Comparação de configurações de operadores genéticos de Algoritmos Genéticos para resolver o Problema do Caixeiro Viajante

Victor Hugo Piontkievitz da Cruz[1]

[1]Universidade Tuiuti do Paraná
Curitiba – PR

{victor.cruz}@utp.edu.br

Resumo. *O trabalho atual se concentra em identificar as diferenças entre os diversos operadores genéticos dos Algoritmos Genéticos, sendo as configurações de crossover, mutação, da inicialização da população e do critério de parada, em relação ao tempo de execução e a qualidade da solução, para o Problema do Caixeiro Viajante. Este problema é frequentemente utilizado em verificações de algoritmos, e consiste em achar o caminho de menor custo possível em um conjunto de cidades, passando por cada uma apenas uma vez a não ser o ponto de origem, qual deve ser o ponto final também.*

1. Introdução

O Problema do Caixeiro Viajante (PCV), ou como é conhecido em língua inglesa, Travelling Salesmen Problem (TSP) é um problema que, dado um roteiro de viagem, deve ter cada destino visitado exatamente uma vez, retornando ao ponto de origem no final. O objetivo é encontrar, dentre os muitos caminhos ou rotas disponíveis, aquele que tem o menor custo. O problema é conhecido por ter classe NP-difícil, mas tem sido empregado esforços para melhorar os seus recursos, pois além de planejar rotas de viagens, também são usados para os movimentos de máquinas automáticas e industriais, como as de perfuração de placas de circuitos [RUSSEL 2010].

2. Algoritmos Genéticos

Os Algoritmos Genéticos são algoritmos propostos pelo pesquisador John H. Holland da universidade de Michigan nos anos 60, baseados no princípio Darwiniano sobre a evolução das espécies e na genética, pretendendo simular o mecanismo da evolução biológica [Goldberg 1989]. Os Algoritmos Genéticos começam com um conjunto de estados, ou indivíduos, gerados que é chamado de população. Cada indivíduo representa um cromossomo, e são representados como uma cadeia sobre um alfabeto finito, que muitas vezes são representadas de forma binária [RUSSEL 2010].

Cada estado é avaliado pela função de adaptação, que possui valores mais altos para estados melhores, e deles, pares são escolhidos para a reprodução, de acordo com algum critério estabelecido, e seguindo algum outro critério, as características dos pais são herdados pelos 2 filhos, gerando assim a próxima geração de estados. Dessa forma, uma evolução é simulada, em que geralmente os melhores indivíduos são escolhidos para reproduzirem com outros melhores indivíduos, focando assim em gerar um individuo cada vez melhor [RUSSEL 2010].

3. Metodologia

O Algoritmo Genético, com as configurações dos operadores de crossover, mutação, inicialização da população e critério de parada foi implementado para resolver o Problema do Caixeiro Viajante, utilizando a linguagem de programação Python. As bibliotecas random, math e time foram utilizadas para permitirem o uso de técnicas e cálculos matemáticos para aperfeiçoar e configurar o algoritmo. O algoritmo possui os operadores genéticos crossover, mutação, inicialização de população e o critério de parada, que podem ser alterados em suas configurações.

Para a representação, cada cidade, junto com as suas coordenadas são mantidas em uma lista, e essa lista então é mantida em outra lista, que representa o conjunto de cidades, sendo um cromossomo, de forma ordenada, que é a forma em que o caminho é feito entre as cidades. A população por sua vez é formada pelo conjunto de cromossomos. Inicialmente, as populações podem ser geradas de forma aleatória ou baseadas em heurística.

Primeiro, o código possui uma função para ler os arquivos Comma-separated values (CSV) que contém as instâncias. Cada instância representa um conjunto de estados, cada um sendo uma cidade e possuindo uma coordenada x e uma coordenada y própria, sendo elas números de ponto flutuante. Em total, são 10 instâncias lidas, que são passadas para o Algoritmo Genético, cada uma por vez, possuindo elas um tamanho variado do conjunto, mas que é escalar.

Uma função chamada *distanciaCid* é usada para receber uma população de cidades e devolver a distância total percorrida pelo caminho tomado. Essa distância é calculada por uma função que calcula a distância euclidiana, da forma:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

entre cada uma das cidades pelas suas coordenadas x e y, e então é somado ao total da distância até o momento. Uma próxima função chamada *escPopulacao* é utilizada para fazer a escolha da população inicial de forma aleatória, recebendo a população e utilizando as técnicas de *shuffle* da biblioteca *random*. Ela também retorna o melhor indivíduo da geração com o *sorted* na lista de cidades.

Para a inicialização da população baseada em heurística, duas funções existem, chamadas *Gulosa* e *escPopulacaoGulosa*. A primeira é relacionada ao algoritmo guloso, que inicia uma população de acordo com sua heurística, e gera algumas sequências, calculando a cidade que tem o menor caminho até a atual, prosseguindo para ela, e repetindo o loop. Porém, apenas uma porcentagem do cromossomo é gerada dessa maneira, que é especificado na segunda função, qual também é responsável por completar as cidades que faltam de forma aleatória.

No *main* da função, os vários parâmetros e as configurações dos operadores são definidos, e os loops para cada instância são chamados. O tamanho da população inicial pode ser definido, tendo o valor base de 2000. A mutação pode ter a sua porcentagem definida, dentro de um valor de 0 a 1.0. O crossover pode receber valores de 1 a 3, que significam respectivamente um ponto, dois pontos e uniforme, e, caso algum outro valor seja passado, o crossover não é utilizado. A variável *estagnacao* representa o critério de parada, no qual se seu valor for 0, é utilizado apenas um número fixo de gerações,

que pode ser alterado na função do Algoritmo Genético, e caso seja 1, também utiliza a estagnação, caso ocorra um determinado número de gerações, que também podem ter seu valor definido. Como base, é utilizado o número fixo de gerações 200, e o número de gerações estagnadas 30.

A inicialização da população pode ser alterada entre os valores 0 e 1, qual representam respectivamente a inicialização aleatória e a baseada na heurística gulosa. Em seguinte, temos o loop que itera pelo número de instâncias, chamando cada função em ordem, lendo o CSV, inicializando a população, e então chamando a função do Algoritmo Genético. No final, são mostradas as informações e a solução alcançada, com o número de gerações levados, o melhor cromossomo inicial, e o melhor cromossomo final.

A função *genetico* é utilizada para representar o Algoritmo Genético, recebendo os operadores genéticos e a população como parâmetros. A geração inicial é a 0, tendo o limite fixo de 200. Um loop *for* é usado na duração do algoritmo até chegar ao limite ou ao ocorrer a estagnação por um número de gerações, caso esteja sendo utilizada. Utilizando o elitismo, os dois melhores cromossomos são inseridos na nova população, que representa a nova geração.

A seleção dos outros indivíduos é feita através de torneio onde pais são selecionados, e o crossover é realizado para cada filho, utilizando a configuração passada, podendo ser a de um ponto, dois pontos ou uniforme. Após, a mutação pode ocorrer caso caia na probabilidade utilizada, e então os novos cromossomos são adicionados na nova população, já com a sua distância total calculada.

Ao fim, a geração é incrementada, e caso esteja sendo utilizado o critério de parada por estagnação, é checado se o melhor cromossomo da população obteve uma solução melhor que o da anterior. Caso não tenha melhorado, significa que houve a estagnação, e uma variavel é incrementada, e caso ela seja igual ao número limite, o algoritmo é finalizado. Caso tenha melhorado, apenas continua. Se este critério não for utilizado, o algoritmo continua até a geração limite. A população final então é retornada no final, de forma ordenada.

4. Resultados

As tabelas mostram a comparação entre as diferentes configurações dos operadores genéticos. Os elementos de comparação representam o melhor cromossomo alcançado na população da geração final e o tempo de execução levado pelo algoritmo em segundos. No entanto, é preciso notar que, como as populações são distintas, cada vez que o algoritmo for utilizado, um resultado diferente pode ser obtido. Cada configuração é comparada com as outras configurações do mesmo operador. Então, quando o operador específico não é o que está sendo medido, os valores padrões do crossover é o de dois pontos, o nível de mutação usado é baixo (1%), a população utiliza inicialização aleatória e o critério de parada é por 30 gerações estagnadas. Somente quando um operador estiver sendo comparado em suas configurações que o valor padrão dele não vai ser utilizado.

A tabela 1 mostra os resultados das configurações do crossover, de um ponto, dois pontos e uniforme. A tabela 2 mostra os resultados das configurações de mutação, sendo o nível baixo com uma probabilidade de 1%, o nível médio de 5% e o nível alto de 20%. A tabela 3 mostra os resultados das configurações de população inicial, gerada

completamente aleatória e com 10% gerado pela heurística gulosa e o resto aleatório. A tabela 4 mostra os resultados das configurações de critério de parada, sendo o por apenas o número fixo de 200 gerações e de estagnação após 30 gerações.

Tabela 1. Operador Crossover.

Instâncias	Um ponto		Dois pontos		Uniforme	
Número	Solução	Segundos	Solução	Segundos	Solução	Segundos
1	173.854	0.3823	173.854	0.4106	173.854	0.3723
2	264.843	0.4165	264.843	0.4736	264.843	0.4452
3	251.382	0.5702	251.382	0.5826	251.382	0.5823
4	341.600	0.6858	341.600	0.7676	341.600	0.8924
5	330.567	0.8870	330.567	1.0449	330.567	1.1150
6	315.906	1.6081	315.906	1.3014	315.906	1.5205
7	382.356	1.6504	370.185	1.9140	373.668	2.2547
8	327.746	2.1220	327.746	2.1902	349.729	3.1109
9	433.201	3.7067	420.820	3.3228	471.892	4.9798
10	576.74	7.6455	502.088	4.0467	493.584	7.6797

Tabela 2. Operador Mutação.

Instâncias	Taxa baixa (1%)		Taxa média (5%)		Taxa alta (20%)	
Número	Solução	Segundos	Solução	Segundos	Solução	Segundos
1	173.854	0.4151	173.854	0.4238	173.854	0.5069
2	264.844	0.4794	264.844	0.4865	264.844	0.5022
3	251.382	0.6276	251.382	0.6039	251.382	0.6384
4	341.600	0.7893	341.600	0.7486	341.600	0.7943
5	330.567	0.8445	330.567	0.9982	330.567	1.0742
6	315.907	1.3507	315.907	1.4409	315.907	1.4027
7	370.185	1.7475	370.185	2.0458	370.185	1.9295
8	327.746	2.0297	327.746	2.1632	327.746	2.5742
9	402.600	2.9673	420.820	2.5340	402.600	3.8465
10	530.780	4.1868	512.846	3.6942	490.509	4.2347

5. Discussão

Observando os resultados obtidos, podemos perceber as diferenças entre as configurações dos operadores genéticos. Em geral, os resultados obtidos entre as diferentes configurações do mesmo operador são semelhantes ou parecidos, com alguma distinção em relação ao tempo consumido, mas não muito alta.

O operador de crossover de dois pontos achou três soluções melhores que o de um ponto e 3 três soluções melhores e uma pior do que a do uniforme, e o de um ponto achou duas soluções melhores e duas piores que a do uniforme. Em relação ao tempo, o de um ponto achou 7 soluções mais rápidas que a de dois pontos, e 8 mais rápidas que o uniforme, e o de dois pontos achou 7 soluções mais rápidas que o uniforme. Dessa forma, os de crossover de um ponto se demonstra mais rápido, e o de dois se demonstra de maior qualidade, porém por pouco.

Tabela 3. Operador Inicialização da População.

Instâncias	Aleatória		Heurística Gulosa	
Número	Solução	Segundos	Solução	Segundos
1	173.854	0.4100	173.854	0.4852
2	264.844	0.5198	264.844	0.5516
3	251.382	0.6260	251.382	0.6903
4	341.600	0.6913	341.600	0.7105
5	330.567	0.9576	330.567	0.8894
6	315.907	1.4761	315.907	1.2998
7	370.185	1.6882	370.185	1.7655
8	327.746	2.0420	349.729	1.9120
9	402.600	2.6047	406.487	2.1934
10	493.585	3.7320	514.428	1.8055

Tabela 4. Operador Critério de Parada.

Instâncias	Número Fixo de 200 Gerações		Estagnação por 30 Gerações	
Número	Solução	Segundos	Solução	Segundos
1	173.854	2.6238	173.854	0.4146
2	264.844	2.8405	264.844	0.4753
3	251.382	3.5270	251.382	0.7118
4	341.600	4.2301	341.600	0.7044
5	330.567	4.7230	330.567	1.0949
6	315.907	5.7105	315.907	1.3750
7	370.185	6.7890	370.185	1.8755
8	327.746	7.4109	327.746	1.9522
9	402.600	9.2286	402.600	3.0792
10	493.585	10.7773	490.509	3.8783

O operador de mutação de taxa baixa achou uma solução melhor e uma pior que o de taxa média, e uma solução pior que o de taxa alta, e o de taxa média achou duas soluções piores que o de taxa alta. Em relação ao tempo, o de taxa baixa foi mais rápido em 6 das soluções do que o de taxa média, e foi mais rápido em todas as instâncias do que o de taxa alta, e o de taxa médio em 8 das soluções em comparação com de taxa alta. A qualidade da solução encontrada entre o de taxa baixa e média são semelhantes, enquanto o de taxa alta apresentou soluções melhores, mas em muita pouca quantidade. Em relação ao tempo, quanto menor a taxa de mutação, menor ele se apresenta.

Em relação ao operador de inicialização da população, a aleatória encontrou 3 soluções diferentes do que a que utiliza heurística, sendo elas melhores. No entanto, a que utiliza heurística chegou no resultado com um tempo mais rápido em 6 das soluções. Embora as 4 primeiras instâncias tenha sido mais lentas para a heurística, como o número de cidades nas instâncias de número mais alto são maiores, estando em ordem crescente, ela se demonstra capaz de resolve-las com um tempo muito menor.

O operador de critério de parada tem apenas uma solução diferente entre suas configurações, com o que utiliza o critério de estagnação possuindo uma melhor qua-

lidade, mas por pouco. Embora a solução não tenha sido muito diferente, a remoção do critério de parada por estagnação se demonstrou ter o o maior impacto do que todas as outras configurações no que se diz ao tempo levado. O critério de número fixo de gerações resultou no tempo sendo mais lento em todas as soluções, com a maior diferença nas primeiras instâncias, e uma menor diferença nas últimas.

Isso se deve ao fato de que, quanto mais instâncias, mais podem ser as alterações, não resultando na estagnação nas gerações mais novas. Conforme o número cresce, mais tarde será a geração em que será parada, e mais esse número se aproxima de 200, que é o limite fixo. Então, nas menores instâncias a estagnação se demonstrou bem mais efetiva, embora não tenha se demonstrado com tanta piora nas últimas, mas isso apenas significa que as instâncias usadas não foram grandes o suficiente para causar a diferença.

6. Conclusão

Observando os resultados obtidos pela comparação das diferentes configurações dos operadores genéticos do Algoritmo Genético, podemos ver as diferenças entre cada um deles. Em geral, as soluções encontradas foram muito semelhantes, com poucas diferenças, mas elas se apresentaram, e o tempo levado certamente se diferenciou em algumas delas. Em relação ao operador de crossover, o de um ponto se demonstrou mais veloz, enquanto o de dois pontos uma melhor solução, com o uniforme se apresentando o pior entre eles. O operador de mutação apresentou semelhança nas soluções encontradas nas instâncias que utilizaram taxas baixas e médias, enquanto o de taxa alta apresentou o melhor resultado entre eles. No entanto, quanto maior a taxa, maior foi o tempo levado para achar a solução.

No operador de inicialização, a população aleatória encontrou soluções melhores que a iniciada com heurística, mas a segunda se demonstrou mais rápida para achar os resultados, se demonstrando ser melhor quanto maior o número de cidades presentes nas instâncias. O operador do critério de parada apenas teve uma solução diferente encontrada, mas em relação ao tempo levado, o que utilizou o critério de estagnação se demonstrou mais rápido para chegar a uma resposta em todas as instâncias. No entanto, conforme o número de instâncias cresce, a diferença de tempo entre as duas configurações diminui, devido ao número de gerações necessárias para ter uma estabilidade aumentar com o número de cidades.

Desse modo, as diferenças entre a qualidade das soluções e o tempo de execução podem ser vistos. Em alguns dos operadores, porém, existem muitas semelhanças. Como a geração da população é diferente para cada vez que o algoritmo é rodado, os resultados obtidos podem acabar sendo diferentes, principalmente em relação ao tempo levado. Para melhores conclusões, mais e maiores instâncias, com problemas mais complexos podem ajudar definir as particularidades de cada configuração dos operadores, podendo chegar a diferentes conclusões sobre seus comportamentos.

Referências

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learnig*. Addison-Wesley.

RUSSEL, Stuart J.; NORVIG, P. (2010). *Inteligência Artificial: Uma Abordagem Moderna.*, volume 3. São Paulo: Prentice Hall.