TÉCNICO LISBOA

PROGRAMAÇÃO ORIENTADA A OBJETOS

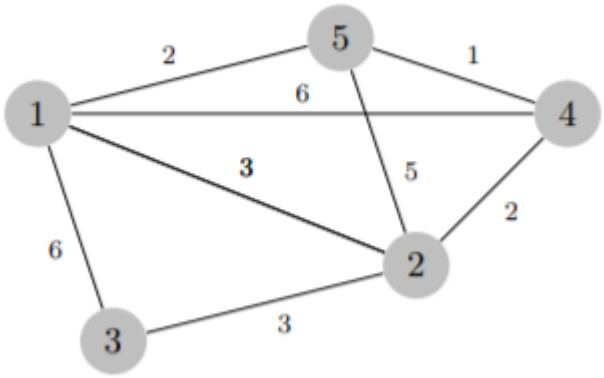
Relatório de Projeto – Problema do Caixeiro Viajante

por Otimização de Colónia de Formigas

Grupo 30
Francisco Silva - 86298
João Caldeira - 78756
Pedro Carvalho - 75340

Parte 1 – Introdução Teórica

O problema proposto para o projecto é o Travelling Salesman problem (TSP) que consiste num grafo ponderado com nós conectados por arestas caracterizadas por uma função que mais tarde será usada como função de custo (neste caso, representando distância entre dois nós) e o objectivo é encontrar o ciclo Hamiltoniano mais curto, passando por todos os nós e retornando depois ao inicial.



Método de Resolução

O método proposto para o projecto é o algoritmo meta-heurístico Ant Colony Optimization (ACO), que como o nome sugere, é inspirado pelo comportamento social e de comunicação de formigas. O algoritmo consiste numa iniciação com as formigas a percorrer caminhos aleatórios, e quando é encontrada uma possivel solução, um rasto de feromona caracteriza aquela solução de acordo com a distância que tem. Se a distancia for curta, a quantidade de feromona será maior e vice-versa. Como o objectivo é encontrar o ciclo mais curto, a probabilidade de a formiga se mover para o próximo nodo será distorcida pela feromona, favorecendo a aresta que tem maior quantidade. Também ocorrerá evaporação da feromona, progressivamente reduzindo o seu valor quando os caminhos não são utilizados ou a quantidade de feromona depositada é menor em iterações posteriores. O algoritmo pode ser descrito nesta forma:

Se existem nodos que ainda não foram visitados no grafo, a formiga escolherá um deles com a probabilidade proporcional à feromona da aresta e inversamente proporcional ao peso (distância) da mesma.

$$P_{ij_k} = \frac{c_{ij_k}}{c_i}$$

Onde:

- c_{ij_k} = α+f_{ij_k} /β+a_{ij_k} ,
- c_i= $\sum_{k=1}^{k=l} c_{ij_k}$
- f_{ij_k} é o nível de feromonas na aresta que liga i a j_k,
- a_{ij_k} é o peso da aresta que liga i a j_k,
- α e β são parâmetros de entrada.

A equação que caracteriza c_{ij_k} apresenta os valores α e β. Estes irão afetar a probabilidade dos nodos assim como a variação de probabilidade que a feromona traz. Quanto mais ou valor de α, menor será a influencia da feromona na probabilidade, visto que o valor desta será diluído por α. Quanto maior for o valor de β, menor será a influência do peso da aresta visto que também será diluído. Terá de existir um balanço entre estes dois valores e a ordem de grandeza do peso e da feromona. Para um funcionamento aperfeiçoado, um incremento dinâmico destes valores ao longo da simulação será ideal, visto que assim reduz a probabilidade de ficar bloqueada num caminho devido ao elevado valor de feromona, que pode não ser o ótimo.

Abordagem

O projecto tem como meta fornecer uma solução ótima usando UML e Java para aplicar o algoritmo ACO. A simulação consiste num grupo de formigas percorrendo o grafo para encontrar um ciclo Hamiltoniano e regressar ao ninho. O ciclo mais curto encontrado desde o início da simulação é guardado e fornecido ao utilizador. Serão realizadas 20 observações com um limite de 800 segundos.

Todas as arestas com valor positivo sofrerão um evento de evaporação, reduzindo a feromona em ρ, com uma distribuição exponencial com média η, entre evaporações.

Alguns parâmetros requeridos ao utilizador:

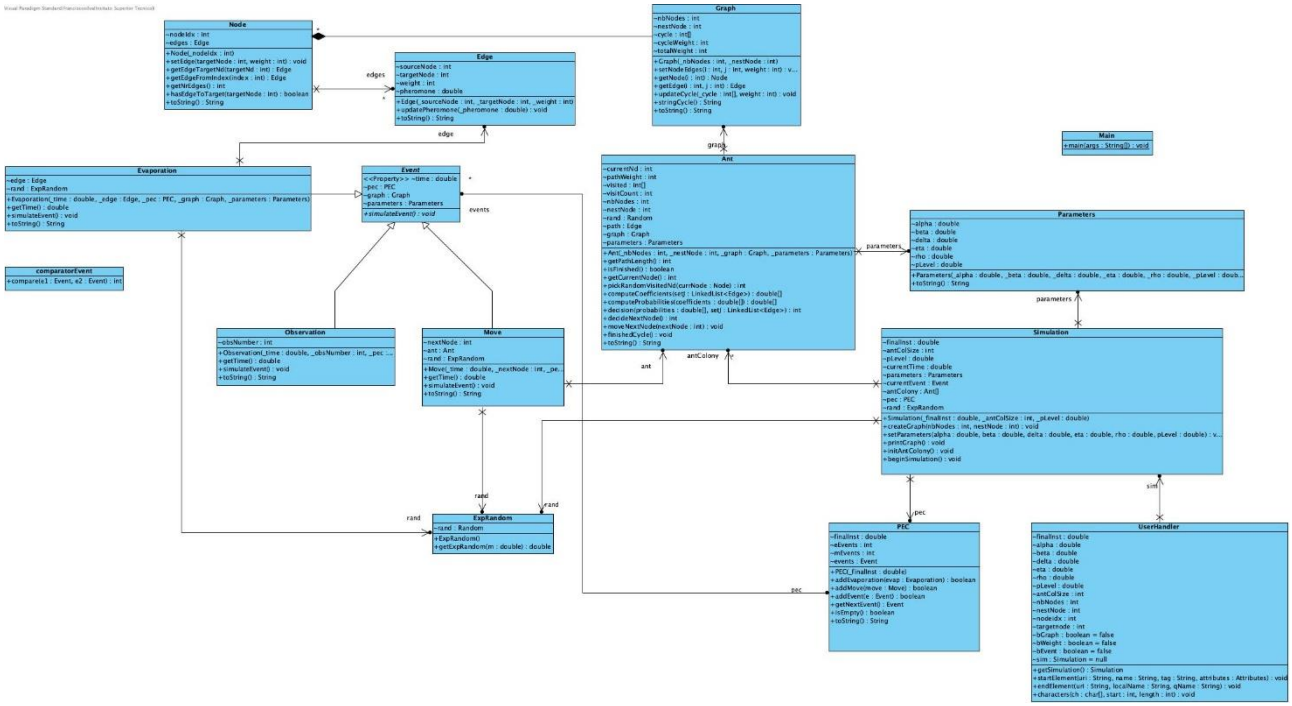
- Número de nodos do grafo;
- O nodo inicial (ninho);
- O peso de atravessar do nodo i para o nodo j;
- Os valores α, β, δ que interferem no movimento das formigas;
- Os valores η, ρ que interferem na evaporação;
- O valor γ, que é responsável pelo depósito de feromona;
- O tamanho da colónia em número, ν;
- O instante final, τ.

O parâmetro δ irá ser multiplicado pelo peso da aresta, fornecendo uma média para a distribuição exponencial para o tempo necessário para percorrer essa aresta.

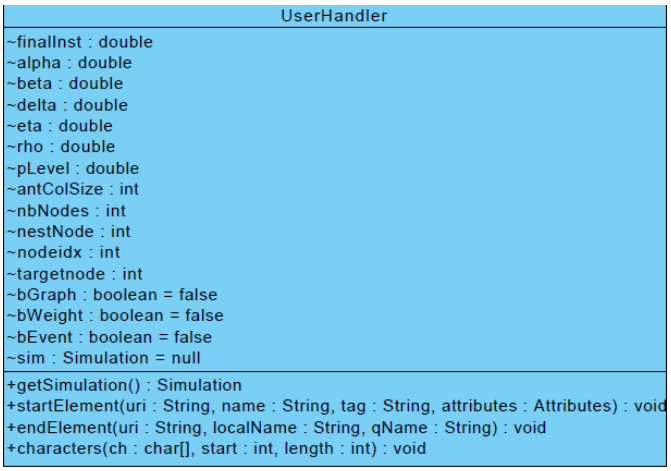
O número de formigas irá afectar o tempo necessário até encontrar uma solução óptima e, até, uma solução. Aumentar o número de formigas irá aumentar a quantidade de processamento necessário e cada iteração irá demorar mais tempo, mas dará informação mais precisa, o número de formigas deve ser proporcional ao tamanho do grafo. Quantos mais nodos, mais formigas serão necessárias para o algoritmo ser eficaz.

Parte 2 – Análise do Código

Nesta secção é analisada a estrutura e organização da resolução do problema, com base nas classes usadas na implementação de cada uma delas, de modo complementar ao fornecido no *Javadoc*. Estas são ainda ilustradas com segmentos do UML projetado pelo grupo.

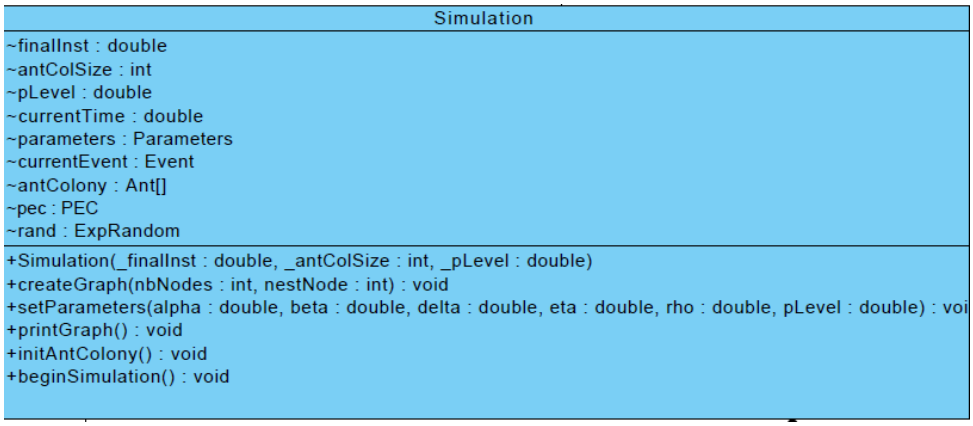


- XML *Parser*



O parsing do ficheiro de entrada, que contém todos os atributos descritos na secção anterior necessários para a simulação é feito com base no *SAX Parser.* O ficheiro é lido, e de seguida o *parser* é instanciado, assim como um *handler* definido na classe *UserHandler*. O ficheiro e o *handler* são então passados ao *parser*, que dará conta de percorrer os elementos do ficheiro XML, extrair os valores dos parâmetros e do próprio grafo e instanciar os objetos necessários. Esta extração é feita com base na *tag* de cada elemento do ficheiro de entrada. No final o *handler* instancia o objeto do simulador (incluindo o grafo pertencente) e retorna o mesmo.

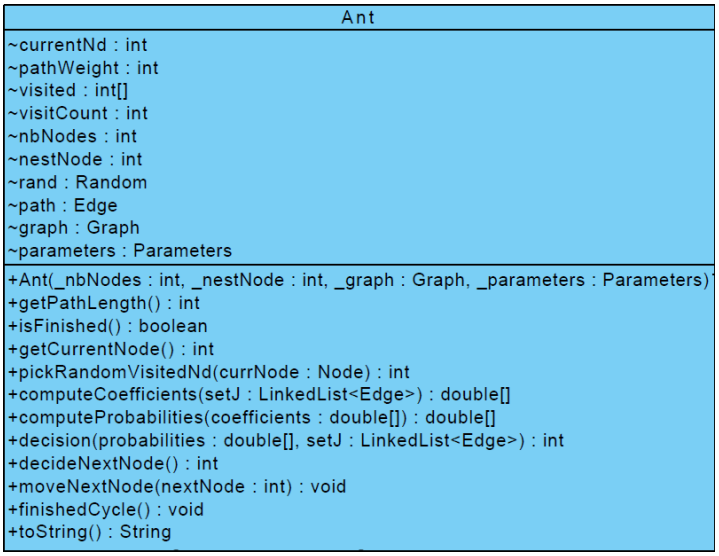
- Simulação



A simulação é definida na classe *Simulation.* Esta classe dá início ao simulador, recorrendo aos parâmetros extraídos da forma descrita no ponto anterior e usando os mesmos para construir a colónia de formigas, definidas como um *Array* de objetos do tipo *Ant,* analisado mais à frente. Para além de controlo da simulação, a principal funcionalidade desta classe é, como referido, o arranque do simulador.

Este é representado pelo método *beginSimulation* que agenda o primeiro evento de movimento para todas as formigas da colónia bem como as 20 observações, implementadas também como eventos.

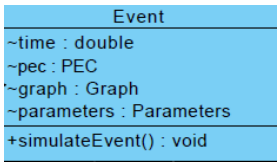
- Formiga



As formigas são definidas na classe *Ant.* Cada uma das formigas possui como atributos informação acerca da sua posição atual, do número de nós do grafo e ainda do ciclo que percorre a um determinado momento. O percurso atual é armazenado como uma *LinkedList* de arestas do grafo e o peso (soma do peso das arestas desse mesmo percurso) é também armazenado. Por outro lado, os nós do grafo são armazenados num vetor de inteiros inicializado a 0, em que os nós visitados são marcados com 1. Desta forma, se o ciclo conter apenas 1's, a formiga terá percorrido a totalidade dos nós.

O processo de decisão de cada movimento está também implementado nesta classe, nomeadamente pelo método *decideNextNode()*. Este método verifica se os nós vizinhos já foram visitados, e decide as probabilidades de escolha com base no resultado. Caso hajam nós vizinhos não visitados é escolhido um deles com base no ponto 3 da descrição do algoritmo, com recurso aos métodos *computeCoefficients, computeProbabilities* e *decision,* que calculam os coeficientes e consequentemente as probabilidades de movimento com base no peso e nível de feromonas das arestas do caminho. Caso todos os nós vizinhos tenham sido visitados é escolhido um uniformemente pelo método *pickRandomVisitedNode.* Por último, esta classe disponibiliza ainda o método *finishedCycle*, que verifica se uma formiga terminou um ciclo e ainda se este tem um custo menor que o ciclo de menor custo encontrado até ao momento, atualizando o mesmo no objecto do Grafo se isso se der e ainda repondo os parâmetros dessa formiga, de modo a que possa iniciar um novo ciclo.

- ## Eventos – Movimento, Evaporação e Observação



Como abordado anteriormente, o algoritmo é baseado na simulação de eventos. Deste modo, foi definida uma classe abstrata *Event.* Esta inclui o tempo de agendamento do evento, os parâmetros de simulação e ainda um método que aplica as alterações programadas no evento, *simulateEvent*.

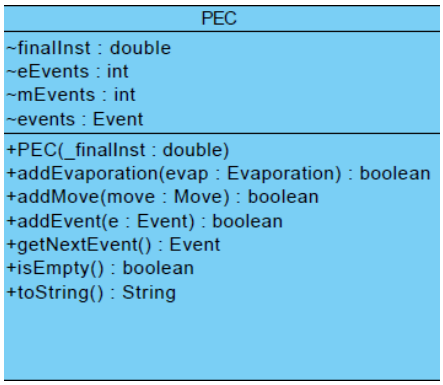
Esta classe possui 3 subclasses que implementam o método de forma distinta: *Move, Evaporation* e *Observation.*

A primeira move uma formiga entre nós e, caso verifique o final de um ciclo, preenche as arestas do caminho que percorreu com feromonas, de acordo com o descrito no ponto 6. Findo este processo, é decidido o próximo nó destino e agendado o próximo movimento.

O evento de Evaporação é também agendado assim que são aplicadas as feromonas sobre a aresta do grafo. O método *simulateEvent*, da classe *Evaporation* verifica apenas se existem feromonas numa determinada aresta, diminuindo o nível das mesmas e agendado a próxima evaporação caso isso se dê.

Por fim, o evento *Observation* efectua a listagem da informação atual do sistema num determinado instante para o terminal, de acordo com os parâmetros estabelecidos no enunciado.

- ## PEC

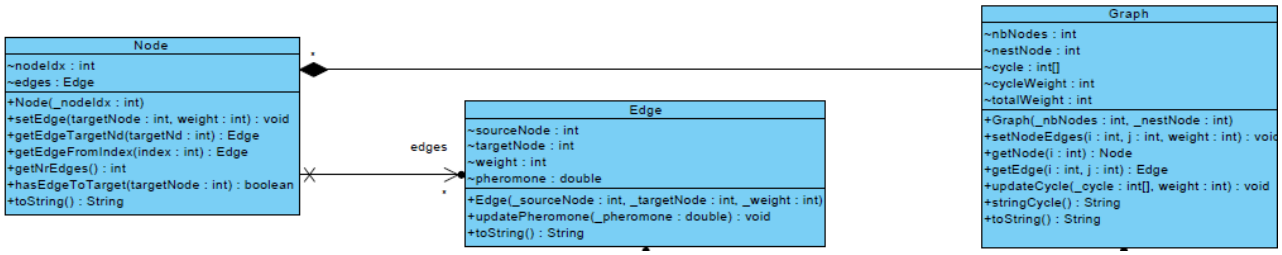


A simulação do algoritmo de optimização foi implementada por meio de um simulador de eventos discreto, o PEC (Pending Event Container). Os eventos vão sendo adicionados ao PEC, que implementa

uma estrutura do tipo PriorityQueue, incluída na *library java.util*. Os eventos são comparados pelo seu atributo *Time,* com base no comparador *comparatorEvent* que implementa um comparador para classes do tipo Event, organizando os mesmos por ordem ascendente de tempo de execução. Os eventos são removidos sucessivamente da cabeça da lista (com base na comparação).

O PEC adiciona e remove eventos até que o instante final seja atingido, contabilizando ainda o número de eventos de evaporação e movimento, para que estes sejam periodicamente mostrados ao utilizador quando se dá um evento de observação.

- ## Grafo – Arestas e Nós



O grafo é definido na classe *Graph* e possui informação acerca da estrutura do grafo extraída dos parâmetros de entrada, bem como do ciclo Hamiltoniano mais curto encontrado dentro do mesmo. Os nós são definidos num *Array* de objetos do tipo *Node,* em que o índex corresponde ao identificador do nó menos uma unidade. Por sua vez, cada nó possui uma coleção de arestas adjacentes, numa *LinkedList.* Cada aresta possui um atributo correspondente ao nível de feromonas.

Esta estrutura de organização do grafo permite um fácil acesso e gestão de percursos, por ser prático percorrer as arestas adjacentes, bem como conhecer os nós origem e nós destino.

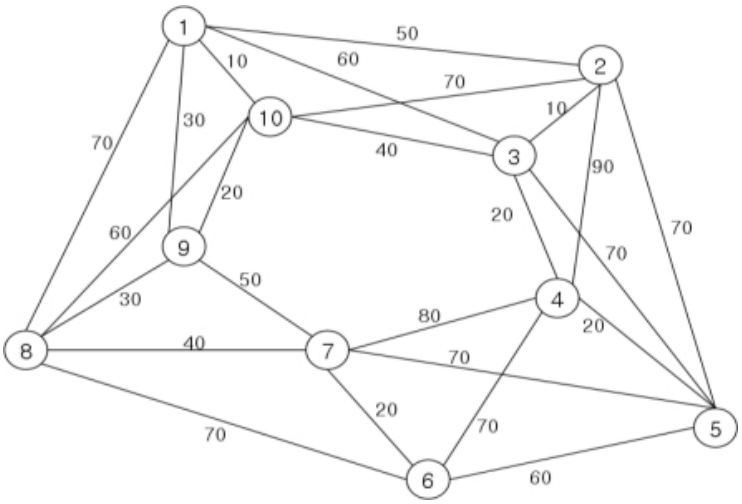
- ## Distribuição Exponencial

Ao longo da simulação, nomeadamente no agendamento de eventos, é necessário gerar números aleatórios distribuídos exponencialmente. Foi para isso implementada uma classe *ExpRandom* que gera números aleatórios utilizando uma transformação da distribuição uniforme de probabilidades definida na classe *Random,* disponível em *java.util*. O construtor desta classe exige que seja fornecida a média desta distribuição.

Parte 3 – Resultados

De forma a ilustrar o funcionamento deste algoritmo, foram criados 5 cenários ilustrativos. Os vários parâmetros de entrada foram alterados, bem como a topologia e dimensão do grafo, de modo a avaliar os resultados e a velocidade de convergência para a solução.

Usando como exemplo o grafo fornecido como exemplo, cujo ficheiro XML foi disponibilizado, verifica-se que o tempo de convergência, ou seja, o tempo de simulação necessário até que seja encontrado o ciclo Hamiltoniano de menor custo, é extremamente baixo. Isto dá-se uma vez que a simulação tem um instante final de 300s e ainda uma colónia de 200 formigas o que para um grafo de apenas 5 nós converge rapidamente para a solução, dado o elevado número movimentos agendado e o baixo número de movimentos exigidos para percorrer um grafo desta dimensão.



Mantendo os mesmos parâmetros mas alterando o grafo para o representado no cenário *test_5.xml,* de 10 nós e custos de arestas superiores, na figura acima (fonte: openi.nlm.nih.gov), os resultados mudam significativamente. Neste caso, nem sempre os 300s foram suficientes para que se encontrasse o caminho de menor custo. Esta lacuna é corrigida se o instante final for alterado para, por exemplo, 800s._Outra solução seria diminuir o parâmetro delta, que afeta diretamente a média de agendamento dos eventos de movimento. Utilizando um delta de 0.1, obtêm-se resultados satisfatórios de convergência deste algoritmo para este grafo.

O parâmetro plevel pode ser também regulado, aumentando o nível de feromonas depositado numa aresta quando um ciclo Hamiltoniano é encontrado, o que sinaliza mais fortemente uma aresta e favorece a probabilidade de ser escolhida aquando do processo de decisão.