Introdução à Inteligência Artificial
Licenciatura em Engenharia Informática, Engenharia Informática – Pós-Laboral e Engenharia Informática – Curso Europeu
2º Ano – 1º semestre
Aulas Laboratoriais

Ficha 7: Computação Evolucionária (C.E.)

1. O Problema da Mochila

Existe um conjunto de objetos, cada um deles com um peso e um valor específicos, e uma mochila com uma capacidade limitada. O objetivo consiste em colocar na mochila a combinação de objetos que maximize o valor, mas sem ultrapassar a sua capacidade. Este problema pode ser descrito da seguinte forma:

- Existe um conjunto I com N objetos, cada um deles caracterizado por um peso W(i) e por um valor V(i), i=1, ..., N;
- Uma mochila com capacidade máxima C.

O objetivo da otimização é encontrar um conjunto $S \subseteq I$ tal que:

- A capacidade da mochila não é excedida:

$$\sum_{obj_i \in S} W(obj_i) \leq C \quad (1)$$

- O resultado da seguinte expressão deve ser o mais elevado possível:

$$\text{Maximizar} \sum_{obj_i \in S} V(obj_i) \quad (2)$$

2. Algoritmo de Computação Evolucionária

Vai ser utilizado um algoritmo de computação evolucionária para efetuar a otimização.

2.1. Representação

Numa instância com N objetos, a representação deve indicar quais os que se encontram na mochila. Para isso, recorre-se a um vetor binário com N posições em que cada bit representa um objeto. Uma posição com o valor '1' indica que esse objeto está mochila. Assim, por exemplo, numa instância com 6 objetos, a solução 101100 especifica que os objetos 1, 3 e 4 se encontram na mochila.
A representação proposta admite a existência de soluções inválidas, uma vez que, em alguns casos, a soma dos pesos dos objetos pode exceder a capacidade da mochila. Por causa disso, a função de avaliação terá que lidar com essa situação.

2.2. Geração da população inicial e condição de paragem

A população inicial (o que está fora e dentro da mochila) é gerada de forma aleatória. O algoritmo deverá terminar a sua execução quando atingir o número máximo de gerações.

2.3. Avaliação e objetivo da otimização

A qualidade de uma solução está diretamente relacionada com o lucro proporcionado pelos objetos escolhidos. No entanto, a função de avaliação deve ter em atenção que existem soluções inválidas no espaço de procura (i.e., soluções cujo peso total excede a capacidade da mochila). Estas soluções devem ser penalizadas.
A função de avaliação proposta obedece a um princípio extremamente simples, as soluções inválidas têm qualidade nula, enquanto as válidas possuem uma qualidade igual ao lucro proporcionado. Assim, a qualidade da solução S (representada como uma sequência binária com N posições) é dada pela seguinte expressão:

$$Qualidade(S) = \begin{cases} \sum_{i=1}^{N} S[i] \times V[Obj_i] & \text{se } \sum_{i=1}^{N} S[i] \times W[Obj_i] \leq C \\ 0 & \text{caso contrário (sol. inválida)} \end{cases} \quad (3)$$

em que:

- *S[i]* representa o valor que se encontra na posição i da sequência que codifica a solução S;
- $V[Obj_i]$ é o valor de cada objeto;
- $W[Obj_i]$ é o peso de cada objeto;
- *C* é a capacidade máxima da mochila.

O objetivo é encontrar uma solução que proporcione lucro máximo (problema de maximização).

2.4. Seleção

É usado um mecanismo de seleção por torneio, com tamanho de torneio 2 (torneio binário). Para selecionar um progenitor, escolhem-se aleatoriamente 2 indivíduos da população atual e o que tiver melhor qualidade será o pai da próxima geração.

2.5. Operadores Genéticos

São utilizados 2 operadores clássicos para representações binárias:

- Operador de recombinação com um ponto de corte;
- Operador de mutação binária.

Cada operador tem associada uma probabilidade de aplicação.

3. Detalhes de Implementação

3.1. Ficheiro de dados:

A implementação do algoritmo, contida no *ficheiro IIA_Ficha7_Codigo_Inicio.zip*, implica a existência de um ficheiro de texto com o seguinte conteúdo:

- Valores para os principais parâmetros do algoritmo;
- Informação sobre a instância do problema a otimizar.

A ordem pela qual a informação surge nesse ficheiro deverá ser a seguinte:

- Tamanho da população;
- Probabilidade de mutação;
- Probabilidade de recombinação;
- Tamanho do torneio;
- Número de gerações;
- Número de objetos do problema a otimizar;
- Capacidade da mochila;

- Informação sobre o peso (*Weight*) e sobre o valor (*Profit*) de cada um dos objetos:
  - Linha de título "*Weight Profit*";
  - N linhas (para uma instância com N objetos). Cada linha possui os valores do peso e do valor do objeto i.

Um exemplo para o ficheiro de texto mencionado, referente a uma instância com 10 objetos, pode ser o seguinte:

```
pop:        20
pm:         0.01
pr:         0.7
tsize:      2
max_gen:    100
obj:        10
cap:        25

Weight          Profit
2               6
5               1
10              10
9               14
3               5
6               6
8               9
...
```

No início da execução, o programa acede a um ficheiro deste tipo e obtém a informação relativa ao problema a otimizar. Assim, são disponibilizadas 4 instâncias, que podem ser usadas nos testes ao algoritmo (*knap_100.txt*, *knap_200.txt*, *knap_500.txt*, *knap_1000.txt*).

3.2. Parâmetros do algoritmo

Os parâmetros do algoritmo são armazenados na variável *EA_param*, do tipo *struct info*. Os campos desta estrutura são:

- Tamanho da população, *popsize*;
- Probabilidade de recombinação (para cada par de progenitores), *pr*;
- Probabilidade de mutação (para cada gene), *pm*;
- Tamanho do torneio, *tsize*;
- Constante para avaliação com penalização, *ro*;
- Número de objetos (tamanho da sequência binária), *numGenes*;
- Capacidade da mochila, *capacity*;
- Número de gerações, *numGenerations*.

A variável *EA_param* é declarada na função *main()*. Os valores para preencher os seus campos são obtidos através da leitura do ficheiro de dados na função *init_data()*. A constante *MAX_OBJ* (módulo *algoritmo.h*) especifica o número máximo de objetos que pode existir nas instâncias a otimizar.

3.3. Indivíduos

Cada indivíduo da população é guardado numa estrutura do tipo *struct individual* (ou *chrom*). *pchrom* é um ponteiro para a estrutura desse tipo. A estrutura possui três campos:

- Vetor binário que representa a solução[1], *p*;

[1] Embora este vetor tenha tamanho MAX_OBJ, numa instância com N objetos (N ≤ MAX_OBJ) apenas são utilizadas as primeiras N posições.

- Qualidade do indivíduo, *fitness*;
- Variável binário que indica se o indivíduo é válido (1) ou não (0), *valido*.

3.4. Código

O projeto em C tem 4 módulos:

- *main.c*
  Contém a função que controla o funcionamento geral do programa. Efetua as chamadas para:
  - Preparação de aplicação do algoritmo:
    - Leitura de dados;
    - Geração e avaliação da população inicial.
  - Ciclo principal de funcionamento do algoritmo de CE;
  - Escrita dos resultados no monitor.
- *funcao.c*
  Contém as funções que permitem avaliar a qualidade e determinar a validade dos indivíduos de uma geração;
- *algoritmo.c*
  Contém as funções utilizadas para o processamento de diferentes passos do algoritmo (seleção e operadores genéticos de recombinação e mutação);
- *utils.c*
  Contém algumas funções auxiliares, como as de geração de números aleatórios, criação da população inicial e leitura do ficheiro de dados.

3.5. Notas

- O nome do ficheiro de dados e o número de repetições a efetuar pelo algoritmo de C.E. podem ser indicados através da linha de comando. Se não existirem argumentos, o nome do ficheiro é solicitado ao utilizador e são efetuadas 10 repetições;
- A população atual é armazenada na tabela *pop* (cada elemento da tabela é uma estrutura do tipo *chrom*). O espaço para guardar e tabela em memória é requisitado dinamicamente pela função *init_pop()*;
- Em cada geração, os progenitores escolhidos pela seleção são guardados na tabela *parents*. O espaço é requisitado dinamicamente pela função *main()*;
- Após a seleção, os descendentes são criados e armazenados na tabela *pop*, substituindo, assim, a população anterior;
- A matriz *mat[MAX_OBJ][2]*, declarada na função *main()* e preenchida na função *init_data()*, armazena o peso e valor de cada objeto 1. Embora este vetor tenha tamanho MAX_OBJ, numa instância com N objetos (N ≤ MAX_OBJ) apenas são utilizadas as primeiras N posições.

3.6. Realização de experiências com algoritmos probabilísticos

O algoritmo de C.E. é um método probabilístico, pelo que deve efetuar várias repetições da mesma configuração para poder avaliar a sua eficácia com precisão. No mínimo devem ser efetuadas 10 repetições (30 é o valor desejável). Existem duas medidas de desempenho essenciais:

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

O algoritmo também indica a percentagem de indivíduos inválidos na última geração.

4. Trabalho a realizar no laboratório

As tarefas 4.1 a 4.4 deverão ser feitas com o ficheiro knap_200.txt (e como trabalho extra-aula, deverão ser repetidas as tarefas para os restantes ficheiros).

4.1. Realização de Experiências

Efetuar algumas experiências para avaliar o efeito que os vários parâmetros têm no desempenho do algoritmo, preenchendo as respetivas tabelas do ficheiro fornecido, *IIA_Ficha7_Resultados.xlsx*. As experiências deverão ser as seguintes:

- Influência da probabilidade de recombinação
  Fixar o número de gerações em 2500, o tamanho da população a 100, a probabilidade de mutação a 1% e ir alterando a probabilidade de recombinação para os valores 30%, 50% e 70%;
- Influência da probabilidade de mutação
  Fixar o número de gerações em 2500, o tamanho da população a 100, a probabilidade de recombinação a 70% e ir alterando a probabilidade de mutação para os valores 0%, 0.1%, 1% e 5%;
- Influência da do tamanho da população
  Fixar as probabilidades de mutação e recombinação a 1% e 70% e ir alterando o tamanho da população para os valores 10 (com 25000 gerações), 50 (com 5000 gerações) e 100 (com 2500 gerações).

Após o preenchimento da tabela, analisar e justificar eventuais diferenças de resultados.

4.2. Avaliação

Efetuar as seguintes alterações na fase de avaliação:

- A penalidade associada às soluções inválidas deve variar de acordo com a violação das restrições. Implemente a seguinte proposta:

$$Fitness(S) = \sum_{i=1}^{N} S[i] \times V[Obj_i] - Pen(S)$$

  A penalidade é calculada da seguinte forma (penalização linear):

  i. Se S for uma solução legal:

$$Pen(S) = 0$$

  ii. Se S for uma solução ilegal:

$$Pen(S) = \rho \times \left(\left(\sum_{i=1}^{N} S[i] \times W[i] \right) - C \right)$$

  O parâmetro ρ (*ro*) é obtido da seguinte forma:

$$\rho = \max_{i=1,\ldots,N} \frac{V[i]}{W[i]}$$

  O parâmetro ρ está disponível na estrutura que contém os parâmetros do algoritmo e os detalhes da instância a otimizar.
  Após a implementação, repetir as experiências com algumas das melhores configurações de parâmetros encontradas em 4.1 e continuar a preencher o ficheiro *IIA_Ficha7_Resultados.xlsx*. Analisar eventuais alterações nos resultados obtidos e a variação na percentagem de indivíduos inválidos, ao mudar a função de avaliação;
- Implementar um algoritmo de reparação que atue antes da avaliação e que garanta que todas as soluções avaliadas são válidas. Um possível algoritmo de reparação, ao atuar sobre uma solução ilegal S, vai retirando objetos da mochila enquanto a

restrição da capacidade não for satisfeita, conforme se poderá ver no seguinte pseudo código:

Enquanto $\left(\sum_{i=1}^{N} S[i] \times W[i] > C\right)$ Repetir:
    Escolher aleatoriamente uma posição A de S que tenha o valor 1
    S[A] $\leftarrow$ 0

Após a implementação, repetir algumas das experiências efetuadas anteriormente e analisar eventuais alterações nos resultados obtidos;

- Alterar o método de reparação descrito na alínea anterior, de tal forma que os objetos sejam retirados da mochila de acordo com uma heurística sôfrega (retire primeiro os elementos de menor valor). Após a implementação, repetir algumas experiências efetuadas anteriormente e analisar eventuais diferenças de resultados.

4.3. Operador de Mutação

Implementar o operador de mutação por troca. Este operador escolhe aleatoriamente 2 objetos (um na mochila e outro fora) e troca-os. Repetir algumas das experiências efetuadas anteriormente e analisar eventuais alterações nos resultados obtidos.

4.4. Operador de Recombinação

Implementar o operador de recombinação com 2 pontos de corte e o operador de recombinação uniforme. Repetir algumas das experiências efetuadas anteriormente e analisar eventuais alterações nos resultados obtidos.

4.5. Seleção

No código disponibilizado apenas é possível efetuar torneios com tamanho 2 (i.e., torneios binários). Alterar a função de seleção para que passe a ser possível efetuar torneios com um número superior de elementos. O campo *t_size* da estrutura *struct info*, que é passada como segundo parâmetro, indica o tamanho de torneio[2].
Após a implementação, repetir algumas das experiências da alínea anterior utilizando diferentes tamanhos de torneio e analisar eventuais alterações nos resultados obtidos.

4.6. Algoritmo híbrido

Adicione um método de pesquisa local (por exemplo, um trepa-colinas) ao algoritmo de computação evolucionária.

- Defina o operador de vizinhança a utilizar pela pesquisa local;
- Defina a arquitetura híbrida:
    i. A pesquisa local pode ser usada para criar as soluções da população inicial;
    ii. A pesquisa local pode ser usada para refinar as soluções da última população;
    iii. A pesquisa local pode ser aplicada a algumas soluções que vão sendo criadas pelo algoritmo evolutivo ao longo das gerações.

Implementar algumas das arquiteturas híbridas e analisar o seu desempenho nas instâncias de teste disponibilizadas.

5. Bibliografia para Consulta

E. Costa e A. Simões (2004). Inteligência Artificial: Fundamentos e Aplicações, FCA – Editora de Informática, secção 6.4.2.1.

[2] Este valor é obtido no início da otimização através da leitura do ficheiro.