Relatório sobre a implementação do Algoritmo Genético aplicado ao problema do Caixeiro Viajante e suas conclusões

Luis P. A. Afonso, Vitor A. C. Silva

Departamento de Informática – Universidade Estadual de Maringá (UEM)
Av. Colombo, 5790 – Jd. Universitário CEP 87020-900
Maringá – Pr - Brasil

{ra77429,ra104409}@uem.br

Abstract. *This report describes the implementation and tests of the genetic algorithm with local search applied to the traveling salesman problem presented for the course "Algorithmic Modeling and Optimization" taught by Prof. Dr. Ademir Aparecido Constantino, as a requirement for completion of the subject.*

Resumo. *Este relatório descreve a implementação e testes do algoritmo genético com busca local aplicado ao problema do caixeiro viajante apresentados para a disciplina "Modelagem e otimização algorítmica" ministrada pelo Prof. Dr. Ademir Aparecido Constantino, como requisito para conclusão da matéria.*

1. Introdução

1.1. Visão geral

Este relatório tem por finalidade descrever e apresentar questões relacionadas ao problema do caixeiro viajante, sendo elas: abordar as implementações propostas do algoritmo meta-heurístico genético com busca local para obter uma solução geral viável para casos conhecidos da literatura; sendo ele implementado usando dois operadores de cruzamento com comparações de desempenho entre ambos.

2. Descrição do problema

2.1. O problema do caixeiro viajante

O problema do caixeiro viajante consiste em visitar n cidades diferentes, iniciando e encerrando sua viagem na mesma cidade (neste trabalho o vértice de origem é 1). As cidades podem ser visitadas em qualquer ordem e de cada uma delas pode-se ir diretamente a qualquer outra.

O problema do caixeiro viajante consiste em descobrir a rota que torna mínima a viagem total. Neste estudo as cidades e distâncias entre elas são representadas por um grafo G = (V,A) onde cada vértice $v \in V$ é representado por coordenadas cartesianas (x,y) e cada aresta $c_{i,j} \in A$ é a distância euclidiana entre um vértice $v_i \in V$ e outro vértice $v_j \in V$.

2.2. Implicações sobre obtenção de resultados viáveis

Formulação de Miller–Tucker–Zemlin (1960) do problema clássico do TSP:

$$\min Z = \sum_{i=1}^{n} \sum_{j=1}^{n} CijXij$$

sujeito a $\sum_{i=1}^{n} Xij = 1$, $j = 1, 2,\ldots,n$

$\sum_{j=1}^{n} Xij = 1$, $i = 1, 2,\ldots,n$

$u_i - u_j + nx_{ij} \leq n - 1, \quad 2 \leq i \neq j \leq n,$

$x_{ij} \in \{0,1\}$ $i,j = 1, 2,\ldots,n, \quad i \neq j$

$u_i \in R^+$ $i = 1,2,\ldots,n.$

O problema do caixeiro viajante é de difícil solução, pois nele é possível obter um grande número de soluções possíveis, principalmente quando é utilizado em grandes e diversas distâncias, considerado sua complexidade NP–Difícil (BELFIORE; FÁVERO, 2013) então, para dimensões elevadas, a resolução por algoritmos exatos fica impeditivo em termos de tempo computacional. Portanto, a abordagem de solução heurística ou meta-heurística é imperativa para a obtenção de resultados aceitavelmente bons.

2.3. O Algoritmo Genético

O algoritmo genético é um método de otimização inspirado no princípio Darwiniano da evolução das espécies e na genética. É um algoritmo meta-heurístico que mantém uma população constituída de cromossomos e utiliza operadores de seleção, cruzamento, mutação e neste trabalho também foi incluída uma busca local na vizinhança do cromossomo que será discutida posteriormente.

Neste trabalho foram implementados os métodos de seleção por torneio e elitismo, atualização da população por *stead stated*. Foram implementados os operadores de cruzamento *Cycle crossover* e *Position based crossover* bem como é feita uma comparação entre ambos por meio de gráficos que serão discutidos adiante.

3. Operadores de cruzamento implementados

3.1. Cycle crossover

Neste algoritmo os genes (cidades para o problema do caixeiro viajante) são subdivididos em ciclos, e os filhos são gerados pela seleção destes ciclos nos cromossomos pais. Desta forma, o CX garante que os elementos presentes nos pais serão herdados pelos filhos mantendo a ordem de ocorrência.

3.2. Position based crossover

O algoritmo seleciona vários genes (cidades) de forma aleatória na estrutura do cromossomo pai e copia os genes presentes nestas posições para o cromossomo filho preservando a ordem de precedência no outro cromossomo pai.

4. Estudo da influência dos parâmetros utilizados

4.1. População

Foram testados os tamanhos populacionais 10, 20 e 25 e 100 e 300 e os experimentos demonstram que populações menores em relação ao tamanho do grafo possuem uma convergência um pouco melhor em relação aos casos onde o inverso ocorre, como mostram os gráficos para o caso pr1002.tsp com o tamanho da população em 300 e 25 respectivamente:

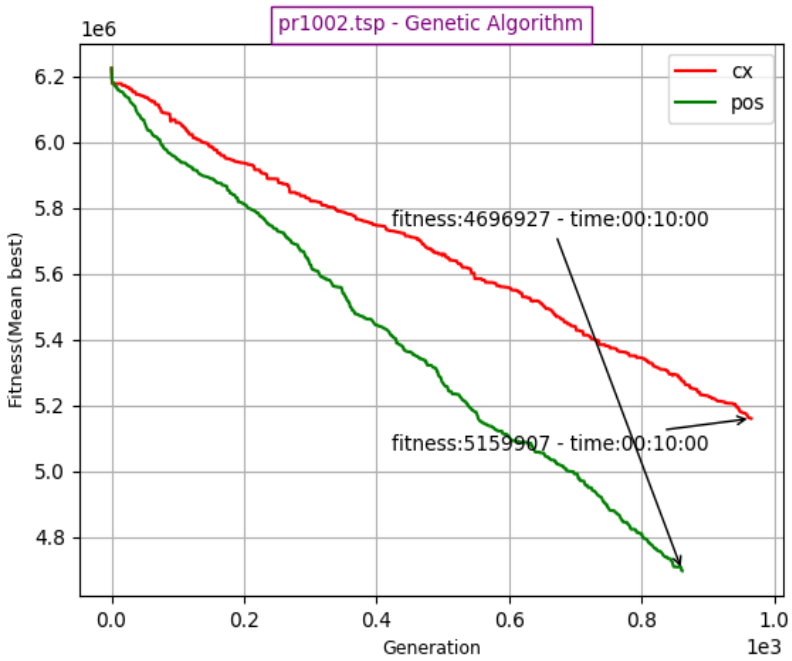

Figura 1 – caso pr1002.tsp tamanho da população:300

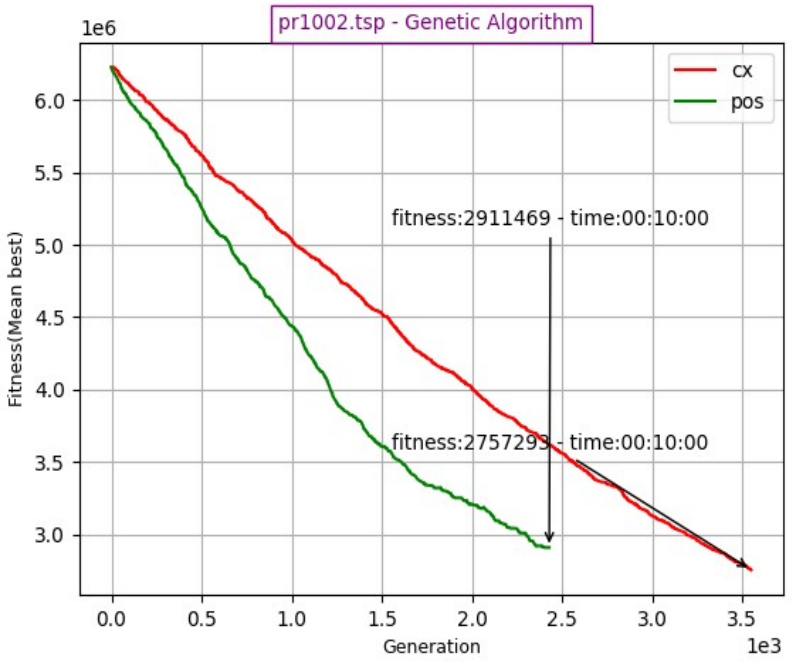

Figura 1 – caso pr1002.tsp tamanho da população:25

Nota-se que populações maiores ocasionaram em perda de eficiência por uma maior frequência na avaliação da função de aptidão de uma forma “cega”. Nota-se que populações maiores geram maior variabilidade (ainda que baixa) entre os valores de *fitness* entre os cromossomos. O valor pode ser passado por parâmetro para o programa

com a flag -pop <int>.

4.2. Mutação

Nos algoritmos genéticos a taxa de mutação dos cromossomos é um valor geralmente baixo, portanto para este trabalho foram considerados os valores(%) 3, 4, 5, 6 e 10 dos quais mostraram oscilações maiores no gráfico de forma proporcional ao tamanho do valor. Fato que comprova que valores maiores de mutação geram maior variabilidade na população o que diminui o desempenho em alguns casos, pois devido a maior variabilidade na população o algoritmo aceita casos piores com maior frequência acarretando um maior "atraso" em relação a execução com valores de mutação menores. Podemos comprovar pelos gráficos a seguir com valores de mutação em 4% e 6% respectivamente:

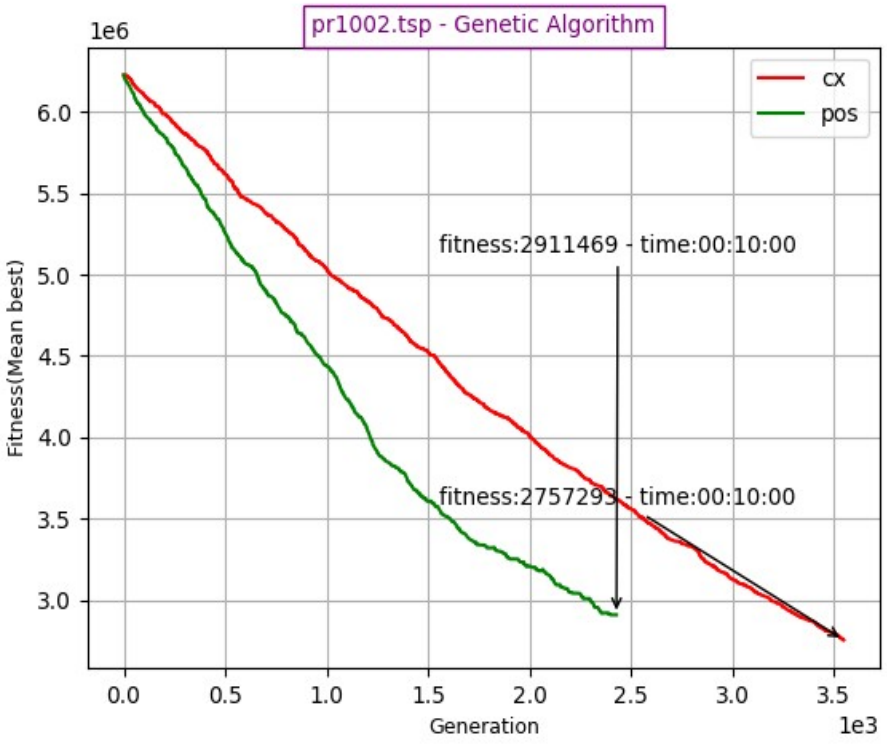

Figura 1 – caso pr1002.tsp com 4% de mutação.

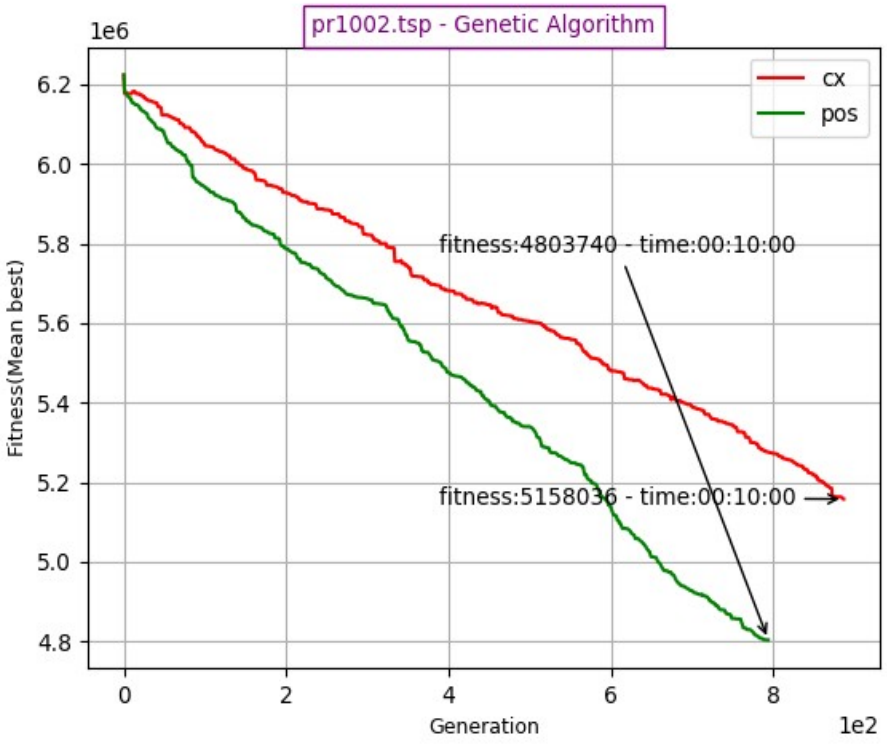

Figura 2 – caso pr1002.tsp com 6% de mutação.

Para fim dos testes finais foi escolhido o valor mediano dentre todos do teste, ou seja 5% de mutação na população. O valor pode ser alterado via parâmetro com a flag -mut <int>.

4.3. Semente aleatória

Foram testadas os valores para a semente aleatória 1, 2, 5 e 10 e possui relevância para o algoritmo para a geração controlada de números pseudo aleatórios, de valor fundamental para a execução do algoritmo e maior facilidade na depuração do código.

Além disso, a semente aleatória está presente no algoritmo de forma que possibilite gerar os mesmos valores aleatórios de entrada para o programa executar de forma sequencial o algoritmo genético com os operadores de cruzamento CX e POS para comparação posterior entre ambos. Pode ser passado por parâmetro com a flag -s <int>.

4.4. Máximo de iterações

Para este trabalho foi definido como padrão o valor $2x10^{6,}$ um valor alto de forma proposital de forma a deixar a execução dos testes apenas por limitação tempo, mas é possível modificá-lo via terminal com a flag -max_it <int>.

4.5. Tempo de execução máximo

Os valores para o tempo de execução variam de caso para caso sendo eles 10 minutos para o caso pr1002.tsp, 2 horas para o caso fnl4461.tsp, 5 horas para o caso pla7397.tsp,

6 horas para o caso brd14051.tsp, 6 horas para o caso d15112.tsp, 4 horas para o caso d18512,tsp e 5 horas para o caso pla33810.tsp.

5. Busca local

O mecanismo de busca local utilizado neste trabalho consiste em buscar melhorias na vizinhança da solução com o algoritmo 2-opt com estratégia *first improvement* implementado no trabalho anterior.

Os experimentos mostram que a busca local torna-se o maior custo do processo na medida que se aproxima de soluções melhores, pois o fitness médio da população melhora conforme avançam as gerações.

6. Casos de testes usados para validação

Os casos de testes requisitados para o relatório foram:

pr1002 (1002-city problem (Padberg/Rinaldi))

fnl4461 (Die 5 neuen Laender Deutschlands (Ex-DDR) (Bachem/Wottawa))

pla7397 (Programmed logic array (Johnson))

brd14051 (BR Deutschland in den Grenzen von 1989 (Bachem/Wottawa))

d15112 (Deutschland-Problem (A.Rohe))

d18512 (Bundesrepublik Deutschland (mit Ex-DDR) (Bachem/Wottawa))

pla33810 (Programmed logic array (Johnson))

pla85900 (Programmed logic array (Johnson))

Os resultados obtidos a partir de cada teste, assim como a execução ou não de cada um será discutido posteriormente.

6.1. Especificações das máquinas usadas para os testes

Por conta da limitação de tempo a execução dos testes foram realizados em duas máquinas: a Máquina 1 com 8 núcleos sempre com 6 testes sendo executados simultaneamente e Máquina 2 com 8 núcleos executando 2 testes simultaneamente.

Máquina 1 - Intel(R) Core(TM) i7-3770 CPU @3.40GHz, 8 Threads

Máquina 2 - Intel(R) Core(TM) i7-6500 CPU @2.40GHz , 4 Threads

7. Resultados obtidos por experimentações

7.1. Considerações sobre a experimentação

As máquinas que foram usadas para a testagem não eram dedicadas para os experimentos. Iterações com pouca relevância então algoritmo limitado por tempo; Contudo,durante as execuções, foi dada prioridade para o processo do teste, assim como o uso da máquina foi reduzido durante o período dos testes, de forma a buscar o desempenho ideal para tais execuções.

Apesar da possibilidade de limitar via parâmetro a execução do algoritmo em termos de número de iterações ou tempo de execução, os experimentos para este trabalho foram executados com a definição padrão do argumento do máximo de iterações elevado propositalmente, para haver limitação apenas por tempo, que variou de acordo com o caso.

7.2. Gráficos comparativos entre Cycle crossover e Position based crossover

7.2.1 Caso PR1002.tsp

Comentado anteriormente.

7.2.2 Caso FNL4461.tsp

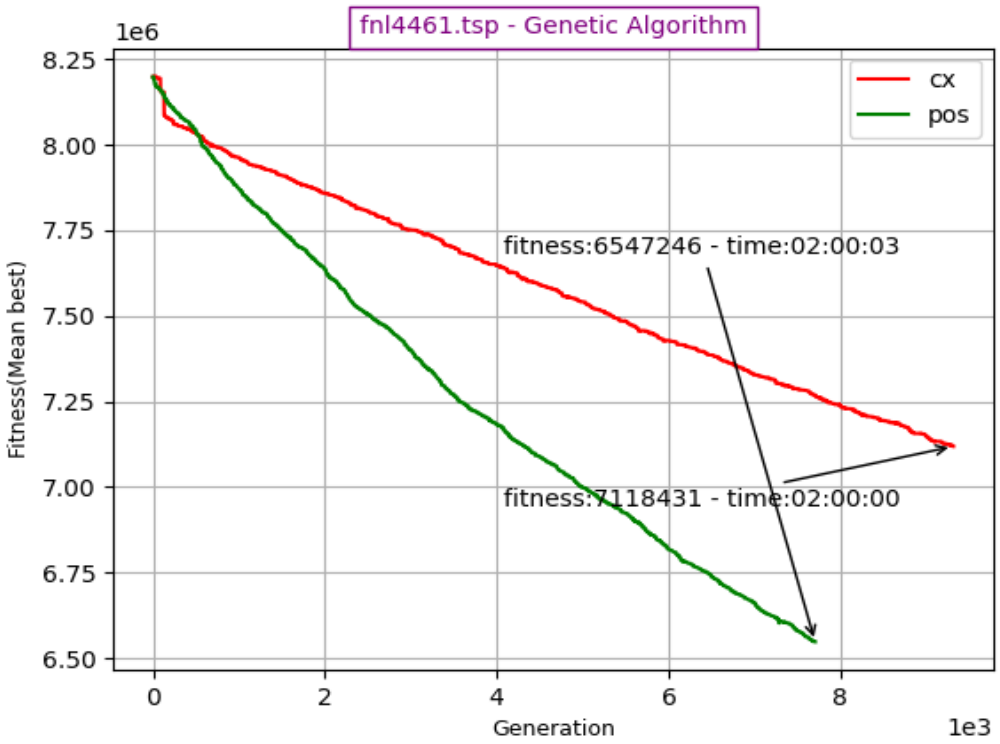

Figura 2 – caso fnl4461 – máquina 1

7.2.3 Caso PLA7397.tsp

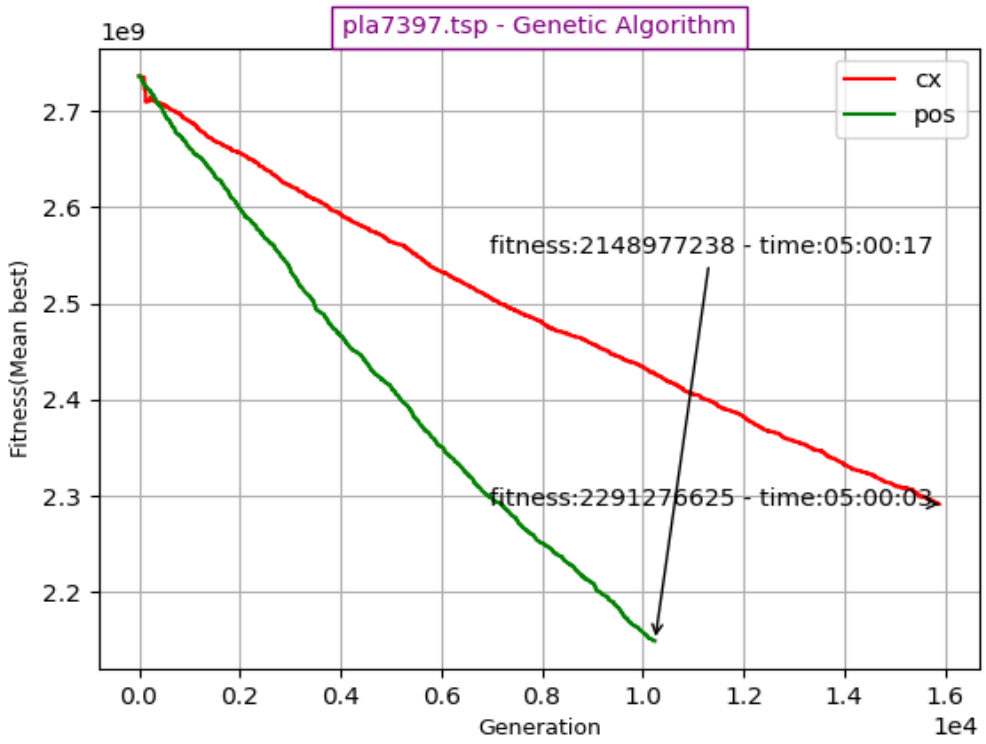

Figura 3 – caso pla7397 – máquina 1

7.2.4 Caso BRD14051.tsp

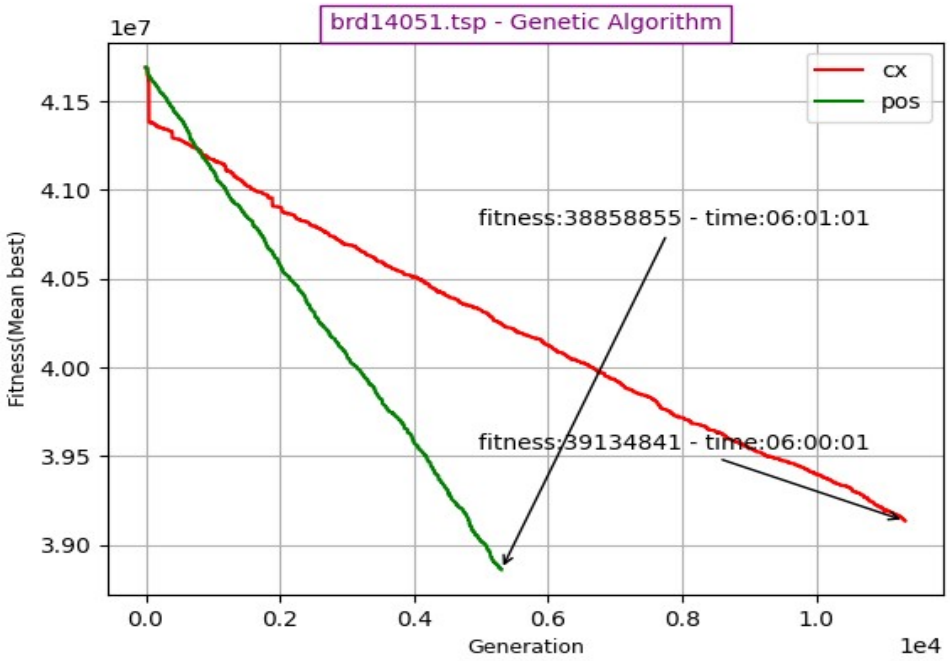

Figura 4 – caso brd14051 – máquina 1

7.2.5 Caso D15112.tsp

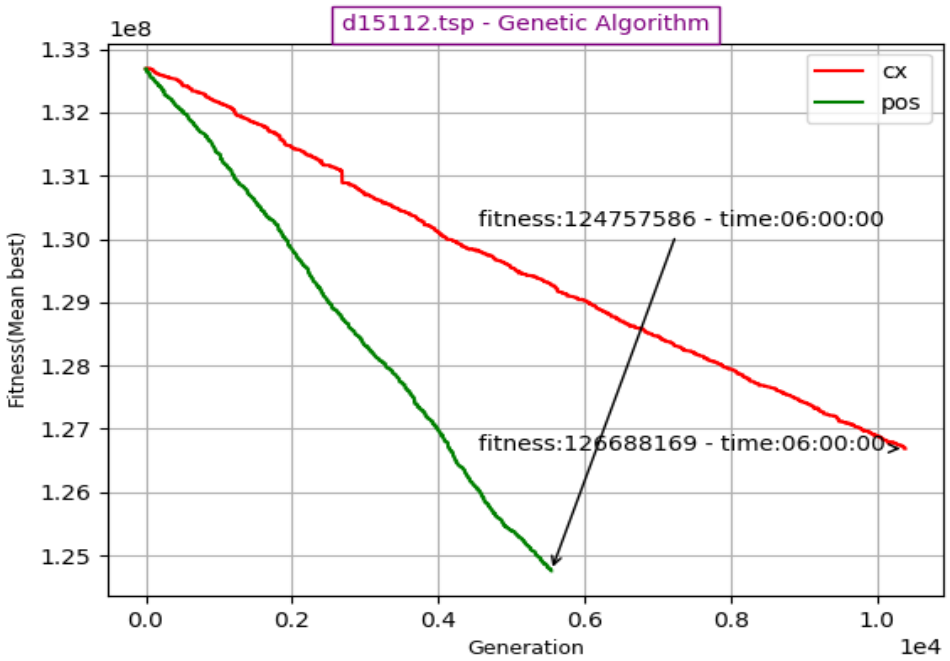

Figura 5 – caso d15112 – máquina 2

7.2.6 Caso D18512.tsp

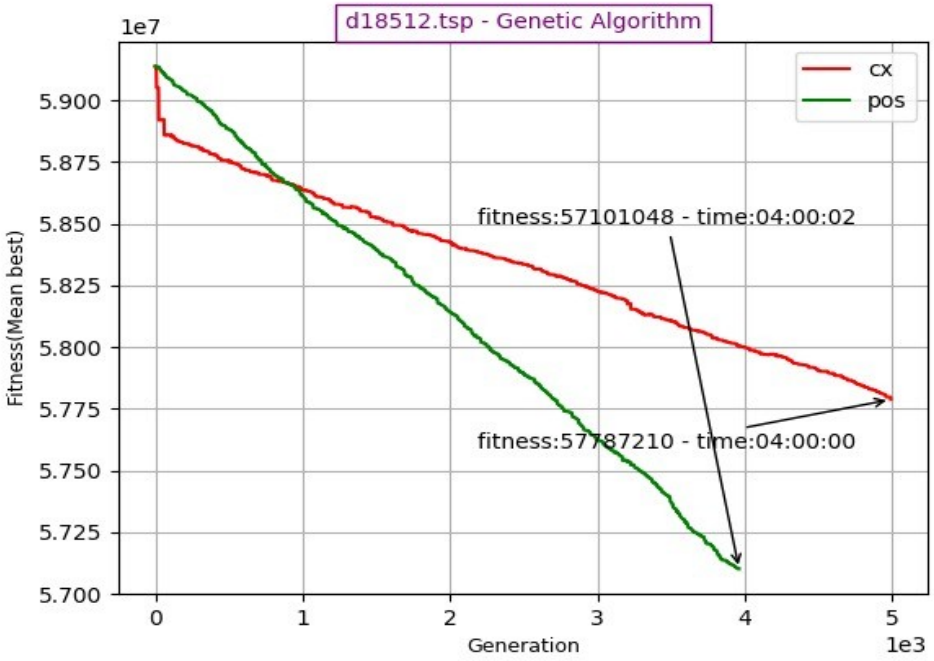

Figura 6 – caso d18512 – máquina 2

7.2.6 Caso PLA33810

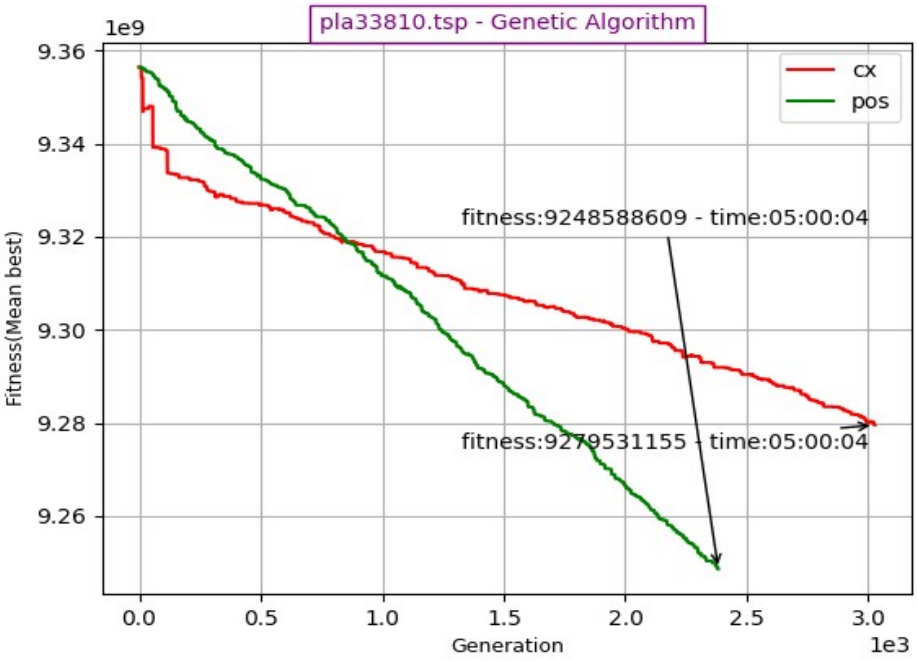

Figura 7 – caso pla33810 – máquina 1

7.3. Tabela comparativa entre os algoritmos

CASO	MS	CX	GAP_CX%	POS	GAP_POS%
pr1002	259045	2757293	964,4	2911469	1023,9
fnl4461	182566	7118431	3799,1	6547246	3486,2
pla7397	81438	2291276625	2813422,7	2148977238	2638689,3
brd14051	46985	39134191	83190,8	38858855	82604,8
d15112	1573084	126188169	268471,1	124757586	7830,7
d18512	645238	57787210	8855,9	57101048	8749,6
pla33810	66048945	9279531155	13949,4	9248588609	13902,6

7.4. Tabelas comparativas entre valores de GAP desse trabalho com os do anterior

Afim de apresentarmos uma perspectiva entre as duas iterações do trabalho, comparemos, usando como representante do trabalho anterior o algoritmo que implementa vizinho mais próximo para a fase construtiva e 2-opt limitado para a fase melhorativa.

CASO	GAP_CX%	GAP_VP_2OPT %	| GAP_CX – GAP_VP_2OPT |
pr1002	964,4	19.10	945.3
fnl4461	3799,1	23.65	3775.45
pla7397	2813422,7	34305.84	2779116.86
brd14051	83190,8	1134.31	82056.49
d15112	268471,1	23.63	268447.47
d18512	8855,9	21.85	8834.05
pla33810	13949,4	-	-

CASO	GAP_POS%	GAP_VP_2OPT %	| GAP_POS – GAP_VP_2OPT |
pr1002	1023,9	19.10	1004.8
fnl4461	3486,2	23.65	3462.55
pla7397	2638689,3	34305.84	2604383.46
brd14051	82604,8	1134.31	81470.49
d15112	7830,7	23.63	7807.07
d18512	8749,6	21.85	8727.75
pla33810	13902,6	-	-

8. Conclusões finais

Dadas os resultados obtidos pela fase de experimentação e o embasamento teórico, também discutido anteriormente, de que dado certos problemas a serem resolvidos computacionalmente, não é possível obter a melhor solução em tempo desejável/possível. Portanto, são propostas heurísticas, que aplicadas a esses problemas, produzem, em questão de qualidade, soluções mistas.

Apesar dos resultados serem obtidos serem muito longe dos ótimos da literatura, podemos notar com clareza que em nenhum caso houve estagnação dos algoritmos a ponto do gráfico permanecer paralelo ao eixo das abcissas (gerações), o que se deve ao fato do algoritmo evoluir a qualidade da população de forma constante mas muito lenta. O contrário ocorre nos algoritmos heurísticos do trabalho anterior, onde é notável muitos pontos de estagnação em quase todos os casos de teste.

Para concluir, também é evidente o desempenho superior do operador POS em relação ao operador CX na maioria dos casos.

Referencias

“Operadores de cruzamento”, Arquivo de material didático disponibilizado pelo professor da matéria de Modelagem e Optimização Algorítmica, no seguinte link restrito: https://moodlep.uem.br/pluginfile.php/197470/mod_resource/content/1/OperadoresCruzamento.pdf.

Marco Aurélio Cavalcanti Pacheco. "Algoritmos genéticos: princípios e aplicações”. Laboratório de inteligência Computacional Aplicada, Departamento de Engenharia Elétrica, PUC-RIO. https://www.inf.ufsc.br/~mauro.roisenberg/ine5377/Cursos-ICA/CE-intro_apost.pdf.