Algoritmos genéticos aplicados ao problema do caixeiro viajante

Júlio Resende, Felipe Faria
Departamento de Ciência da Computação
Universidade Federal de São João Del Rei, Minas Gerais, Brasil
{julio.cmdr, felipevisu}@gmail.com

Abstract

O Problema do Caixeiro viajante é uns dos problemas mais estudados por pesquisadores da área de ciência da Computação. Métodos presentes na literatura são capazes de fornecer soluções ótimas, mas com complexidade que pode inviabilizar a execução do mesmo. Nesse contexto, esse trabalho descreve um método utilizando algoritmos genéticos para se encontrar soluções aproximadas em tempo viável.

1 Introdução

Desde os primórdios é natural do ser humano buscar por melhorias em sua vida. Tais melhorias são possíveis através da capacidade que o mesmo tem de otimizar suas técnicas e tarefas, tornando assim a otimização uma qualidade valorizada desde os tempos mais antigos.

Nos tempos atuais, com a presença da tecnologia, é possível buscar o aperfeiçoamento de trabalhos do dia a dia através da execução de algoritmos que buscam facilitar o cálculo matemático necessário para a resolução de muitos problemas comuns, como por exemplo, um trajeto mínimo a ser percorrido. No entanto, grande parte desses problemas são de complexidade exponencial, sendo considerados como problemas da classe NP-Completo.

Ma teoria da complexidade computacional, é definido como verdade que se caso alguma solução seja apresentada para um problema da classe NP-Completo em tempo polinomial, será possível resolver todos os problemas dessa classe em tempo polinomial também, sendo o problema do caixeiro viajante o mais estudado devido a simplicidade de se entender o desafio proposto pelo mesmo.

Todavia, a única forma de se obter uma solução ótima para todos os problemas da classe NP-Completo é aplicando o método da força bruta, no qual analisa todas as possíveis soluções com a finalidade de encontrar a melhor. Mas apesar do método sempre encontrar a solução ótima, a execução pode chegar a demorar muito a ponto de se tornar inviável caso o número de soluções possíveis para o problema seja alto. Dessa forma, muito se é estudado sobre a criação de algoritmos que encontrem uma solução aproximada, mas que executem em tempo viável.

Grande parte desses algoritmos encontram como soluções máximos e mínimos locais, podendo não resultar em uma solução aproximada da ótima. Apesar disso, existem também métodos capazes de realizar uma busca mais robusta em um espaço de solução, os quais são classificados como Metaheurísticas[Glover and Kochenberger, 2003].

Exemplos clássicos de Metaheurísticas são: GRASP [Feo and Resende, 1989], Busca Tabu [Glover, 1986], Colônia de Formigas [Dorigo and Stützle, 2004] e Algoritmos Genéticos [Holland, 1975].

Portanto, o objetivo deste trabalho é realizar uma investigação sobre a aplicação de algoritmos genéticos ao problema do caixeiro viajante, a fim de se obter uma solução ótima ou aproximada em tempo viável.

2 Referencial Teórico

O Problema do Caixeiro Viajante (PCV) é um problema NP-Completo clássico de otimização, no qual propõe que seja encontrado uma rota que passe por um certo número de cidades apenas uma vez, e volte para a cidade de partida. Sabendo-se da distância entre cada uma das cidades, o desafio do problema é selecionar o percurso de menor tamanho dentre os viáveis. Na teoria de grafos este problema pode ser modelado como uma busca pelo menor ciclo hamiltoniano, e possui grandes aplicações, como por exemplo roteamento de veículos, projeto de circuitos integrados, dentre outros,

2.1 Algoritmos Genéticos

Proposto em 1975, os algoritmos genéticos foram criados utilizando os conceitos da Biologia de evolução, e hoje são amplamente aplicados a diversas áreas como: Engenharia, Música, Telecomunicações, setores de petróleo, gás e Saúde [Redusino, 2009]. A ideia principal do algoritmo consiste em criar um conjunto de soluções que evolui a cada geração, no qual recebe o nome de população.

Cada solução presente na população representa um indivíduo, sendo este composto por um cromossomo que permite que seja possível avaliar o indivíduo de acordo com sua composição genética. Tal avaliação é realizada através de uma função de aptidão comumente denominada de *fitness*. Em linguagens de programação, o cromossomo pode ser representado como um vetor ou uma lista encadeada que, para o problema do caixeiro viajante, pode ser uma sequência de cidades a se percorrer [Guedes *et al.*, 2005].

A evolução da população ocorre através de operações como:

- Seleção: Seleciona dois indivíduos aleatoriamente para serem pais de novos indivíduos. Apesar da escolha ser aleatória, os indivíduos com a melhor aptidão possuem mais chances de serem escolhidos. Métodos comuns de seleção são: roleta, proporcional, escalonamento, Boltzmann, ranqueamento e torneio [Rodrigues *et al.*, 2004].
- Cruzamento: Esta operação funde os dois pais a fim de obter dois filhos para a nova população. Na literatura os métodos mais utilizados são o cruzamento por dois pontos, e um ponto. A figura 1 ilustra um exemplo de cruzamento de um ponto.
- Mutação: Altera um ou mais genes de um determinado cromossomo de forma aleatória.

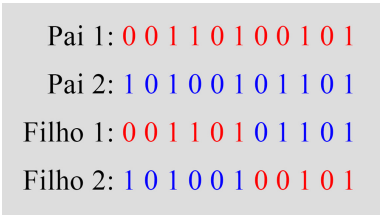

Figura 1: Exemplo de cruzamento de um ponto

Trabalhos anteriores como de [Rosendo and Pozo, 2010] e [Potvin, 1996] aplicaram algoritmos evolutivos para a resolução do PCV, obtendo em ambos resultados satisfatórios. No entanto, poucos foram os trabalhos que realizaram um comparativo efetivo entre operadores de cruzamentos para a resolução do PCV.

3 Metodologia

Por se tratar de um ciclo, é conveniente definir para a modelagem do problema uma cidade fixa de partida e chegada, diminuindo assim em n o número de possibilidades possíveis. Dessa forma foi definido como cromossomo dos indivíduos uma sequência de cidades a se percorrer que pertence ao intervalo $[1, n-1]$, uma vez que a cidade 0 foi marcada como ponto de partida e chegada, e portanto não é incluída no cromossomo. Um exemplo de cromossomo para um problema que visa otimizar o ciclo hamiltoniano entre 7 cidades pode ser visto em 1.

$$[1, 6, 5, 3, 2, 4] \tag{1}$$

Cada indivíduo, além de possuir o cromossomo que representa uma possível solução para o problema, contém também marcadores que o identificam de acordo com a geração e a aptidão do mesmo.

As configurações iniciais do algoritmo genético são definir o número de indivíduos existentes de cada população e o número gerações que serão criadas. Analisando o problema que consistia em um percurso de 12 cidades foi definido um número de 40 indivíduos por geração e de 100 gerações por execução.

Após gerar a primeira população aleatoriamente, o algoritmo genético realiza repetidamente as seguintes etapas:

1. Função de aptidão;
2. Seleção dos pais;
3. Crossover;
4. Mutação;
5. Definição da população sobrevivente;
6. Critério de parada.

Estas etapas serão detalhadas uma a uma nas sub-sessões seguintes.

3.1 Função de aptidão

Esta função toma como medida o custo para percorrer a sequência de cidades apresentada no cromossomo. É importante perceber que para o problema do caixeiro viajante o indivíduo que apresenta a melhor solução é aquele com a menor nota de aptidão, uma vez que esta nota representa a distância a ser percorrida e o objetivo do problema é encontrar a menor delas. Posteriormente a população é ordenada do melhor indivíduo para o pior (menor distância para a maior). Esta ordenação terá influência no algoritmo seguinte que consiste em selecionar os pais para o cruzamento.

3.2 Seleção dos pais

As gerações seguintes serão sempre compostas por indivíduos gerados à partir do algoritmo de cruzamento, mas antes é preciso escolher quais pares de indivíduos serão os reprodutores.

A função de seleção desempenha um papel importante no algoritmo genético, baseado no conceito de que indivíduos geneticamente evoluídos tendem a gerar descendentes que também são e assim evoluir a espécie como um todo a cada geração. O objetivo é obter um método aleatório onde tais indivíduos geneticamente evoluídos possuem mais chances de serem escolhidos mas sem ignorar o restante da população, e para isto foi utilizado o método da roleta viciada.

Considerando que os indivíduos estão ordenados em uma lista do pior par o melhor, esta função deve retornar com maior frequência aqueles que estão no final da lista. O valor base da roleta é formado por uma sequência de fibonacci gerada a partir do número de indivíduos e posteriormente multiplicada por um número aleatório entre 0 e 1. Inicialmente o índice do indivíduo escolhido é o último, então um somatório é iniciado a partir do final da lista e enquanto este somatório for menor que o valor base da roleta o índice do indivíduo escolhido é decrementado.

Esta solução não descarta completamente os indivíduos mais fracos, isto porque mesmo mal classificados, dois indivíduos ainda podem se reproduzir e gerar filhos satisfatórios, porém com uma frequência menor.

3.3 Crossover

Neste algoritmo foi utilizado um método de reprodução sexuada onde são necessários dois indivíduos diferentes para realizar a reprodução. Desta reprodução são sempre gerados dois filhos, o que significa que todas gerações possuem o mesmo número de indivíduos.

Vários algoritmos de cruzamento existem na literatura, para fins de comparação e análise de desempenho foram utilizados três deles: cruzamento de ponto único (OPX), cruzamento parcialmente combinado (PMX) e cruzamento baseado na ordem (OBX).

Cruzamento de ponto único (OPX)
Trata-se do algoritmo de cruzamento mais popular para algoritmos genéticos, no qual um único ponto denominado ponto de corte é gerado aleatoriamente nos cromossomos de ambos os pais. Os dados antes do ponto no pai 1 são combinados com os dados além do ponto no pai 2, disto resulta o primeiro filho. O segundo filho é gerado de forma inversa, sendo composto pelos dados após o ponto no pai 1 e antes do ponto no pai 2. Este processo já foi ilustrado na Figura 1.

Um problema surge neste método, a união das partes de dois cromossomos pode gerar valores repetidos, um tratamento adicional para este caso consiste em eliminar tais valores e substitui-los pelos que ficaram faltando.

Cruzamento Parcialmente Combinado (PMX)
O algoritmo parcialmente combinado, também conhecido como parcialmente mapeado, é considerado um dos mais eficientes para uma ampla variedade de aplicações. O pai 1 doa uma faixa de seu cromossomo para o filho e esta mesma faixa corresponde no pai 2 é inserida aleatoriamente nos espaços restantes. Aqui um tratamento para prevenir valores repetidos também é necessário. Uma execução deste algoritmo pode ser visualizada na Figura 2.

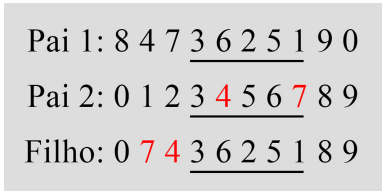

Figura 2: Cruzamento parcialmente combinado

Cruzamento Baseado na Ordem (OBX)
Este é um método de reprodução assexuada o que significa que os indivíduos não compartilham dados entre si, a reprodução é feita modificando apenas um indivíduo e gerando um filho. Semelhante ao processo de mutação, pontos aleatórios são definidos em posições iguais em ambos os pais, posteriormente estas posições são embaralhadas entre si conforme ilustra a Figura 3.

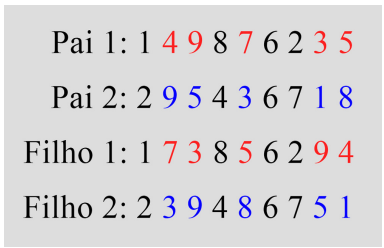

Figura 3: Cruzamento baseado em ordem

3.4 Mutação

Na natureza os filhos podem sofrem mutações e apresentar características que não estavam presentes nos pais, talvez porque gerações anteriores aos pais possuíam estas características. Por exemplo pode acontecer de um filho ter olhos verdes mesmo se ambos os pais tiverem olhos castanhos. Tal característica é representada nos algoritmos genéticos aplicando pequenas modificações nos resultados logo após o cruzamento.

O método utilizado foi o de mutação baseada na posição, onde duas posições aleatórias do cromossomo são trocadas de lugar. Isto é ilustrado na figura 4. É preciso definir uma taxa de mutação correspondente a frequência com que esta mutação pode ocorrer, para o problema proposto foi definida uma taxa de 5%. Configurar estes valores corretamente tem influência no desempenho do algoritmo, a mutação ajuda e manter diversidade na população prevenindo que os indivíduos se tornem cada vez mais parecidos ao longo das gerações.

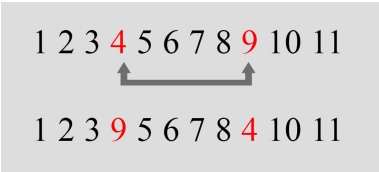

Figura 4: Exemplo de mutação em um cromossomo

3.5 Definição da população sobrevivente

Após a reprodução e a mutação o novo indivíduo gerado apenas fará parte da população seguinte se o mesmo possuir um desempenho superior ao de seus pais, do contrário o indivíduo é descartado e a nova população mantêm os indivíduos reprodutores.

Durante a fase de testes em execuções onde este método não foi aplicado foi possível observar que as populações não evoluíam tão satisfatoriamente. Inserir os filhos na população seguinte apenas se eles forem melhores que os pais garante que ao decorrer da gerações a população seja cada vez mais evoluída, não regredindo em termos do critério de aptidão.

3.6 Critério de parada

O algoritmo se encerra quando o número de gerações se iguala a 100. Este número foi definido por uma análise empírica dos dados onde foi possível observar que um número maior de gerações surtia pouco efeito nos resultados. Antes de 100 gerações o melhor resultado obtido já se aproxima de um resultado ótimo e a medida que isto acontece o número de gerações necessárias para obter qualquer melhoria aumenta consideravelmente.

4 Experimentos e análise dos resultados

Em todos os experimentos detalhados nesta seção foi utilizado uma mesma entrada para o problema, garantindo assim uma equiparidade entre os experimentos. A entrada possui 12 cidades, e a distância do menor percurso possível é

equivalente a 29.248733 unidades de medida, e o maior percurso equivale a 70.681161 unidades de medida. Os resultados foram gerados a partir da implementação dos algoritmos na linguagem python, e os gráficos gerados pela biblioteca mathplotlib[Hunter, 2018].

Foram realizados vários experimentos utilizando a mesma entrada com a finalidade de se descobrir qual o operador de cruzamento mais eficiente (cruzamento de ponto único, cruzamento parcialmente combinado ou cruzamento baseado na ordem), de forma que os algoritmos foram executados por 50 vezes e foi utilizado como comparação a média dos resultados finais, e a média do número de gerações que cada uma das 50 execuções necessitou para se estabilizar.

As Figuras 12, 6 e 7 exibem gráficos que mostram em azul o valor obtido pela função aptidão do melhor indivíduo final de cada uma das 50 execuções, e em vermelho a média obtida por essas execuções. A figura 12 se refere as execuções utilizando cruzamento de ponto único, figura 6 cruzamento parcialmente combinado e figura 7 cruzamento baseado na ordem. A tabela 1 relaciona os resultados obtidos pelos três métodos de cruzamento.

Cruzamentos	Média de aptidão dos melhores indivíduos
OPX	31.8
PMX	34.6
OBX	35.6

Tabela 1: Comparativo entre os três métodos de cruzamento

Como observado na tabela 1, o algoritmo utilizando o cruzamento de ponto único foi o que obteve o melhor desempenho. A justificativa de tal desempenho pode ser visualizada nas figuras 8, 9 e 10, no qual exibem gráficos de todos os indivíduos de todas as populações para os três métodos, e é possível perceber que o cruzamento de ponto único foi o único dos três métodos que gerou evolução para a população.

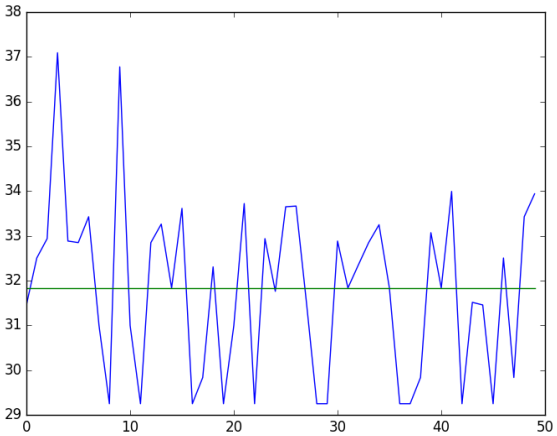

Figura 5: Amostragem de 50 execuções do algoritmo genético utilizando cruzamento de ponto único

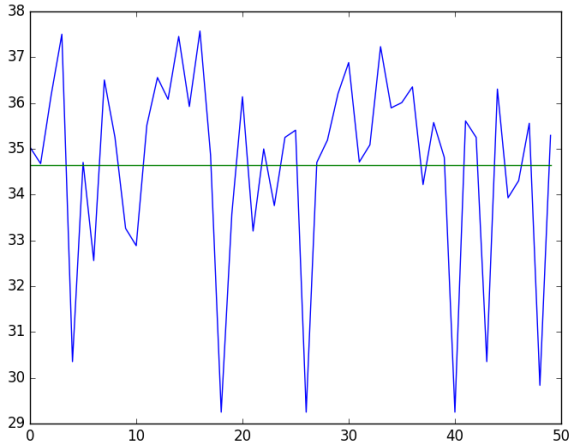

Figura 6: Amostragem de 50 execuções do algoritmo genético utilizando cruzamento parcialmente combinado

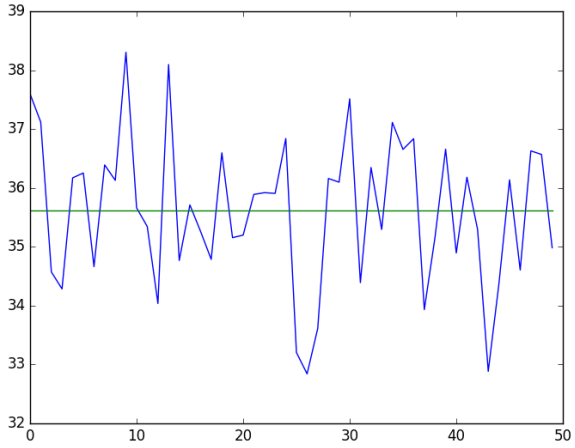

Figura 7: Amostragem de 50 execuções do algoritmo genético utilizando o cruzamento baseado na ordem

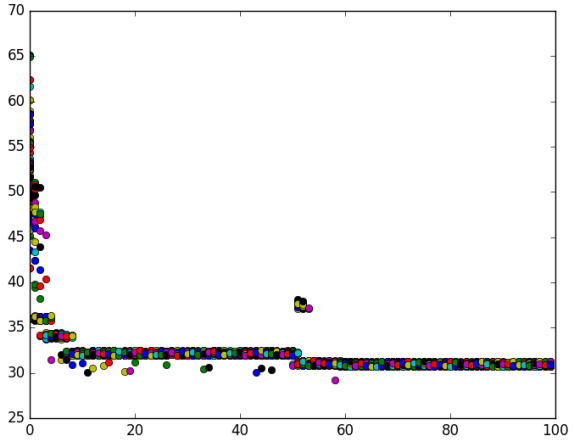

Figura 8: Função de aptidão dos indivíduos de cada população utilizando o cruzamento de ponto único

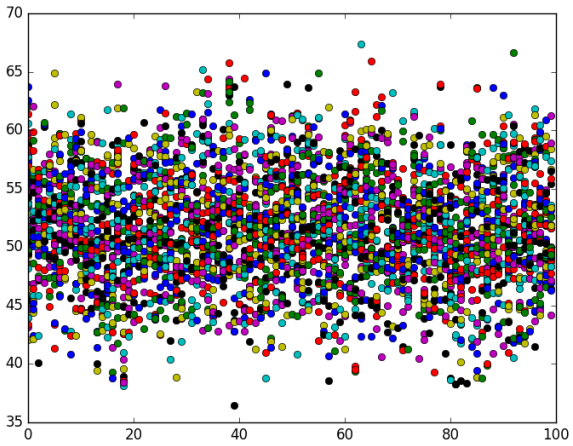

Figura 9: Função de aptidão dos indivíduos de cada população utilizando o cruzamento parcialmente combinado

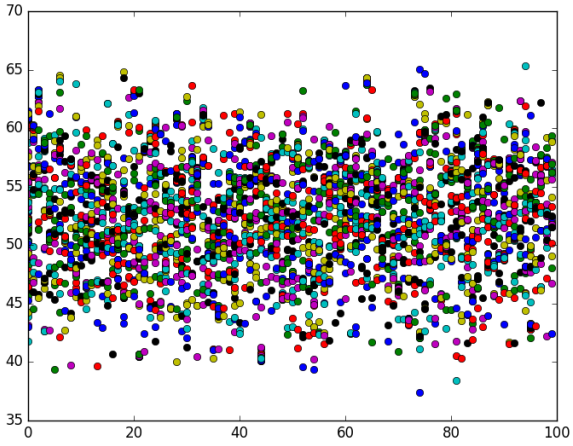

Figura 10: Função de aptidão dos indivíduos de cada população utilizando o cruzamento baseado na ordem

4.1 Análise de tempo

Como dito anteriormente, a principal motivação deste trabalho é construir um algoritmo para encontrar soluções eficientes, sendo o tempo de execução do algoritmo um fator primordial. Portanto foi realizada uma análise comparando o tempo de execução do algoritmo genético com o método da força bruta.

Como a execução do método força bruta demora muito quando o número de cidades é alto, não foi possível mensurar o tempo de execução para variados número de cidades, sendo necessário realizar uma estimativa de tempo de acordo com a complexidade do algoritmo. Para um conjunto de 12 cidades, o método força bruta demorou 3.3 segundos, e para 13 cidades demorou 40 segundos. A proporção entre os dois tempos respeita de forma bem satisfatória a complexidade do algoritmo que é de $(n-1)!$, o que permite que a mesma seja usada para a definição de tempo de execução do algoritmo. De acordo com esses dados, foi possível construir a equação 2, no qual retorna o tempo de execução do algoritmo em função do número de cidades n, e a partir disso construir o gráfico da figura 11, em que o eixo X representa o número de cidades, e o eixo Y o tempo de execução em segundos.

$$t(n) = \frac{3.3 * (n-1)!}{11!} s \tag{2}$$

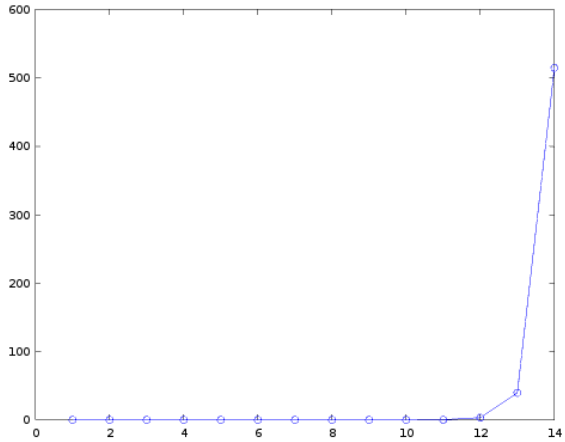

Figura 11: Gráfico que representa estimativa de tempo do método da força bruta

Obter a função complexidade para algoritmos genéticos é uma tarefa difícil. Portanto, foram realizados testes com o número de cidades variando de 4 a 20, sendo possível obter 16 pontos que posteriormente foram utilizados para realizar uma regressão polinomial, no qual foi obtida utilizando o software Libre Office Calc. Feito isso foi possível se obter a função 3, que estima o tempo de execução do algoritmo para todos valores de n. O gráfico da figura 12 exibe esse processo, em que o eixo X representa o número de cidades, e o eixo Y o tempo de execução em segundos.

$$t_2(n) = 0,0006802336n^2 + 0,0228366701n + 0,0327495929 \tag{3}$$

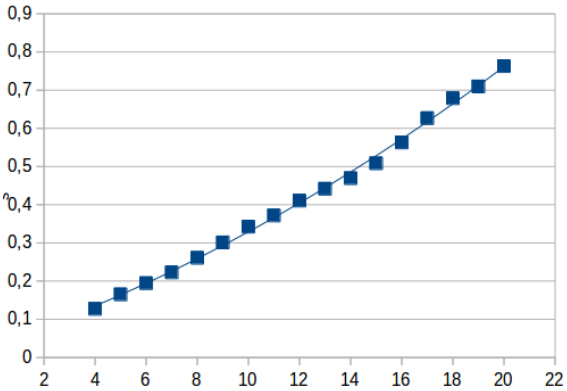

Figura 12: Gráfico que representa o processo de regressão polinomial para estimativa de tempo do algoritmo genético

Para uma quantidade de 20 cidades, enquanto o algoritmo genético demora cerca de 0,76 segundos para executar, o método da força bruta demora cerca de 318,28 anos para executar, sendo uma alternativa inviável.

5 Conclusão

O algoritmo proposto apresenta soluções que demonstram ser bastante satisfatórias, sendo altamente aplicável em problemas que buscam otimizar ciclos hamiltonianos que possuem um grande número de vértices. Para o sucesso do método foi de grande importância realizar experimentos com diferentes técnicas de cruzamentos, pois este é um dos fatores que garante a evolução das populações.

Melhorias no algoritmo ainda podem ser feitas, sendo uma delas a modelagem de uma função que relacione o número de cidades com o número de indivíduos necessário por população, obtendo assim bons desempenhos para variadas situações.

Referências

[Dorigo and Stützle, 2004] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization (MIT Press)*. A Bradford Book, 2004.

[Feo and Resende, 1989] Thomas A Feo and Mauricio G.C Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, apr 1989.

[Glover and Kochenberger, 2003] Fred W. Glover and Gary A. Kochenberger. *Handbook of Metaheuristics*. Springer US, 2003.

[Glover, 1986] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, jan 1986.

[Guedes *et al.*, 2005] Allison C. B. Guedes, Jéssica N. F. Leite, and Dario J. Aloise. Um algoritmo genético com infecção viral para o problema do caixeiro viajante. *PublICa*, 1:16–24, 2005.

[Holland, 1975] John H. Holland. *Adaptation in Natural and Artificial Systems (Complex Adaptive Systems)*. A Bradford Book, 1975.

[Hunter, 2018] John D. Hunter. Mathplotlib 2.2.2. `https://matplotlib.org/index.html`, 2018.

[Potvin, 1996] Jean-Yves Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63:339–370, 1996.

[Redusino, 2009] Augusto C. E. Redusino. Aplicações de algoritmos genéticos. *Revista de Sistemas de Informação da FSMA*, 3:34–39, 2009.

[Rodrigues *et al.*, 2004] Flávio L. Rodrigues, Helio G. Leite, Heleno N. Santos, Agostinho L. Souza, and Gilson F. Silva. Metaheurística algoritmo genético para solução de problemas de planejamento florestal com restrições de integridade. *Revista Árvore*, 28(2):233–245, abril 2004.

[Rosendo and Pozo, 2010] Mateus Rosendo and Aurora T. Ramirez Pozo. Um algoritmo de otimização por nuvem de partículas para resolução de problemas combinatórios. 2010. Dissertação de Mestrado (Informática), UFP.