Uma imagem com Tipo de letra, Gráficos, texto, design gráfico

Descrição gerada automaticamente

Departamento de Engenharia Informática e Sistemas (DEIS)

Introdução à Inteligência Artificial 2023/2024

Trabalho Prático nº 2

**Problema de Otimização**

André Dias – a2021140917

João Pascoal – a2017009420

## ÍNDICE

[Introdução 3](#_Toc153736004)

[Algoritmo de pesquisa local 4](#_Toc153736005)

[**Inicialização da matriz** 4](#_Toc153736006)

[**Algoritmo** 4](#_Toc153736007)

[**Funções mais importantes** 4](#_Toc153736008)

[**Resultados** 5](#_Toc153736009)

[Algoritmo Evolutivo 7](#_Toc153736010)

[**Funções mais importantes** 7](#_Toc153736011)

[**Resultados** 8](#_Toc153736012)

# Introdução

Neste relatório vamos analisar o Subconjunto de custo mínimo.

O desafio do "Subconjunto de Custo Mínimo" em grafos não direcionados busca identificar um grupo de vértices de tamanho k, interconectados por arestas, minimizando o custo total interno. Esta busca pela eficiência máxima requer métodos de otimização capazes de encontrar soluções de alta qualidade para diferentes cenários do problema. Este estudo visa conceber, implementar e testar tais métodos, oferecendo soluções de qualidade excecional para este desafio complexo da teoria dos grafos

# Algoritmo de pesquisa local

### **Inicialização da matriz**

* Foi criada uma matriz V \* V contendo os valores que foram disponibilizados nos ficheiros de teste.

### **Algoritmo**

* Um algoritmo de pesquisa local, em termos genéricos, é um algoritmo que recebe um problema como entrada e retorna uma solução válida para o mesmo, depois de resolver um certo número de possíveis soluções.
* Neste trabalho o algoritmo que utilizámos foi o trepa-colinas. O trepa-colinas parte de um estado inicial aleatório, define um critério de vizinhança, avalia todos os seus vizinhos e vê qual o melhor, se houver algum melhor ele aceita o mesmo. De seguida é iniciada uma nova iteração e repete esse procedimento até chegar a um estado em que todos os vizinhos sejam de qualidade inferior para o contexto do problema apresentado.

### **Funções mais importantes**

“**initDados**” nesta função é onde vamos ler os ficheiros de texto e preencher a matriz com os respetivos valores, além de atribuir o custo na posição correspondente.

“**geraSolInicial**” esta função recebe um array onde vamos guardar as soluções, o número de vértices e o valor de “k”, a função vai depois preencher esse mesmo array.

“**calculaFit**” esta função vai calcular o valor de ajuste dado uma solução representada pelo array “a”. Primeiro inicia-se com o valor 0 o array “con” que vai servir para verificar se um vértice está conectado a outro vértice. De seguida percorremos todos os vértices do grafo e verificamos o vértice “i” está incluído na solução e depois verificamos as suas conexões. Se o vértice “j” também estiver incluído na mesma solução(“a[j] == 1”) e se houver uma aresta entre i e j (“mat[i \* vert +j] !=0”) então ambos os vértices são marcados como conectados(“con[i] = 1” e “con[j] = 1”) e o custo da aresta é adicionado á variável total. A função vai devolvar o valor de total/2 para evitar a contagem duplicada de arestas. Nesta função é ainda ou se vai implementar as penalizações e os mecanismos de reparação.

“**geraVizinho**” é gerado um ponto p1 aleatório entre 0 e n-1, e de seguida acontece o mesmo com um ponto p2. O p2 vai ser gerado num ciclo enquanto for igual a p1, ou seja, vai garantir que sejam índices diferentes. Quando se der essa situação é feita uma substituição dos valores dos respetivos índices, trocando p1 com p2 gerando assim um novo vizinho.

### **Resultados**

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente

Podemos observar que o trepa-colinas lida bem com o problema e em todos os ficheiros conseguimos atingir a solução ideal, ou então chegar relativamente perto, possivelmente até atingindo com mais iterações.

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente

Nesta experiência fizemos com que o trepa-colinas aceita-se soluções de custo igual, mas não se notou nenhuma diferença significativa, portanto podemos concluir que não alterou em muito os resultados.

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente

De seguida, além de aceitar situações de custo igual transformamos o trepa-colinas num trepa-colinas probabilístico, mas também não houve uma melhoria significativa nos resultados.

# Algoritmo Evolutivo

### **Funções mais importantes**

“**initDados**” nesta função é onde vamos ler os ficheiros de texto e preencher a matriz com os respetivos valores, além de atribuir o custo na posição correspondente. Nesta situação os resultados foram todos guardados numa estrutura de dados do tipo “info”.

“**geraSolInicial**” esta função recebe um array onde vamos guardar as soluções, o número de vértices e o valor de “k”, a função vai depois preencher esse mesmo array.

“**calculaFit**” esta função vai calcular o valor de ajuste dado uma solução representada pelo array “a”. Primeiro inicia-se com o valor 0 o array “con” que vai servir para verificar se um vértice está conectado a outro vértice. De seguida percorremos todos os vértices do grafo e verificamos o vértice “i” está incluído na solução e depois verificamos as suas conexões. Se o vértice “j” também estiver incluído na mesma solução(“a[j] == 1”) e se houver uma aresta entre i e j (“mat[i \* vert +j] !=0”) então ambos os vértices são marcados como conectados(“con[i] = 1” e “con[j] = 1”) e o custo da aresta é adicionado á variável total. A função vai devolver o valor de total/2 para evitar a contagem duplicada de arestas. Nesta função é ainda ou se vai implementar as penalizações e os mecanismos de reparação.

“**initPop**” esta função recebe uma estrutura que contém os parâmetros do problema. Vamos ter uma variável “indiv” que é um ponteiro para uma estrutura do tipo “chrom” que representa um individuo na população, depois vamos alocar dinamicamente essa variável para armazenar a população de indivíduos. Depois temos um ciclo sobre cada individuo em que cada individuo vai chamar a função “geralSolInicial” para preencher o seu “gene” com valores binários.

Foram ainda implementados 2 métodos de seleção diferentes (Tournament e Tournament2), 2 operadores de recombinação diferentes (crossover e crossover2) a diferenças entre as recombinações são: A função crossover utiliza um ponto aleatório enquanto a crossover2 utiliza dois pontos aleatórios, mas garante que sejam diferentes e ordena-os. Além disto temos ainda 3 operadores de mutação diferentes (Mutation, Mutation2 e Mutation\_por\_troca. A Mutation inverte independentemente os valores dos genes, a Mutation2 inverte os valores entre uma posição de valor 0 e uma posição de valor1, e a Mutation\_por\_troca troca os valores entre duas posições.

### **Resultados**

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente

Nesta experiência testamos para um número diferente de runs os operadores genéticos crossover(recombinação) e mutation(mutação) com as suas probabilidades ao máximo. Podemos observar que sem penalização ou reparação nos ficheiros maiores os valores ficam longe do desejado.

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente

Nesta experiência baixámos os valores das probabilidades de recombinação e mutação, e utilizámos a mutação por troca, e podemos ver que os valores especialmente nos ficheiros mais pequenos chegaram aos valores ideais, e no “file4.txt” chegou bastante perto podendo até assumir que chegaria lá com um número mais elevado de runs.

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente

Nesta experiência além de usarmos um método de recombinação e mutação alternativos, utilizámos também a penalização, e embora para os ficheiros mais pequenos os valores sejam próximos do ideal, podemos observar que para o ficheiro “file5.txt” os valores ficaram bastante longes do esperado, só melhorando com 20 runs, mas longe do esperado.

Uma imagem com texto, captura de ecrã, número, Paralelo

Descrição gerada automaticamente

Enquanto que nesta experiência utilizámos um método de seleção alternativo (Tournament2) para verificar o impacto que teria nos resultados, e podemos verificar que os resultados não melhoraram como alguns até pioraram portanto concluído assim que não trás vantagens para a resolução do problema.

### **Conclusão**

Uma conclusão que podemos tirar no algoritmo evolutivo é que baixando as probabilidades de recombinação e de mutação os valores melhoram, assim como quanto maior o número de runs melhores os resultados. Também podemos tirar a conclusão de que a mutação por troca é aquela que gera melhores resultados.

Quanto ao trepa-colinas, podemos observar que teve uma boa eficácia, mas que nem aceitar soluções de custo igual, nem aumentar o número de vizinhanças, nem transformar em trepa-colinas probabilístico afeta muito os resultados, mas o numero de iterações afeta sim proporcionalmente o resultado.