UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA ELÉTRICA

GUSTAVO ALVES PACHECO

PROBLEMA DO CAIXEIRO VIAJANTE

Uberlândia-MG

2019

GUSTAVO ALVES PACHECO

PROBLEMA DO CAIXEIRO VIAJANTE

Trabalho apresentado ao curso de Engenharia de Computação da Universidade Federal de Uberlândia, como requisito parcial de avaliação da disciplina de Algoritmos Genéticos.

Prof.: Keiji Yamanaka

Uberlândia-MG

2019

SUMÁRIO

1. INTRODUÇÃO

Até o presente momento, preocupou-se com o valor de cada posição em uma *string*, por exemplo, ao se calcular a aptidão dos indivíduos. No geral, era importante o estado de cada gene, e o que esse estado, nessa posição, representava. Entretanto, novos problemas surgem, nos quais já não é prioridade a avaliação de cada valor nas posições, e sim a ordem nos quais determinados valores discretos aparecem.

Um exemplo clássico desse tipo de problema é o TSJ (Travelling Salesman Problem), o problema do caixeiro viajante. Nele, deseja-se descobrir qual o menor trajeto entre n cidades, visitando todas elas.

Assim como anteriormente, alguns elementos utilizados precisam receber uma nova análise. É o caso principalmente do design dos cromossomos, da função de otimização, da forma como o crossover e a mutação vão acontecer e a maneira de visualização dos dados.

Para o crossover, deve-se observar que já não é mais possível utilizar a técnica anterior, já que poderia ocorrer duplicidade no cromossomo filho, eliminando algumas das cidades a serem visitadas, ocasionando erro. Já para a mutação, é necessário pensar em alguma forma de gerar aleatoriedade os valores sem que haja duplicidade de genes. Ambas as novas estratégias são especificadas em mais detalhes nas seções 3 e 4.

Nesse caso, um quesito importante a ser levado em conta é a complexidade de tempo do algoritmo, que chega a ordens exponenciais.

2. OBJETIVOS

- Modificar o algoritmo genético implementado até o momento para que resolva um problema de otimização de rotas
- Desenvolver novo design para os cromossomos
- Formular a equação a ser trabalhada
- Implementar crossover PMX
- Implementar mutação Swap
- Desenvolver forma de visualização dos resultados
- Verificar impacto dos parâmetros do algoritmo genético, no processo de *tuning*
- Observar tempo de execução, tentando maximizar a velocidade do código.

3. MATERIAIS E MÉTODOS

A linguagem utilizada foi o Racket, uma linguagem de uso geral, baseada em Scheme e Lisp. Como interface de desenvolvimento, foi utilizado o DrRacket.

Para confecção da interface gráfica, utilizou-se a linguagem *racket/gui*, um recurso nativo da própria linguagem Racket. O processo de plotagem foi realizado através das funções do pacote *plot*.

Como métodos, utilizou-se uma abordagem *top-down*, com auxílio de um framework SCRUM para desenvolvimento ágil. Em algumas situações, uma visão *bottom-up* foi adicionada ao projeto, na parte de incremento da própria linguagem, ao criar novos operadores e funcionalidades. O código produzido segue majoritariamente uma abordagem funcional, mas possui elementos procedurais, para facilitar a leitura de algumas funções.

Durante o desenvolvimento, a documentação de funções se mostrou fundamental. Principalmente, para que algumas estruturas de dados fossem melhor especificadas, e a execução do código fosse correta.

Para o crossover, foi utilizado um método chamado de PMX (Partially Matched Crossover), no qual os cromossomos pais são, ao início do processo, divididos em três partes. A parte central de ambos os indivíduos é trocada, assim como acontecia no crossover de dois pontos. Entretanto, diferente do método antigo, o PMX avalia todos os outros genes desse novo cromossomo, de forma a evitar alguma duplicidade no indivíduo. Para cada gene, é verificado a presença do mesmo no resto da representação. Caso seja encontrado, ele é substituído pelo gene que se encontra na posição equivalente da duplicata, no outro indivíduo.

Ou seja, implicitamente é criada uma tabela de conversão, no qual as partes centrais dos pais são mapeadas entre si, funcionando como uma estrutura *hash*, de chave e valor. Por exemplo, sejam 2 *strings*,

A: “9|8|4|5|6|7|1|3|2|10”

B: “8|7|1|2|3|10|9|5|4|6.

Caso as posições selecionadas sejam 2 e 5, temos a seguinte divisão (o símbolo * mostra a posição de corte do cromossomo):

A: “9|8|4*5|6|7*1|3|2|10”

B: “8|7|1*2|3|10*9|5|4|6”

É criada uma tabela de conversão da forma:

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

Portanto, caso algum desses elementos apareçam fora da área central de corte, ele é substituído pelo equivalente na tabela. Logo os filhos ficariam da forma

F1: “9|8|4|2|3|10|1|6|5|7”

F2: “8|10|1|5|6|7|9|2|4|3”

Percebe-se que no F1, o 3 foi substituído pelo 6, o 2 pelo 5 e o 10 pelo 7. No segundo, o 7 foi trocado pelo 10, o 6 pelo 3 e o 5 pelo 2.

O método utilizado pela mutação foi o de swap, que apenas troca dois genes de lugar, aleatoriamente, de acordo com um parâmetro de probabilidade, estabelecido inicialmente.

4. RESULTADOS E DISCUSSÕES

Inicialmente, planejou-se como seria representado cada cromossomo, e optou-se por deixar cada gene como uma coordenada x e y, nos quais ambos os valores variam entre 0 e 1000. Esse valor representa a posição da cidade em um mapa e é implementado com uma lista de dois inteiros. Cada indivíduo seria composto por uma lista de pontos e cada população, por uma lista de indivíduos. Como já se sabe a forma do retorno do algoritmo genético, no caso uma lista de populações, a animação para visualização dos dados foi implementada, e os gráficos de performance foram adaptados.

Para essa visualização, cada cidade representava um vértice nas linhas que representavam os caminhos. Todas as rotas de cada geração eram impressas simultaneamente, com uma opacidade baixa, que resultava na sobreposição de vários trajetos. Ao final, o melhor trajeto era selecionado em azul.

Em seguida, implementou-se a função para definição do *fitness.* Nesse caso, deseja-se minimizar a distância total, consequentemente (como feito anteriormente), maximizar a função *offset – distância.* Funções auxiliares precisaram ser criadas, em especial as de agrupamento de elementos em uma lista, para que a distância entre pontos fosse calculada.

A seguir, outros aspectos do A.G. inicial foram modificados, como as funções que trabalhavam com valores binários, que puderam ser removidas. Estando preparada a lógica do algoritmo, voltou-se a atenção para a codificação do crossover PMX e da mutação Swap.

Ambas as funções foram relativamente simples de implementar, em termos de código, mas exigiram um certo trabalho de pensamento lógico, principalmente por serem compostas de recursões duplas, nos quais a troca de itens entre cromossomos ocorre de forma recursiva, para melhorar a performance (o compilador está equipado com algoritmos que otimizam recursões de cauda, deixando-as tão rápidas quanto código sequencial).

Após o processo de adaptação do algoritmo, alguns testes foram feitos. Em vários testes, notou-se resultados de performance bem semelhantes aos encontrados anteriormente, nos outros problemas. Itens como seleção por torneio, elitismo habilitado, crossover alto e mutação baixa foram os que propiciaram os melhores resultados. É interessante notar que o aumento no número de cidades é acompanhado de um aumento no tempo de convergência.

Tal fato é trivial, mas representa um impacto muito grande no sistema, visto que se torna necessário aumentar o número de indivíduos, também. Isso gera execuções bem longas. Outro detalhe que se percebe ao analisar o tempo de execução é o da natureza exponencial do algoritmo. Ao se aumentar o número de cidades, o tempo de execução também aumenta consideravelmente. Entretanto, diferente do que acontece em algoritmos específicos para o problema, a complexidade obtida foi quase polinomial, chegando a ser linear para valores abaixo de 1000 cidades (com simulações executadas em quatro núcleos de processamento).

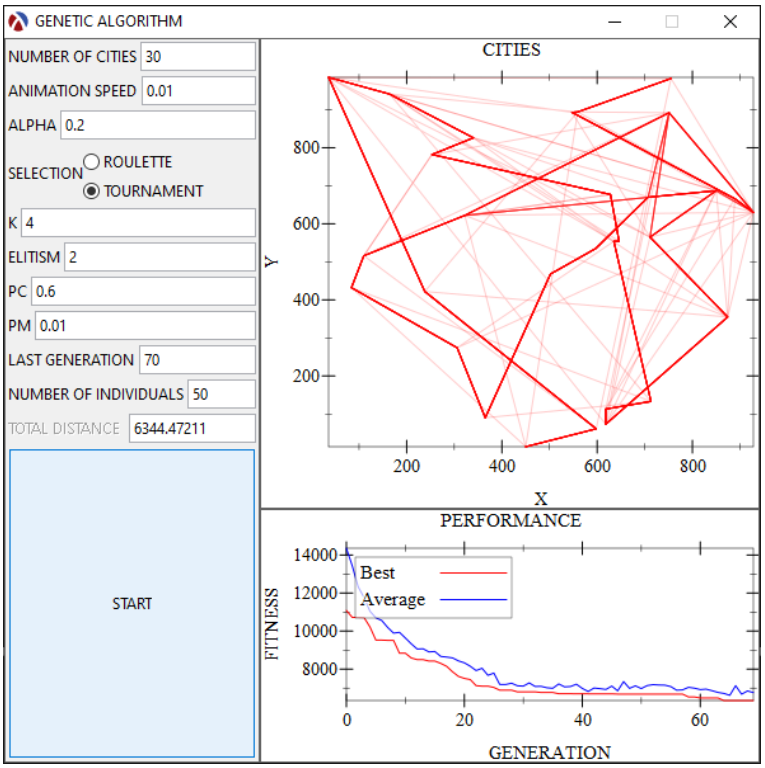

Figura 1: Algoritmo em execução, convergência rápida e média das populações estável e próxima do valor máximo

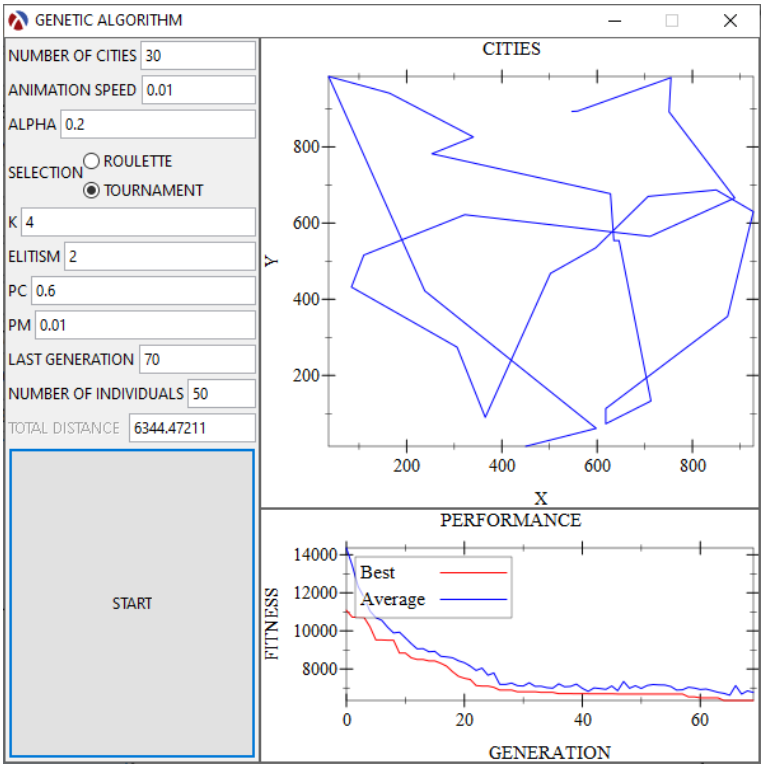

Figura 2: Fim da execução, melhor trajeto marcado em azul e distância total percorrida exibida à esquerda

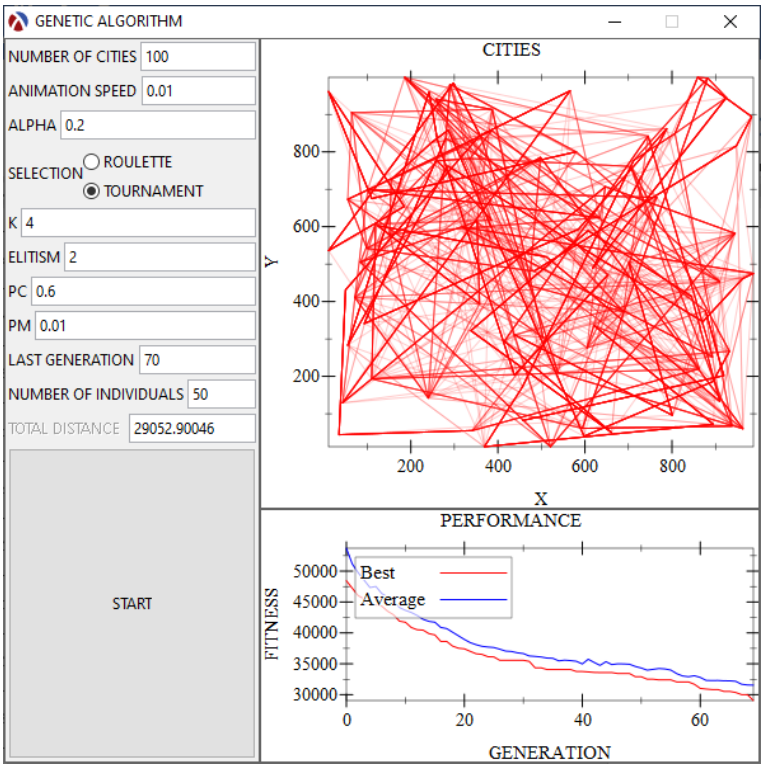

Figura 3: Um aumento no número de cidades está atrelado a um período maior para convergência

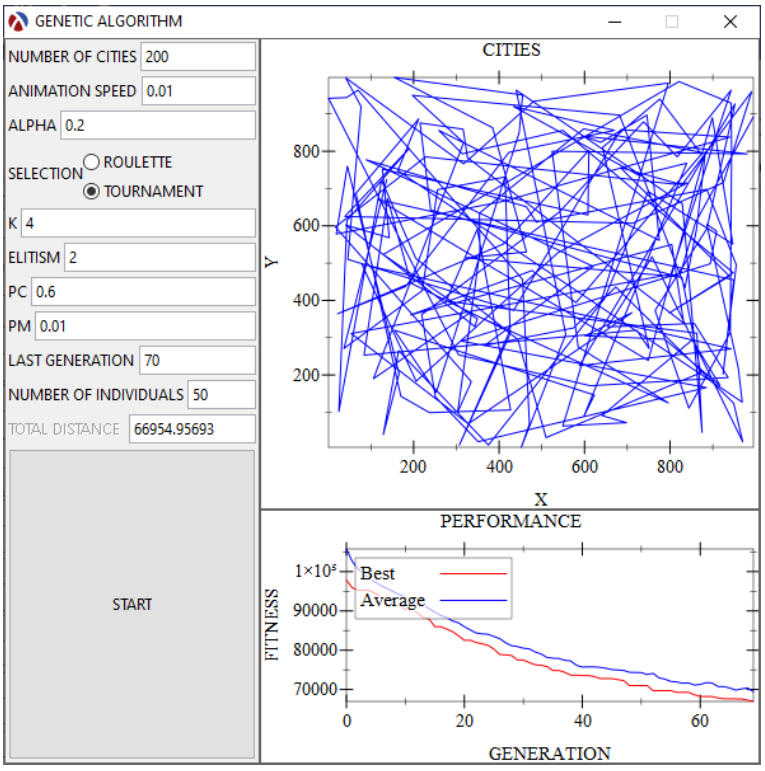

Figura 4: Monitoramento do tempo de execução em ms para 200 cidades.
cpu time: 2032 real time: 2030 gc time: 203

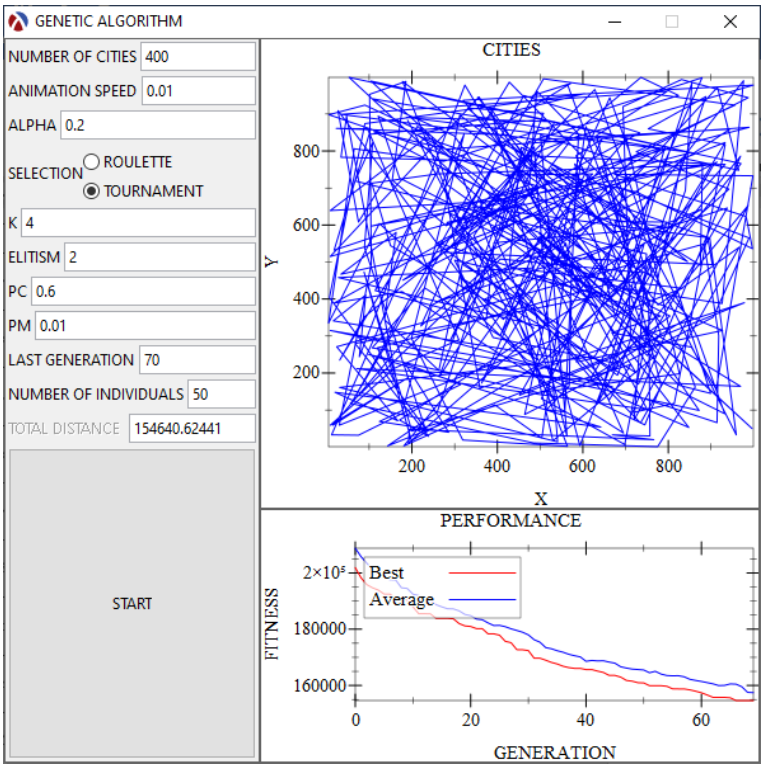

Figura 5: Monitoramento do tempo de execução em ms para 400 cidades. Nota-se que o dobro do valor gera um tempo de execução três vezes maior.
cpu time: 6266 real time: 6283 gc time: 512

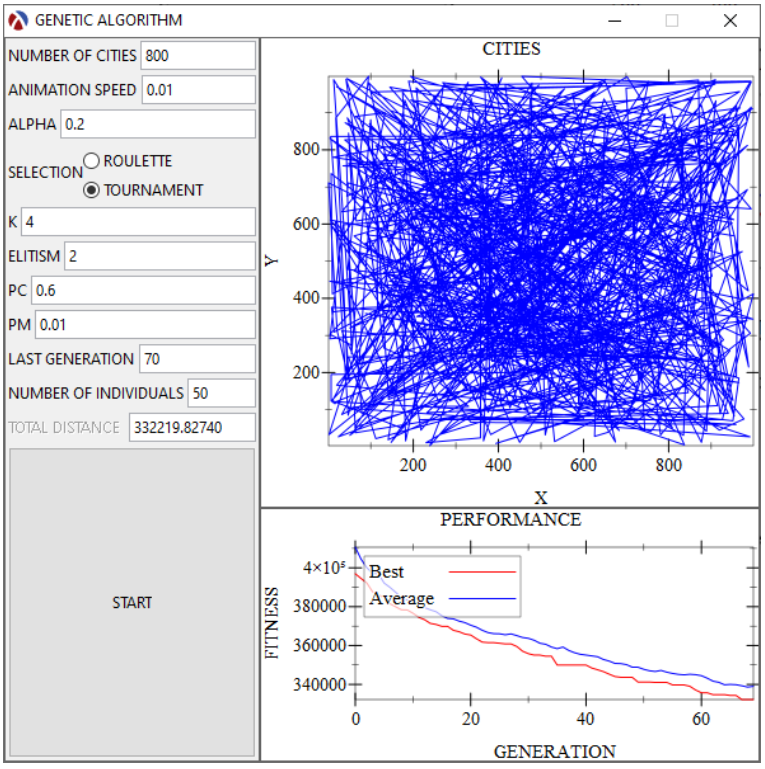

Figura 6: Monitoramento do tempo de execução em ms para 800 cidades.
cpu time: 20391 real time: 20486 gc time: 1554

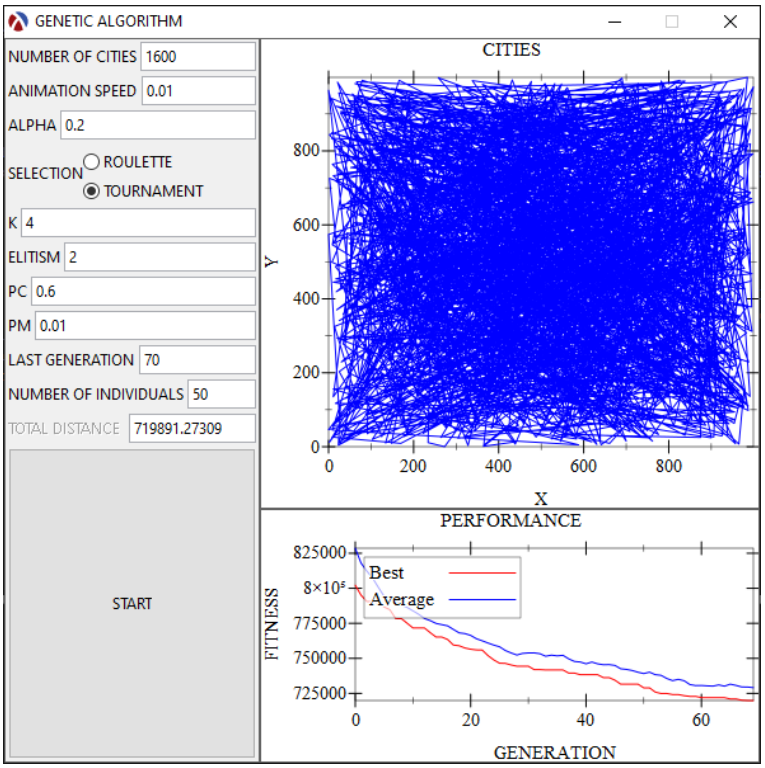

Figura 7: Monitoramento do tempo de execução em ms para 1600 cidades. Relação de 2:3 ainda está sendo mantida, assim como nos testes anteriores. Tempo extremamente rápido, se comparado a outros algoritmos do tipo.

cpu time: 71828 real time: 72037 gc time: 4331

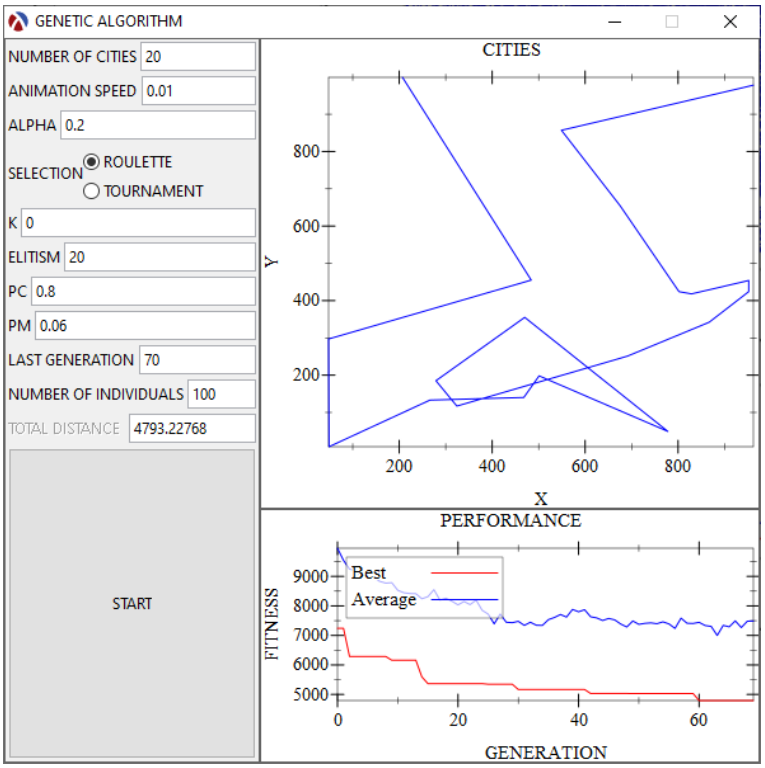

Figura 8: Teste utilizando roleta, elitismo, PC e PM consideravelmente altos. Observa-se que a média se mantém distante da curva do melhor indivíduo

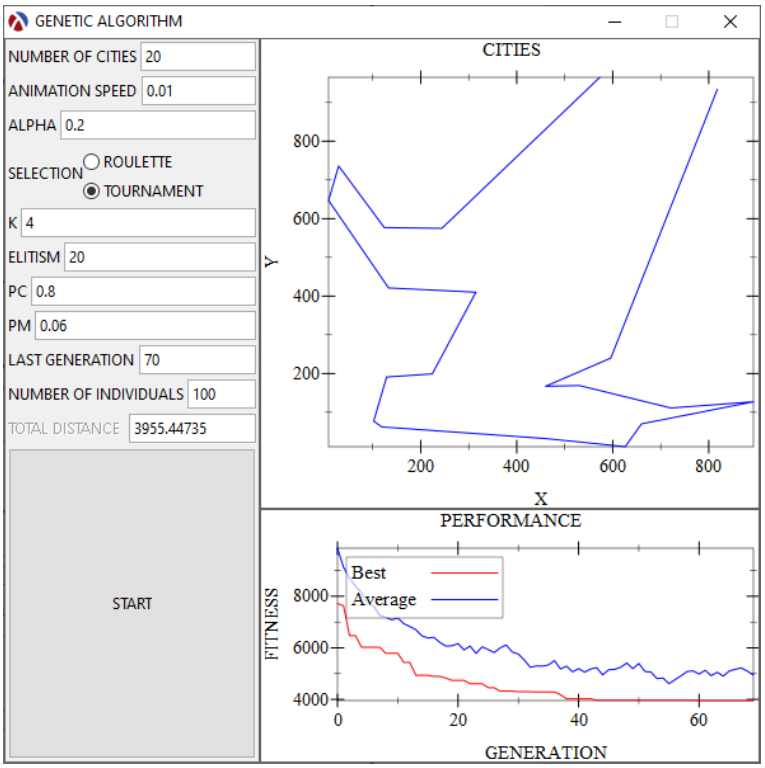

Figura 9: Teste anterior utilizando torneio. Observa-se melhoria considerável na média

5. CONCLUSÃO

No contexto atual, no qual é preciso a otimização de recursos a todo momento, a análise de rotas se torna indispensável para minimizar o custo com transporte, por exemplo.

Apesar de existirem outros algoritmos específicos que desempenham a mesma tarefa, devido ao grande processamento em paralelo, e possibilidade de adaptação dos algoritmos genéticos, os últimos tem-se mostrados extremamente eficientes e eficazes na solução desse tipo de problema.

Além disso, os métodos e técnicas aprendidos durante a execução da prática se mostraram altamente aplicáveis em outros problemas que podem não ser semelhantes à primeira vista, mas que abrem possibilidade de aplicação dos algoritmos genéticos.

No geral, a resolução de um problema utilizando um A.G. tem-se mostrado eficiente e eficaz. Na maioria dos casos, desempenha um papel ainda melhor que os algoritmos específicos, além de apresentarem confiabilidade relativamente alta.

6. REFERÊNCIAS BIBLIOGRÁFICAS

Mallawaarachchi, V. (7 de Julho de 2017). *Introduction to Genetic Algorithms*. Fonte: Towards Data Science: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

Montesanti, J. d. (s.d.). *Seleção natural*. Fonte: InfoEscola: https://www.infoescola.com/evolucao/selecao-natural/

Tomassini, M. (s.d.). A Survey of Genetic Algorithms. *Annual Reviews of Computational Physics, Volume III*.

Yamanaka, P. K. (Agosto de 2017). *ALGORITMOS GENÉTICOS: Fundamentos e Aplicações*.

Zini, É. d., Neto, A. B., & Garbelini, E. (18 de Novembro de 2014). ALGORITMO MULTIOBJETIVO PARA OTIMIZAÇÃO DE PROBLEMAS RESTRITOS APLICADOS A INDÚSTRIA. *Congresso Nacional de Matemática Aplicada à Indústria*.