Relatório de Modelagem e Otimização Algorítmica
Problema Caixeiro Viajante

Matheus Augusto – 107115, Gabriel de Melo Osório – 107862

Introdução

A matéria de modelagem e otimização algorítmica busca resolver problemas de otimização da vida real por meio de um modelo matemático, com a solução ótima para o problema. Nem sempre o tempo de execução desses algoritmos é viável, obrigando o uso de heurísticas, que nada mais são que premissas adotadas que não garantem o resultado ótimo, mas sim, um resultado muito próximo do ótimo com um tempo aceitável. Dentro do campo dos algoritmos heurísticos surgiu a meta-heurística, que basicamente são modelos gerais que servem como guia para a construção de algoritmos heurísticos. Isto posto, o presente relatório busca comparar os resultados da implementação de um algoritmo genético sem busca local com um algoritmo genético utilizando busca local, utilizando duas opções de cruzamento para resolver o problema do caixeiro viajante. O algoritmo genético é um algoritmo meta-heurístico e foi batizado desta forma pois sua origem vem de uma analogia com a evolução da biologia, possibilitando explorar uma maior variabilidade de soluções do que a busca local e, consequentemente, aumentando as chances de respostas melhores.

Descrição do Problema

O problema tratado por este relatório trata-se do problema de otimização do caixeiro viajante. O problema consiste em um caixeiro viajante, que deseja visitar *n* cidades, apenas uma vez, e retornar para sua cidade de origem com o menor custo possível.

Descrição do Algoritmo

Um algoritmo genético pode ser utilizado para diversas funções, neste caso, ele está sendo usado como uma otimização para a busca local no caixeiro viajante. Foi definido como um cromossomo uma solução para o problema do caixeiro viajante contendo como gene a lista de vértices, sua aptidão, que representa o custo final, e uma lista de conexões de vértices adjacentes, que armazena apenas os índices do vértice. A população inicial foi obtida por meio do algoritmo construtivo do trabalho anterior e utilizaram-se como critérios de

parada um tempo máximo de execução, um limite de iterações e uma verificação se os últimos dez filhos gerados estagnaram, os valores destes critérios foram definidos arbitrariamente.

Enquanto estes critérios não são violados, o algoritmo seleciona aleatoriamente dois pais que irão realizar o cruzamento. Uma vez selecionados, ele utiliza um dos operadores de cruzamento do código. Caso seja o cycle crossover, ele define o filho 1 como cópia do pai 2. Após isto, a primeira posição no filho 1 é substituída pelo elemento de mesmo índice do pai 1, alterando seu conteúdo. Como a primeira posição foi alterada, existem elementos repetidos neste filho e, portanto, procura-se a posição que este elemento está no pai 2. Ao achá-la, seu índice é usado para procurar um elemento em pai 1 e colocá-lo no mesmo índice do filho 1, até que este processo retorne à posição inicial do filho 1. Quando ele volta para a posição inicial significa que o filho está pronto e, portanto, o processo se repete para o segundo filho, até que o mesmo também esteja pronto.

Caso o operador de cruzamento seja o Order Based Crossover, inicialmente são selecionadas posições no pai 1 e posições no pai 2 que serão fixadas. Uma vez selecionadas as posições do pai 1 e do pai 2, os elementos respectivos às posições de pai 1 são procurados no pai 2 e, quando encontrados, suas posições são travadas no filho 1. Deste modo, o filho 1 recebe em suas posições não travadas todos os genes do pai 1 e, para as posições travadas, os valores são preenchidos com o gene de mesmo índice do pai 2. O mesmo processo ocorre com o filho 2, invertendo as posições entre pai 1 e pai 2.

Com os dois filhos criados, a etapa de cruzamento é finalizada e o algoritmo entra na fase de mutação que possui uma taxa de 15% de chance que, caso seja aceita, recebe duas conexões diferentes e altera o vértice de destino da primeira com a da segunda. O mesmo se aplica para a segunda conexão.

Uma vez que o processo de mutação tenha sido tratado, o algoritmo avalia se a população precisa ser atualizada com os novos filhos e, caso algum dos filhos possua uma aptidão menor que a população atual, o cromossomo com o maior valor nominal é removido, visto que o algoritmo do caixeiro viajante procura o menor custo. Após isso, o algoritmo atualiza as variáveis da interação, tempo decorrido e estagnação para verificar se algum dos critérios de parada foi violado.

Resultados observados na utilização do CX

Casos de Teste	Operação de Cruzamento	CX		Melhor caso conhecido
	Busca Local	*nda*	*Utilizando*	
Pr1002	***Custo***	312.237,27	269.669,90	259045
	Tempo (s)	2,13	572,04	
	Gap%	20,53	4,10	
Fnl4461	***Custo***	225.248,00	194.524,48	182566
	Tempo (s)	36,15	3.625,18	
	Gap%	23,38	6,55	
Pla7397	***Custo***	27.694.589,91	24.270.672,59	81438
	Tempo (s)	364,62	3.641,97	
	Gap%	33.906,96	29.702,64	
Brd14051	***Custo***	572.114,53	496.530,64	46985
	Tempo (s)	50,19	4.200,46	
	Gap%	1.117,65	956,79	
D15112	***Custo***	1.937.877,02	1.937.877,02	1573084
	Tempo (s)	1.755,80	3.656,69	
	Gap%	23,19	23,19	
D18512	***Custo***	794.121,33	783.892,71	645238
	Tempo (s)	71,84	5.365,02	
	Gap%	23,07	21,49	
Pla33810	***Custo***	77.327.460,49	77.327.460,49	66048945
	Tempo (s)	291,64	5.578,00	
	Gap%	17,08	17,08	
Pla85900	***Custo***	163.504.086,39	150.535.568,52	142382641
	Tempo (s)	991,42	20.973,67	
	Gap%	14,83	5,73	

Resultados observados na utilização do OX2

Casos de Teste	Operação de Cruzamento	OX2		Melhor caso conhecido
	Busca Local	*nda*	*Utilizando*	
Pr1002	***Custo***	312.237,27	266.629,90	259045
	Tempo (s)	17,04	2.686,64	
	Gap%	20,53	2,93	
Fnl4461	***Custo***	225.248,00	198.253,94	182566
	Tempo (s)	171,98	3.627,54	
	Gap%	23,38	8,59	
Pla7397	***Custo***	27.694.589,91	25.803.133,06	81438
	Tempo (s)	441,19	3.695,74	
	Gap%	33.906,96	31.584,39	
Brd14051	***Custo***	572.114,53	499.730,87	46985
	Tempo (s)	1.483,92	3.787,09	
	Gap%	1.117,65	963,60	
D15112	***Custo***	1.937.877,02	1.937.877,02	1573084
	Tempo (s)	3.015,78	4.033,75	
	Gap%	23,19	23,19	
D18512	***Custo***	794.121,33	773.826,90	645238
	Tempo (s)	2.309,18	3.826,78	
	Gap%	23,07	19,93	
Pla33810	***Custo***	77.327.460,49	72.511.349,47	66048945
	Tempo (s)	3.600,63	5.949,51	
	Gap%	17,08	9,78	
Pla85900	***Custo***	163.504.086,39	163.504.086,39	142382641
	Tempo (s)	3.628,16	20.144,00	
	Gap%	14,83	14,83	

Comparação dos melhores resultados entre Trabalho 1 e 2

Casos de Teste	Melhor Resultado Trabalho 1	Melhor Resultado Trabalho 2	Best
Pr1002	272.078	266.629,90	259.045
Fnl4461	196.881,99	194.524,48	182.566
Pla7397	24.907.408,07	24.270.672,59	81.438
Brd14051	499.730,86	496.530,64	46.985
D15112	1.698.892,19	1.937.877,02	1.573.084
D18512	690.572,32	773.826,90	645.238
Pla33810	70.950.017,20	72.511.349,47	66.048.945
Pla85900	150.476.292,41	150.535.568,52	142.382.641

Comparação entre os tempos dos melhores resultados do Trabalho 1 e 2

Casos de Teste	Tempo do Melhor Resultado T1	Tempo do Melhor Resultado T2	Best
Pr1002	1,314	2.686,64	259.045
Fnl4461	27,68521	3.625,18	182.566

Pla7397	69,01628	3.641,97	81.438
Brd14051	274,11505	4.200,46	46.985
D15112	404,72601	1.755,80	1.573.084
D18512	380,72122	3.826,78	645.238
Pla33810	1242,47572	5.949,51	66.048.945
Pla85900	8087,58881	20.973,67	142.382.641

Gráficos dos resultados a cada iteração

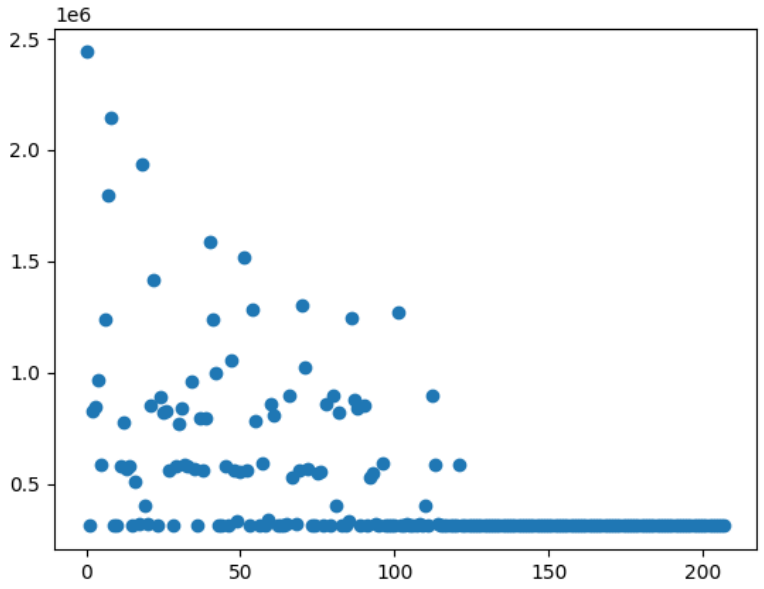

Figura 1 - pr1002 com função de cruzamento CX, sem busca local

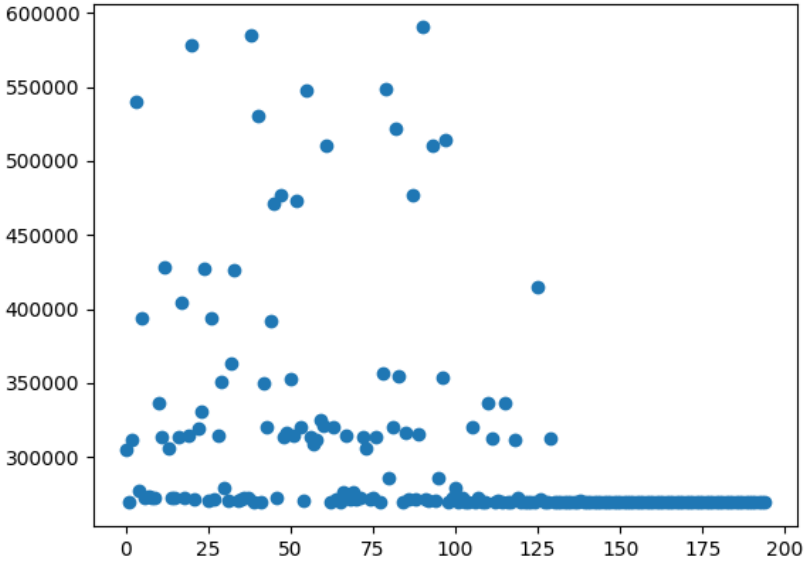

Figura 2 - pr1002 com função de cruzamento CX e busca local

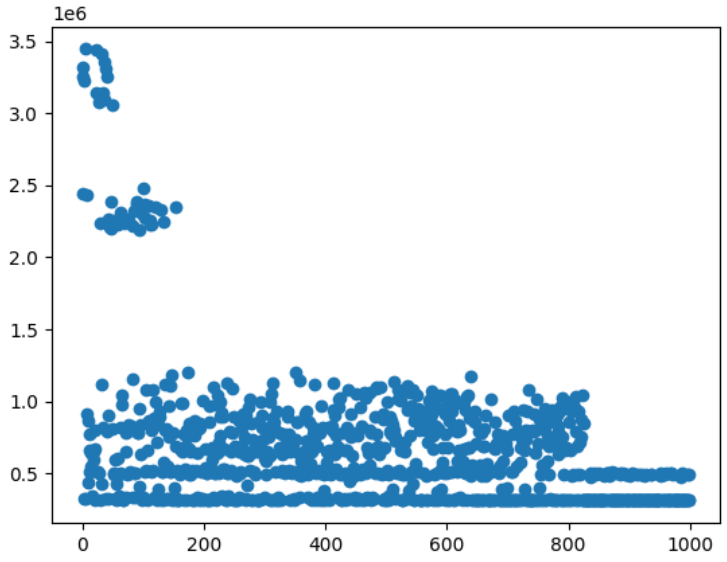

Figura 3 - pr1002 com função de cruzamento OX2, sem busca local

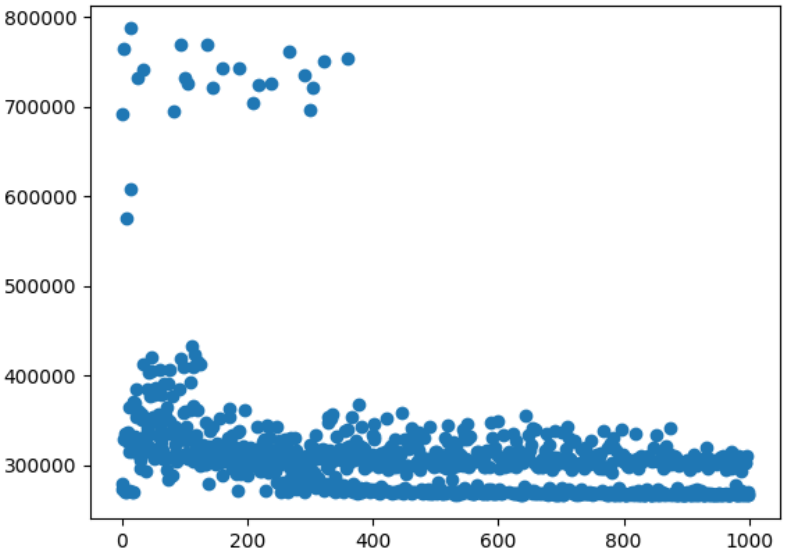

Figura 4 - pr1002 com função de cruzamento OX2 e busca local

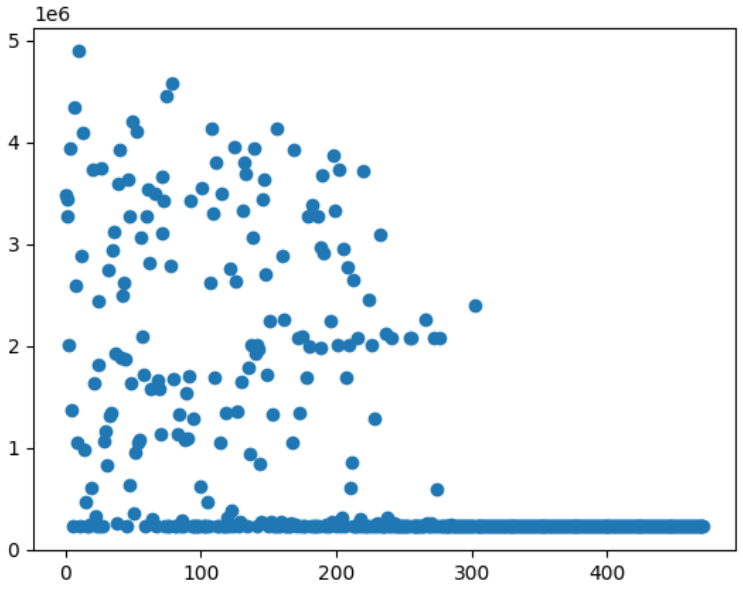

Figura 5 - fnl4461 com função de cruzamento CX, sem busca local

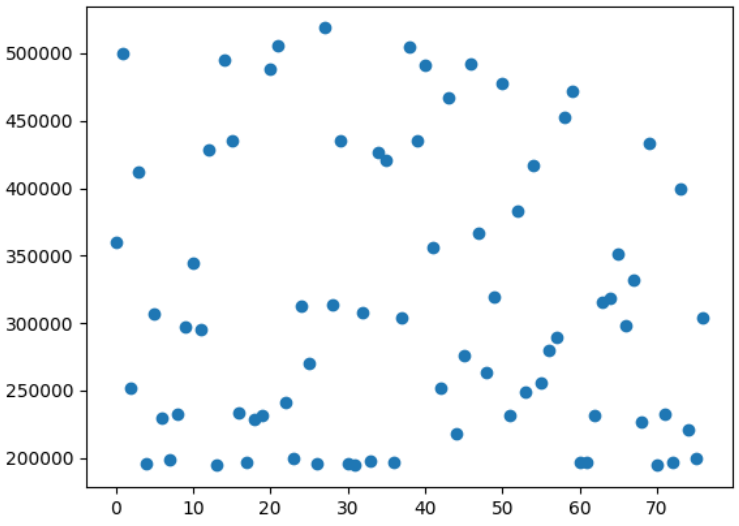

Figura 6 - fnl4461 com função de cruzamento CX e busca local

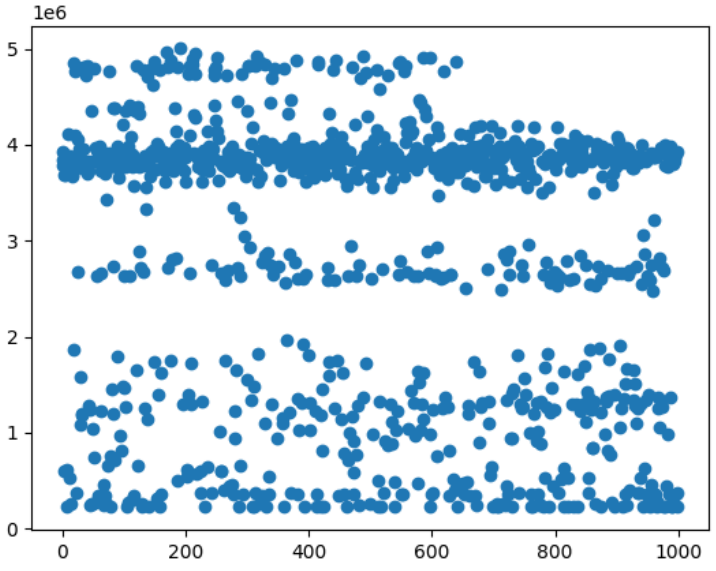

Figura 7 - fnl4461 com função de cruzamento OX2, sem busca local

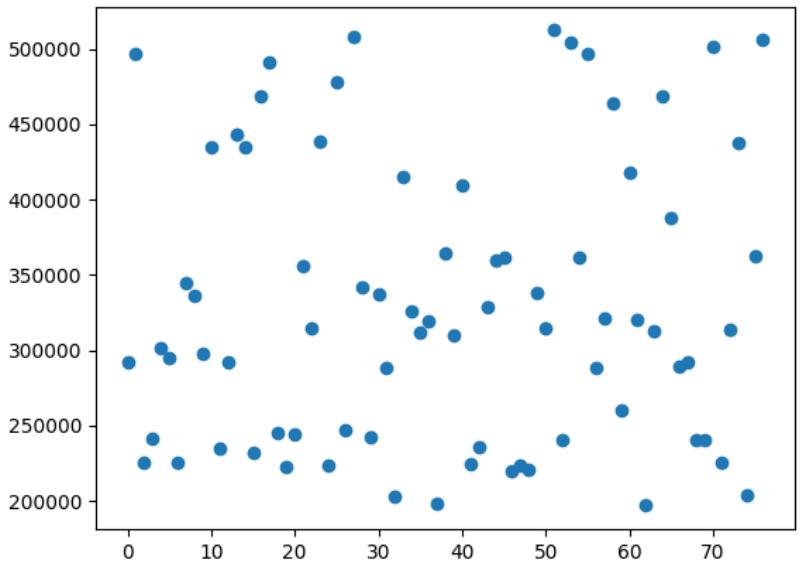

Figura 8 - fnl4461 com função de cruzamento OX2 e busca local

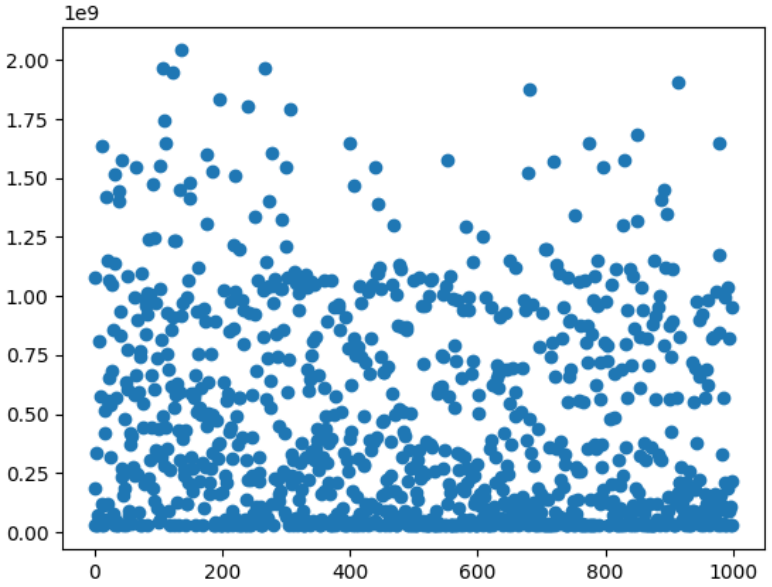

Figura 9 - pla7397 com função de cruzamento CX, sem busca local

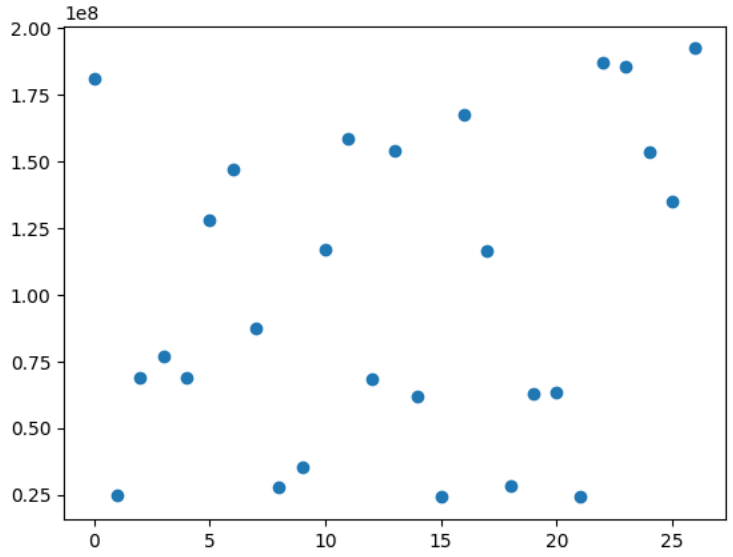

Figura 10 - pla7397 com função de cruzamento CX e busca local

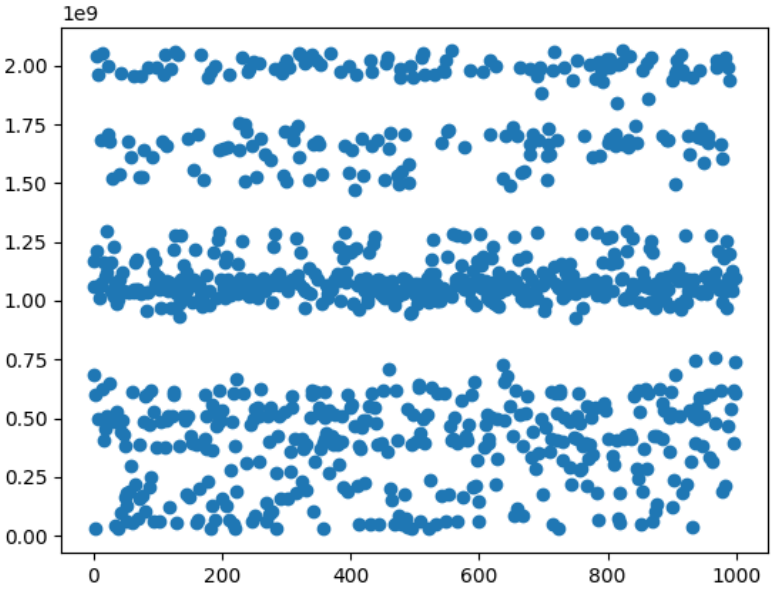

Figura 11 - pla7397 com função de cruzamento OX2, sem busca local

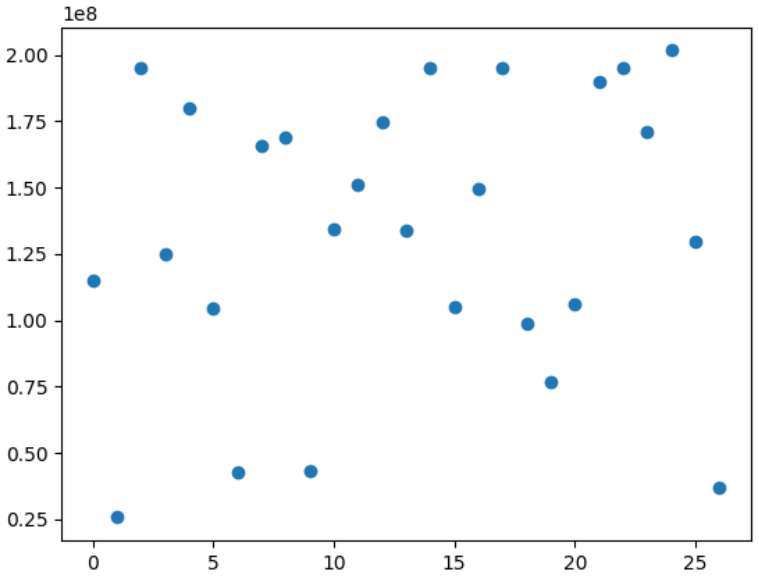

Figura 12 - pla7397 com função de cruzamento OX2 e busca local

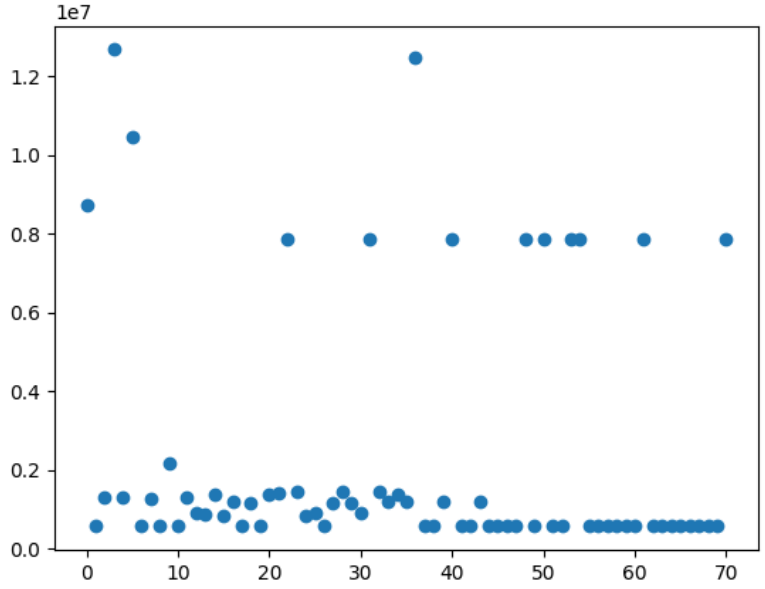

Figura 13 - brd14051 com função de cruzamento CX, sem busca local

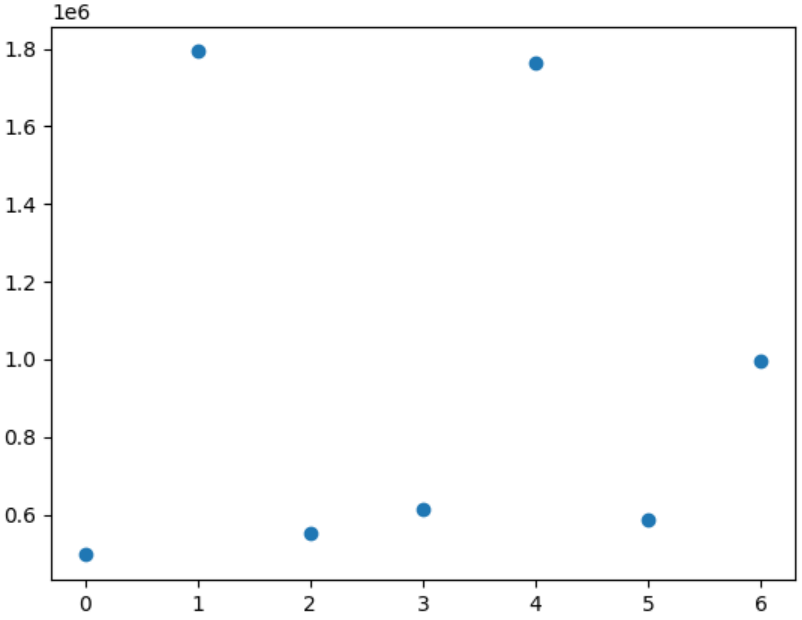

Figura 14 - brd14051 com função de cruzamento CX e busca local

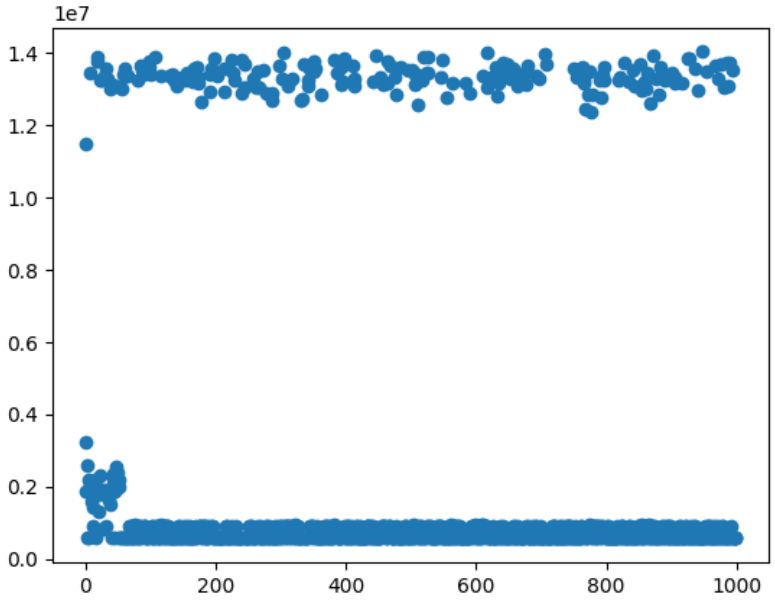

Figura 15 - brd14051 com função de cruzamento OX2, sem busca local

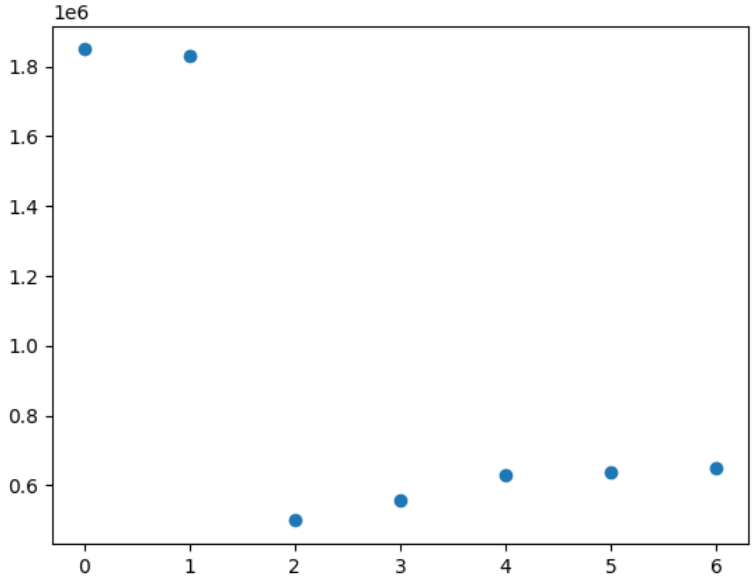

Figura 16 - brd14051 com função de cruzamento OX2 e busca local

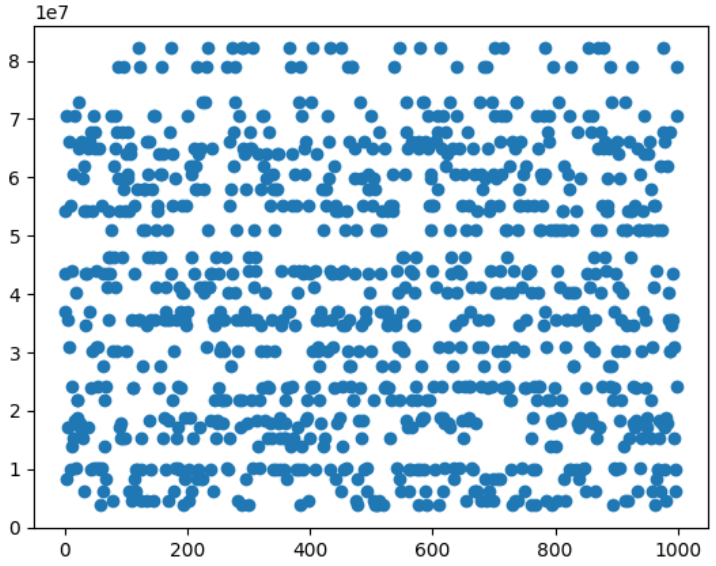

Figura 17 - d15112 com função de cruzamento CX, sem busca local

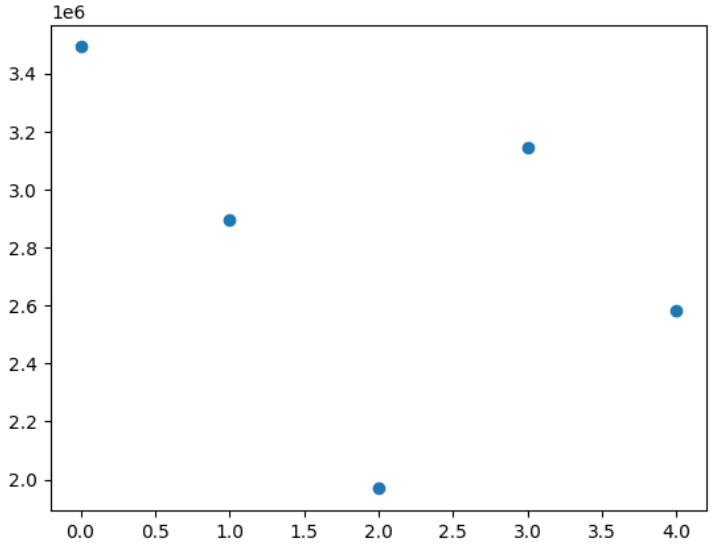

Figura 18 - d15112 com função de cruzamento CX e busca local

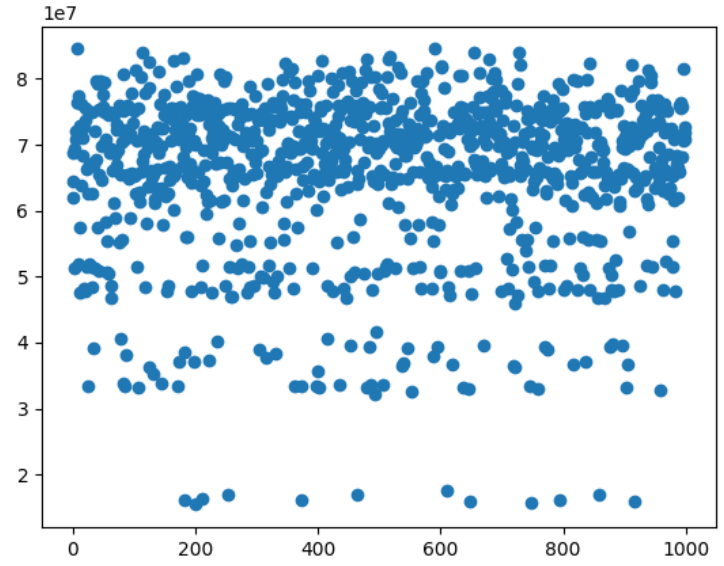

Figura 19 - d15112 com função de cruzamento OX2, sem busca local

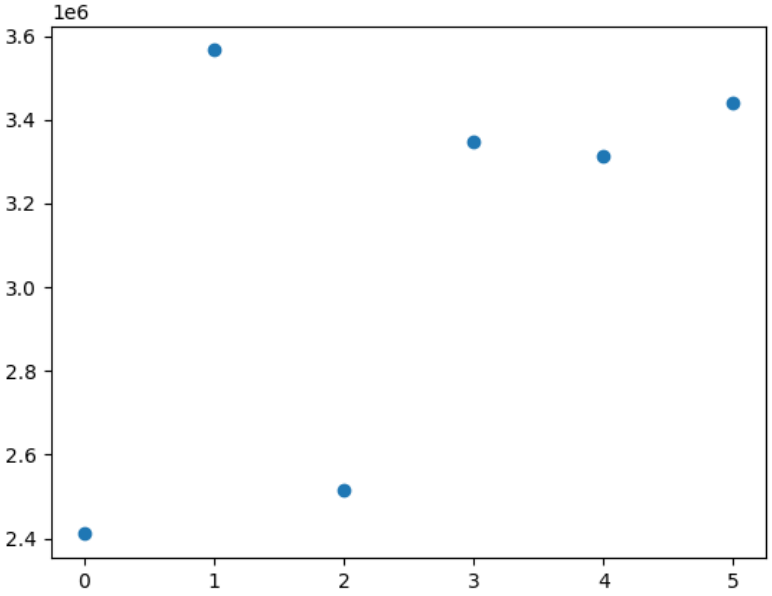

Figura 20 - d15112 com função de cruzamento OX2 e busca local

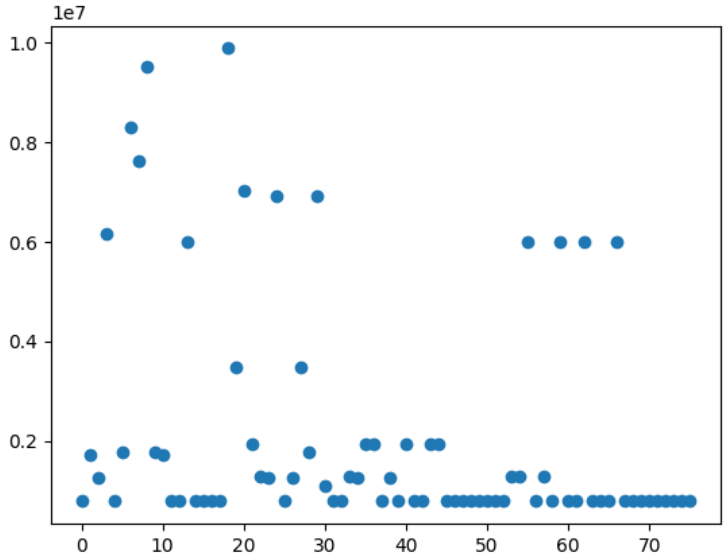

Figura 21 - d18512 com função de cruzamento CX, sem busca local

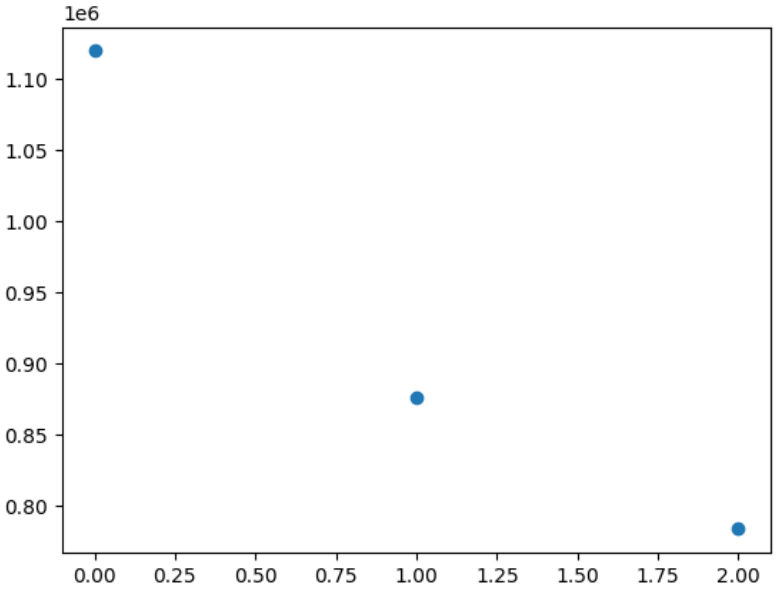

Figura 22 - d18512 com função de cruzamento CX e busca local

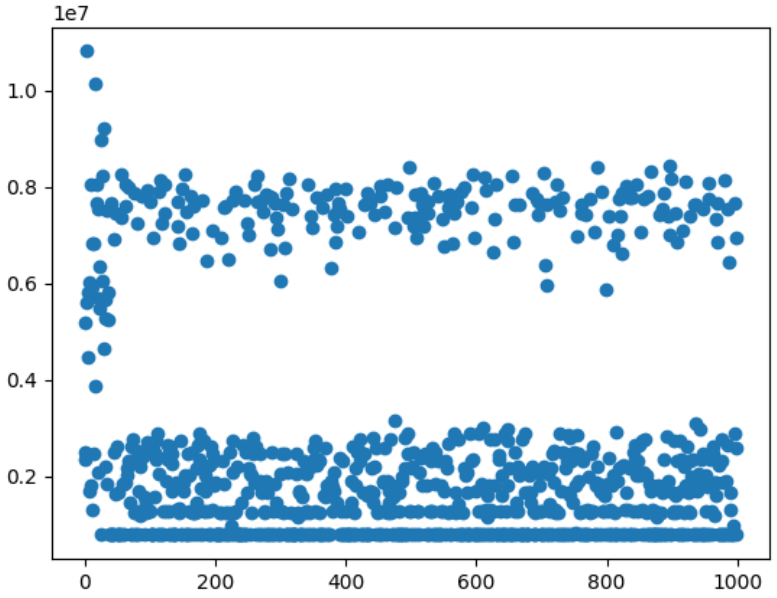

Figura 23 - d18512 com função de cruzamento OX2, sem busca local

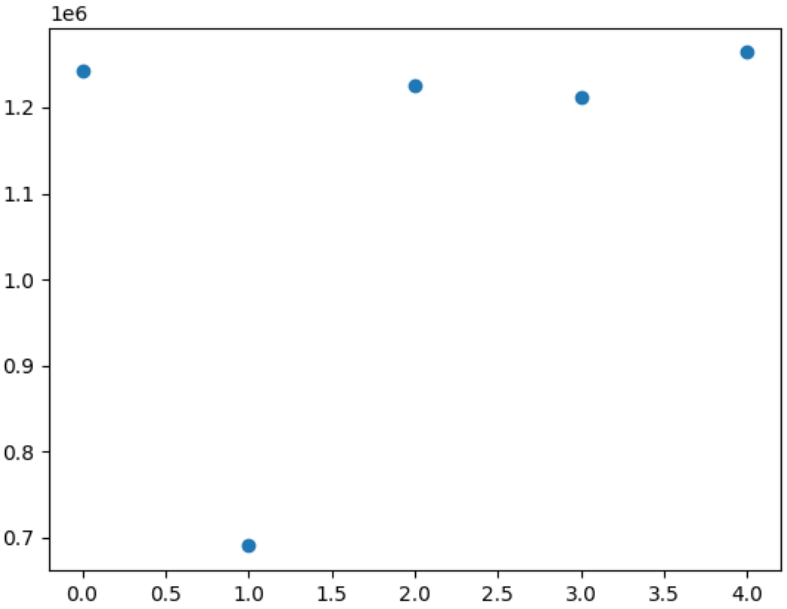

Figura 24 - d18512 com função de cruzamento OX2 e busca local

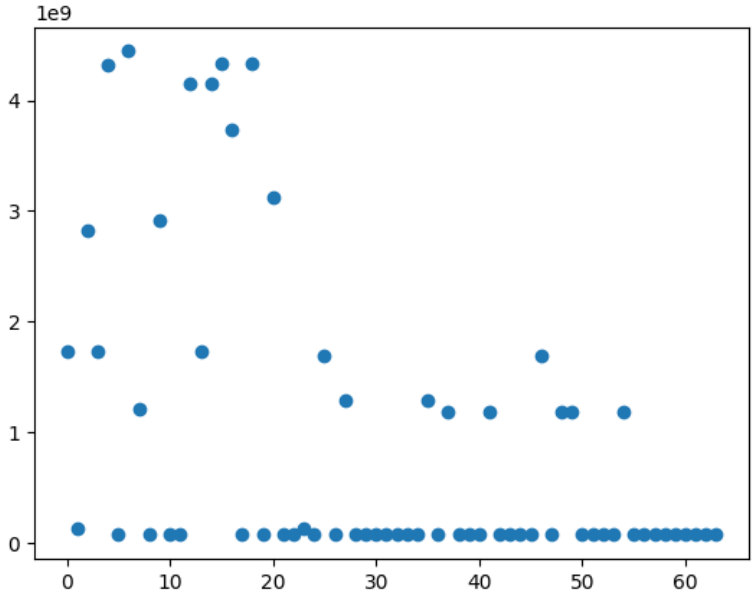

Figura 25 - pla33810 com função de cruzamento CX, sem busca local

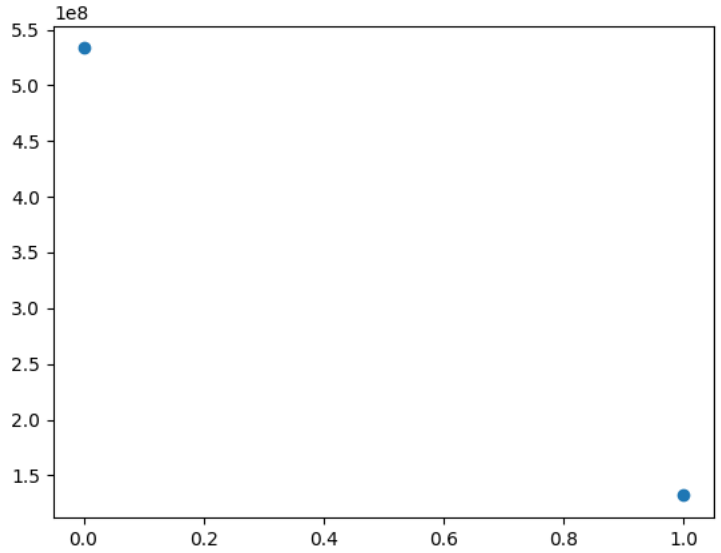

Figura 26 - pla33810 com função de cruzamento CX e busca local

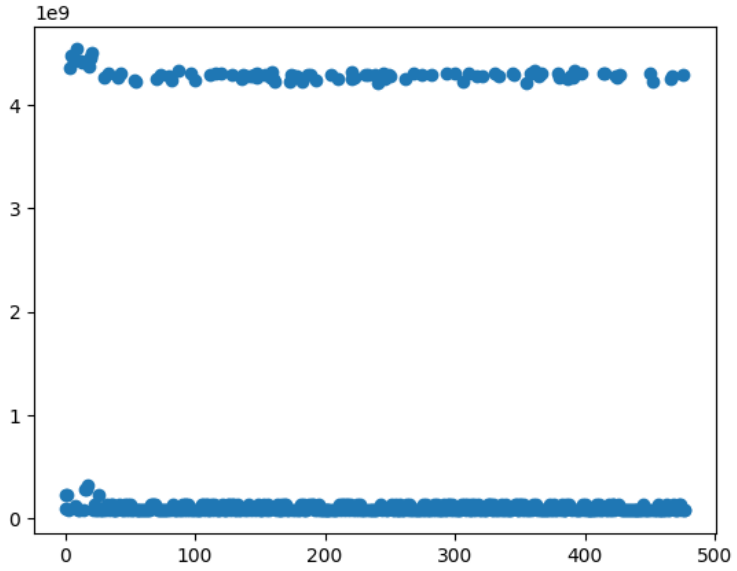

Figura 27 - pla33810 com função de cruzamento OX2, sem busca local

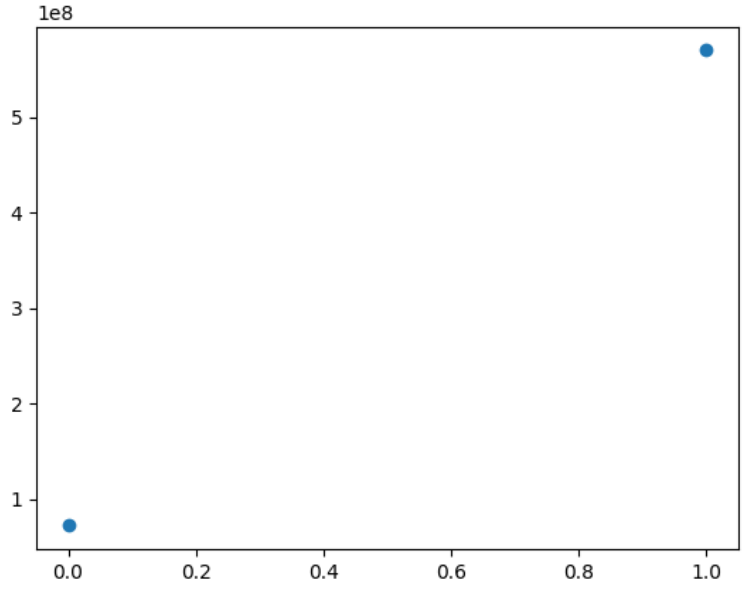

Figura 28 - pla33810 com função de cruzamento OX2 e busca local

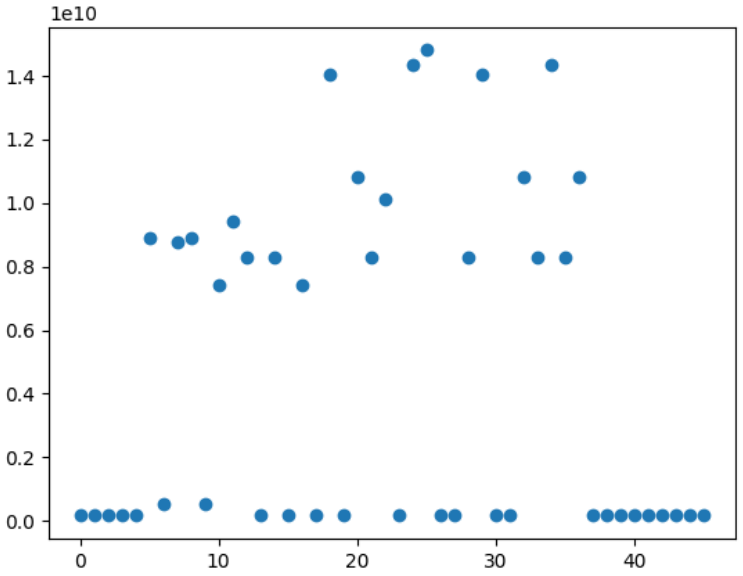

Figura 29 - pla85900 com função de cruzamento CX, sem busca local

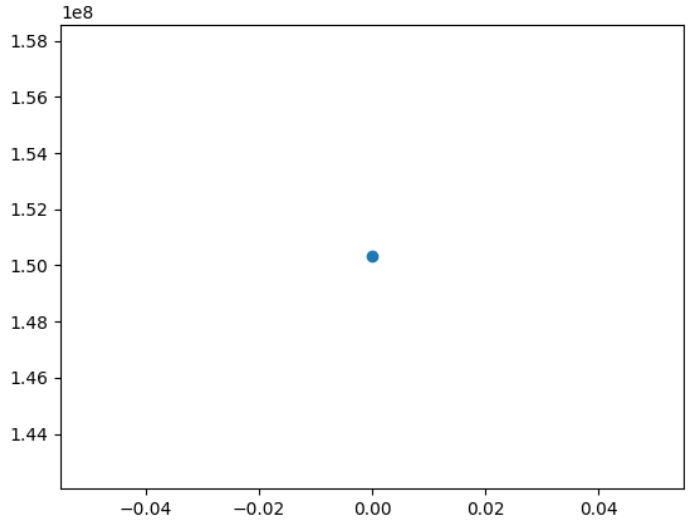

Figura 30 - pla85900 com função de cruzamento CX e busca local

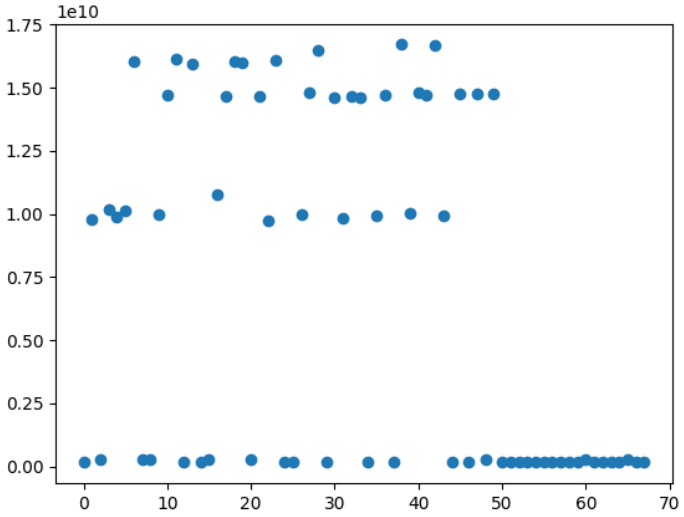

Figura 31 - pla85900 com função de cruzamento OX2, sem busca local

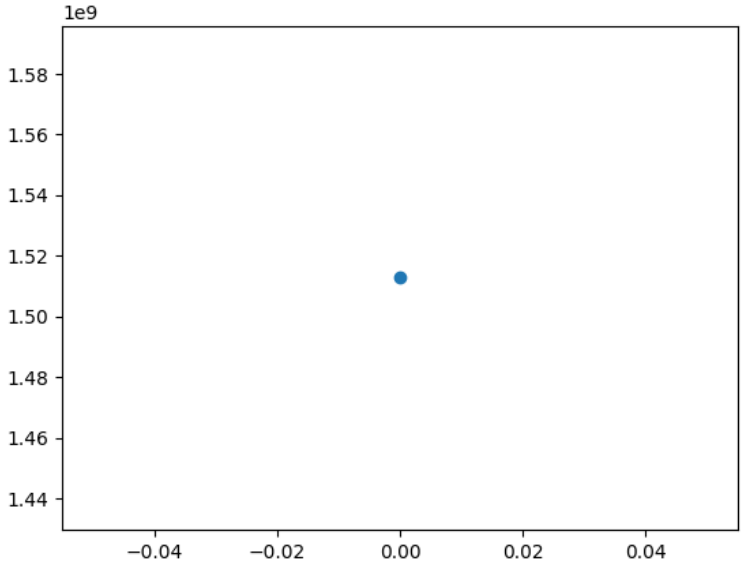

Figura 32 - pla85900 com função de cruzamento OX2 e busca local

Conclusão

Inicialmente o programa construía uma população de cromossomos de tamanho igual a quantidade de vértices n, contudo o tempo de construção da população ficava extremamente custoso então a equipe decidiu criar uma população de tamanho raiz de n arredondada para cima, entretanto ainda era muito, no caso mais extremo o Pla85900 eram necessários criar e manter em memória 294 listas com tamanho igual 85900, então a quantidade de da população foi limitada a 10 indivíduos. Como já esperado pela equipe não era interessante utilizar algoritmos genéticos sem busca local, mas com a utilização de busca local surpreendentemente nos casos menores era possível obter resultados com melhor precisão que no trabalho anterior, infelizmente não é possível dizer o mesmo sobre o tempo, todos os algoritmos do trabalho 1 obtém os resultados com consideravelmente menos tempo e com pouca diferença de precisão, é possível afirmar isso pois antes a medida de tempo do primeiro trabalho era em milisegundos, este é segundos e a diferença do gap varia de 0,5% até 3% de melhoria em relação a um aumento muito grande de tempo.

De fato, algoritmos genéticos são uma meta-heurística muito interessante, mas para o caso do caixeiro viajante a equipe concluiu que há estratégias melhores, caso houvesse uma enorme quantidade de tempo disponível para o algoritmo genético trabalhar ele poderia eventualmente achar uma solução melhor mas para o nosso caso com limite de tempo do cruzamento não é uma opção viável.