resolução do PROBLEMA DO ENTREGADOR VIAJANTE

MARCO AURÉLIO PACHECO, RICARDO FUKASAWA

Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225, Gávea, Rio de janeiro, RJ, CEP. 22453-900, BRASIL
E-mail: marco@ele.puc-rio.br / ricfuka@attglobal.net

Resumo: Este artigo descreve os experimentos feitos tentando aplicar algoritmos genéticos para resolver uma modificação do problema do caixeiro viajante, chamada problema do entregador viajante. Foi feito um programa modificado para ajustar-se às condições que resolve o problema do caixeiro viajante achando soluções (sub) ótimas. Este programa emprega técnicas gerais de algoritmos genéticos e técnicas específicas. Assim como no caixeiro viajante, o problema se propõe a encontrar uma rota que passe por todas as cidades, e que tenha custo mínimo. A diferença é que este problema enfoca menos o lado do entregador e mais o lado do "cliente".

Palavras chave: algoritmos genéticos, caixeiro viajante.

Abstract:

This article describes the experiments conducted to attempt to apply genetic algorithms to solve a modification of the traveling salesman problem, called the problem of delivery traveler. A modified program was made to adjust to the conditions to solve the traveling salesman problem by finding optimal *solutions (sub). This program uses general techniques of genetic algorithms and techniques. As the traveling salesman* problem proposes to find a route passing through every town, and at minimum cost. The difference is that this issue focuses less on the side of delivery and more on the side of the "client".

Keywords: genetic algorithms, traveling salesman.

1. Introdução:

O problema do caixeiro viajante é um problema muito famoso e muito estudado. Um caixeiro viajante, partindo de sua cidade, deve visitar exatamente uma única vez cada cidade de uma dada lista e retornar para casa tal que a distância total percorrida seja a menor possível. Este problema tem inúmeras aplicações práticas, como minimização de rotas de veículos, confecção de sistemas digitais, sequenciamento de atividades e outros.

O problema do entregador viajante é uma modificação do problema do caixeiro viajante, em que o objetivo é minimizar o tempo médio de atendimento a n cidades ou clientes. Como pode-se notar por este enunciado, este problema se aplica nas situações em que, não se quer minimizar é o custo de quem visita, de quem oferece o serviço – situação do caixeiro viajante. O que se quer minimizar é o incômodo dos clientes, por exemplo, o tempo média de espera de cada cliente a ser atendido. Exemplos: Entrega, atendimento, vistoria...

A motivação para a utilização de algoritmos genéticos neste problema é por ser um problema aparentemente simples, mas não existir algoritmo eficiente para

resolvê-lo. O melhor algoritmo documentado, descrito em [1], é um algoritmo que utiliza técnicas de Branch-and-Bound, aliadas a um algoritmo $O(n^3)$ para achar limites inferiores para o problema. Tipicamente, algoritmos deste tipo são muito ineficientes, apesar de serem os melhores encontrados até agora. O algoritmo conhecido resolve problemas da ordem de 30 vértices, possuindo heurística que fornece boas soluções para mais vértices. O desenvolvimento de um algoritmo genético que pudesse encontrar a solução ótima, ou pelo menos perto da ótima, ou melhor do que a heurística existente no algoritmo tradicional poderia ser de grande valia, seja substituindo o algoritmo tradicional, ou auxiliando-o (por exemplo fornecendo um limite superior válido bom).

O propósito deste artigo é descrever os resultados obtidos, bem como os métodos desenvolvidos para a resolução deste problema.

A seção seguinte descreve o problema de forma mais detalhada. A seção 3, mostra os operadores utilizados para resolver o problema, e toda a implementação feita. Na seção 4 são apresentados os melhores resultados obtidos, e finalmente na seção 5, as conclusões tiradas desta experiência.

2. O Problema

Um entregador, saindo de sua cidade (depósito), deve passar em outras n cidades para fazer entregas. O objetivo deste entregador é minimizar o tempo médio que cada uma das cidades espera para receber sua encomenda.

Este problema pode ser visto como uma modificação do problema do caixeiro viajante, já que eles têm uma natureza muito semelhante (é um problema de ordenar n cidades em uma rota).

Para facilitar a compreensão do problema e de sua aplicação, o custo de cada arco ij pode ser entendido como o tempo que se demora para ir da cidade i para a cidade j.[1]

Para o cálculo do tempo neste problema específico, foi considerado que o tempo de ir de uma cidade a outra é proporcional à distância entre as cidades. Como apenas uma constante multiplicando todos os custos não altera a solução ótima ou nenhuma das outras soluções, simplesmente considera-se o tempo igual à distância.

Este cálculo do tempo foi usado apenas por simplicidade, mas vale ressaltar que qualquer valor poderia ser utilizado.

Abaixo, há um exemplo de como calcular o tempo médio para uma rota com 5 cidades mais a cidade origem.

Suponha que a rota seja:

$0 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 5 \rightarrow 3 \rightarrow 0$

Seja C_{ij} o tempo total para ir da cidade i para a cidade j. Então, a cidade 4 espera C_{04} para ser visitada. Já a cidade 2 espera $C_{04}+C_{42}$. A cidade 1, $C_{04}+C_{42}+C_{21}$, e assim por diante.

[1] *Note que aqui, como em todo o resto do artigo, custo se refere a qualquer grandeza a ser minimizada (tempo, distância, etc.), e não apenas dinheiro.*

Com isto, pode-se concluir que o tempo médio de espera é:

	Espera da cidade 0	- 0
	Espera da cidade 4	- C_{04}
	Espera da cidade 2	- $C_{04}+C_{42}$
	Espera da cidade 1	- $C_{04}+C_{42}+C_{21}$
	Espera da cidade 5	- $C_{04}+C_{42}+C_{21}+C_{15}$
+	Espera da cidade 3	- $C_{04}+C_{42}+C_{21}+C_{15}+C_{53}$

Tempo total de espera = $5.C_{04} + 4.C_{42} + 3.C_{21} + 2.C_{15} + C_{53}$
Tempo médio = Tempo total / 5.

De maneira geral, sendo a rota:
$0 \rightarrow i_1 \rightarrow i_2 \rightarrow ... \rightarrow i_{n-1} \rightarrow i_n$

O tempo médio de espera seria:

$$\frac{n.C_{0,i_1} + (n-1).C_{i_1,i_2} + (n-2).C_{i_2,i_3} + ... + C_{i_{n-1},i_n}}{n}$$

Esta é, portanto a função objetivo. Por simplicidade, pode-se eliminar o denominador, pois é uma constante, e então apenas o numerador deve ser minimizado.

Pode-se ver que cada custo tem seu peso na função objetivo. Consequentemente, ao contrário do problema do caixeiro viajante, deve ser definido claramente um ponto de partida neste problema, pois os custos dependem da posição. Como o problema é um modelo para um entregador, que tem que visitar n clientes, pode-se considerar que ele sempre sai de uma cidade fixa (o depósito) e retorna a ela no final, sendo que o custo de retornar ao depósito não é considerado.

3. Descrição da Solução

Para a resolução deste problema, foram utilizados algoritmos genéticos, que são modelos probabilísticos de busca e otimização inspirados nos princípios da evolução natural e da genética. Pressupõe-se que o leitor já está familiarizado com os conceitos e o funcionamento de um algoritmo genético. Para maiores detalhes, recomenda-se o estudo de [3].

Foram usados operadores que explorem a posição das cidades e suas adjacências, dois aspectos importantes deste problema.

Representação e decodificação

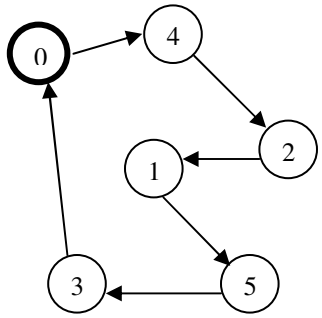
O cromossoma que representa a rota do entregador é o mais intuitivo possível. Ele é composto de apenas um gene, que possui uma ordenação entre as cidades

a serem visitadas. O único cuidado que deve ser tomado é que a cidade origem é fixa, e portanto ela não entra no cromossomo. Ela é guardada à parte em um outro lugar do programa, para que não haja nunca alteração nela.

Portanto, o cromossoma:

4	2	1	5	3

Representa a trajetória:



Operadores Genéticos

A representação escolhida é extremamente simples. O seu problema é que operadores tradicionais de crossover e mutação geram soluções inválidas. Para resolver este problema, foram utilizados operadores criados para o TSP, que alteram a ordem dos indivíduos no cromossoma.

Abaixo, há uma breve descrição dos operadores utilizados.

Crossover de Ordem – PMX (Partially Mapped Crossover)

O Crossover PMX faz um crossover parcialmente mapeado. Ele usa dois pontos de corte e faz o crossover normalmente entre os dois pontos. Neste processo, se trocou um vértice do pai 1 por um vértice do pai 2 para cada posição da rota entre os pontos. Cada uma destas trocas define um mapeamento, ou seja, se o vértice i estava na posição k no pai 1, e o vértice j estava na posição k no pai 2, e a posição k está entre os dois pontos de corte, i está mapeado em j e vice-versa.

Após feito o crossover normal, provavelmente as rotas geradas serão inválidas. Para corrigir este erro, deve-se substituir todos os vértices repetidos que estão fora dos pontos de corte pelos vértices aos quais eles estão mapeados.

Exemplo:

Pais:

9 8 4 | 5 6 7 | 1 3 2 10
8 7 1 | 2 3 10 | 9 5 4 6

O operador troca a cadeia de números entre os dois pontos de corte, e repete as cidades dos pais que não são repetidas:

9 8 4 | 2 3 10 | 1 x x x
8 x 1 | 5 6 7 | 9 x 4 x

A troca entre os pontos de crossover define alguns mapeamentos:

$2 \leftrightarrow 5,\ 3 \leftrightarrow 6,\ 10 \leftrightarrow 7$

As cidades repetidas dos cromossomas originais são substituídas pelas cidades mapeadas:

9 8 4 | 2 3 10 | 1 6 5 7
8 10 1 | 5 6 7 | 9 2 4 3

Crossover de Adjacências

Desenvolvido para o problema do caixeiro viajante, pois foi observado que a grande importância neste problema era a adjacência entre duas cidades.
O algoritmo para o cruzamento é:

1. Elabora-se uma lista de adjacências de cada um dos vértices dos pais.
2. Escolhe-se os vértices que possuem menos elementos na lista de adjacências, para formar um conjunto de candidatos a próximo vértice
3. Entre os elementos deste conjunto, escolhe-se aleatoriamente, com mesma probabilidade, um vértice para ser o primeiro vértice do cromossoma filho. Note que independente da posição original do vértice escolhido, ele será colocado na primeira posição do cromossoma filho, e as posições seguintes serão preenchidas sequencialmente conforme os próximos passos.
4. Retira-se o vértice escolhido da lista de adjacências.
5. Dentre os vértices adjacentes ao último vértice escolhido, forma um conjunto de candidatos a próximo vértice, com os vértices que possuem menos elementos na lista de adjacências.
6. Entre os elementos deste conjunto, escolhe-se aleatoriamente, com mesma probabilidade, um vértice para ser o próximo vértice do cromossoma filho.
7. Repete os passos de 4 a 6 até o cromossoma filho estar completo, ou até o conjunto de candidatos a próximo vértice estar vazio. No primeiro caso, o crossover termina. No segundo, vai para o passo 8.
8. Escolhe aleatoriamente um vértice entre todos os vértices ainda não escolhidos para o cromossoma filho, Todos os vértices restantes têm igual probabilidade. Volte para o passo 4

Crossover de Adjacências modificado

Foi criado devido ao baixo desempenho observado nos outros operadores. É uma modificação do crossover de adjacências, específico para este problema.

Conforme pode-se observar, o crossover de adjacências não se importa com a posição dos vértices na rota, apenas com suas adjacências. No caso específico deste problema, a posição também é um fator importante.

Para agregar este conhecimento, foi desenvolvido o crossover de adjacências modificado, que, na lista de adjacências guarda não somente as adjacências, mas também a posição em que a aresta se encontrava originalmente nos pais.

Então, após ser escolher o primeiro vértice do cromossoma filho, ao invés de ser colocado na primeira posição, ele é colocado na posição em que estava em um dos cromossomas pais e as posições vão sendo preenchidas sequencialmente a partir desta.

Note-se também que no passo 6, escolhia-se aleatoriamente o próximo vértice, sendo que cada um dos candidatos tinha iguais chances de ser escolhido. Neste crossover modificado, os candidatos que estão mais próximos da próxima posição a ser ocupada tem maiores chances de serem escolhidos. Mais especificamente, a chance de ser escolhido é proporcional ao quadrado da proximidade entre a posição original do vértice e da posição em que será colocado o próximo vértice.

Mutação por rotação (direita/esquerda)

É escolhido um número aleatório n de posições e o cromossoma é girado (todos as posições são deslocadas de n) para a direita/esquerda.

Mutação por troca

Esta mutação simplesmente troca duas cidades escolhidas aleatoriamente de lugar.

Também foram feitos experimentos com um outro tipo de crossover, o de ciclo. Porém, como os resultados obtidos com ele foram ruins, não é dada importância a ele. O leitor que estiver interessado neste e em outros operadores para problemas de ordem, ou em maiores detalhes sobre os outros operadores utilizados deve procurar [3] e [4]

Função de avaliação

A função de avaliação, conforme já foi descrito na seção anterior, é:

$$n.C_{0,i_1} + (n-1).C_{i_1,i_2} + (n-2).C_{i_2,i_3} + ... + C_{i_{n-1},i_n}$$

4. Resultados

Foram estudados os casos para os quais temos a solução ótima (até 30 vértices). Para os casos de até 20 vértices, o algoritmo obteve a solução ótima sem muito esforço da parte de ajustes de parâmetros. Por esta razão, não nos são citados aqui em maiores detalhes estes experimentos. Exploramos sim o problema

de 30 vértices, para o qual uma grande quantidade de esforço teve de ser feito para chegar a melhores resultados.

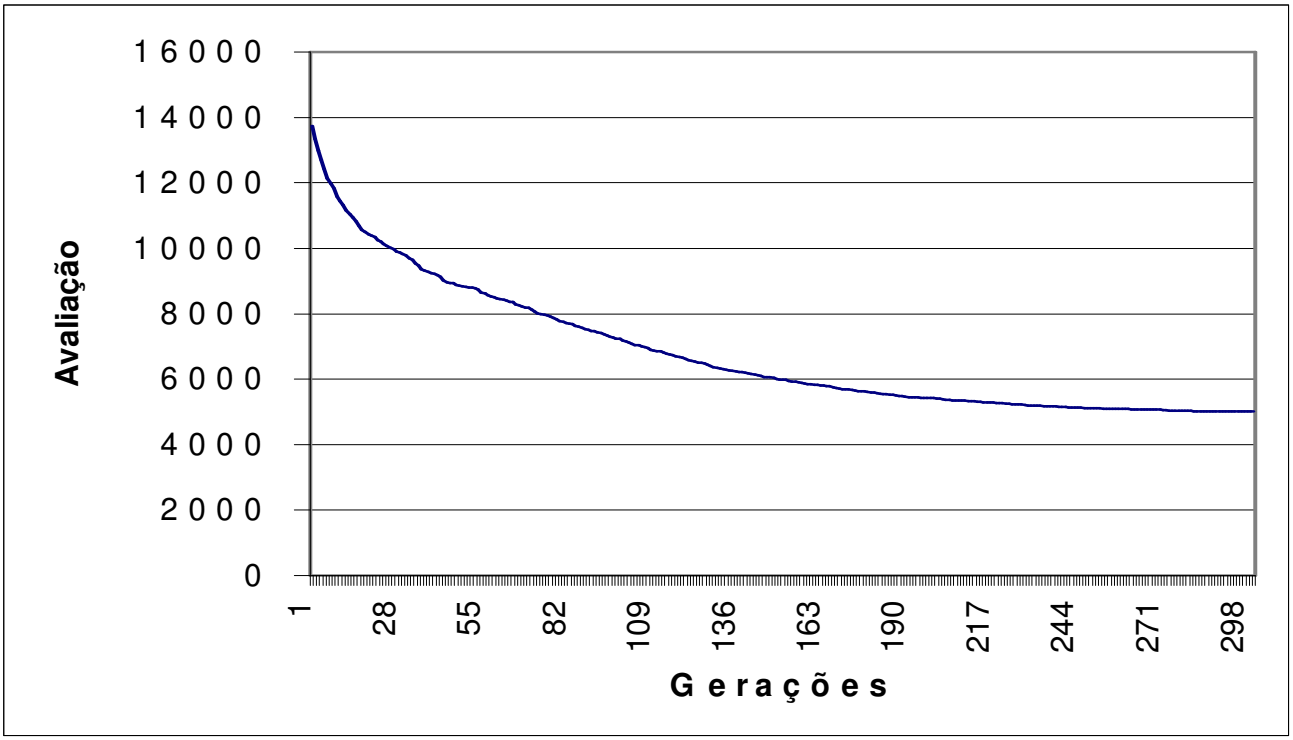
Foram feitos diversos experimentos, utilizando vários recursos diferentes. Foi observado que o melhor tamanho de população era 200 e o número de gerações 300. Números maiores que estes não produziam resultados melhores. Foi observado também que após 300 gerações, o algoritmo não evoluía mais, nem mesmo utilizando-se semeamento, e portanto, este recurso não foi utilizado.

Curiosamente, os experimentos em que foi usado steady state obtiveram piores resultados do que aqueles que não utilizaram (só utilizaram elitismo). Este resultado apesar de curioso, também já havia sido observado em [2] e inclusive a partir daí é que surgiu a idéia de não usar steady state.

A normalização linear forneceu resultados melhores, principalmente quando foram alterados os limites de máximo e mínimo para que o “range” fosse duas vezes maior que o tamanho da população.

Outro recurso que produziu resultados melhores foi o da interpolação de parâmetros (mais crossover e menos mutação no começo da evolução, e mais mutação e menos crossover no final). Também foi utilizada a combinação de vários tipos diferentes de crossover e mutação, com probabilidades diferentes de serem escolhidos (seleção de operadores).

Abaixo, mostramos o gráfico da média de 30 rodadas para a melhor combinação de parâmetros encontrada.



OBS: Apesar de a análise do gráfico mostrar que possivelmente o algoritmo ainda poderia evoluir com mais de 300 gerações, na prática foram feitos experimentos com mais de 300 gerações e não foi obtido melhor resultado.

A combinação de parâmetros usada para esta solução foi:

- Porcentagem de Crossover:
  Variação de 0.8 a 0.6
- Porcentagem de Mutação:
  Variação de 0.1 a 0.3
- Seleção de Operadores:
  0.8 de Crossover PMX e 0.2 de Adjacências Modificado
  0.6 de mutação por troca, 0.2 de rotação para a direita e 0.2 de rotação para a esquerda

O melhor indivíduo encontrado foi:
16-20-30-22-10- 4-19- 9-27-12-29-21-11- 3- 5- 6-18- 7-28-15-17-14-23- 2- 1- 8-26-25-13-24
com avaliação 4626

Comparando com o indivíduo ótimo:
16- 7-18- 6- 5-20-30-22-10- 4- 3-11-21-29-12-27-19- 9- 8-26- 1- 2-23-14-15-28-24-17-13-25
com avaliação 4545

Diferença entre o ótimo e o melhor indivíduo:
1,78%

Analisando ambos os cromossomos (em cima o obtido pelo algoritmo genético e em baixo o ótimo), à primeira vista, parece não haver muita semelhança. Porém, fazendo uma análise mais minuciosa, pode-se observar a existência de padrões (marcados em cores) que se repetem nos dois cromossomas, e que portanto devem ser explorados. Isto é uma indicação de um aspecto que pode ser explorado no futuro para melhorar o desempenho do GA.

16-20-30-22-10- 4-19- 9- 27-12-29-21-11- 3- 5- 6-18- 7-28-15-17-14-23- 2- 1- 8-26-25-13-24
16- 7-18- 6- 5-20-30-22-10- 4- 3-11-21-29-12-27-19- 9- 8-26- 1- 2-23-14-15-28-24-17-13-25

Semelhanças:

- cidade 16 na primeira posição
- o padrão em vermelho se repete nos dois cromossomas
- os padrões em roxo e azul estão invertidos nos cromossomas
- o padrão em verde está invertido no segundo cromossoma, apenas trocando a cidade 26 de posição com a 8

5. Conclusões

A utilização de algoritmos genéticos para resolver o problema do entregador viajante foi motivada por existir um algoritmo que consegue resolver o problema ótimo para apenas poucos vértices (da ordem de 30). Para problemas maiores, possui-se apenas heurísticas para obter soluções sub-ótimas. Com o algoritmo genético esperava-se obter resultados melhores do que as heurísticas

(possivelmente ótimos), que tivessem um custo computacional inferior ao algoritmo que obtém o ótimo.

O algoritmo, programado em Visual C++ 6.0 processa 20 rodadas em aproximadamente 30 minutos num Pentium II 500 Mhz. Este tempo computacional é considerado alto quando comparado ao tempo do algoritmo tradicional (aproximadamente 10 minutos na mesma máquina). Porém, para instâncias maiores, o crescimento do tempo computacional do algoritmo genético é baixo, enquanto o do tradicional é muito alto. Com isto, espera-se que o tempo computacional de um algoritmo genético se torne muito menor para instâncias maiores.

O grande problema encontrado foi que o algoritmo genético não encontrou resultados melhores nem comparados às heurísticas do algoritmo tradicional, que é muito rápida.

Podemos concluir que o problema não é tão simples quanto parece, e possui complicações que não eram observadas no problema do Caixeiro Viajante. Os resultados não foram satisfatórios, e isto pode ser devido a várias diferentes razões que devem ser estudadas com mais profundidade para se obter melhores resultados.

Apesar dos resultados não muito satisfatórios, pode-se observar a existência de padrões entre o cromossoma ótimo e o melhor cromossoma achado pelo GA. Isto pode indicar uma direção na qual se pode melhorar o algoritmo genético, implementando operadores que levem isto em conta.

Outra idéia possível é o desenvolvimento de algoritmos híbridos.

Enfim, o problema ainda está em aberto, e ainda há muito campo para ser explorado, tanto em algoritmos genéticos, quanto em algoritmos tradicionais, ou híbridos.

6. Bibliografia.

[1] A. Lucena, *Time-Dependent Traveling Salesman Problem – The deliveryman case*, Networks, Vol. 20, 753-763. John Wiley & Sons Inc - 1990

[2] A. Rodrigues, M.A. Pacheco, *Um estudo sobre a aplicação de algoritmos genéticos ao problema do caixeiro viajante,* PUC-Rio 1997

[3] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learnin*g,
Addison-Wesley 1989.

[4] Z. Michalewicz, *Genetic Algorithms+Data Structures=Evolution Program*s,
Springer-Verlag-1994