Introdução à Inteligência Artificial

Licenciatura em Engenharia Informática, Engenharia Informática – Pós-Laboral e Engenharia Informática – Curso Europeu

2º Ano – 1º semestre

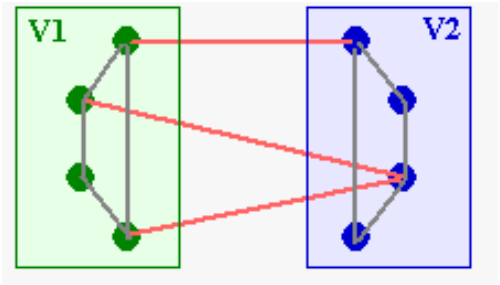
Aulas Laboratoriais

Ficha 6: Pesquisa Local

1. O problema da bissecção mínima de um grafo

A bissecção de um grafo é um problema de optimização que surge frequentemente em situações do mundo real (por exemplo, em problemas de escalonamento ou *design* de circuitos VLSI) e que consiste em dividir os vértices por dois conjuntos com o mesmo número de elementos, de tal forma que o custo do corte efetuado seja mínimo.

Este problema pode ser definido da seguinte forma:



- Existe um grafo G = (V, A) com um número par de vértices, V e com um número arcos, A;
- Pretende-se dividir o conjunto V em dois conjuntos disjuntos V1 e V2 com o mesmo número de vértices;
- O custo da bissecção é dado pelo número de arcos que efetuam ligações entre vértices que pertencem a conjuntos diferentes (ou seja, arcos que ligam um vértice de V1 a um vértice de V2);
- O objetivo da otimização é minimizar o custo da bissecção.

2. Componentes dos algoritmos de optimização

Vão ser utilizados diferentes algoritmos de pesquisa local para efetuar a otimização. As componentes descritas a seguir são comuns a todos eles.

2.1. Representação

Numa instância com N vértices, a representação de uma solução deve indicar quais os N/2 vértices que pertencem ao conjunto V1 e quais os N/2 vértices que pertencem ao conjunto V2.
Para isso, recorre-se a um vector binário com N posições em que cada bit representa um vértice do grafo. Uma posição com o valor '0' indica que esse vértice pertence ao conjunto V1, enquanto o valor '1' indica que pertence a V2.

> **Exemplo**: Numa instância com 6 vértices, a solução 101100 especifica que os vértices 1, 3 e 4 pertencem a V1 e os vértices 2, 5 e 6 a V2.
> Para que as soluções sejam válidas é necessário garantir que o número de '0' e de '1' é sempre igual.

2.2. Avaliação e objetivo da otimização

A qualidade de uma solução é igual ao custo da bissecção proposta. O objectivo é encontrar uma solução em que este custo seja mínimo (problema de minimização).

2.3. Vizinhança inicial

Para obter um vizinho de uma solução deve proceder-se da seguinte forma:

- Escolher uma posição P1 do vector binário com valor '0';
- Escolher uma posição P2 do vector binário com valor '1';
- Trocar os valores dos bits nas posições P1 e P2.

Na prática, esta vizinhança escolhe um vértice em cada um dos conjuntos V1 e V2 e troca-os. Com esta vizinhança garante-se que o número de '0' e '1' se mantém igual.

3. Trepa-colinas *first choice*

O primeiro algoritmo a utilizar na optimização é o trepa-colinas *first choice*.

3.1. Detalhes de implementação

Ficheiros de dados:
No início da execução, o programa tem que aceder a um ficheiro de texto com informação sobre o problema a otimizar e usar essa informação. O ficheiro deverá ser estruturado com os seguintes elementos importantes (por esta ordem):

- Número de máximo de iterações;
- Número de vértices do problema;
- Matriz de adjacências (ou de ligações).

Para um problema com N vértices, a matriz de adjacências (ou de ligações) tem dimensão N×N. O valor '1' na posição [i, j] indica que existe uma ligação entre o vértice i e o vértice j.
Um exemplo de um ficheiro deste tipo pode ser visto de seguida. Como se pode verificar o ficheiro informa que o algoritmo de otimização deverá efetuar 100 iterações (primeira linha do ficheiro), que o problema terá dez vértices (segunda linha

do ficheiro) e que, por exemplo, o vértice 1 tem ligações com os vértices 2, 3, 4, 5, 6, 9 e 10 (colunas da primeira linha da matriz de adjacências que têm o valor '1').

```
100

10

0  1  1  1  1  1  0  0  1
   1
1  0  1  1  1  1  1  0  0
   0
1  1  0  1  1  1  1  1  0
   0
1  1  1  0  1  1  1  1  1
   0
1  1  1  1  0  1  1  1  1
   1
1  1  1  1  1  0  1  1  1
   0
0  1  1  1  1  1  0  1  1
   0
0  0  1  1  1  1  1  0  1
   0
1  0  0  1  1  1  1  1  0
   0
1  0  0  0  1  0  0  0  0
   0
```

Código:

O projecto em C tem 4 módulos:

- *main.c*
  Contém a função que controla o funcionamento geral do programa, efetuando as seguintes chamadas:
  - Funções de preparação de aplicação do algoritmo (leitura de dados, geração da solução inicial);
  - Aplicação do trepa-colinas;
  - Escrita dos resultados finais no monitor.
- *funcao.c*
  Contém a função que avalia a qualidade de uma solução.
- *algoritmo.c*
  *C*ontém a função principal de aplicação do trepa-colinas e a função que gera um vizinho de uma determinada solução.
- *utils.c*
  Contém algumas funções auxiliares como a de geração de números aleatórios, criação da primeira solução e leitura do ficheiro de dados.

Notas sobre a implementação:

- O nome do ficheiro de dados pode ser especificado como primeiro argumento da linha de comando. Se não existirem argumentos, o utilizador deve indicar o nome do ficheiro no início da execução do programa;

- O número de vértices do problema a ser resolvido é obtido pela função *init_dados()* através da leitura do ficheiro e passado como argumento para a função *trepa_colinas()*;
- A matriz de adjacências é preenchida na função *init_dados()* e passada como argumento para a função *trepa_colinas()*;
- Na função *trepa_colinas()*, o vector *sol* armazena a melhor solução encontrada até ao momento. A nova solução obtida na vizinhança é armazenada no vector *nova_sol*. Se, após a avaliação se verificar que esta é melhor, é efetuada a troca;
- O espaço ocupado pelos vectores onde são armazenadas as soluções e pela matriz de adjacências é requisitado de forma dinâmica.

Realização de experiências com algoritmos probabilísticos

O trepa colina *first-choice* é um método probabilístico, pelo que uma única execução do algoritmo não permite obter resultados estatisticamente válidos. Sem esses resultados é impossível avaliar a eficácia do algoritmo com precisão. Sempre que o algoritmo contenha componentes probabilísticas, é obrigatório efetuar várias repetições da mesma configuração. No mínimo devem ser efectuadas 10 repetições (30 é o valor desejável). Ao comparar 2 algoritmos probabilísticos (ou 2 configurações diferentes para o mesmo algoritmo), existem duas medidas de desempenho essenciais:

- Melhor solução obtida – qualidade absoluta da melhor solução encontrada;
- Média da melhor avaliação (*Mean best fitness*) – média obtida a partir das melhores soluções encontradas em cada uma das repetições.

Na implementação disponibilizada é possível indicar através da linha de comando o número de repetições a efetuar (segundo argumento da linha de comando). Se este argumento não for especificado, serão realizadas 10 repetições (valor por defeito).

4. Trabalho a realizar no laboratório

4.1. Realização de Experiências

- Efetuar algumas experiências com os 5 ficheiros de teste disponibilizados: grafo_20.txt, grafo_50.txt, grafo_100.txt, grafo_150.txt e grafo_250.txt. As situações armazenadas nestes ficheiros possuem respectivamente 20, 50, 100, 150 e 250 vértices;
- Preencher a respetiva tabela (no ficheiro *IIA_Ficha6_Resultados.xlsx*) para cada uma das instâncias de teste. Nela pode analisar a influência que o número máximo de iterações tem no desempenho do algoritmo. Tente justificar eventuais diferenças de resultados.

4.2. Aceitar soluções com o mesmo custo

- Alterar o código de modo a permitir que o trepa-colinas aceite vizinhos com o mesmo custo;
- Efetuar algumas experiências, preenchendo a respetiva tabela (no ficheiro *IIA_Ficha6_Resultados.xlsx*), com os grafos de 100 e 150 vértices, e verificar se existem alterações nos resultados.

4.3. Nova vizinhança

- Implementar uma nova vizinhança para a representação utilizada;

- Repetir as experiências realizadas na secção 4.1., preencher a respetiva tabela (no ficheiro *IIA_Ficha6_Resultados.xlsx*) para cada uma das instâncias de teste e analisar eventuais diferenças em relação aos resultados anteriores.

4.4. Trepa-colinas probabilístico

- Alterar o código de modo a permitir que o trepa-colinas aceite a possibilidade de ter soluções piores. Essa aceitação deverá ser baseada numa determinada probabilidade fixa;
- Efetuar algumas experiências, preenchendo a respetiva tabela (no ficheiro *IIA_Ficha6_Resultados.xlsx*), com os grafos de 100 e 150 vértices, e verificar se existem alterações nos resultados.

4.5. Recristalização simulada

- Implementar um algoritmo de recristalização simulada para este problema;
- Realizar experiências com diferentes opções para os componentes do algoritmo:
    - TMax;
    - TMin;
    - Função de arrefecimento;
    - Número de iterações do ciclo interno.
- Repetir as experiências realizadas com o trepa-colinas;
- Analisar e justificar os resultados obtidos.

Pode utilizar a função *rand_01()* do módulo *utils.h* para ajudar a decidir se o vizinho é ou não aceite como nova solução.