ALGORITMOS E ESTRUTURAS DE DADOS

ASSIGNMENT PROBLEM

Licenciatura em Engenharia Informática

Pedro Miguel Bastos Almeida – 93150 – 33,3%

Eduardo Henrique Ferreira Santos – 93107 – 33,3%

José Vaz – 88903 – 33,3%

Índice

[1.1. – O que é o Assignment Problem 3](#_Toc24559454)

[1.2 - Objetivos coletivos para o projeto 4](#_Toc24559455)

[2.1 - Brute Force 4](#_Toc24559456)

[2.2 - Branch And Bound 5](#_Toc24559457)

[2.3 - Random permutations 5](#_Toc24559458)

[3.1 – Resultados 6](#_Toc24559459)

[4.1 – Código implementado em C (comentado com explicações). 9](#_Toc24559460)

1

# 1.1. – O que é o Assignment Problem

Assignment problem é um problema de otimização combinatória. Consiste em encontrar a melhor combinação possível no menor tempo possível. Neste caso, são dados um certo número de agentes e de tarefas (n). cada tarefa (t) que é atribuída a um agente (a) tem um custo C (a, t). O objetivo é encontrar um custo mínimo atribuindo cada tarefa a um agente diferente, ou seja, cada tarefa só pode ser atribuída a um e um só agente que ainda não tenha nenhuma tarefa (função bijetiva). O custo total é o somatório dos custos de cada assignment feito. Por exemplo, com a tabela de custos:

|  |  |  |  |
| --- | --- | --- | --- |
| a\t | 0 | 1 | 2 |
| 0 | 3 | 8 | 6 |
| 1 | 4 | 7 | 5 |
| 2 | 5 | 7 | 5 |

t(0) = 0, t(1) = 1, t(2) = 2, com o custo 3 + 7 + 5 = 15

t(0) = 0, t(1) = 2, t(2) = 1, com o custo 3 + 5 + 7 = 15

t(0) = 1, t(1) = 0, t(2) = 2, com o custo 8 + 4 + 5 = 17

t(0) = 1, t(1) = 2, t(2) = 0, com o custo 8 + 5 + 5 = 18

t(0) = 2, t(1) = 0, t(2) = 1, com o custo 6 + 4 + 7 = 17

t(0) = 2, t(1) = 1, t(2) = 0, com o custo 6 + 7 + 5 = 18

Neste caso, o custo mínimo seria 15, encontrado nas duas primeiras permutações.

# 1.2 - Objetivos coletivos para o projeto

Com este projeto nós pretendemos aprofundar os conhecimentos na linguagem C, aperfeiçoando também a prática em programar utilizando novos métodos necessários para a resolução deste problema.

Tendo em conta que nunca tínhamos utilizado esta linguagem de programação, foi uma boa oportunidade para o fazer, visto que esta continua a ser uma das mais utilizadas, sendo uma das mais eficientes devido ao fácil acesso à memória e trabalho com a mesma.

2.

# 2.1 - Brute Force

Para uma primeira resolução deste problema, recorremos ao algoritmo mais óbvio e simples: o brute force. Este apenas consiste em percorrer todas as opções possíveis (permutações) e calcular o custo para cada permutação, verificando se é menor que o mínimo até aquela permutação e, se for menor, substituir pelo novo mínimo.

Contudo, esta solução é a menos eficiente que vamos utilizar. Isto porque a complexidade deste problema é fatorial em relação ao número de agentes e tarefas (n). Por exemplo, se aplicarmos este algoritmo para 14 agentes e 14 tarefas, a complexidade do problema será 14! ( O(n!) ). Dito isto, este algoritmo é ineficiente e demoroso.

Para este algoritmo, usamos a função *generate\_all\_permutations*, parte dela já disponibilizada pelo professor, e altermos a parte dentro do else, de modo a que calculasse o custo mínimo e o custo máximo para cada n e guardasse o assignment correspondido a cada um. Além disso, complementamos a função para gerar o histograma para cada n.

# 2.2 - Branch And Bound

Após termos utilizado o Brute Force e termos comprovado a ineficiência do mesmo, começámos a investigar um novo método denominado Branch And Bound, consistindo este em:

Para cada permutação, calcular o custo parcial até à posição atual e se, somando a esse custo o custo mínimo das tarefas restantes, já for maior que o custo mínimo atual, passar à permutação seguinte. Basicamente, é um algoritmo parecido com o brute force mas que descarta as soluções que já se sabemos que são impossíveis.

Quanto ao código, criamos a função generate\_all\_permutatios\_branch\_and\_bound (adaptada da generate\_all\_permutations) que tem como argumento extra o custo parcial. Como no brute force, também calculamos o custo mínimo e o seu assignment correspondente.

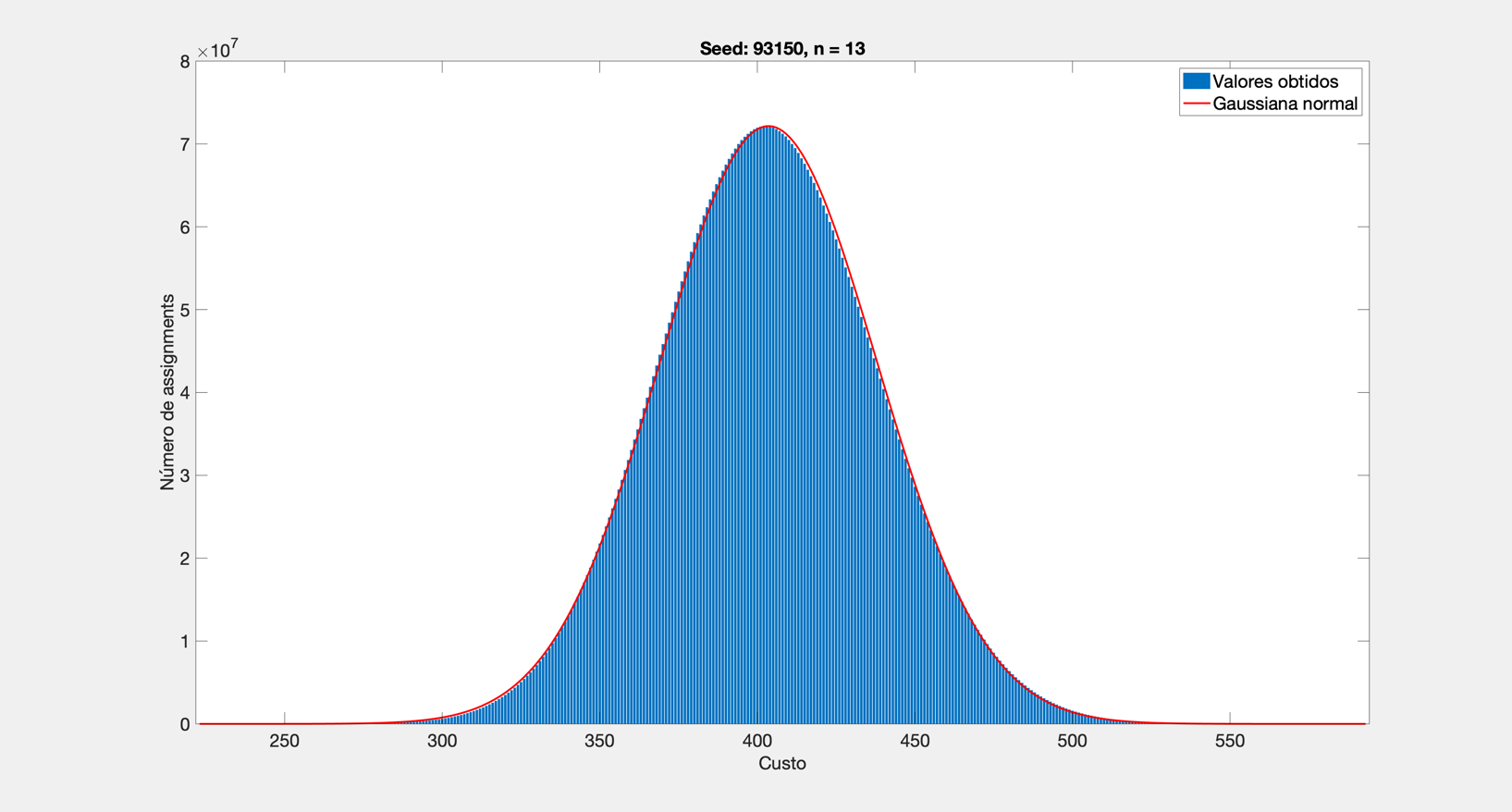
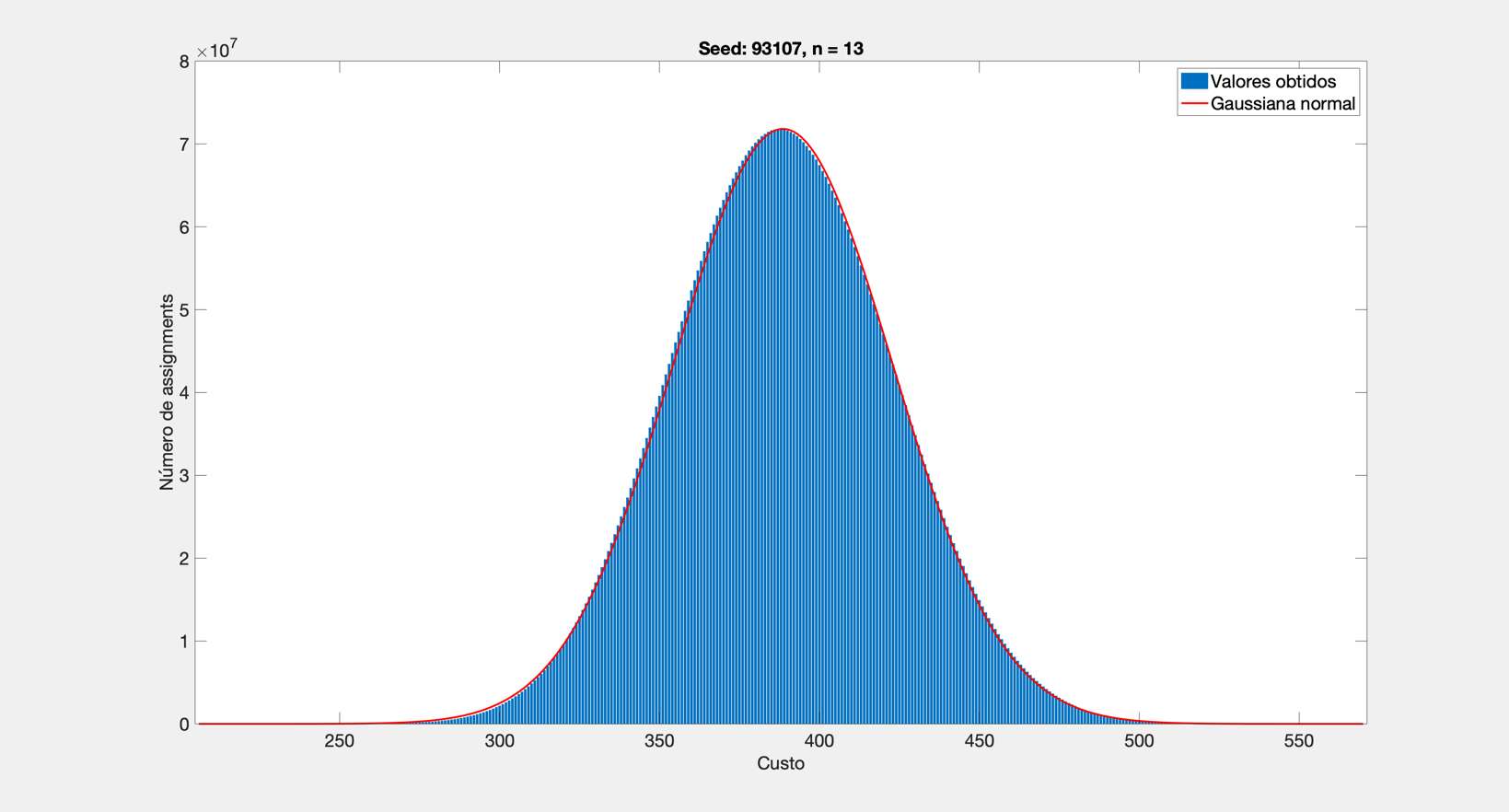
# 2.3 - Random permutations

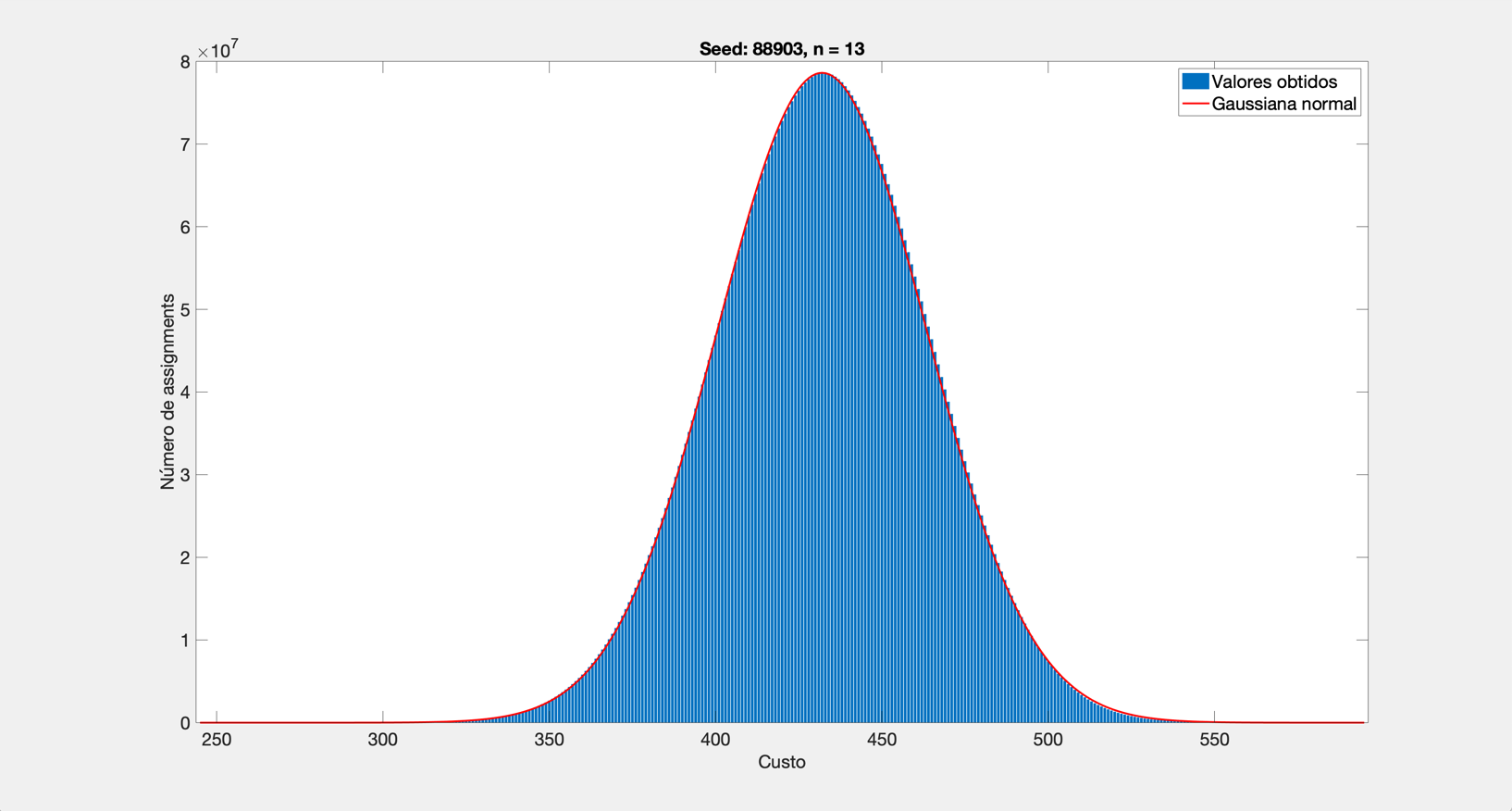
Para este algoritmo, vamos escolher aleatoriamente um valor elevado de permutações e calcular a permutação com o custo mínimo. Basicamente, para cada permutação escolhida calculamos o custo e verificamos se é menor que o mínimo que tínhamos e, se for, substituímos o custo mínimo e o seu assignment correspondente.

No código, usamos a função (dada pelo professor) random\_permutation que retorna uma permutação aleatória e calculamos, para 1 milhão de permutações aleatórias, o custo mínimo e máximo e os seus assignments correspondentes.

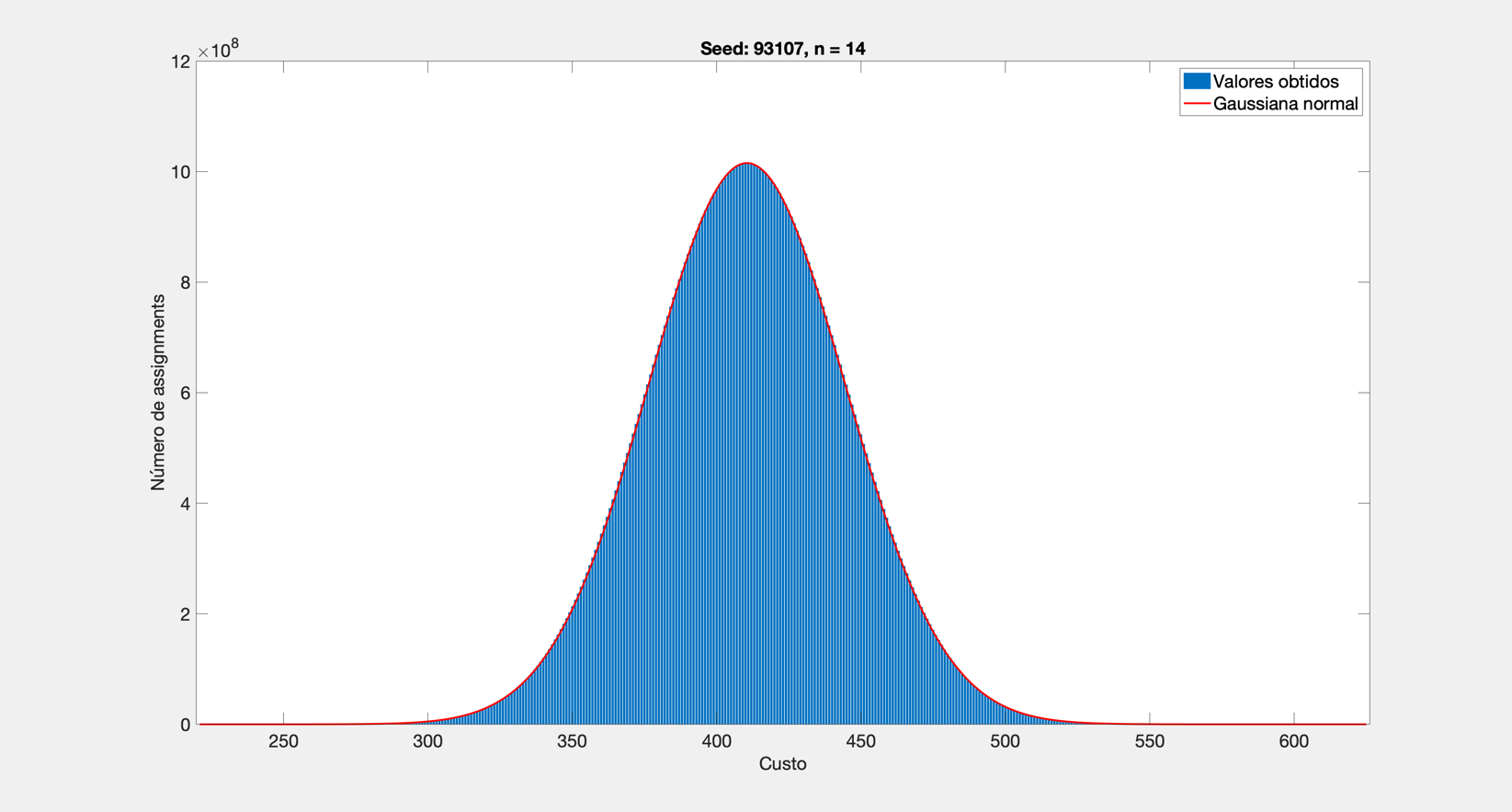
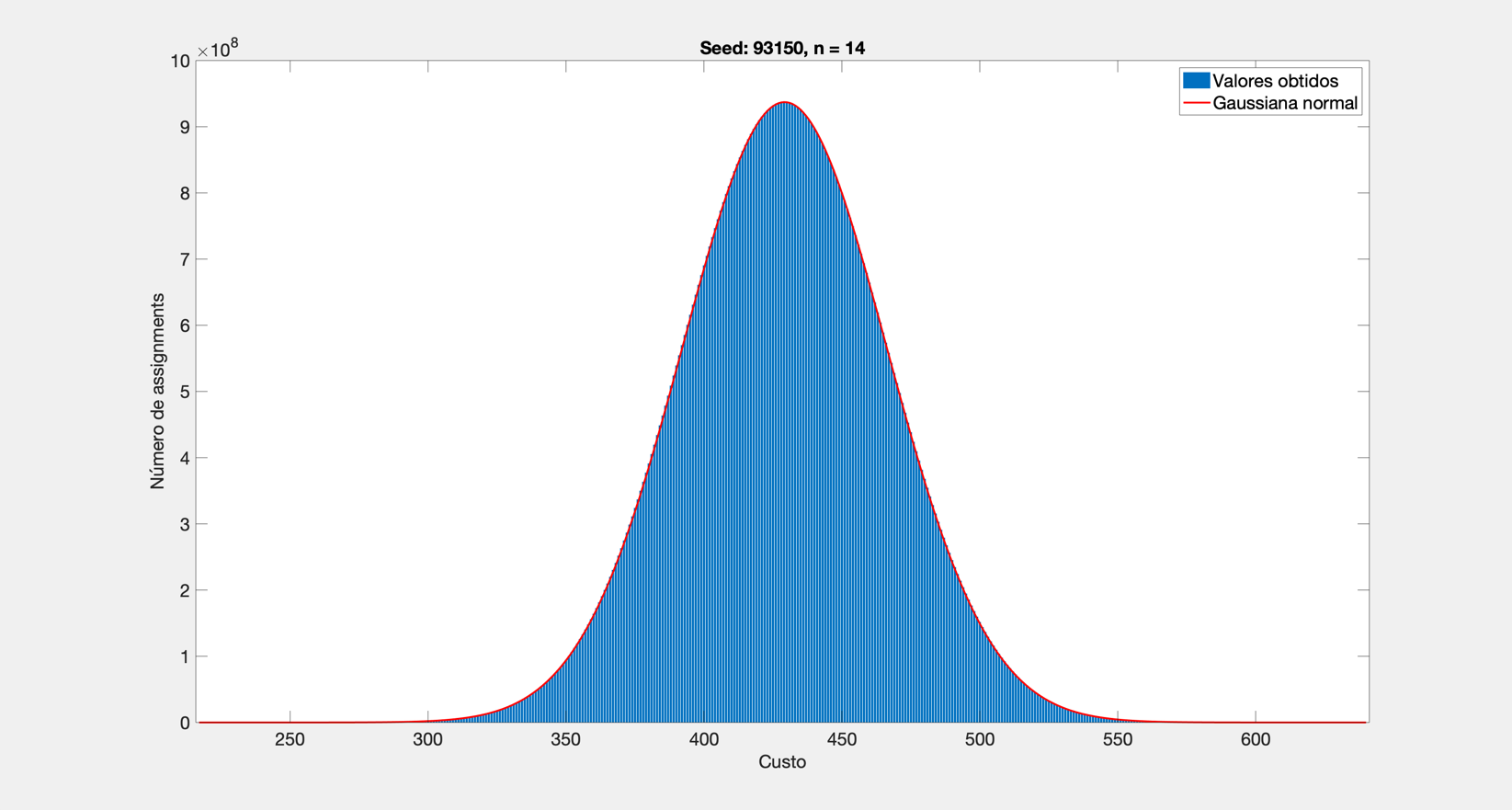
3.

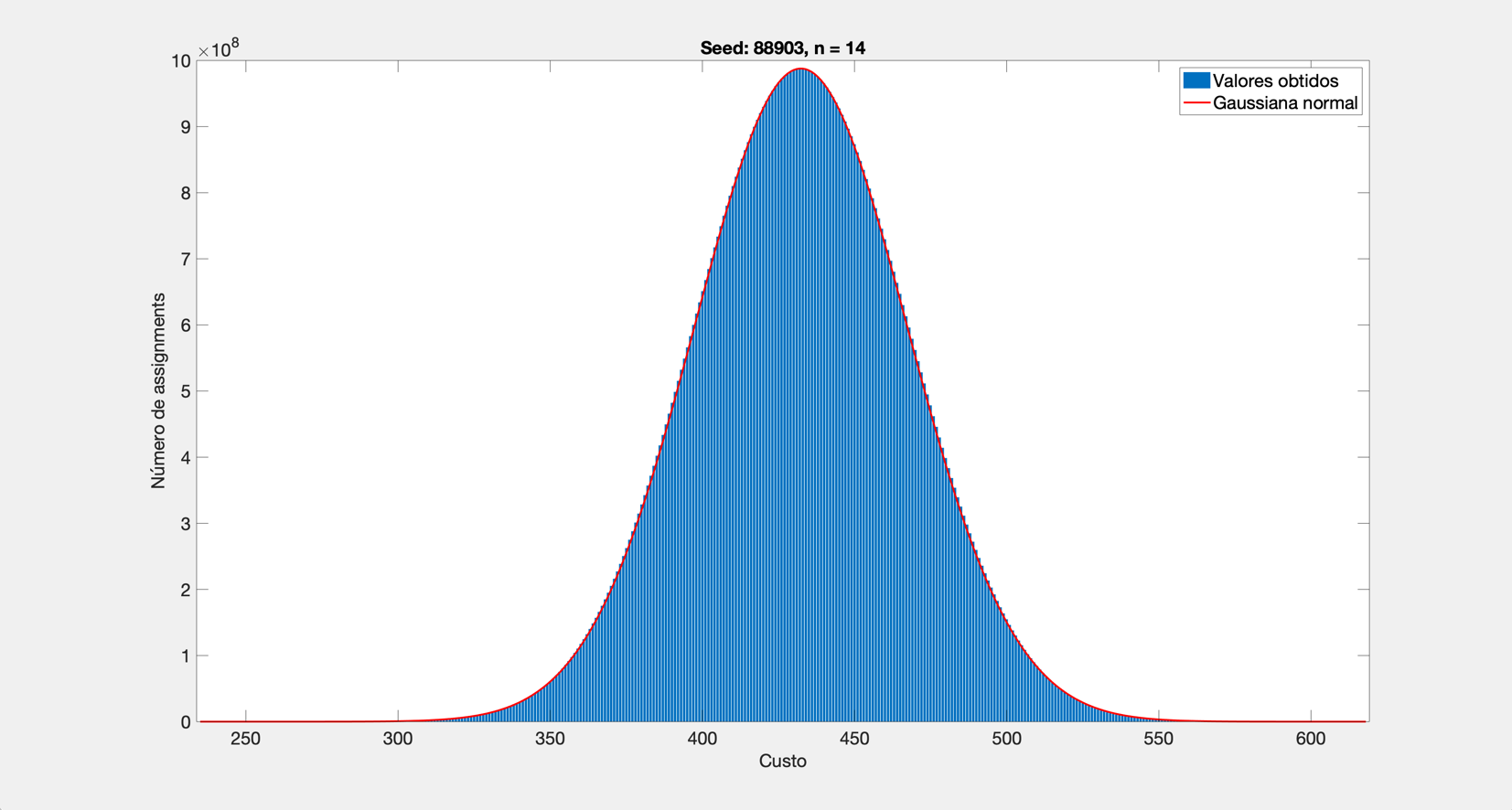
# 3.1 – Resultados

Histogramas e Gaussianas para cada número mecanográfico com n=13:



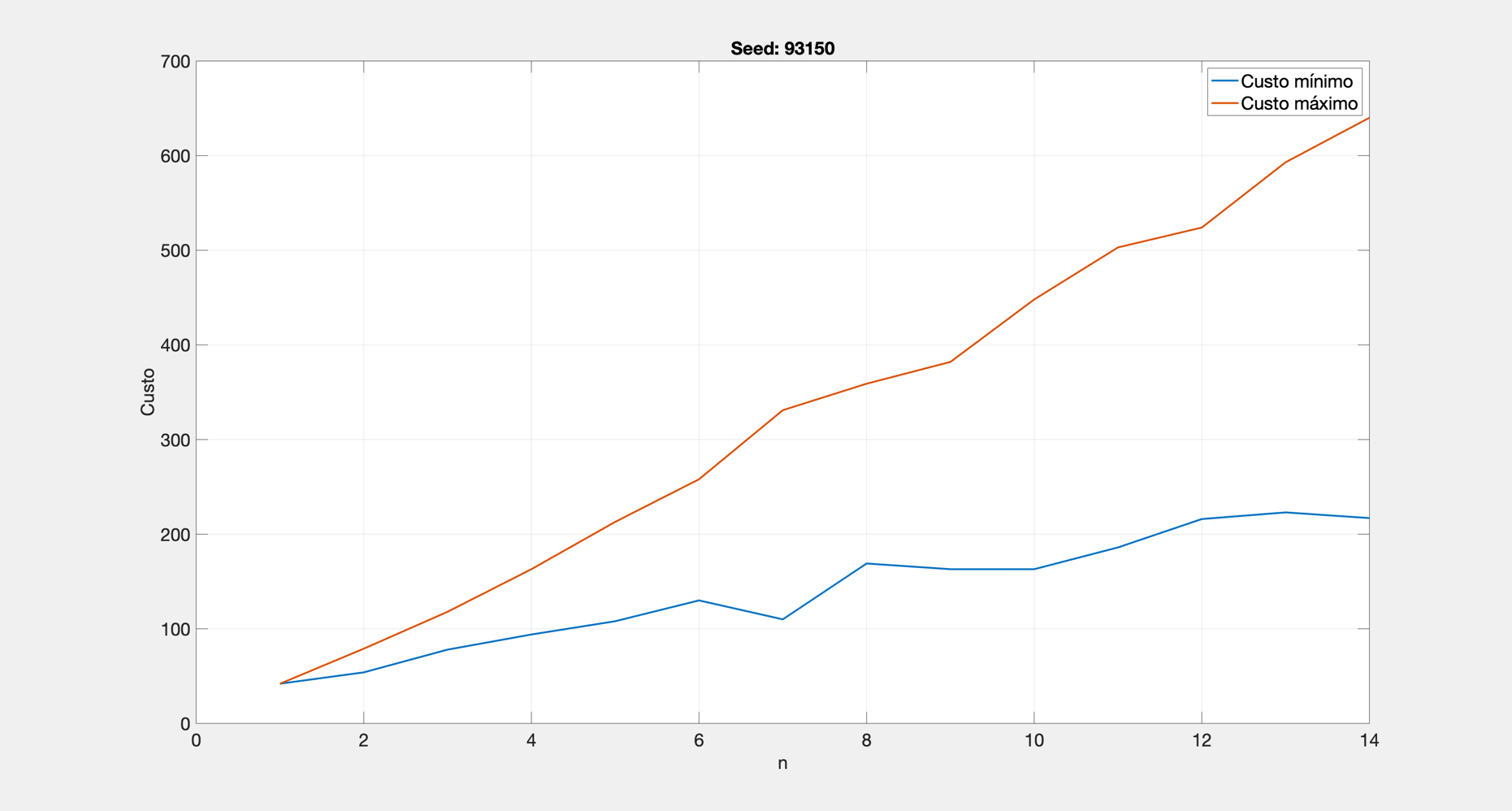
Histogramas e Gaussianas para cada número mecanográfico com n=14:

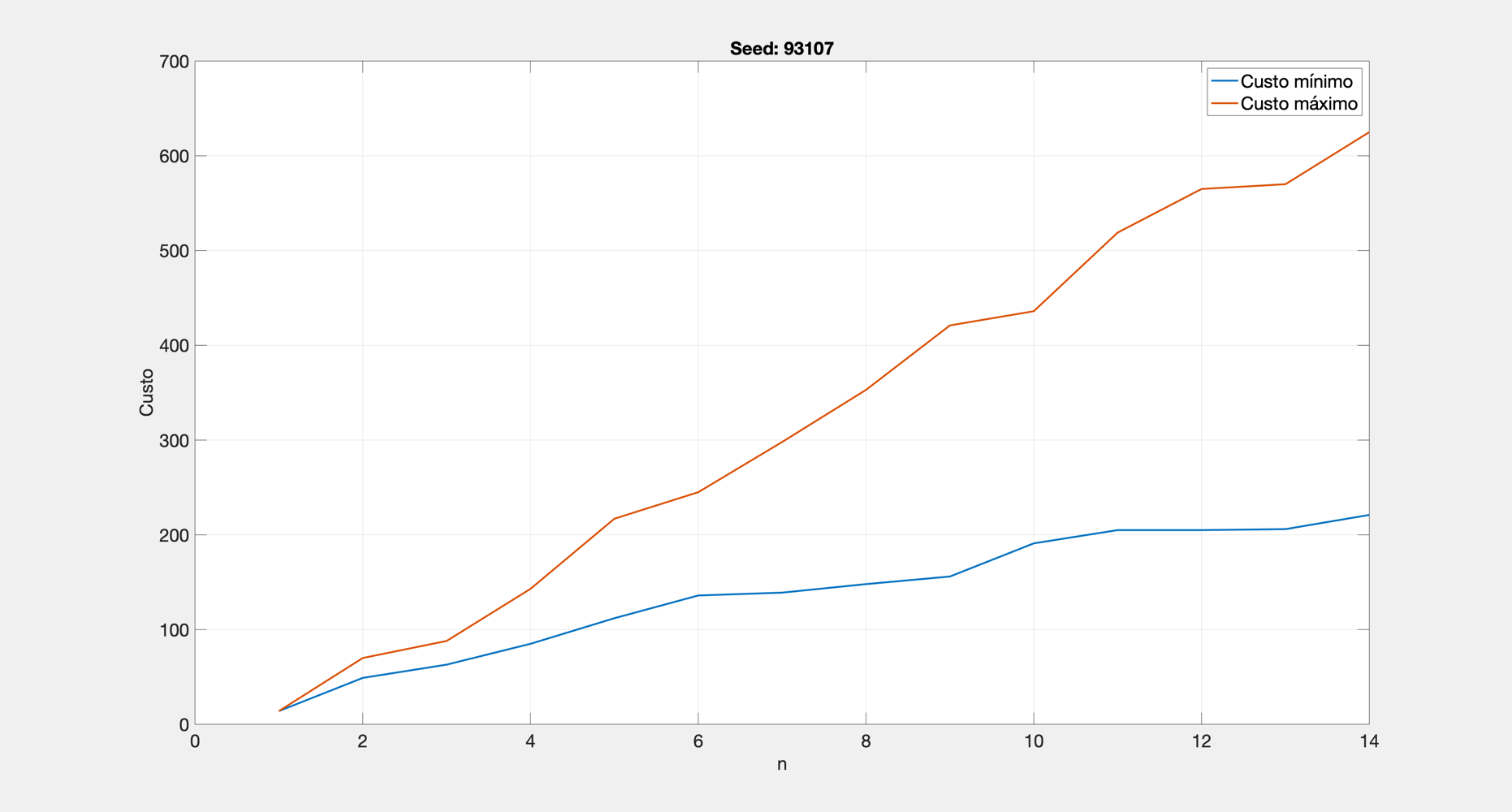


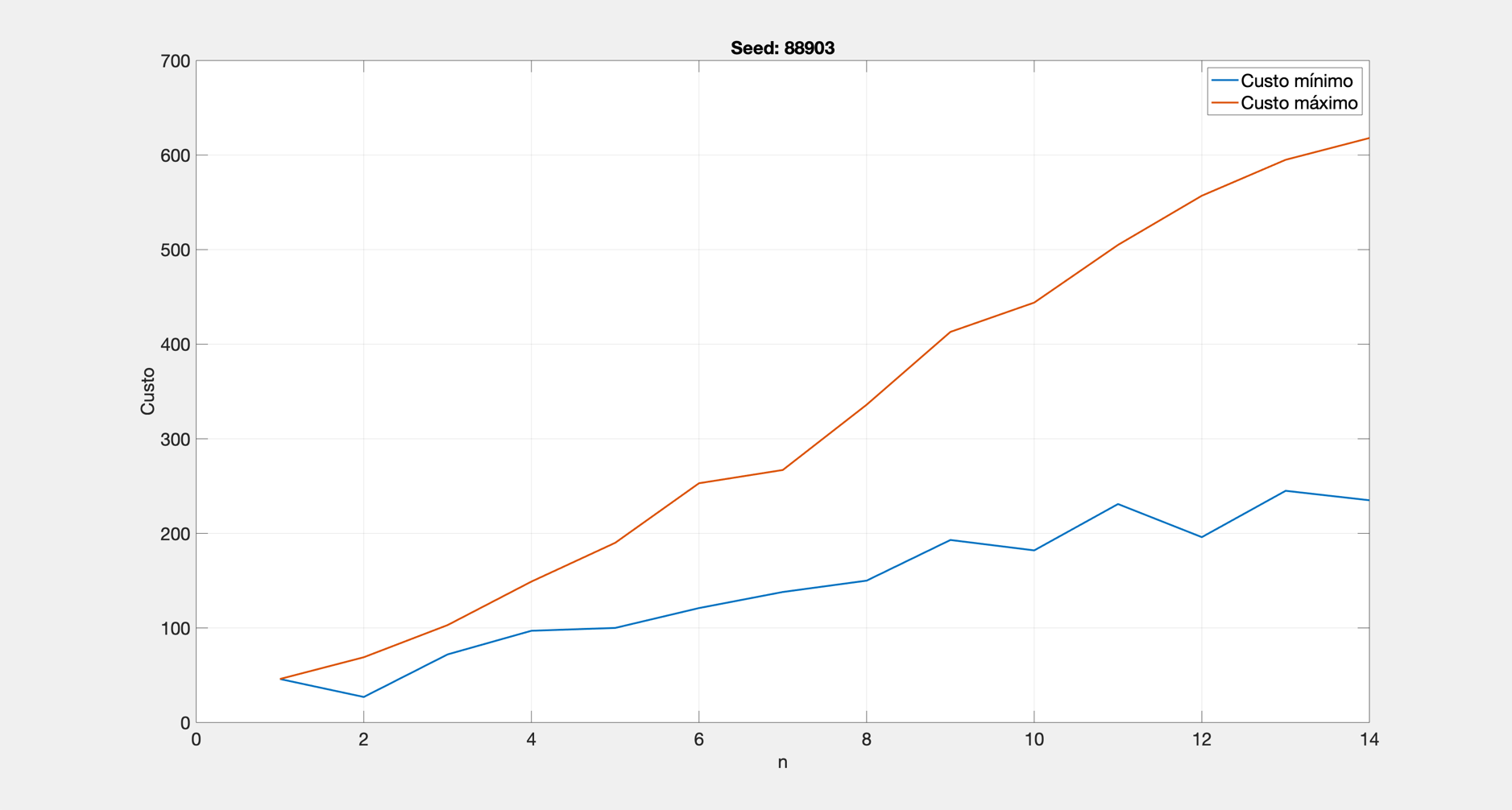


Como se pode verificar, os histogramas do número de assignments em função do custo estão muito semelhantes aos valores teóricos (representados pelas Gaussianas normais).

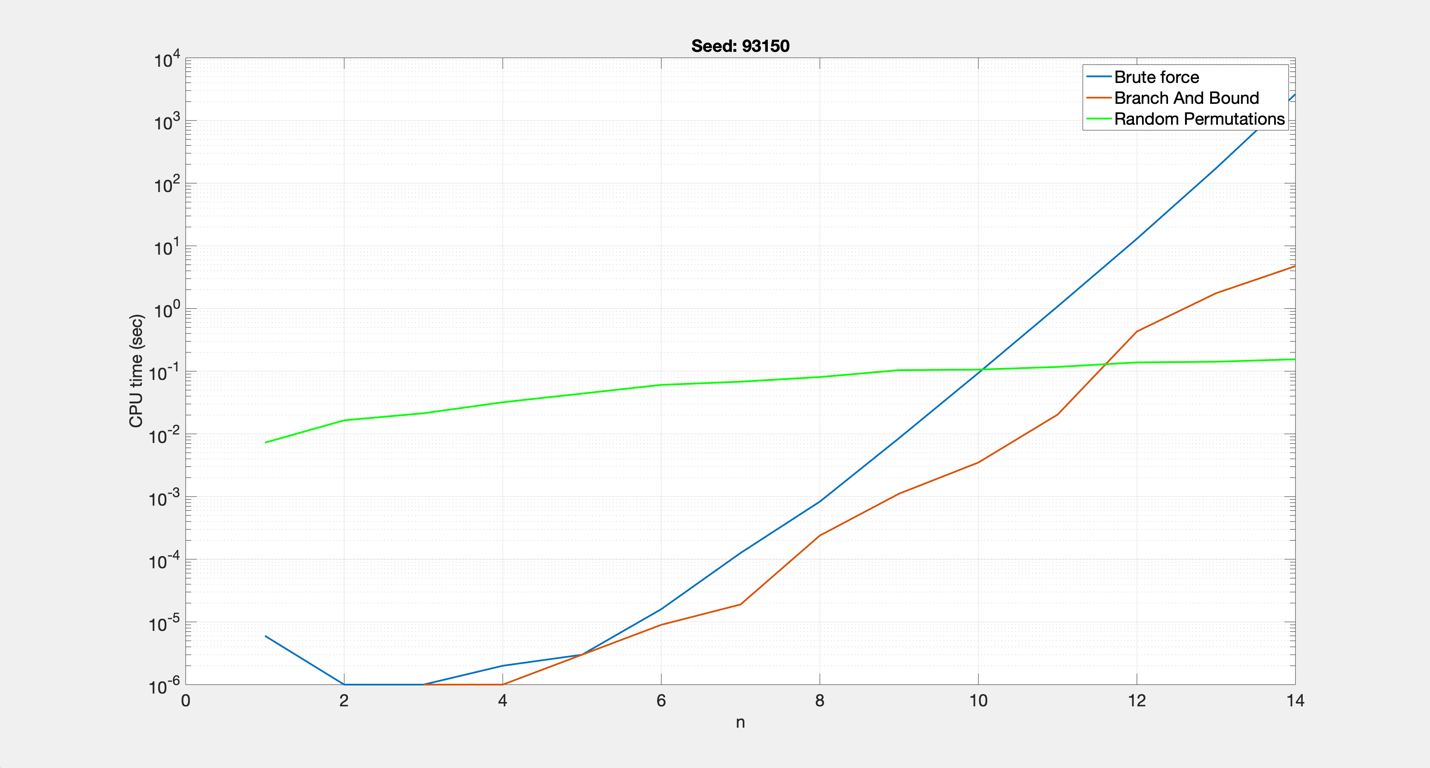
Gráficos que comparam custos mínimos e máximos em função de n para cada número mecanográfico:

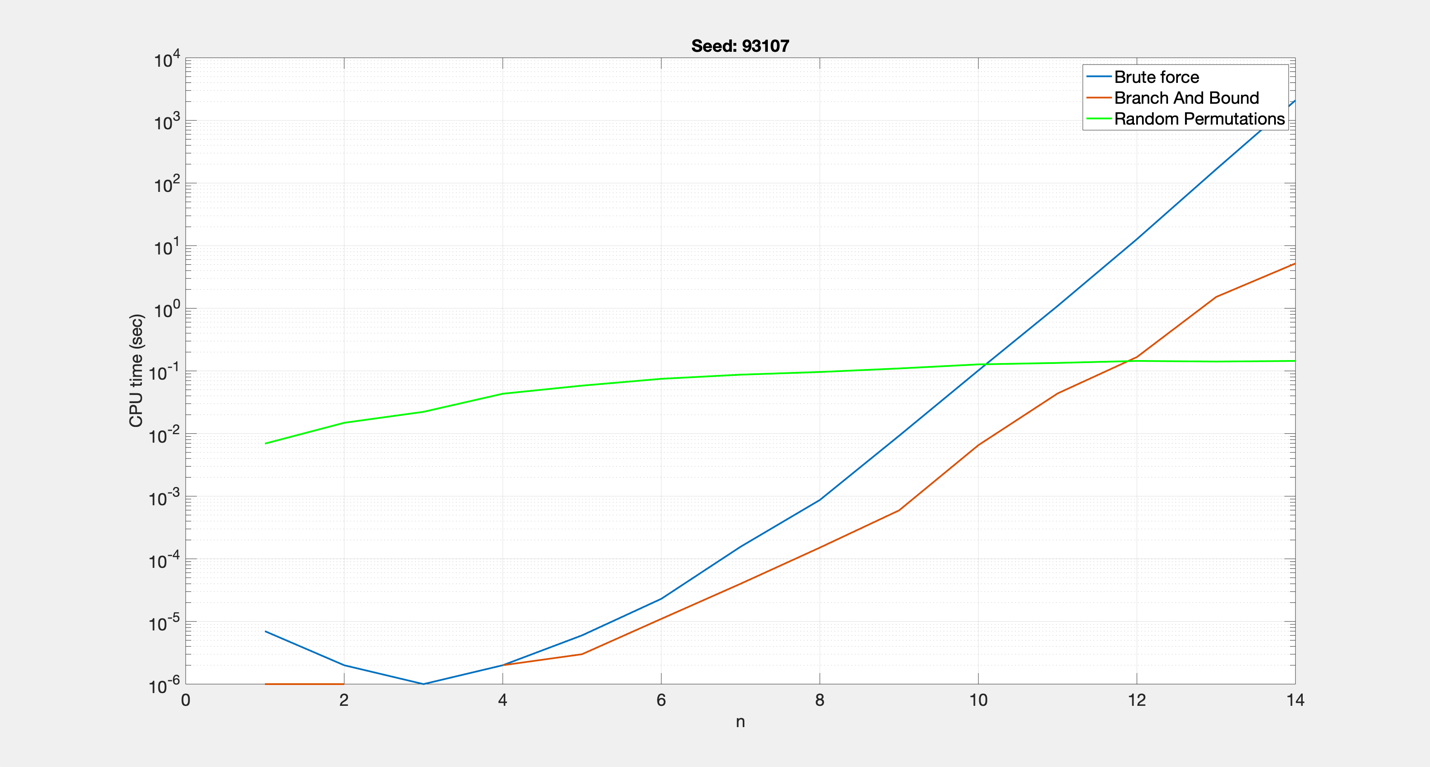
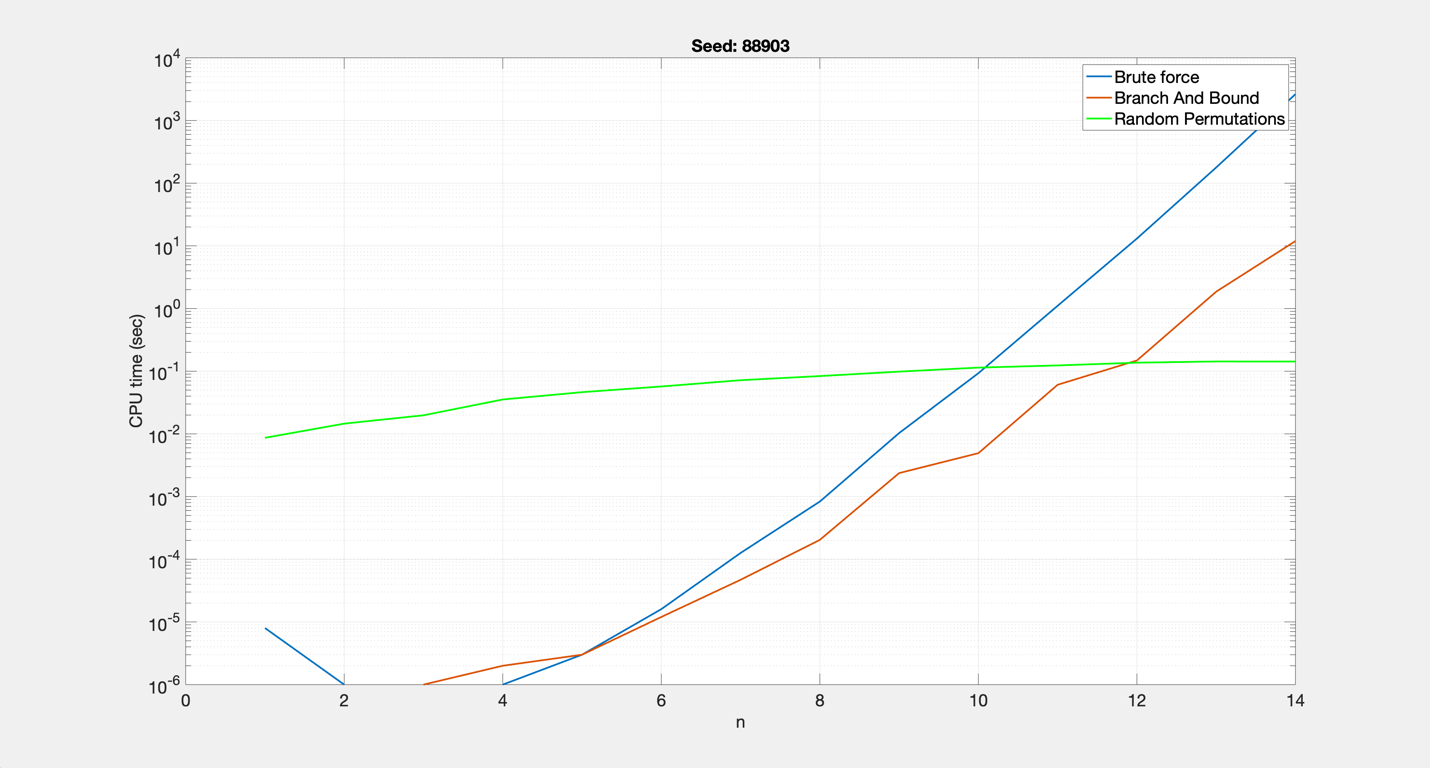






Quanto aos custos mínimos e custos máximos, quanto maior for o n, mais diferença há entre eles, como seria de esperar. É também visível que a função do custo máximo cresce muito mais rapidamente que a do custo mínimo.

Gráficos dos tempos de execução em função do n comparando os 3 algoritmos usados, para cada número mecanográfico:



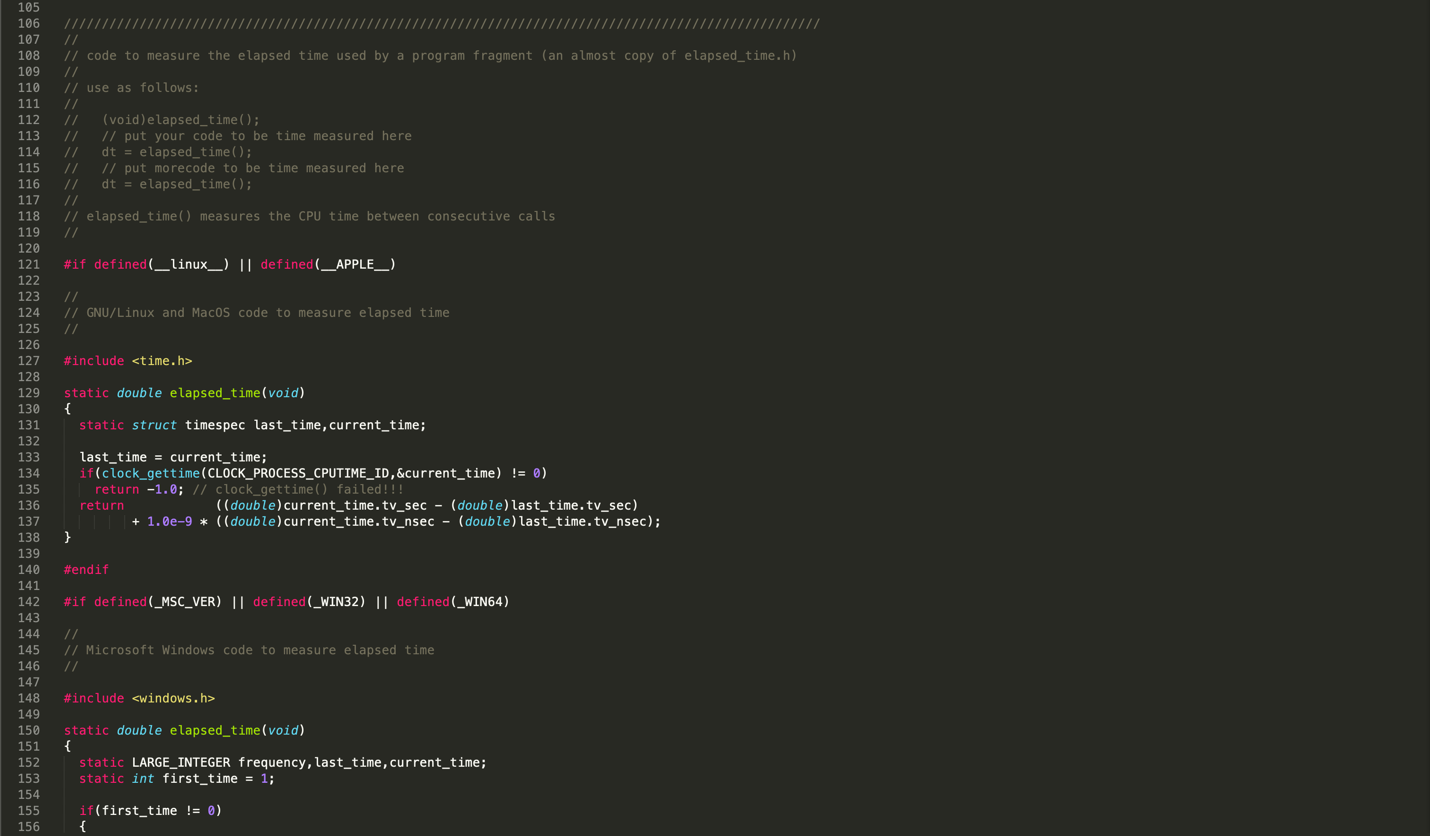
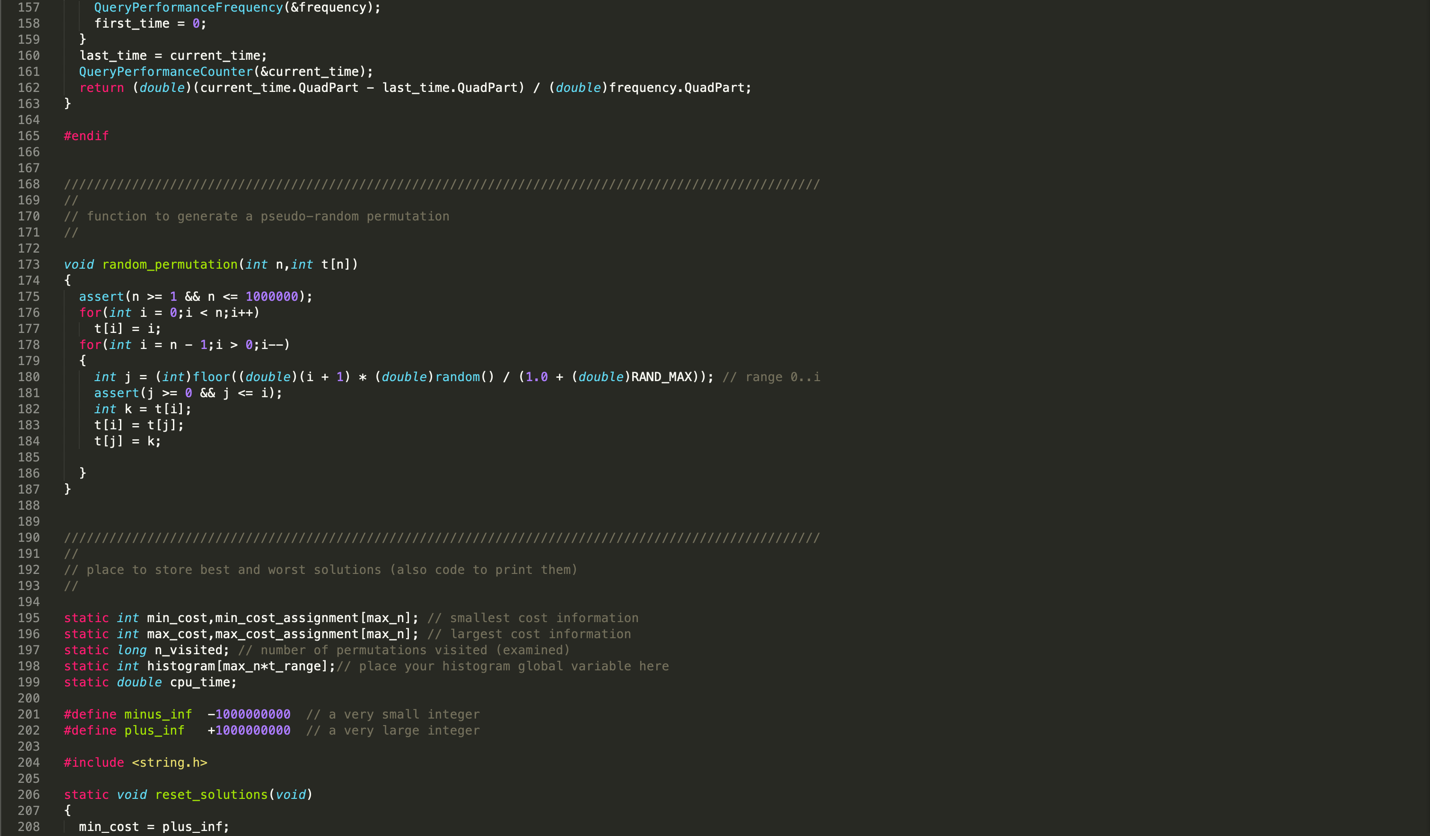
Como se pode verificar, o branch and bound é muito mais eficiente que o brute force, pois é significativamente mais rápido a encontrar as soluções pretendidas, sendo na mesma 100% fiel.

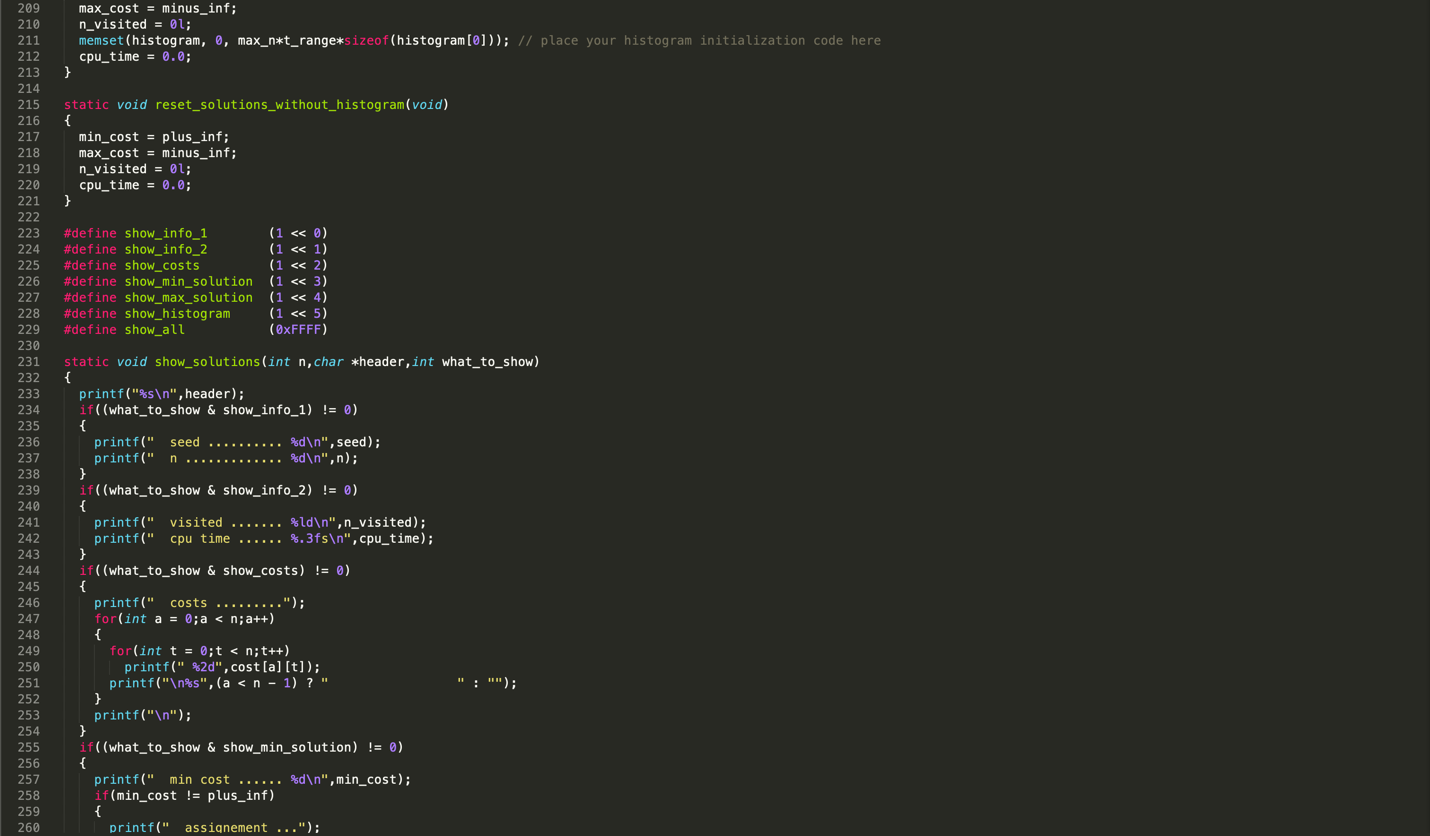
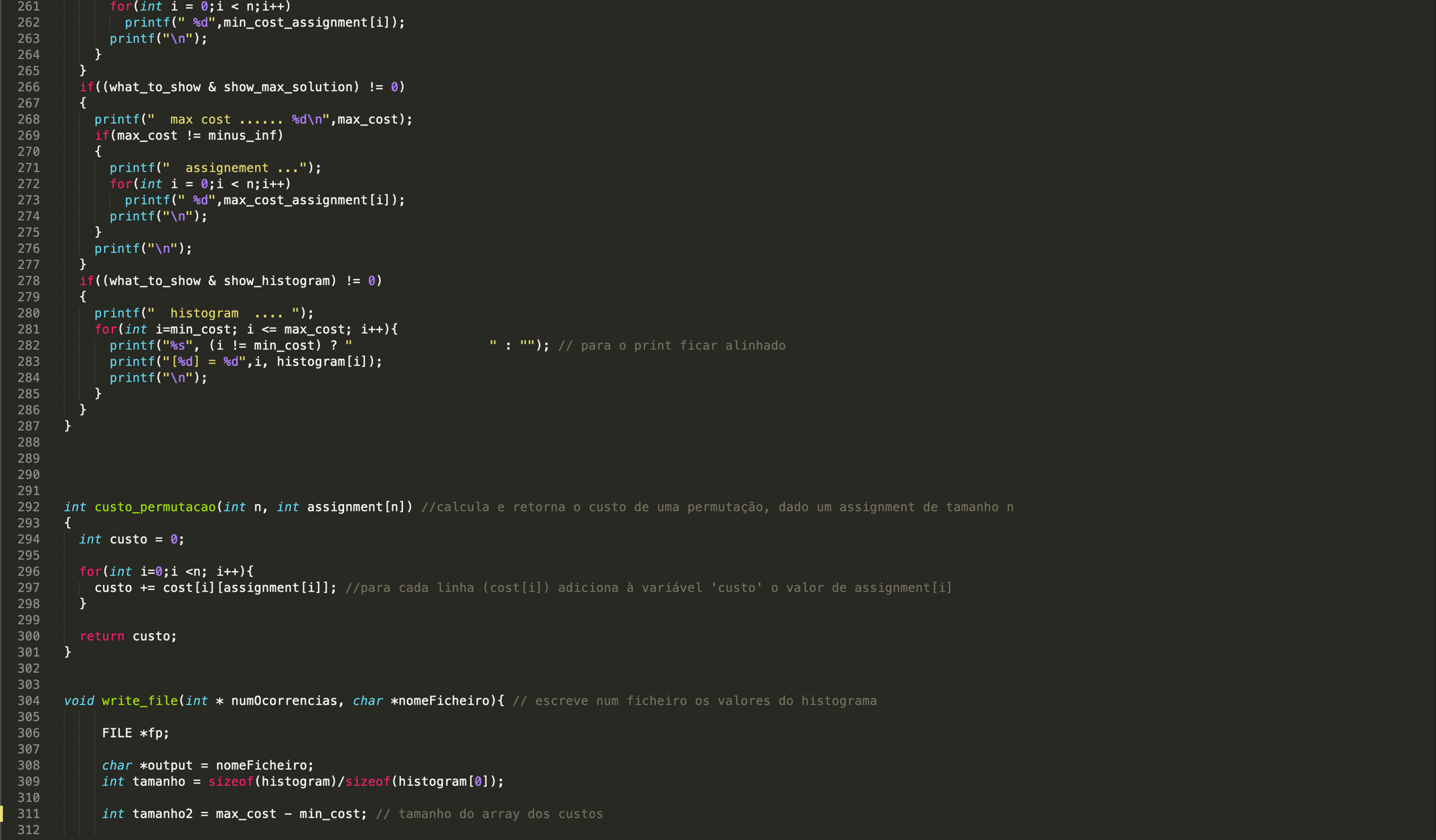
Quanto ao uso de random permutations, acaba por ser mais rápido para n>12, contudo como não são percorridas todas as permutações, apenas obtemos um valor próximo do custo mínimo e do custo máximo.

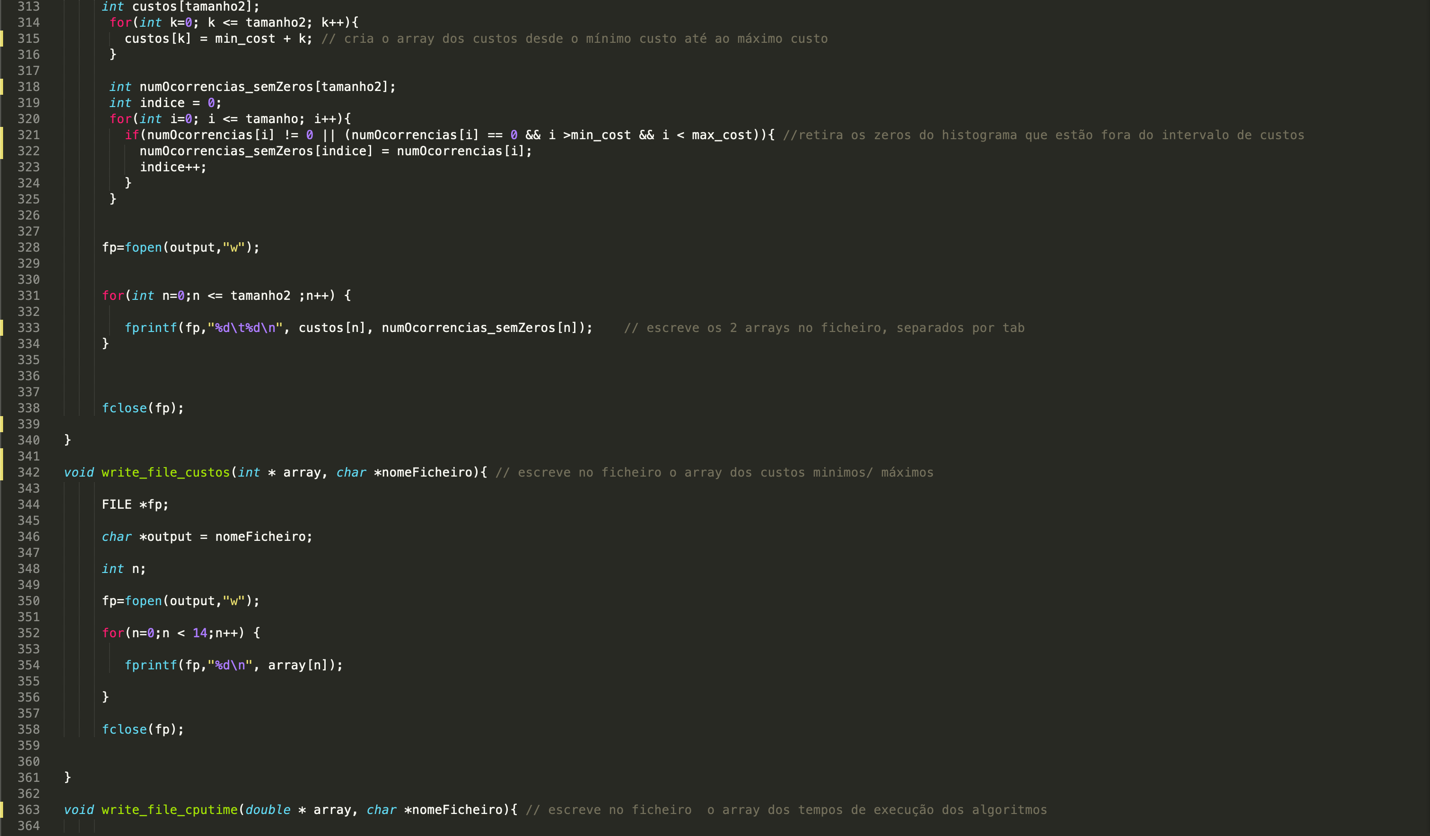
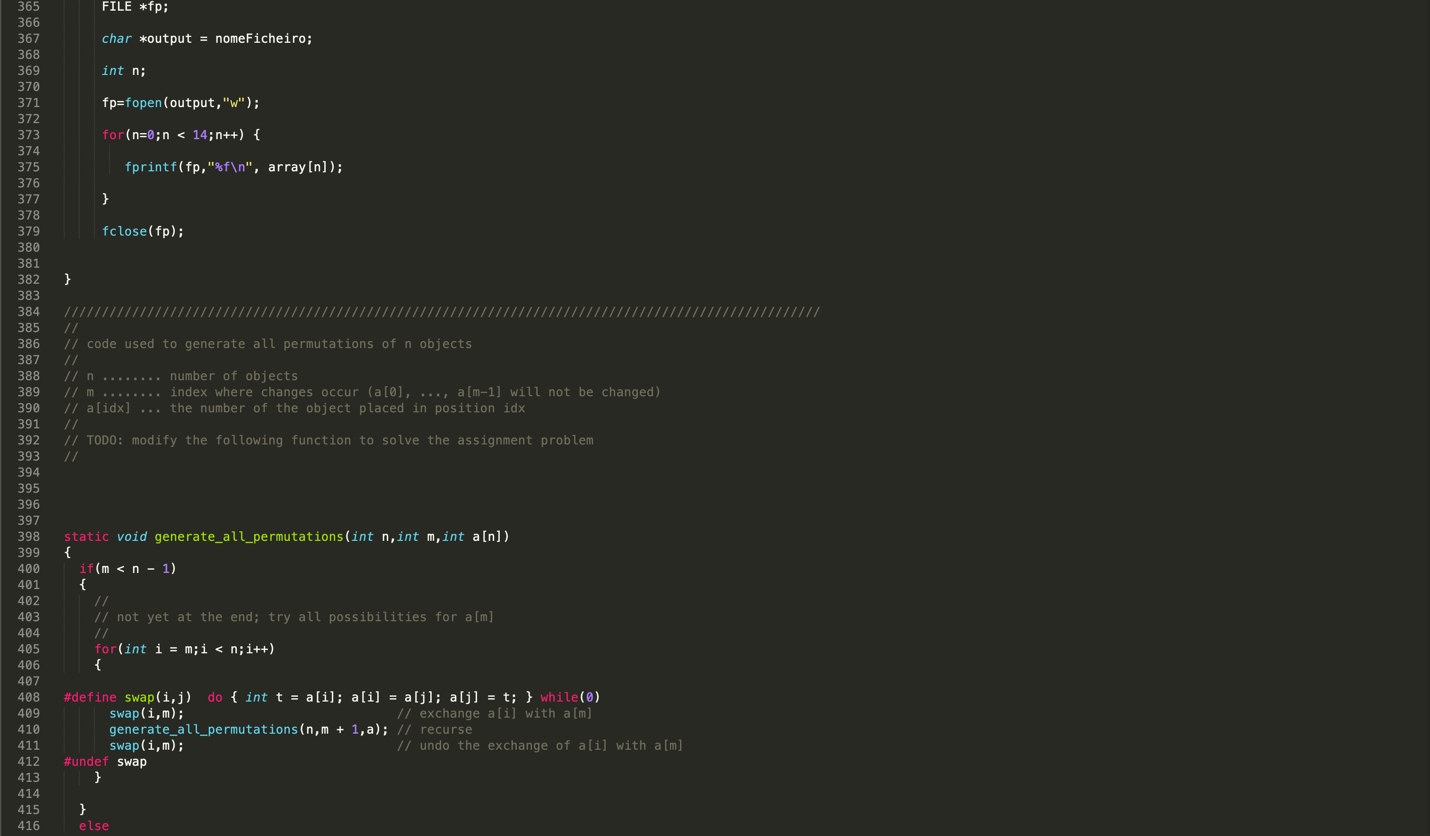
4.

# 4.1 – Código implementado em C (comentado com explicações).

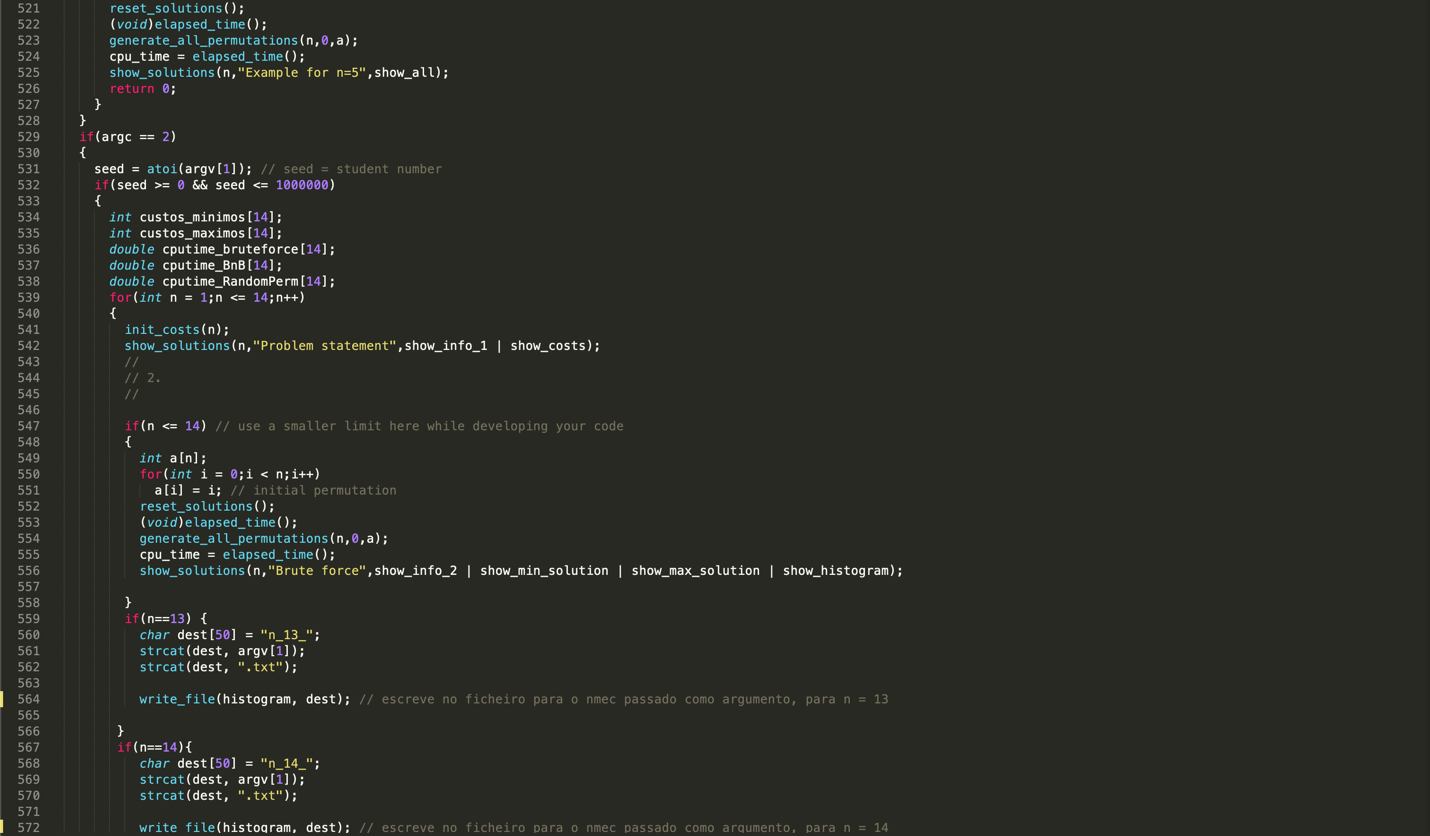




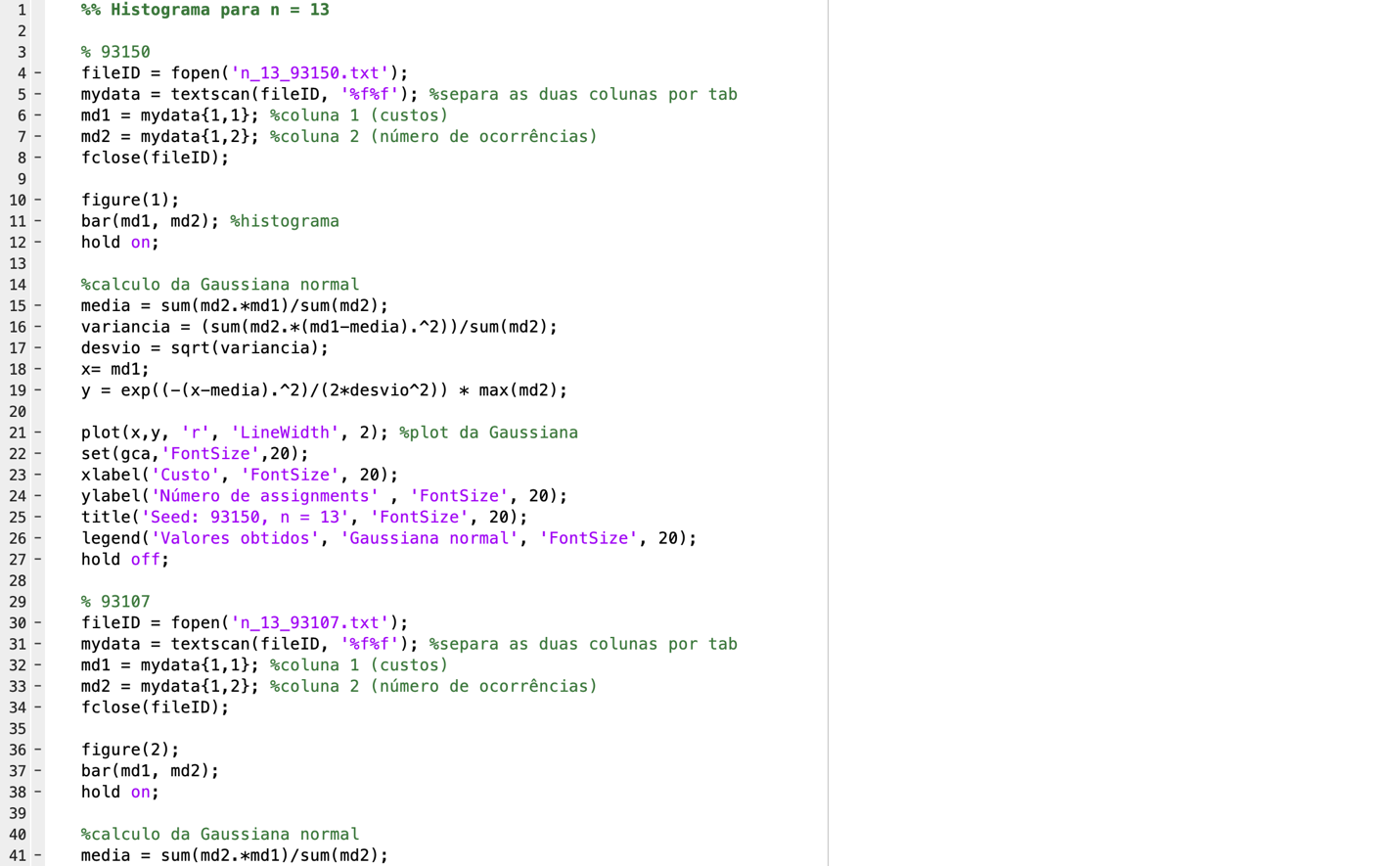


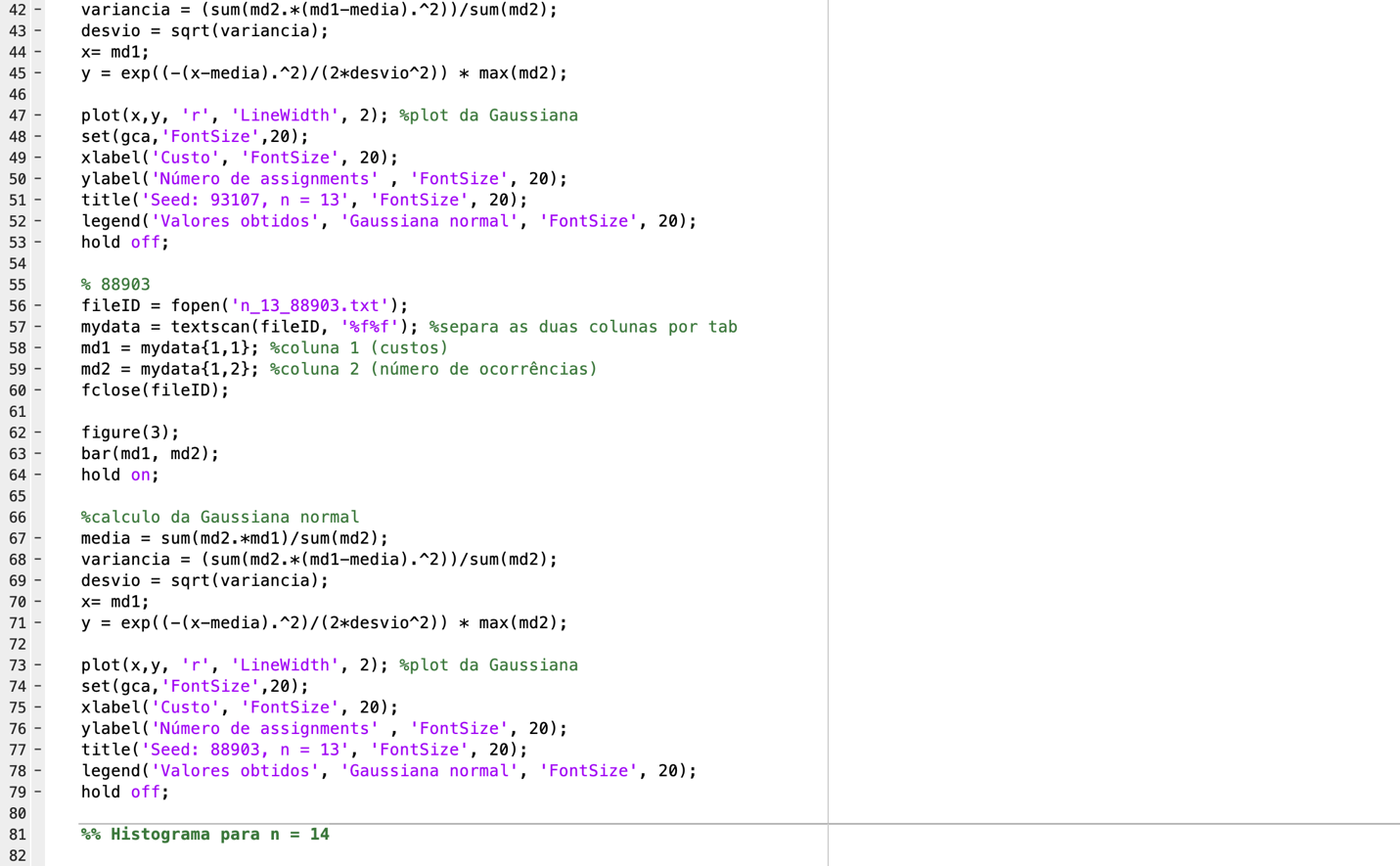
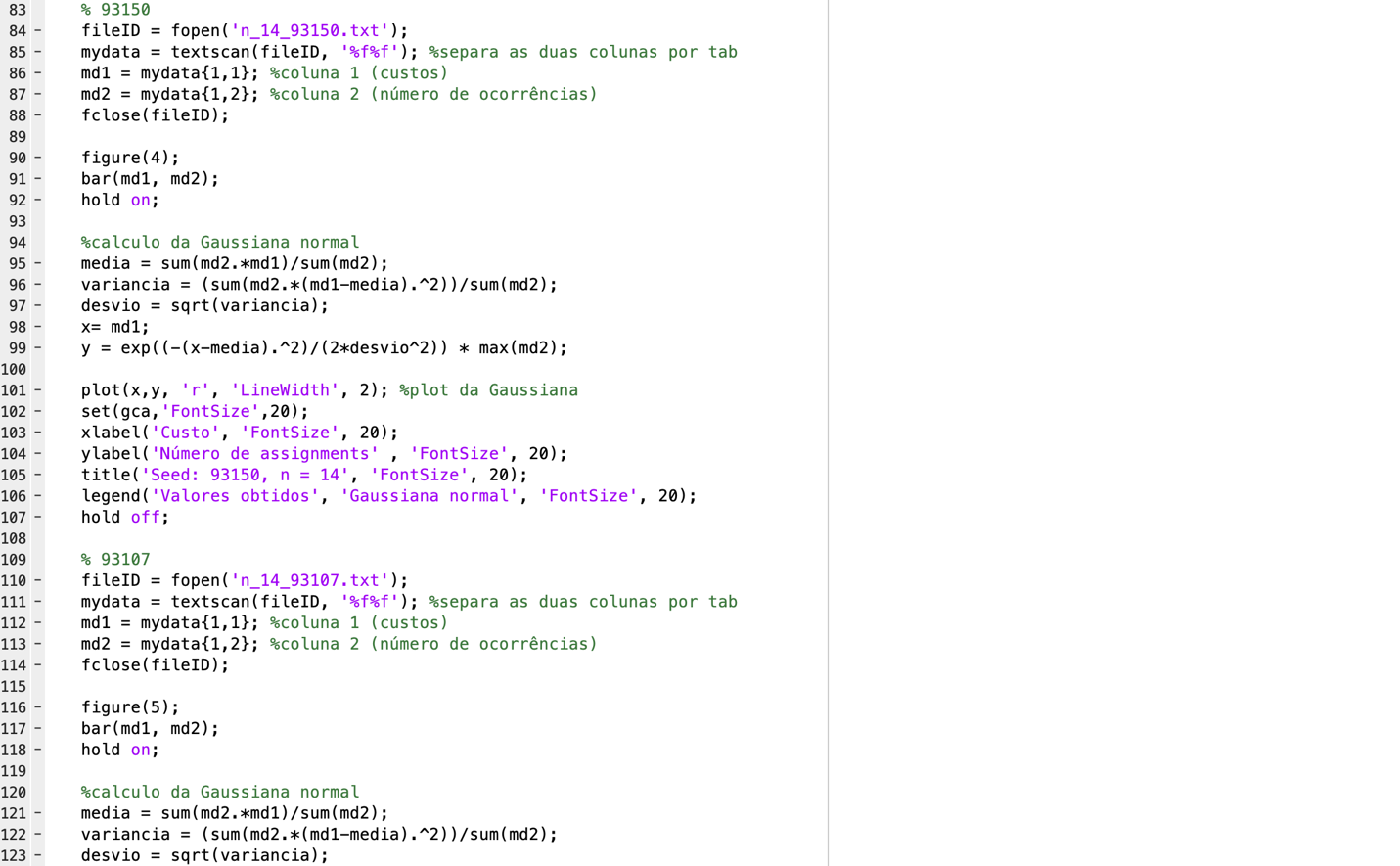


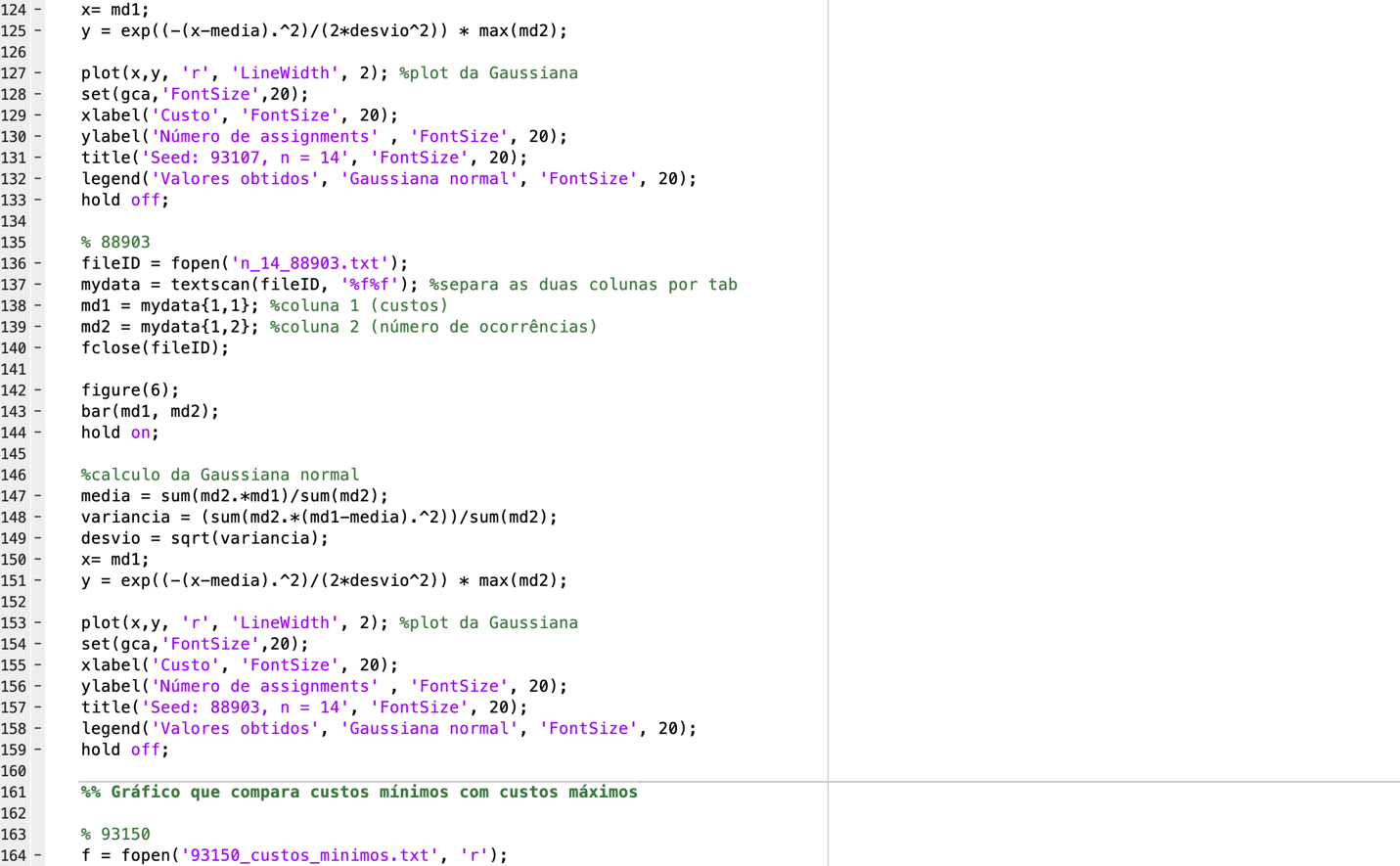
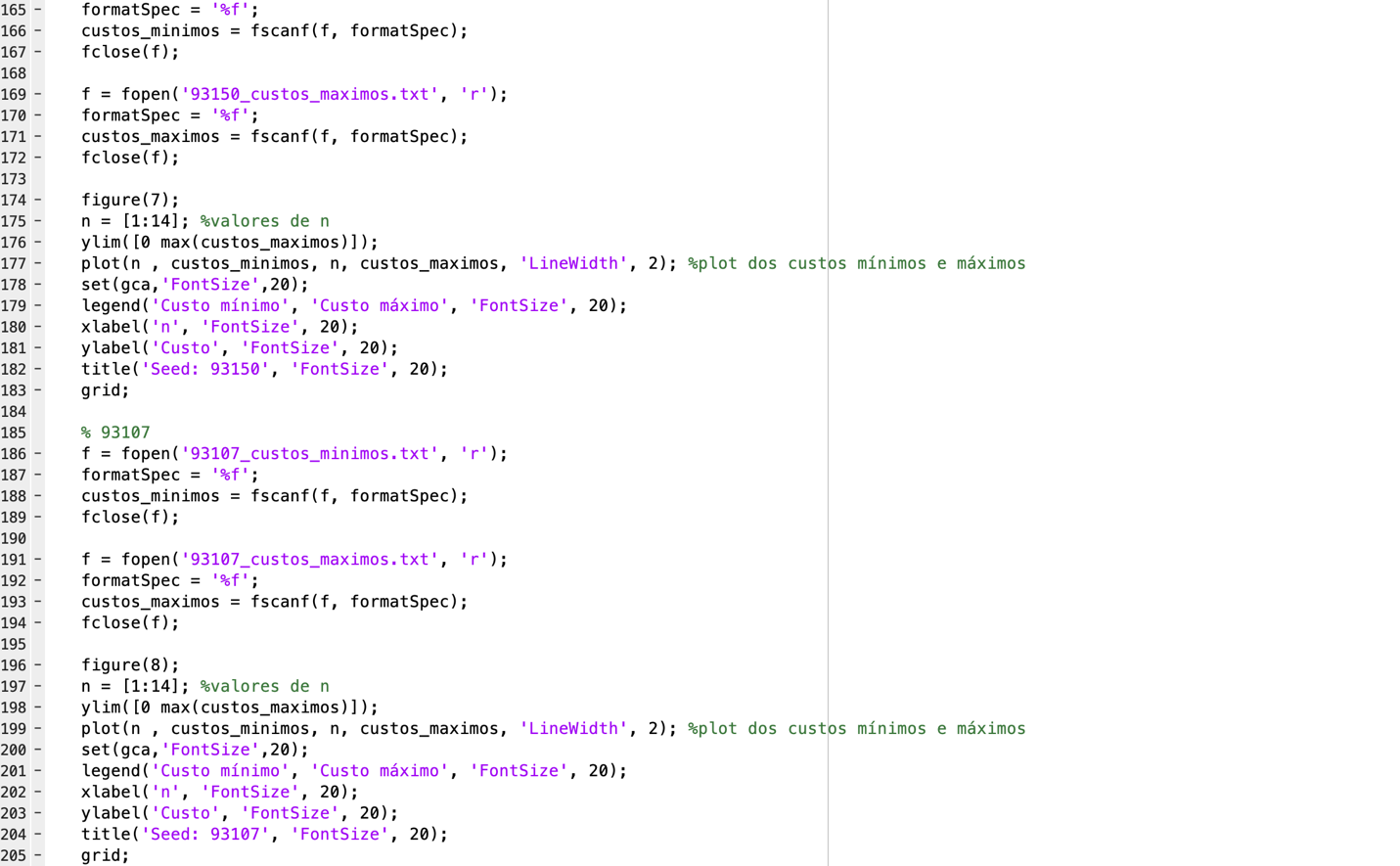


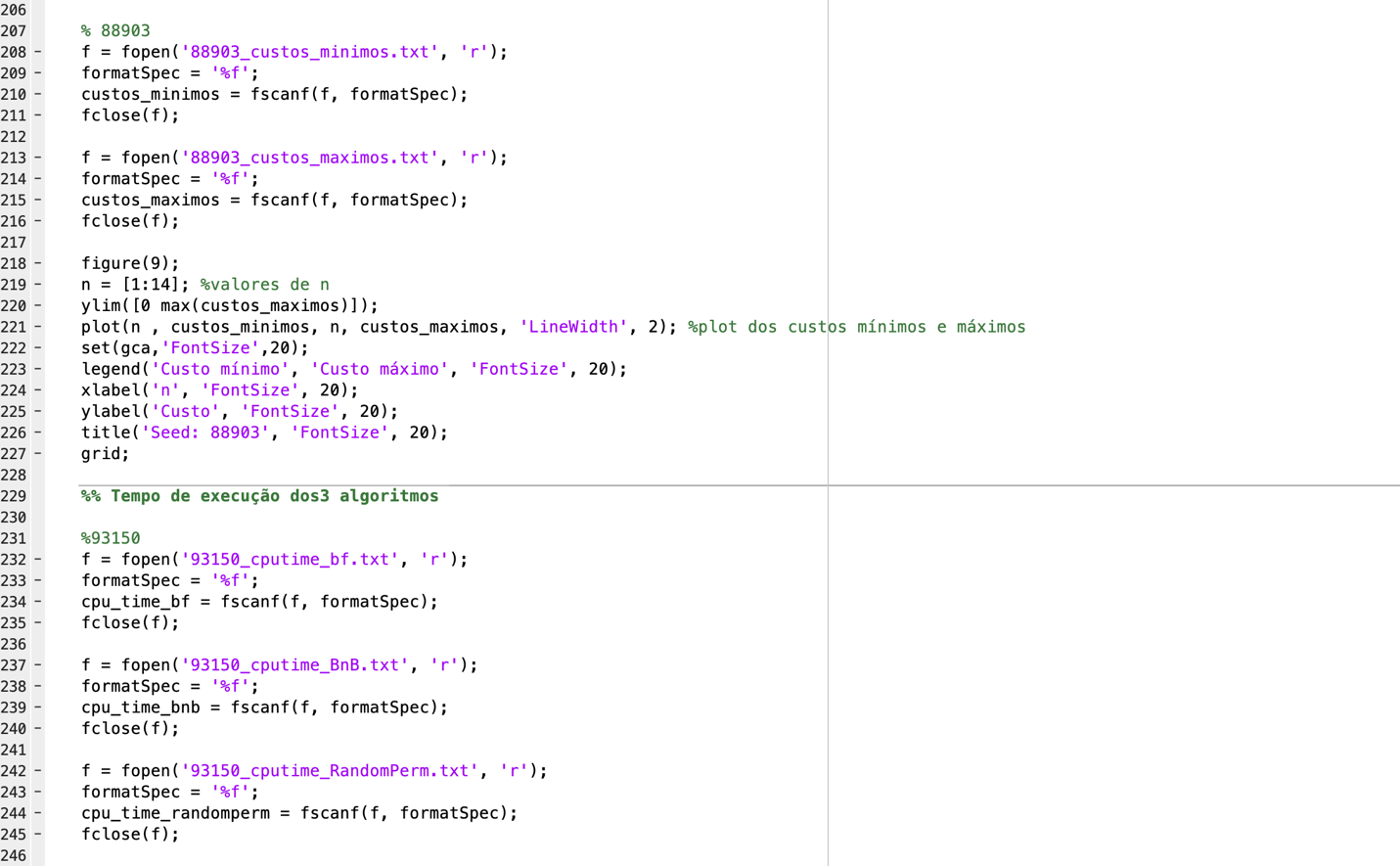




4.2 – Código implementado em Matlab (comentado com explicações)









5.

5.1 – Conclusões

Com este trabalho facilmente conseguimos perceber que o melhor algoritmo utlizado é o de Branch and Bound. Isto porque nos dá o resultado 100% certo (o random permutations não é) e muito mais eficiente que o brute force.

Resumidamente, gostamos de trabalhar neste projeto pois foi uma experiência enriquecedora tanto a nível das linguagens de programação C e Matlab como a nível do trabalho em equipa e gestão do tempo.