Slide 1

O problema do caixeiro viajante requer a determinação a menor rota que passe uma única vez em cada cidade e no final, retorne para a cidade de partida. É a busca por um ciclo hamiltoniano de custo mínimo em um dado grafo completo.

É um problema de grande complexidade porque a quantidade de caminhos possível é igual ao fatorial do número de cidades. Por exemplo, com 7 cidades já temos 5040 caminhos possíveis.

Pertence à classe NP-Completo porque:

* Não existe um algoritmo de tempo polinomial que o resolva.
* É um problema considerado difícil porque até o momento, possui apenas algoritmos exponenciais para resolve-lo.

Slide 2

Pertence aos algoritmos evolucionários. Eles usam técnicas de busca baseadas em metáforas de processos biológicos, como a evolução. Combinam a sobrevivência entre os melhores como uma forma estruturada de troca de informações genéticas entre 2 indivíduos. Essa troca de informação se dá pela seleção, elitismo, cruzamento e mutação.

O algoritmo genético trabalha com transições probabilísticas.

* Primeiro, cria uma população de indivíduos com cromossomos aleatórios, em um dado conjunto, nesse caso, se tenho 7 cidades, cada cromossomo terá uma ordem aleatória de 1 a 7, sem repetir qualquer número.
* Para a seleção, escolhe uma quantidade ‘n’ de indivíduos, sendo que ‘n’ é decidido pelo usuário. Mas os ‘n’ indivíduos são selecionados de forma aleatória em meio a população, e depois, é verificado quais deles tem melhor aptidão e o melhore é escolhido para testar para o cruzamento.
* No cruzamento e mutação: depois de escolher os indivíduos para essas etapas, eles ainda tem que vencer uma disputa aleatória, aonde é gerado um número aleatório e ele precisa ser igual ou menor que a taxa de cruzamento e mutação escolhido pelo usuário. Não há uma regra pre determinada para saber se haverá ou não cruzamento e mutação. Se houver qualquer um desses processos, escolhe aleatoriamente quais partes de cromossomo serão trocadas entre os indivíduos.

O elitismo segue uma regra, que é a escolha dos indivíduos mais aptos, nesse caso, de menor distancia de viagem entre as cidades. O programa ordena a população do melhor para o pior e seleciona os ‘n’ melhores indivíduos para coloca los na nova população, eles substituem os piores indivíduos da nova população, tornando ela mais eficaz em comparação se não usarmos elitismo.

O resultado é avaliado de acordo com uma regra de parada, nesse caso uso quantidade de evoluções. Cada vez que rodo o programa, escolho uma quantidade de vezes que serão feitos os processos de seleção, elitismo, cruzamento e mutação, daí o programa varre a população e escolhe o indivíduo de melhor aptidão (melhor fitness ou menor caminho).

Ele codifica um conjunto de soluções possíveis dentro de um parâmetro, nesse caso, encontrar o menor caminho possível. Não interessa a questão de otimização, depois de encontrar um conjunto de respostas, o AG não verifica se possui a melhor resposta.

O resultado desse algoritmo é um conjunto de soluções possíveis, chamado de população. A população é um conjunto de indivíduos, cada um representa uma solução, e nesse indivíduo, temos o seu cromossomo, que é um vetor binário ou alfabético ou numérico, contendo as informações da solução. No caso do caixeiro viajante, o cromossomo contem a sequência de cidades que devem ser visitadas. Ainda é possível usar essa sequência para calcular as distancias entre cada cidade e a distância total.

O passo a passo do AG é:

* Cria uma população de indivíduos de tamanho decidido pelo usuário. Cada indivíduo terá um cromossomo com genes aleatórios e não repetidos. Apesar de não repetir genes, pode haver repetição de indivíduos com cromossomos identificos, por exemplo, se testo o AG para 4 cidades, a quantidade de rotas possíveis é 24, mas decido criar uma população com 50 indivíduos, são 50 possibilidades de solução para um problema cujo conjunto de possibilidades é 24, certamente haverá indivíduos idênticos na população.
* O programa verifica se há elitismo, também decidido pelo usuário. A população é ordenada do melhor para o pior e copia para a nova população os ‘n’ melhores indivíduos.
* Depois usa um FOR que repete por até a metade do tamanho da população, pois em cada iteração “pega” 2 pais para gerar 2 filhos.
* Cada iteração chama o método de seleção 2 vezes para pegar 2 pais e testar para saber se eles vão fazer o cruzamento, como foi explicado antes, é um processo aleatório. Se der certo, os 2 filhos gerados são testados para mutação e independente se der certo ou não, eles substituem esses pais na população (ocupam a posição no vetor que era de seus pais, sobrescrevendo eles). Se der errado, se não cair na taxa de cruzamento, esses pais são mantidos na população.
* Depois desse FOR, a nova população será dos filhos gerados e dos pais que não cruzaram. Se optar por elitismo, os selecionados por esse método substituirão os menos aptos da nova população para terminar de compor a nova população.
* Há também a possibilidade de testar mutação para toda a nova população, isso é decidido pelo usuário.
* Tudo isso é uma evolução, cada execução do programa faz uma quantidade ‘n’ de evoluções determinada pelo usuário. Depois de fazer a última evolução, o programa escolhe o indivíduo de melhor aptidão como resposta.

Slide 3

A aptidão ou fitness é o somatório das distancias entre cada cidade, quanto menor, melhor.

Para esse programa eu calculo também a aptidão média da população e ploto isso num gráfico, apenas para mostrar que a aptidão geral da população tende a melhorar a medida que o algoritmo genético roda.

A distância de um ponto ao outro (de cidade a cidade) é calculada usando essa formula.

Já falei dos processos de seleção e elitismo, vou explicar o cruzamento. Eu usei o cruzamento por mapeamento parcial (PMX). Depois de selecionar os 2 melhores indivíduos da população atual, gero 2 filhos. Cada um é uma cópia parcial do seu pai. A outra parte dos cromossomos de um filho será doado pelo outro pai.

O filho 1 é uma cópia do indivíduo 1, mas parte do seu cromossomo vem do indivíduo 2, e vice versa. Essa parte é escolhida de forma aleatória, não há regras determinísticas no AG.

Mas perceba que esse cruzamento pode gerar genes iguais no cromossomo. O filho 1 tem 2 genes “3” e “4”, e o filho 2 tem 2 “1” e “5”.

Para resolver esse problema uso uma tabela auxiliar que relaciona as posições dos genes dos pais. O gene 2 do pai 1 está na mesma posição do gene 2 do pai 2.

Então, eu substituo o gene repetido por seu equivalente nessa tabela, mas observe que eu altero apenas os genes repetidos que não vieram do cruzamento.

Já a mutação é mais simples, 2 pontos distintos do vetor de cromossomo são escolhidos aleatoriamente, não podem ser iguais e o primeiro não pode ser maior que segundo, e os genes dessas posições são trocados de lugar, veja que mais uma vez o processo evolutivo é aleatório.

Explicação do código

1 – Cada ponto desenhado na tela cria uma cidade num plano 2D, com coordenadas X e Y, que servem para calcular a distância entre 2 pontos. É preciso criar pelo menos 3 cidades para liberar a opção “criar população” porque é inútil querer achar o melhor caminho com apenas 2 pontos.

2 – Depois cria a população e abre opção para executar o AG (linha 214 em form1.cs). Os valores da interface são gravados nas variáveis. E o método “executeGA” (linha 282) é chamado.

3 – (linha 39 em GeneticAlgorithm.cs)

Passa a população do parâmetro para uma lista temporária (trabalhar com a lista é mais fácil do que com os objetos de Indivíduo dentro de um objeto População);

Verifica se vai fazer elitismo;

Inicia o FOR para testar cruzamento;