Relatório

Introdução à Inteligência Artificial

Trabalho Prático nº 2

Problema de Otimização

Trabalho realizado por:

Gonçalo Ramalho – 2019106561

Índice

Introdução

Este relatório tem como base o estudo de algoritmos de otimização para encontrar soluções de boa qualidade, efetuando um estudo comparativo e aprofundado sobre o desempenho de cada um.

A base providenciada pelos docentes consiste num problema de otimização em que o objetivo é maximizar a solução do problema. É necessário utilizar linguagem C para resolver o problema em questão. É-nos dado vários ficheiros de texto, com complexidades diferentes que temos de usar para obtermos os dados iniciais. A partir destes temos preparar o programa para lidar com estes dados.

Temos três tarefas para fazer, a primeira é criar um algoritmo de pesquisa local que explore várias vizinhanças, a segunda é implementar um algoritmo evolutivo em que deve ser testado com vários operadores genéticos, e a terceira é combinar estes dois algoritmos, criando um algoritmo híbrido.

Este trabalho foi idealizado e realizado plenamente pelo discente deste relatório através do zoom, devido às circunstâncias atuais.

Enunciado da atividade proposta

O enunciado providencia um problema de diversidade máxima de grupos. É pretendido dividir um conjunto de elementos em subconjuntos, em que estes últimos terão de ter o mesmo número de elementos. Esta informação é-nos dada através de ficheiros de texto, que contêm o número de elementos e subconjuntos. A qualidade de uma solução é igual à soma da diversidade da cada conjunto, que por sua vez é igual à soma das distâncias entre todos os pares de elementos que o constituem.

Objetivo

O objetivo deste trabalho prático é encontrar a solução com a qualidade de maior valor, usando um algoritmo de pesquisa local, um evolutivo e outro combinando estes dois.

Preparação do Problema

Para resolver este problema de otimização, e antes de serem planeados os algoritmos principais, foi necessário proceder à criação de código para escolha dos métodos, bem como a leitura de ficheiros de texto e funções úteis como randomizar números e copiar matrizes.

Defines a serem considerados

Os defines criados para este trabalho para ajudar a escolher os parâmetros iniciais, bem como a escolha do método de otimização a usar, foram os seguintes:

`#define FILE`: O utilizador insere aqui o nome do ficheiro de texto que contém os dados iniciais do problema.

`#define ALGORITMO`: O utilizador pode inserir 1 para pesquisa local e 2 para algoritmo evolutivo (com possibilidade de ser híbrido, dependendo do define `M_HIBRIDO` explicado mais à frente).

`#define DEFAULT_RUNS`: O utilizador introduz aqui as runs a considerar.

`#define RUNS_TREPA`: O utilizador introduz aqui o número de vizinhanças a pesquisar pelo algoritmo de pesquisa local.

`#define PROB_ACEITAR_PIOR`: O utilizador introduz aqui a probabilidade de o algoritmo de pesquisa local considerar uma solução nova gerada pior à solução anterior a esta.

`#define TAM_POPULACAO`: O utilizador introduz aqui o tamanho da população inicial a considerar pelo algoritmo genético.

`#define NUM_GERACOES`: O utilizador introduz aqui o número de gerações a criar pelo algoritmo genético.

`#define TIPO_MUT`: O utilizador introduz aqui 0 para não considerar qualquer mutação, 1 para mutação binária, e 2 para mutação sôfrega.

`#define M_HIBRIDO`: O utilizador introduz aqui 0 para desativar, 1 para ativar a pesquisa local para definir a população inicial, 2 para ativar a pesquisa local para melhorar os indivíduos no final de cada geração, e 3 para ativar a pesquisa local para melhorar os indivíduos da população final.

Preparação dos ficheiros

Assim que o programa é ligado, este guarda o número de elementos e de subconjuntos do ficheiro de texto selecionado, através da função **leitura_inicial**.

Funções úteis

Das funções presentes no ficheiro utils.c, destacam-se os seguintes: a função **gera_matriz_inicial** que como o nome sugere, gera a matriz aleatória considerando quais os subconjuntos usados e o número de elementos; a função **calcula_solucao** cuja função é calcular a qualidade de uma solução; e a função **retorna_distancia**, que calcula a distância entre dois elementos, do mesmo subconjunto.

Métodos de Otimização

Foram desenvolvidos dois métodos de otimização (algoritmo de pesquisa local e algoritmo genético) e um método híbrido que é a combinação dos dois. São descritos da seguinte maneira:

Algoritmo de Pesquisa Local

Para a pesquisa local criado foi um algoritmo do tipo Trepa Colinas Probabilístico. Este método consiste em pesquisar uma solução vizinha com melhor que a sua atual.

Este algoritmo começa por criar uma matriz igual à matriz inicial introduzida, que irá servir como backup da matriz original, para o caso da solução final gerada ser pior que a solução inicial gerada. São depois feitas randomizações acerca do subconjunto a usar, e também sobre o primeiro elemento escolhido.

Escolhido o elemento e o subconjunto, a matriz entra então num ciclo que irá calcular a melhor distância, em elementos que não pertençam ao subconjunto escolhido, para aumentar assim a otimização, e ao mesmo tempo, procura as piores distâncias, mas apenas do subconjunto a que pertence.

Encontradas as melhores e as piores distâncias, é feita uma troca de elementos. Por exemplo considerando que o subconjunto escolhido foi o 2, a melhor distância encontrada, num elemento que não é do subconjunto 2, vai passar a ser deste, e o subconjunto que estava neste elemento, passa a pertencer ao elemento a que o algoritmo encontrou a pior distância. Nesta forma é-nos permitido evitar o acontecimento de soluções inválidas.

Se então o custo desta matriz for maior que a original é completado um ciclo, e a iteração repete-se. Senão esta matriz volta a conter os valores iniciais. Sendo este Trepa Colinas Probabilístico, está preparado para aceitar soluções piores, consoante a probabilidade inserida no define `PROB_ACEITAR_PIOR`.

Acabadas todas as iterações do algoritmo a função devolve o custo da solução que irá servir para calcular o Mean Best Fitness (MBF) e para guardar o valor da melhor solução gerada.

Por fim, é mostrada na consola a melhor matriz gerada com a sua solução, e o MBF.

Algoritmo Evolutivo

Para este algoritmo foi criada uma `struct individual` com as seguintes variáveis: um ponteiro para inteiros, que irá ser usado como matriz, um inteiro onde irá ser guardado a qualidade da solução e outro inteiro que irá servir de flag, usado no torneio e no crossover, para distinção de quais os pais a usar.

Esta estrutura irá ser alocada com um tamanho estabelecido no define `TAM_POPULACAO` e depois será gerada uma matriz aleatória para cada um, criando assim, uma população inicial de pais. Entramos depois num ciclo onde vão ser avaliadas quantas gerações quisermos, a partir da população inicial.

Estes pais são então colocados num torneio binário, onde será calculada a solução de cada um bem como um reset geral à flag (int valido) de validade para 0. É depois calculada os dois pais com melhores soluções, comparando o rácio da sua qualidade com a qualidade somada da população. Encontrados os dois pais que irão criar descendência, as suas flags de validade passam a conter 1.

Passamos então ao crossover, onde os dois pais, irão dar a cada filho as suas melhores distâncias, para cada subconjunto. Se o filho contiver exatamente a mesma distância, nada alterar. Se contiver parte dela, ou nada de todo, é assim substituído os elementos pelos do pai. Tudo isto com controlos para não existirem soluções inválidas.

É então mudado de subconjunto a considerar no pai, e a iteração volta ao início. Acabando todas as iterações é devolvida uma população de filhos.

Seguidamente estes filhos serão passados por mutações (ou não, se for especificado no define `TIPO_MUT`). Esta função tem dois tipos de mutações. A primeira é uma mutação que altera aleatoriamente a matriz não olhando a distancias piores nem melhores, e a segunda é uma sôfrega, que vai escolher um subconjunto aleatório e retirar a pior distância desse subconjunto, e coloca esse elemento noutra posição que não seja igual à posição onde estava, nem igual à posição de um elemento do mesmo subconjunto. Estas duas mutações criadas estão preparadas para evitar soluções inválidas.

Método Híbrido Combinado

Para o método híbrido combinado foi usado o algoritmo genético em conjunção com o algoritmo de pesquisa local.

Foram feitas modificações para existir três tipos deste método, o primeiro usa pesquisa local para definir a população inicial, o segundo usa pesquisa loca para melhorar os indivíduos no final de cada geração, e o terceiro usa a pesquisa local para melhorar os indivíduos da população final, antes do algoritmo terminar. O utilizador pode escolher cada método a usar através do define `M_HIBRIDO`.

Resultados

Todos os resultados derivados dos testes estão contidos no ficheiro *best solutions.xlsx* em anexo a este relatório. Foram testados todos os algoritmos, com diferentes runs, vizinhanças, tamanho de população inicial, número de gerações, tipos de mutações, e diferentes métodos híbridos.

Os testes ao método híbrido, foram apenas feitos com a mutação 1 (mutação binária) e com a mesma run (4), devido ao elevado número de testes que seriam necessários para obter uma tabela completa.

Devido aos ficheiros com maiores instâncias, como o n120.txt e o n240.txt, foram testados com valores menores ao dos restantes ficheiros, devido ao tempo médio de espera pelos resultados ser consideravelmente longo.

Através da análise do ficheiro com os resultados atrás mencionado, concluímos que:

- O algoritmo de pesquisa local é mais eficiente a achar o melhor resultado quando a probabilidade de aceitar soluções piores é igual a 0.
- Com o algoritmo evolutivo, o que apresenta melhores resultados é quando combinado com a mutação 2 (sôfrega). A mutação 1 (binária) tanto pode apresentar resultados melhores ou piores em relação à mutação 0 (nenhuma mutação), pois é completamente aleatória.
- Com o método híbrido combinado, não é o que apresenta melhores resultados. As melhores soluções e o MBF de cada ficheiro é relativamente baixo em relação aos outros algoritmos, quando isolados. Pensamos que estes resultados se devem ao facto de termos considerado apenas a mutação mais penalizadora (mutação binária) a correr com os testes deste método.

Conclusão

Com este trabalho prático foi possível pôr em prática a matéria lecionada pela cadeira de Introdução à Inteligência Artificial. Implementei os três métodos de otimização pedidos pelo enunciado.

Pelos resultados encontrados, podemos considerar com sucesso que os algoritmos funcionam de forma desejada. Por cada iteração que fazemos, aumentamos sempre o nosso MBF, apesar de por algumas vezes, a melhor solução encontrada não corresponde automaticamente ao maior número de iterações, devido à aleatoriedade presente nos algoritmos.

Graças a este trabalho prático também adquiri mais competências lógicas para resolução de problemas com restrições, e também aumentei as minhas capacidades de programação em C devido à complexidade dos algoritmos realizados.