Programação Orientada por Objectos

Projeto Final

Engenharia Eletrotécnica e de Computadores - Instituto Superior Técnico

Grupo 21

João Ferreira, 70547
José Silva, 84109
Miguel D'Ajuda, 84144

9 de Maio, 2019

Abstract

O objetivo deste documento será fornecer documentação de apoio à solução implementada do problema *traveling salesman problem* (TSP) utilizando um algoritmo baseado numa colónia de formigas. Para além do referido, também serão explicitadas as razões para a forma como certos módulos foram implementados.

Contents

1 Introdução **3**

2 Implementação em alto nível **3**
2.1 pec . . . 3
2.2 events . . . 3
2.3 graphs . . . 4
2.4 ants . . . 4
2.5 simulation . . . 4
2.6 Processamento de erros de utilização . . . 5

3 Testes **6**
3.1 Situação exemplo . . . 6
3.2 Grafo circular . . . 7
3.3 Cenário caótico . . . 8

4 Discussão **8**
4.1 Resultados . . . 8
4.2 Decisões . . . 9

1 Introdução

O objetivo deste projeto é desenhar um programa em Java e UML que aplique um algoritmo que simule uma colónia de formigas, *ant colony optimization* (ACO), ao conhecido problema de otimização TSP. Esta solução é uma alternativa eficiente à utopia de resolução em tempo polinomial, ainda não alcançada. É também tido como foco a extensibilidade do código, para que este possa ser facilmente adaptado a diferentes tipos de simulações pretendidas pelo utilizador.

Tendo como parâmetros do problema dados referentes ás formigas e aos caminhos que estas irão percorrer (caminhos que representam os grafos do TSP), usando o rasto deixado por feromonas e os seus tempos de evaporação, este programa adquire os trajetos ótimos para o TSP. O caminho percorrido pelas formigas é baseado numa simulação estocástica discreta, sobre a qual uma análise mais matemática e profunda se encontra no enunciado deste projeto.

Este trabalho serve também como um desafio para implementar uma solução orientada por objetos, de forma a cumprir o último ponto do parágrafo anterior.

2 Implementação em alto nível

O principio da solução implementada, baseia-se numa simulação estocástica discreta. O funcionamento desta teve como base um *Pending Event Container* (PEC). Para além do PEC, também terão de ser implementados os eventos da simulação sendo estes o movimento da formiga e a evaporação de feromonas. Será também implementado o grafo onde estarão representados os nós que cada formiga terá de visitar, assim como as ligações entre estes. Com isto, a solução poderá ser dividida em 5 *packages*,

- **pec** : Responsável pelo comportamento do PEC;
- **events** : Inclui a implementação dos eventos da simulação;
- **graphs** : Implementa uma estrutura representativa do grafo da simulação;
- **ants** : Implementa o comportamento das formigas na simulação;
- **simulation** : Faz uso das *packages* anteriores para implementar a solução do problema.

2.1 pec

Esta possui uma única *class* denominada de **PEC**, na qual será implementada a funcionalidade total da *package*.

De forma a gerir os eventos da simulação, será necessário armazenar os eventos pendentes por ordem crescente do instante de tempo em que terminam (caso terminem antes do instante final da simulação e caso já não estejam presentes na lista). Será também necessário remover o próximo evento a ser simulado, sendo este o primeiro elemento da lista.

2.2 events

Visando a extensibilidade da solução, foi definida uma interface na qual um evento será definido por dois métodos, um que o simula e outro que indica o instante de tempo em que acaba.

Visto que os dois eventos necessários para o problema em questão são bastante semelhantes, foi considerado um polimorfismo. A super classe implementa as características em comum de

ambos eventos, sendo estas uma variável responsável por guardar o tempo final do evento, um método capaz de o retornar e um outro método estático auxiliar que retorna a duração do evento com base na sua duração média. Existem também um conjunto de variáveis estáticas na superclasse referentes a parâmetros da simulação e um método responsável por atribuir os respetivos valores a estas variáveis.

As subclasses não só implementam os métodos referidos na interface como contêm métodos e atributos específicos à simulação do evento específico. Possuem também uma variável estática responsável por contar o número de simulações de cada evento e um método que o retorna.

Foi também implementado um evento adicional que responsável por imprimir periodicamente o estado da simulação

2.3 graphs

Foi escolhida uma implementação com base numa lista de adjacências em vês de uma matriz de adjacências, de forma a evitar usos excessivos de memória diminuindo no entanto a rapidez do programa. O grafo será então um objeto, no qual estarão contidos um conjunto de nós em que cada um deles possuirá uma lista com as suas ligações. Será importante referir que o presente grafo é duplamente ligado, o que significa que para a implementação através de uma lista de adjacências, uma ligação no grafo estará duplamente representada, algo que será crítico considerar na implementação de algumas das funcionalidades do mesmo.

A *package* será então constituída por três classes, a classe **Graph**, **Node** e **Link**. A função de cada uma delas será implementar o funcionamento do grafo, dos nós e das ligações entre nós respetivamente.

Ao contrário da *package* anterior, a extensibilidade do código não foi um aspeto tão importante a ter em conta, tendo apenas sido implementadas duas exceções de forma a que o utilizador possa salvaguardar a execução do programa caso pretenda implementar os seus próprios métodos.

2.4 ants

Uma vez mais, de forma a tornar a solução o mais extensível possível, foi implementada um interface. Nesta, o comportamento de uma formiga é definido pelo seu movimento sendo necessários dois métodos para o realizar, um que determina o nó para o qual a formiga se deve deslocar e outro que realiza o movimento para o nó desejado. Outros dois métodos são necessários para caracterizar o movimento, um atribui um valor quantitativo ao mesmo e outro que indica o ponto de partida.

Para a presente solução, o movimento da formiga segue as regras 2-4 do enunciado sendo que o ponto de parida do mesmo será sempre o *nestnode* da simulação para todas as formigas da colónia. O peso das ligações percorridas no menor ciclo Hamiltoniano encontrado pela colónia será o parâmetro usado para quantificar o movimento.

Na implementação da interface serão usadas um conjunto de variáveis estáticas, que definem os parâmetros que caracterizam o movimento assim como um método que as define. Para além disso, serão também armazenados o trajeto atual da formiga e o menor ciclo Hamiltoniano encontrado pela colónia com o respetivo peso e o nó de partida, estes últimos 3 são implementados de forma estática.

2.5 simulation

A função desta *package* será juntar as restantes de forma a implementar a solução do problema. O comportamento de uma simulação será apenas ser corrida. Será então fornecida uma

interface que define o comportamento referido sendo que a classe **Simulation** a implementa.

A função **main** da implementação geral estará também nesta *package* na classe **Main** sendo esta responsável por ler e validar o conteúdo do ficheiro de configuração XML através do *SAX parser*, assim como correr a simulação através da interface e implementação desenvolvida. A função espera como argumentos um vetor de 2 strings, sendo a primeira o nome do ficheiro XML a interpretar e a segunda o número de observações que devem ser apresentadas no terminal.

2.6 Processamento de erros de utilização

Como um dos principais pontos tidos em conta na implementação da solução foi a utilização e modificação do código por parte de diferentes utilizadores, tiveram de ser impostos alguns *failsafes* que impeçam o programa de correr caso exista algo fora de norma.

Os erros mais comuns de utilização são por norma no ficheiro de configuração, seja na estrutura do mesmo, seja nos seus valores ou até mesmo a fornecer-lo ao programa. Caso exista algum problema a carregar a informação do ficheiro XML, o programa termina retornando o código adequado.

Exceções

Foram desenvolvidas algumas exceções no que toca à criação e manipulação da estrutura do grafo, nomeadamente a **NoSuchNodeException** que permite, juntamente com as exceções nativas, salvaguardar o programa caso o utilizador tente aceder ao grafo de forma inesperada. De referir também a **ExistingLinkException** que permite evitar que sejam criadas ligações repetidas no XML.

Para além do referido até agora, existe a possibilidade de que o utilizador não consiga garantir a existência de pelo menos um evento no PEC ao implementar os seus próprios eventos. Caso aconteça, o programa termina retornando o código adequado. Em baixo seguem-se os diferentes códigos que o programa pode retornar e o seu significado.

- 0: Programa correu conforme a norma
- -1: Erro na leitura do XML (valor no formato errado)
- -2: Número incorreto de parâmetros de entrada (!=2)
- -3: Erro ao abrir o ficheiro XML
- -4: Erro na estrutura do ficheiro XML errada
- -5: Tentativa de aceder ao PEC vazio ou fora do grafo

3 Testes

3.1 Situação exemplo

Fornecendo ao programa o ficheiro **test_1.xml**, representativo da situação exemplificada no enunciado do problema, obtiveram-se os seguintes resultados.

```
» java -jar ProjetoPOO.jar TESTS/test_1.xml 20
```

```
Observation 1:
    Present instant: 1.0
    Number of move events: 8
    Number of evaporation events: 0

Observation 2:
    Present instant: 2.0
    Number of move events: 12
    Number of evaporation events: 0
    Hamiltonian cycle: 1,5,4,3,2

Observation 3:
    Present instant: 3.0
    Number of move events: 15
    Number of evaporation events: 2
    Hamiltonian cycle: 1,5,4,3,2

Observation 4:
    Present instant: 4.0
    Number of move events: 16
    Number of evaporation events: 2
    Hamiltonian cycle: 1,5,4,3,2

Observation 5:
    Present instant: 5.0
    Number of move events: 21
    Number of evaporation events: 2
    Hamiltonian cycle: 1,5,4,3,2

Observation 6:
    Present instant: 6.0
    Number of move events: 29
    Number of evaporation events: 2
    Hamiltonian cycle: 1,5,4,3,2

Observation 7:
    Present instant: 7.0
    Number of move events: 35
    Number of evaporation events: 3
    Hamiltonian cycle: 1,5,4,3,2

Observation 8:
    Present instant: 8.0
    Number of move events: 39
    Number of evaporation events: 4
    Hamiltonian cycle: 1,5,4,3,2

Observation 9:
    Present instant: 9.0
    Number of move events: 43
    Number of evaporation events: 4
    Hamiltonian cycle: 1,5,4,3,2

Observation 10:
    Present instant: 10.0
    Number of move events: 50
    Number of evaporation events: 5
    Hamiltonian cycle: 1,5,4,3,2

Observation 11:
    Present instant: 11.0
    Number of move events: 52
    Number of evaporation events: 6
    Hamiltonian cycle: 1,5,4,3,2

Observation 12:
    Present instant: 12.0
    Number of move events: 56
    Number of evaporation events: 6
    Hamiltonian cycle: 1,5,4,3,2

Observation 13:
    Present instant: 13.0
    Number of move events: 58
    Number of evaporation events: 7
    Hamiltonian cycle: 1,5,4,3,2

Observation 14:
    Present instant: 14.0
    Number of move events: 61
    Number of evaporation events: 8
    Hamiltonian cycle: 1,5,4,3,2

Observation 15:
    Present instant: 15.0
    Number of move events: 65
    Number of evaporation events: 9
    Hamiltonian cycle: 1,5,4,3,2

Observation 16:
    Present instant: 16.0
    Number of move events: 68
    Number of evaporation events: 10
    Hamiltonian cycle: 1,5,4,3,2

Observation 17:
    Present instant: 17.0
    Number of move events: 71
    Number of evaporation events: 10
    Hamiltonian cycle: 1,5,4,3,2

Observation 18:
    Present instant: 18.0
    Number of move events: 76
    Number of evaporation events: 10
    Hamiltonian cycle: 1,5,4,3,2

Observation 19:
    Present instant: 19.0
    Number of move events: 82
    Number of evaporation events: 12
    Hamiltonian cycle: 1,5,4,3,2

Observation 20:
    Present instant: 20.0
    Number of move events: 86
    Number of evaporation events: 15
    Hamiltonian cycle: 1,5,4,3,2
```

3.2 Grafo circular

Neste caso será testado um caso extremamente simples de um grafo circular constituído por 5 nós contido no ficheiro **test_2.xml**, a ser percorrido por uma formigas.

```
» java -jar ProjetoPOO.jar TESTS/test_2.xmls 10
```

```
Observation 1:
    Present instant: 1.3
    Number of move events: 5
    Number of evaporation events: 0
    Hamiltonian cycle: 1,5,4,3,2

Observation 2:
    Present instant: 2.6
    Number of move events: 12
    Number of evaporation events: 1
    Hamiltonian cycle: 1,5,4,3,2

Observation 3:
    Present instant: 3.8999999
    Number of move events: 17
    Number of evaporation events: 1
    Hamiltonian cycle: 1,5,4,3,2

Observation 4:
    Present instant: 5.2
    Number of move events: 22
    Number of evaporation events: 1
    Hamiltonian cycle: 1,5,4,3,2

Observation 5:
    Present instant: 6.5
    Number of move events: 30
    Number of evaporation events: 2
    Hamiltonian cycle: 1,5,4,3,2
```

```
Observation 1:
    Present instant: 1.3
    Number of move events: 5
    Number of evaporation events: 0
    Hamiltonian cycle: 1,5,4,3,2

Observation 2:
    Present instant: 2.6
    Number of move events: 12
    Number of evaporation events: 1
    Hamiltonian cycle: 1,5,4,3,2

Observation 3:
    Present instant: 3.8999999
    Number of move events: 17
    Number of evaporation events: 1
    Hamiltonian cycle: 1,5,4,3,2

Observation 4:
    Present instant: 5.2
    Number of move events: 22
    Number of evaporation events: 1
    Hamiltonian cycle: 1,5,4,3,2

Observation 5:
    Present instant: 6.5
    Number of move events: 30
    Number of evaporation events: 2
    Hamiltonian cycle: 1,5,4,3,2
```

3.3 Cenário caótico

Neste caso, irá ser testado o comportamento da solução num caso difícil correspondente ao ficheiro **test_3.xml**. O caso em questão corresponde 20 formigas num grafo circular como o anterior, com 10 nós cujas ligações têm o peso 1. Para além disso, cada número par encontra-se ligado aos restantes pares com ligações de peso 3 e cada número ímpar ligado aos restantes ímpares com ligações de peso 2. Os resultados obtidos foram os seguintes.

```
» java -jar ProjetoPOO.jar TESTS/test_3.xmls 20
```

```
Observation 1:
    Present instant: 25.0
    Number of move events: 1580
    Number of evaporation events: 18
    Hamiltonian cycle: 5,6,2,1,10,9,8,7,3,4

Observation 2:
    Present instant: 50.0
    Number of move events: 3254
    Number of evaporation events: 51
    Hamiltonian cycle: 5,6,7,8,9,10,1,2,3,4

Observation 19:
    Present instant: 475.0
    Number of move events: 29990
    Number of evaporation events: 664
    Hamiltonian cycle: 5,6,7,8,9,10,1,2,3,4

Observation 20:
    Present instant: 500.0
    Number of move events: 31556
    Number of evaporation events: 703
    Hamiltonian cycle: 5,6,7,8,9,10,1,2,3,4

(Observações 4 a 17 foram omitidas visto que são idênticas as observações 18 19 e 20)
```

4 Discussão

4.1 Resultados

Situação exemplo

Esta situação serve para demonstrar o correto funcionamento da solução. Com base no *feedback* dado pela simulação, pode ser afirmado que esta correu conforme o esperado tendo o menor ciclo Hamiltoniano sido encontrado em tempo útil e em todas as execuções do programa.

Grafo circular

Esta situação serve para testar o funcionamento específico de certas características desejadas ao programa nomeadamente o *timming* dos eventos. Ora se o tempo médio que uma formiga demora a realizar o seu movimento é de $0.2s$, o valor esperado de movimentos no final da simulação seria $\frac{10}{0.2} = 50$. Ora sendo que após $9.75s$ a formiga efetuou 49 deslocações, pode afirmar-se que a solução encontra-se corretamente implementada.

Quanto aos eventos de evaporação, verifica-se que o primeiro ciclo foi encontrado perto do instante $1.5s$, como tal seria de esperar que no instante $11.5s$, tivessem já sido concluídas 5 evaporações, o que se verifica.

De notar que, ao contrário dos outros dois casos apresentados, este caso apenas apresenta o *feedback* de 17 observações. Isto porque a duração da simulação é demasiado pequena face a duração de cada evento e como tal existem casos em que entre a simulação de dois eventos deveriam ser dadas mais do que uma observação, no entanto estas não são apresentadas porque não houveram eventos a ser simulados.

Cenário caótico

O objetivo deste cenário seria de demonstrar que mesmo em situações complexas, a solução encontra o menor ciclo Hamiltoniano, demonstrando a sua convergência. Ora é evidente que

a solução encontrou de facto o menor ciclo Hamiltoniano no final da simulação, no entanto o primeiro ciclo encontrado não corresponde à solução desejada. é então visível as pequenas alterações que são feitas ao longo do decorrer do programa até se chegar à solução ótima. No entanto, verifica-se que com os parâmetros fornecidos ao programa, ele nem sempre encontra a solução ótima retornando um conjunto de soluções diferentes com cada execução o que permite verificar o carácter estocástico da simulação.

Com base nos resultados obtidos nos três casos apresentados pode-se afirmar que a solução implementa todos os aspetos pretendidos de forma correta.

4.2 Decisões

Ao longo deste documento foram registadas algumas das decisões tomadas na implementação da solução, decisões as quais serão nesta secção discutidas. A primeira decisão referida foi a escolha da representação do grafo do problema, decisão a qual já foi justificada e com base nos tempos de execução do programa, pode ser afirmado que não afetou negativamente o resultado das simulações.

Uma escolha relevante justificar será o porquê de se ter implementado interfaces em grande parte dos módulos da solução e não no Grafo. Foi considerado que a parte fundamental do problema seria implementar um sistema capaz de simular uma colónia de formigas num grafo. Com isto, os aspetos considerados críticos de serem extensíveis foram a simulação, os eventos da mesma e o comportamento das formigas. O grafo foi considerado um elemento fora do alcance do utilizador visto que não pertence a nenhum dos pontos referidos anteriormente. Para além disso, os elementos do grafo são altamente requisitados pelos restantes módulos da solução e como tal seria necessário muito mais tempo para implementar a sua interface que o despendido nas restantes. Devido a estes dois aspetos, foi então decidido não implementar uma interface nos para os elementos do grafo.

Pode ser afirmado que ao bloquear o acesso ao ambiente físico da simulação, o utilizador fica extremamente restringido no que toca ao comportamento das formigas, o que é impactado caso o utilizador deseje que a sua colónia interaja com o grafo. No entanto esta interação pode ser de certa forma feita através das interfaces fornecidas.