Resolvendo Caixeiro Viajante com Algoritmo Genético

Alexandre Mendonça Fava[1]**; Gustavo Diel**[1]

[1]Universidade do Estado de Santa Catarina - UDESC
Programa de Pós-graduação em Computação Aplicada - PPGCA
Joinville - SC - Brasil CEP: 89.219-710

{alexandre.fava, gustavodiel}@hotmail.com

Abstract. *Genetic Algorithms are inspired by the Darwinian principle of species evolution and genetics. They are probabilistic algorithms that provide a parallel and adaptive search engine based on the survival principle of the fittest and reproduction. This article introduces the genetic algorithm technique applied to the Traveling Salesman Problem.*

Resumo. *Algoritmos Genéticos são inspirados no princípio Darwiniano da evolução das espécies e na genética. São algoritmos probabilísticos que fornecem um mecanismo de busca paralela e adaptativa baseado no princípio de sobrevivência dos mais aptos e na reprodução. O presente trabalho apresenta a aplicação da técnica dos Algoritmos Genéticos no problema do Caixeiro Viajante.*

1. Introdução

Há anos o ser humano já está familiarizado com a Teoria da Evolução e as ideias do naturalista Birtânico Charles Darwin. Considerando que os indivíduos mais adaptados são os que sobrevivem, pensou-se que alguns problemas poderiam ser tratados de forma similar, onde as melhores soluções para um determinado problema seriam as mais "adptadas", cujo as quais, no final acabariam "sobrevivendo".

O problema da Teoria da Evolução é que ela carece de um formalismo matemático mais detalhado, todavia isso não impede que os Algoritmos Genéticos se demonstrem bem efetivos na solução de determinados problemas.

Os algortimos evolutivos acabaram por ser muito usados na computação para a resolução de problemas que demandariam um grande consumo de tempo ou memória, mesmo que na maioria dos casos, a solução gerada não seja a ótima.

2. Caixeiro Viajante

O Problema do Caixeiro Viajante (em inglês, TSP) é um problema da classe NP-Difícil que modela diversas aplicações práticas (MORAIS, 2011). O problema do caixeiro-viajante consiste em, dado uma lista de cidades e suas posições, encontrar o menor caminho que, partindo de uma cidade inicial, passe por todas as outras cidades apenas uma vez e que termine o ponto de partida (NILSSON, 1982).

Esse problema pertence a classe de problemas de otimização combinatória por buscar um caminho otimizado em um conjunto finito de cidades. Esse problema possui diversas aplicações, tais como planejamento e manufatura de microchips. Também é um sub-problema na área de sequenciamento de DNA.

2.1. Força Bruta

Uma solução que não envolve heurísticas para resolver o problema do caixeiro-viajante é utilizando o método de força bruta. Seu algoritmos consiste em escolher uma cidade inicial, e a partir dela, tentar todos os caminhos possíveis. O algoritmo utilizado nos testes está apresentado no Algoritmo 1. É possível notar que o algoritmo é $O(n!)$, sendo **n** o número de cidades. Esse crescimento fatorial é um dos piores na computação, tanto que com apenas 12 cidades, temos um aproximado de $479.001.600$ iterações.

```
1  def find_paths(current_city, cities, path, distance):
2    path.append(current_city)
3    if len(path) > 1:
4      distance += Distance(path[-2], current_city)
5    if (len(cities) == len(path)):
6      path.append(path[0])
7      distance += Distance(last_city, path[0])
8      routes.append([distance, path])
9      return
10
11   for city in cities:
12     if (city not in path):
13       find_paths(city, cities, list(path), distance)
```

Listing 1. Algoritmo de Força Bruta

3. Algortimos Genéticos

Os Algortimos Genéticos foram introduzidos por Holland (1975) e popularizados por um dos seus alunos, Goldberg (1989). Estes algoritmos seguem o princípio da seleção natural e da brevivência apenas do mais apto, ideia cunhada em 1859 pelo naturalista e fisiologista inglês Charles Darwin.

Os Algortimos Genéticos (AGs) são particularmente aplicados em problemas complexos de otimização: problemas com diversos parâmetros ou características que precisam ser combinadas em busca da melhor solução; problemas com muitas restrições ou condições que não podem ser representadas matematicamente; e problemas com grandes espaços de busca (PACHECO et al., 1999). As aplicação práticas dos Algortimos Genéticos são das mais váriadas, indo desde a área médica (GHAHERI et al., 2015) até o o desenvolvimento de antenas pela NASA (HORNBY et al., 2006).

O primeiro passo de um Algoritmo Genético típico é a geração de uma população inicial de cromossomos, que é formada por um conjunto aleatório de cromossomos que representam possíveis soluções de um determinado problema a ser resolvido. Durante o processo evolutivo, esta população é avaliada e cada cromossomo recebe uma nota, refletindo a qualidade da solução do cromossomo (indivíduo).

Em geral, os indivíduos mais aptos são selecionados e os menos aptos são descartados (em geral, pois o comportamento do algoritmo é não-determinístico). Os membros

passam por um processo de seleção, podendo após isso, sofrer modificações em suas características fundamentais através dos operadores de reprodução/cruzamento (crossover) e mutação, gerando descendentes para a próxima geração (LACERDA; CARVALHO, 1999). Frequentemenete, soluções de Algoritmos Genéticos implementam elitismo, isto garante que pelo menos uma cópia sem alterações da melhor solução da geração é passada para a próxima população, de forma que a melhor solução possa sobreviver a sucessivas gerações.

4. Experimentos

Para a implementação do algoritmo evolutivo foi escolhida a linguagem de programação C, por causa de uma série de fatores. A primeira é a familiaridade dos autores com essa linguagem estrutural e procedural criada por Ritchie (1993). Outros fatores para a escolha de tal linguagem são:

- **Difusão**: A linguagem C é amplamente difundida em todo o mundo, sendo a segunda linguagem mais popular no mundo, logo atrás de Java. A popularidade da linguagem é de extrema importância, pois quanto mais popular uma linguagem for, mais pessoas estarão aptas para compreender o algoritmo escrito em tal linguagem.
- **Portabilidade**: é possível compilar o mesmo código C em praticamente todos os tipos de computadores e sistemas operacionais sem grandes alterações (às vezes, sem nenhuma alteração).
- **Programação modular**: essa característica permite ligar o código C com o produzido em outras linguagens como Assembler ou C++.
- **Velocidade**: por ser diretamente compilada em código de máquina, a linguagem não passa por um interpretador, eliminando qualquer *overhead* desnecessário.

4.1. Implementação

O algoritmo em C foi implementado para receber duas entradas por parâmetro. Uma das entradas diz respeito a um arquivo texto e a outra diz respeito a quantidade inicial de indivíduos da população inicial. O arquivo em questão deve estar estruturado de tal modo que cada linha do arquivo contenha apenas dois valores. O primeiro valor de uma linha remete a posição no eixo das abscissas (X) de uma cidade, enquanto o segundo valor remete a posição no eixo das ordenadas (Y) desta mesma cidade. Nesse sentido, cada linha diz respeito as coordenadas de uma cidade.

Após executado, o algoritmo inicialmente aprensenta ao usuário o arquivo informado, ao prosseguir a população inicial então é gerada. Cada indíviduo corresponde a uma possível solução do problema, ou seja, o tamanho do cromossomo (índivíduo) corresponde ao caminho entre as cidades. O cromossomo é representado por um vetor, onde cada elemento do vetor se conecta com seus adjacentes, destaca-se nesse sentido que a primeiro posição do vetor se conecta diretamente com a última posição, formando desta forma um ciclo (caminho do caixeiro viajante).

A determinação dos melhores indivíduos é definida pela aptidão, onde no caso do algoritmo desenlvoldido é realizada por um cálculo de distância euclidiana bidimensional, sendo assim, é calculada a distância entre os genes (cidades) de um determinando

cromossomo. Os cromossomo (soluções) que obtiverem um cálculo menor de distância, são as melhores soluções, portanto, com maior probabilidade de reproduzirem.

A geração da população inicial é feita de tal maneira que não existam genes (cidades) repetidos no cromossomo (solução) sendo assim, além de tratar isso na geração da população é preciso tratar na hora do cruzamento e na hora da mutação. No cruzamento isso se corrigi utilizando o PMX (Partially Matched Crossover), já a mutação precisa ser feita por 'troca de números', onde: se um gene X for mutato para um valor Y, o gene Y do mesmo cromossomo será mutato para o valor de X, preservando assim a unicidade de cada gene.

Para adentrar na etapa de cruzamento, é necessário antes passar por um processo de seleção. O algoritmo desenvolvido apresenta dois processos de seleção: Roleta e Torneio. Na Roleta, cada indivíduo da população é representado na Roleta proporcionalmente ao seu índice de aptidão. Assim, aos indivíduos com alta aptidão (soluções melhores) possuem uma uma maior porção da roleta, enquanto aos de aptidão mais baixa (soluções piores) possuem uma porção relativamente menor da roleta. Os melhores indíviduos possuem maiores chances de serem selecionados pela Roleta e passarem seus genes adiante.

No Torneio são selecionados diversos pequenos subconjuntos da população, sendo selecionado o indivíduo de maior adequação de cada um desses grupos. Desta forma, um indivíduo "batalha"contra outro(s), de modo que o vencedor é o que apresenta a melhor distância euclidiana. O Algoritmo desenvolvido, permite a personalização da quantidade de "batalhadores"por Torneio.

Após a seleção e a reproduzação (cruzamento), os indivíduos da próxima geração passam por um processo de mutação, com a probabilidade de 2% de alterarem algum de seus genes. Ao término, o algoritmo passa por uma etapa final de elitismo (ou não, personalizável pelo usuário). O elitismo é um processo para garantir a sobrevivência do melhor indivíduo da geração passada, retomando a mesmo para aposição que ocupava.

4.2. Testes iniciais

Para comparar o desempenho do algoritmo genético em relação ao algoritmo de força bruta, alguns testes foram feitos. O primeiro é executar ambos os algoritmos com instâncias de 9, 10, 11 e 12 cidades. O outro teste é para verificar até onde o algoritmo genético consegue ir, e para isso foram utilizados casos maiores que serão tatados na seção 4.3. As instâncias da primeira etapa de testes podem ser vistas na Figura 1.

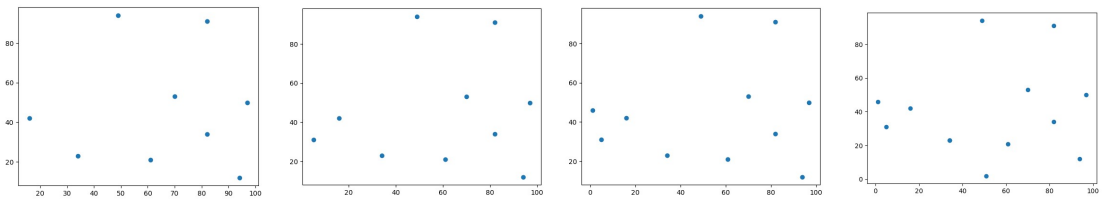

Figura 1. Mapas das cidades

A Figura 1 apresenta quatro diferente mapas, cada qual com 9, 10, 11 e 12 cidades respectivamente. Os pontos em azul representanto as cidades as quais foram colocadas no mapa de acordo com suas coordenadas. Após quatro teste rodados por 50 gerações com uma população inicial de 10 indivíduos para cada uma das instâncias, alguns caminhos subótimos foram encontardos como mostra a Figura 2.

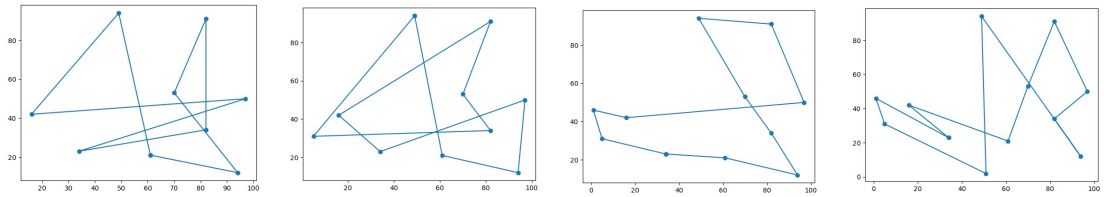

Figura 2. Mapas de algumas soluções encontradas

Dos testes realizados as melhores soluções encontram-se na Figura 3. Os testes foram realizados com uma probabilidade de 2% de mutação, usando seleção por roleta e torneio, reprodução PMX, mutação *swap*, e com/sem elitismo.

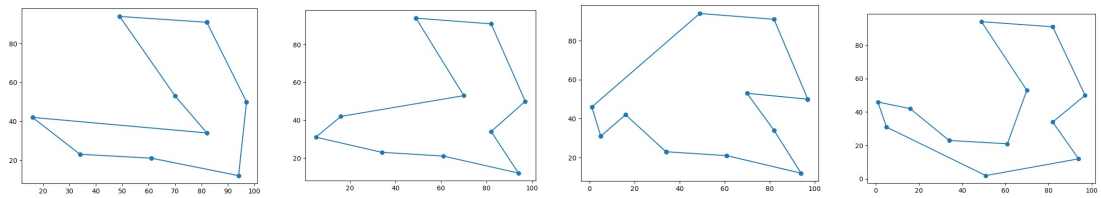

Figura 3. Mapas das melhores soluções encontradas

O primeiro mapa da Figura 3 foi alçado com uso da seleção por roleta, todos os demais são de soluções usando torneio. Com exeção do terceiro mapa, todos são do cenário com elitismo. A Figura 4 mostra mais graficamente a verdadeira influência do elitismo.

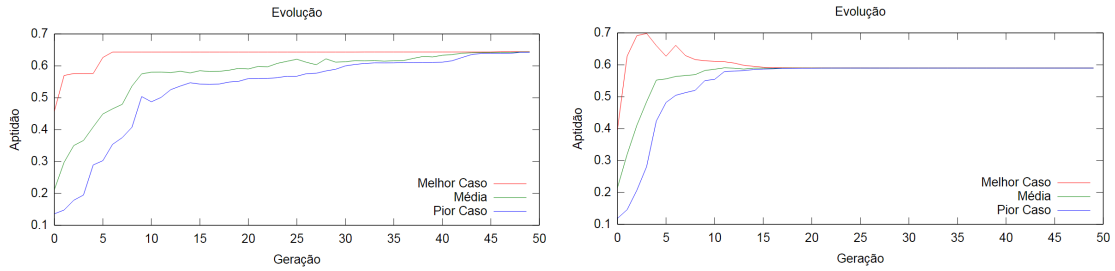

Figura 4. Diferença entre usar e não usar elitismo

A Figura 4 mostra dois teste (9 cidades), o da esquerda com seleção por roleta e com elitismo e o da direita com seleção por torneio (4 indivíduos) sem elitismo. Como é possível constatar, o gráfico da esquerda, nunca abandona a melhor solução encontrada, já o gráfico da direita se desfaz da melhor solução em prol da convergência da população. A diferença entre a roleta e o torneio é melhor constatado na Fgiura 5.

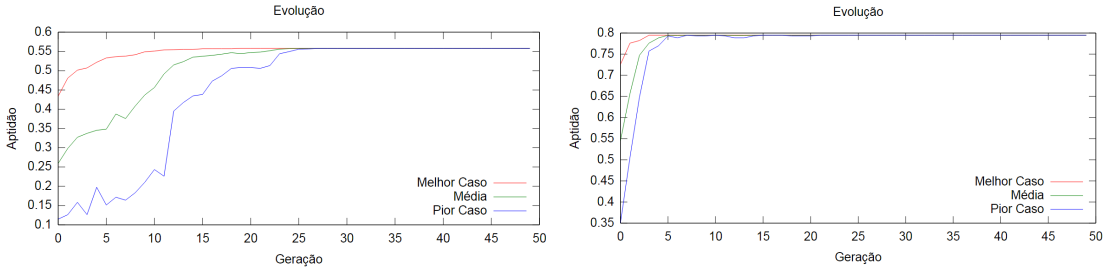

Figura 5. Diferença entre usar Roleta e Torneio

A Figura 5 mostra dois teste (10 cidades), ambos com elistismo, porém o da esquerda com a seleção por roleta e o da direita por torneio. Como é possível observar nos gráficos, usando torneio de 4 indivíduos, o algoritmo alcança uma convergência mais rápida da população (soluções).

A presente seção foi apenas para exemplificar didaticamente como deveria ser o comportamento do algoritmo genético sobre determinadas condições. Usando uma população pequena de apenas 10 individuos é possível notar a convergencia rápida da população, além do peso e da influeência da roleta e do torneio. Somando, o efeito de cada opção de elitismo podem ser observados de forma mais detalhada.

A presente leva de destes, executou relativamente rápido no uso continuo do computador sem consumir muita memória. A médica de tempo foi praticamente equivalente a todos os testes, quase que idependendo da quantidade de cidades. Cada problema, demorou em média 4 segundos para rodar por completo.

4.3. Testes finais

Os testes práticosforam tirados de *datasets* e durante os experimentos usou-se uma população de 200 indivuidos até geração 150ª. Os testes foram realizados com uma probabilidade de mutação de 2%, utilizando apenas o método de seleção de torneio com 4 participantes. Três instâncias do problema foram postas para testes, cada qual com 15, 29 e 38 cidades. A primeira instância foi obtida por meio do seguinte site: https://people.sc.fsu.edu/ jburkardt/datasets/tsp/tsp.html. As demais instâncias foram extraídas do seguinte domínio: http://www.math.uwaterloo.ca/tsp/world/countries.html. As melhores soluções de acordo com seus respectivos sites são elecandas na Figura 6.

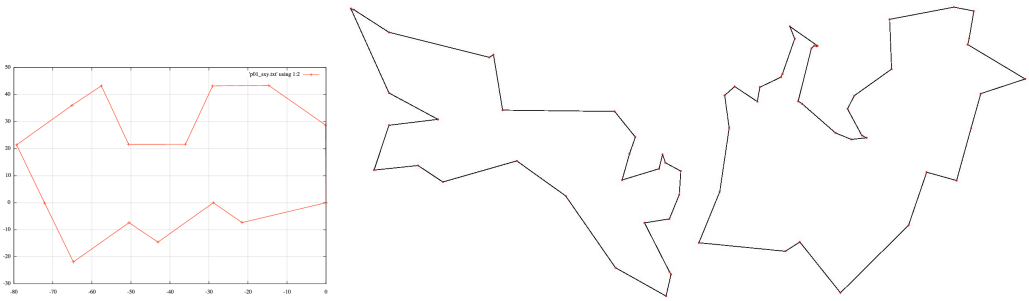

Figura 6. Melhores soluções dadas pelos sites

A Figura 6 apresenta as melhores soluções para as instâncias de 15, 29 e 38 respectivamente. Já a Figura 7 apresenta as melhores soluções encontradas pelo algoritmo genético.

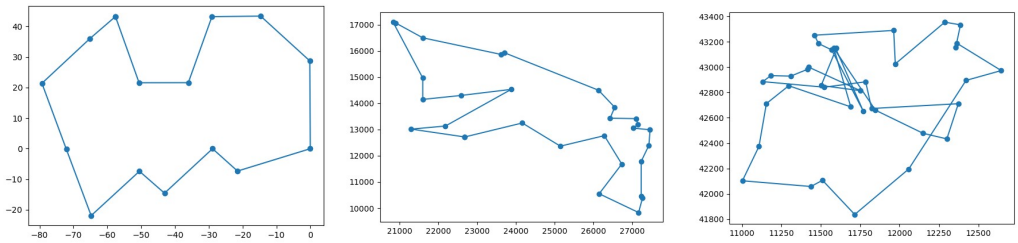

Figura 7. Melhores soluções do AG

O primeiro mapa da Figura 7 se equipara em termos matemáticos ao primeiro mapa da Figura 6, demonstrando que essa foi a única instância que alcançou a solução ótima, todas as demais chegram apenas ao subótimo. O cálculo da distância euclidiana revelou uma distância mínima para o problema de 15 cidades de 291 unidades de medida. Para o problema com 29 cidades, o custo mínimo possível era de 27.603 unidades, com o menor resultado do AG alcançando 29.654. Por fim, para o problema de 38 cidades o menor caminho seria de 6.656 unidade, onde o genético chegou em 9.354. Os gráficos de um dos testes realizados para cada instância encontram-se apresentados respetivamente nas Figuras 8, 9 e 10.

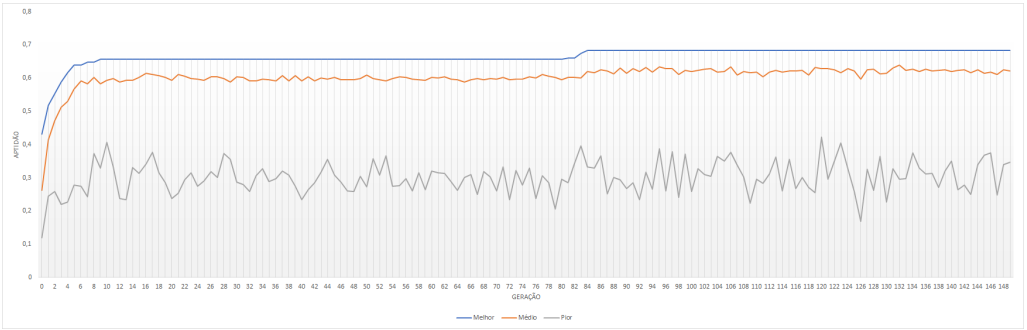

Figura 8. 15 cidades, com elistimos

O teste com 15 cidades é mostrado na Figura 8, o elitismo demonstra que nunca a melhor solução será perdida de uma geração para a próxima. O gráfico em questão representa a solução ótima encontrada para a instância de 15 cidades. Destaca-se que a linha superior representa a aptidão da melhor solução, a linha do meio representa a média das aptidões entre todos os indivíduos, enquanto a última linha representa a pior solução.

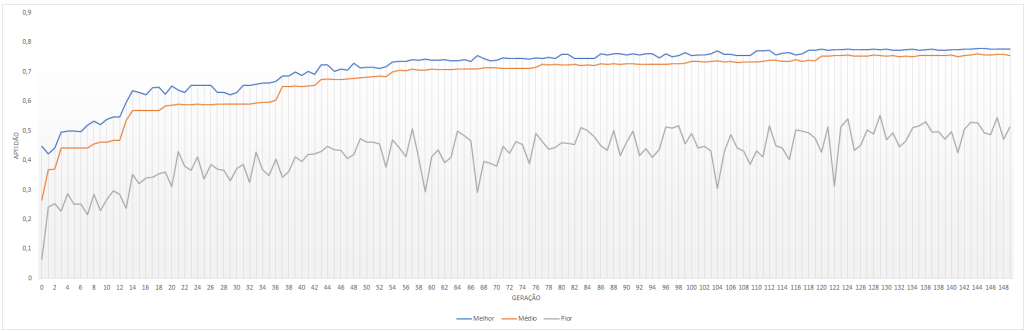

Figura 9. 29 cidades, sem elistimos

A Figura 9 representa o teste com 29 cidades. É possível notar uma convergência relativa da população, porém não uma convergência completa como vista na seção 4.2. Com uma popupalação de 200 individuos, numericamente mais individúos sofrem mutação do que 50 indiviuduos, desta forma, mutações supostamente ruins podem prevalecer por uma maior tempo.

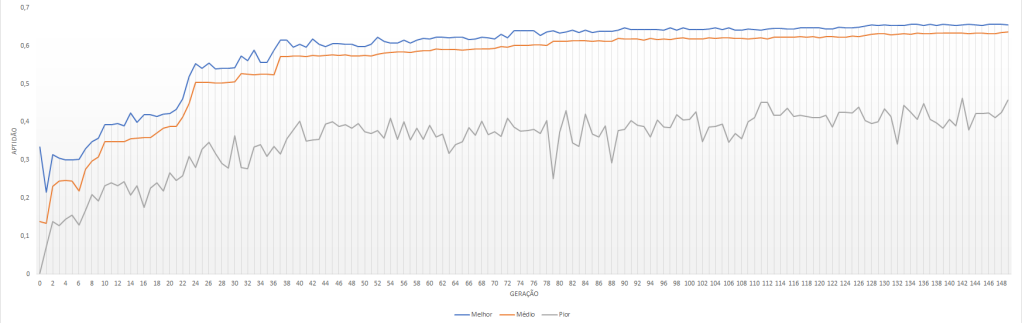

Figura 10. 38 cidades, sem elistimos

A Figura 10 representa o teste com 38 cidades. A aptidão foi calibrada de modo a deixar o pior indivíduo o mais próximo do zero do gráfico. De todas os testes, este foi o mais demorou para executar, levendo em média 1 minuto. Cabe resaltar que, desta leva de testes, os tempos foram praticamente equivalentes.

O AG demonstra-se um bom algoritmo para encontra o subótimo (às vezes o otimo também). Como observado nas Figuras 8, 9 e 10, para instâncias grandes, o problema demora para convergir.

5. Conclusão

A área de computação evolutiva demonstra-se extremamente promissora na solução de problemas que demandem um grande tempo de execução ou um grande uso de memória, onde os algoritmos genéticos acabam por vir para fornecer uma boa solução (às vezes uma ótima solução) em curto periodo de tempo.

Os testes realizados ajudaram o autor a perceber as pequenas diferenças entre uma dada execução e outra com pequenas mudanças nos parâmetros, onde o autor pode concluir que embora uma mutação alta atrapalhe muito a convergência do problema, as chances de aparecer um ótimo valor aumentam (ainda mais com elitismo para não deixar o melhor valor ser mutado para um valor ruim), mesmo em casos com gerações iniciais ruins de população. Todavia é importante equilibrar a mutação de modo que ela não seja muito grande, pois do contrário, entraria-se na área de busca aleatória.

Referências

GHAHERI, A. et al. The applications of genetic algorithms in medicine. *Oman medical journal*, Oman Medical Specialty Board, v. 30, n. 6, p. 406, 2015.

GOLDBERG, D. Genetic algorithms in search. optimization and machine learning. *Reading, MA*, 1989.

HOLLAND, J. H. Adaptation in natural and artificial systems. *The University of Michigan Press*, v. 1, p. 975, 1975.

HORNBY, G. et al. Automated antenna design with evolutionary algorithms. In: *Space 2006*. [S.l.: s.n.], 2006. p. 7242.

LACERDA, E. G. de; CARVALHO, A. de. Introdução aos algoritmos genéticos. *Sistemas inteligentes: aplicaçoes a recursos hıdricos e ciências ambientais*, v. 1, p. 99–148, 1999.

MORAIS, J. L. M. *Problema do caixeiro viajante aplicado ao roteamento de veículos numa malha viária*. Dissertação (B.S. thesis), 2011.

NILSSON, N. J. *Principles of Artificial Intelligence: With 139 Figures*. [S.l.]: Springer-Verlag, 1982.

PACHECO, M. A. C. et al. Algoritmos genéticos: princípios e aplicações. *ICA: Laboratório de Inteligência Computacional Aplicada. Departamento de Engenharia Elétrica. Pontifícia Universidade Católica do Rio de Janeiro. Fonte desconhecida*, p. 28, 1999.

RITCHIE, D. M. The development of the c language. *ACM Sigplan Notices*, v. 28, n. 3, p. 201–208, 1993.