;
3. $|V|$ - Número de vertices;

Sendo que, no pior caso,

$$T_{exec} = \Theta(|V|^3 \times |E| \times N).$$

¹²Este é um problema do programa como será abordado na conclusão.

5 Extras

Para além das funcionalidades pedidas no enunciado foram também desenvolvidas algumas funcionalidades extra. De todas realça-se a interface com o utilizador, na qual se dispendeu bastante tempo de desenvolvimento para tornar a tarefa do utilizador o mais simples possível. É possível observar que existe a noção de projecto, do qual é possível carregar a informação gravada anteriormente. Para além disso, a linha de comandos está bastante robusta. É possível configurar certos aspectos da aplicação, tais como as constantes abordadas no enunciado, isto é, é possível modificar o número máximo de UCE's por horário, bem como o número de horários disponíveis.

6 Conclusão

Os objectivos deste projecto foram cumpridos na sua plenitude. Para além disso, realça-se o aspecto lúdico no qual se apoiava este projecto. Foi necessário proceder ao estudo de várias matérias tendo como objectivo o resultado final. Foi bastante cativante implementar esta aplicação atendendo à sua complexidade, que necessitou de uma atenção redobrada, bem como, pela forma minuciosa como se atacou cada problema.

Alguns dos conceitos abordados no contexto deste projecto foram novidades contribuindo desta forma para tomar conhecimento de alguns problemas bastante conhecidos no mundo da informática, realçando-se sem dúvida o problema da coloração de grafos e cálculo do maior clique.

Futuro Em termos de futuro, é possível melhorar esta aplicação em várias frentes, desde a apresentação ao funcionamento. Contudo, no contexto desta disciplina seria interessante optar por implementar uma redução do problema de coloração de grafos ao problema de SAT. Poder-se-ia ainda aumentar as condições sobre as alocações, como por exemplo, adicionar a capacidade de não permitir a sobreposição de UCE's em concreto. Foi descoberto um problema com a aplicação na altura de avaliação do tempo de execução, isto porque, não foi considerada a situação de que remover duas arestas com menor peso pode resultar numa maior insatisfação que a remoção de uma aresta de peso superior, aliás a ordem pela qual se removem as arestas, isto é, se forem removidas duas ou mais arestas também surgem como falha na implementação desta aplicação, sendo que, seria de facto obrigatório implementar estas situações.

Referências

- [1] Coloração de grafos - construção de horários.
- [2] Walter Klotz. Graph coloring algorithms. *Internet*, 1998.
- [3] Mike Trick. Clique and coloring problems. *Internet*, 1992.