Computação e Programação

Projecto de Biocomputação 2014/15

Departamento de Matemática, IST, Portugal

Considere o seguinte problema:

> Um caixeiro-viajante tem de visitar *n* cidades, regressando no final à cidade origem. Que percurso deve tomar para minimizar a distância a percorrer?

Este problema, conhecido na literatura como "Travelling Salesman Problem" (TSP), tem aplicações importantes em diversas áreas, como por exemplo:

- planeamento de rotas de distribuição de mercadorias;
- optimização de utilização de recursos em fábricas;
- planemanento de rotas de aviões;
- análise de estrutura de cristais.

Desenvolver bons algoritmos para resolver o TSP (rápidos e que encontrem boas soluções) é uma área ativa de investigação em Computação.

O objetivo deste projecto é, utilizando princípios de Programação Evolutiva, desenvolver em F um programa para encontrar uma solução do TSP, dadas as coordenadas das *n* cidades. Considera-se que a distância entre duas cidades é medida em linha recta.

A ideia é gerar aleatoriamente no instante zero uma população de ν percursos Hamiltonianos e fazê-la evoluir até ao instante final τ. Um percurso Hamiltoniano consiste em percorrer todas as cidades uma única vez, voltando ao início. Cada indivíduo *z* evolui de acordo com o conforto

$$\varphi(z) = \frac{\omega + (c_{\min} / c(z))^2}{1 + 2\omega}$$

(em que $c(z)$ é o custo do seu percurso, isto é, a distância total desse percurso e c_{min} é o menor dos custos de todos os percursos existentes em cada momento) através dos mecanismos aleatórios exponenciais seguintes:

- **Possibilidade de morte**, com tempo médio $(1-\log(1-\varphi(z)))\mu$ entre eventos. A morte ocorre com probabilidade $1-\varphi(z)^2$.

- **Possibilidade de mutação**, com tempo médio $(1-\log(\varphi(z)))\delta$ entre eventos. A mutação ocorre com probabilidade $1-\varphi(z)^2$ e consiste em trocar duas cidades aleatoriamente escolhidas no percurso do indivíduo. Quando ocorre uma mutação, dá-se imediatamente uma segunda mutação com probabilidade p_2 e (caso esta ocorra) uma terceira com probabilidade p_3 (p_2 e p_3 são parâmetros da população). No caso de não ocorrer nenhuma mutação, o tempo até à próxima reprodução é reduzido a 90%.

- **Reprodução**, com tempo médio $(1-\log(\varphi(z)))(N/\nu_{max})\rho$ entre eventos, em que N representa o tamanho atual da população. Da reprodução surge um novo indivíduo que difere do progenitor por uma mutação.

Quando na população surge um indivíduo com percurso *z* (na inicialização ou ulteriormente por nascimento ou mutação) a que está associado um custo $c(z) < c_{min}$, o valor c_{min} da melhor solução encontrada é actualizado para $c(z)$.

A população evolui em função da evolução individual dos seus elementos e ainda por ocorrência de epidemias. Quando o número de indivíduos excede um máximo ν_{max}, ocorre uma epidemia. À epidemia sobrevivem sempre os 5 indivíduos com melhor conforto. Dos restantes, a probabilidade de sobrevivência é $\varphi(z)^2$.

O programa deve:

- Receber interativamente o conjunto seguinte de dados:
  - Número de cidades n;
  - Lista de coordenadas (x_i,y_i) de cada cidade, $i=1,...,n$;
  - Instante final $\tau > 0$ da evolução;
  - Termo $\omega > 0$ de normalização do conforto;
  - Lista $\nu, \nu_{max}, \mu, \delta, \rho, p_2, p_3$ dos parâmteros da população.

- Devolver o melhor percurso z encontrado e o seu custo $c(z)$.
- Mostrar, quando solicitado, o resultado de observações da população, realizadas de $\tau/20$ em $\tau/20$ unidades de tempo. Cada observação deve incluir o instante atual, o número de eventos já realizados, a dimensão da população e o custo do melhor percurso.

Desenvolva o programa seguindo o método de programação por camadas centrado nos dados:

1. Comece por identificar os objetos de trabalho, nomeadamente caminhos, indivíduo e população.
2. Implemente esta camada sobre a camada básica da linguagem F.
3. Desenvolva de seguida o programa abstrato pretendido sobre a camada que disponibiliza estes objectos.
4. Integre o programa obtido em 3 com os módulos desenvolvidos em 2.
5. Experimente o programa desenvolvido em 4 com diversos conjuntos de dados à sua escolha. Em particular, considere os conjuntos:
   - Conjunto A:
     - $n=8$;
     - cidades colocadas nas coordenadas seguintes:
       (2,2), (18,2), (2,18), (18,18), (10,4), (10,16), (8,10) e (12,10)
     - $\tau=0.5$;
     - $\omega=1/1000$;
     - $(\nu, \nu_{max}, \mu, \delta, \rho, p_2, p_3) = (25, 50, 0.5, 0.05, 0.1, 0.3, 0.15)$

   - Conjunto B:
     - $n=15$;
     - cidades colocadas aleatoriamente no rectângulo $[0,20] \times [0,20]$:
     - $\tau=300$;
     - $\omega=1/1000$;
     - $(\nu, \nu_{max}, \mu, \delta, \rho, p_2, p_3) = (25, 50, 300, 1, 5, 0.5, 0.5)$

Apêndice

Relativamente às variáveis aleatórias, recorra à subrotina da linguagem F `random`, para implementar as seguinte subrotinas:

- para simular uma variável aleatória uniforme no intervalo [*a, b*] use a subrotina

```
subroutine vunif(a,b,x)
real, intent(in) :: a,b
real, intent(out) :: x

call random_number(x)
x=a+(b-a)*x
end subroutine vunif
```

- para simular uma variável aleatória exponencial com valor médio *m* use a subrotina

```
subroutine vexp(m,x)
real, intent(in) :: m
real, intent(out) :: x

call random_number(x)
x=-m*log(x)
end subroutine vexp
```

- para gerar aleatoriamente um número inteiro entre *1* e *k*

```
subroutine randinteger(k,r)
integer, intent(in) :: k
integer, intent(out) :: r
real :: x

call random_number(x)
r=int(1+k*x)
end subroutine randinteger
```

Nota: serão em breve disponibilizadas descrições detalhadas dos módulos relativos a caminhos, indivíduo e população a desenvolver.