Algoritmos e Estruturas de Dados

Primeiro Trabalho Prático

The Assignment Problem

9 de Novembro de 2019

Trabalho Realizado por:

* Daniel Gomes, Nmec 93015, Contributo:50%
* Rui Fernandes, Nmec 92952, Contributo: 50%

Índice

**Introdução3**

O que é o Assignment Problem?3

**Métodos Utilizados4**

Brute Force4

Branch and Bound4

Random Permutations5

Outros Métodos e Notas Interessantes5

**Resultados Obtidos7**

Comparação de Eficiência dos Métodos7

Análise de Histogramas9

**Código Desenvolvido11**

Asignment.c11

Código Matlab20

1. Introdução
   1. O que é o Assignment Problem?

O objetivo do primeiro trabalho prático foi estudar e analisar o “*Assignment Problem”.* Neste problema temos por base um conjunto deagentes e outro de tarefas, no caso específico do nosso estudo analisamos o “*Balanced Assignment Problem”* implicando que os conjuntos têm o mesmo número de elementos, denominado por ***n***.

Atribuir um agente a uma tarefa tem um determinado custo, isto para cada uma das combinações agente-tarefa. Tendo em conta que cada tarefa só pode ser atribuída uma vez, o objetivo do problema é atribuir um agente a cada tarefa de tal modo que o custo total (soma dos custos de cada atribuição) seja o mínimo possível.

1. Métodos Utilizados
   1. Brute Force

Um método de simples implementação do problema, denominado “*Brute Force”,* baseia-se em percorrer todas as permutações possíveis de agentes-tarefas e encontrar a que corresponde ao custo total mínimo (sendo também fácil encontrar o custo total máximo).

Pode-se definir em 4 passos o algoritmo genérico de *“Brute Force”*:

“

1. *primeiro* (*P*): gera o primeiro candidato à solução de *P*.
2. *próximo* (*P*, *c*): gera o próximo candidato de *P* depois de *c*.
3. *válido* (*P*, *c*): verifica-se o candidato *c* é a solução de *P*.
4. *saída* (*P*, *c*): usa a solução *c* de *P* como for conveniente para a aplicação. “

Citado de [https://pt.wikipedia.org/wiki/Busca\_por\_força\_bruta](https://pt.wikipedia.org/wiki/Busca_por_for%C3%A7a_bruta)

Este método tem o benefício de ser muito fácil de implementar em código e chegar sempre ao valor certo, no entanto é extremamente demoroso, tendo uma complexidade algorítmica de O(n\*n!).

* 1. Branch and Bound

O *“Branch and Bound Algorithm”* é uma técnica que pode ser aplicada a diversos tipos de problemas nomeadamente na área de otimização combinatória, onde o problema em causa, *“Assignment Problem”*, é um exemplo.

Este método assemelha-se a princípios que caracterizam o *Brute Force* mas com a ligeira diferença de que não são analisados os casos que, à partida, já são do nosso conhecimento que nunca poderão ser solução. Assim, são, portanto, casos descartáveis para solucionar o problema em causa, pelo que apenas consideramos as opções que têm a possibilidade de serem candidatas a ser a solução ou que preencham alguns requisitos iniciais necessários para tal.

Ao contrário do *“Brute Force”*, o “*Branch and Bound”* não apresenta uma complexidade computacional definida, segundo os *computer scientists.* Isto acontece porque calcular a complexidade deste algoritmo é um processo extremamente difícil e que pode ser variável dependendo da sua aplicação.

* 1. Random Permutations

Um método diferente de resolver o problema é ao invés de percorrer todas as permutações possíveis, gerar um número fixo (no nosso estudo utilizámos 1e6) de permutações aleatórias e calcular o custo mínimo utilizando as mesmas.

Apesar deste método ter uma complexidade algorítmica extremamente menor (ou seja, é muito menos demoroso), não garante que se encontre o verdadeiro custo mínimo, apenas próximo.

Quando o número de permutações aleatórias é consideravelmente maior que o número total de permutações possíveis é muito provável o valor calculado ser o custo mínimo real e, pelo contrário, quanto menor for mais distante fica o valor calculado do valor real.

É de notar que se se aumentar este valor fixo de permutações para igualar o número total, torna-se mais demoroso e menos eficiente criar e analisar mais permutações.

* 1. Outros Métodos e Notas Interessantes

Por falta de tempo e nalguns casos muita dificuldade de implementação ou teste, não testámos os seguintes métodos e soluções do *“Assignment Problem”*, no entanto achámos importante referi-los neste relatório.

Um dos métodos mais eficiente a resolver o *“Assignment Problem”,* com uma complexidade algorítmica de O(n^3), é o *“Hungarian Algorithm”* desenvolvido por Harold Khun.

O algoritmo assenta nos seguintes passos:

1. For each row of the matrix, find the smallest element and subtract it from every element in its row.
2. Do the same (as step 1) for all columns.
3. Cover all zeros in the matrix using minimum number of horizontal and vertical lines.
4. *Test for Optimality:* If the minimum number of covering lines is n, an optimal assignment is possible and we are finished. Else if lines are lesser than n, we haven’t found the optimal assignment, and must proceed to step 5.
5. Determine the smallest entry not covered by any line. Subtract this entry from each uncovered row, and then add it to each covered column. Return to step 3.

Citado de

<https://www.geeksforgeeks.org/hungarian-algorithm-assignment-problem-set-1-introduction/>

Um teste muito interessante de efetuar seria o Teste de Kolmogorov-Smirnov, onde a estatística utilizada para o teste é :





onde:



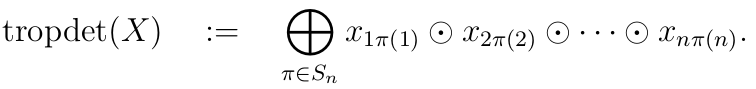
e em que***sup*** é osupremo do conjunto de distâncias.

Este teste observa a máxima diferença absoluta entre a função de distribuição acumulada assumida para os dados, e a função de distribuição empírica dos dados. Como critério, comparamos esta diferença com um valor crítico, para um dado nível de significância.

Adaptado de

<http://www.portalaction.com.br/inferencia/62-teste-de-kolmogorov-smirnov>

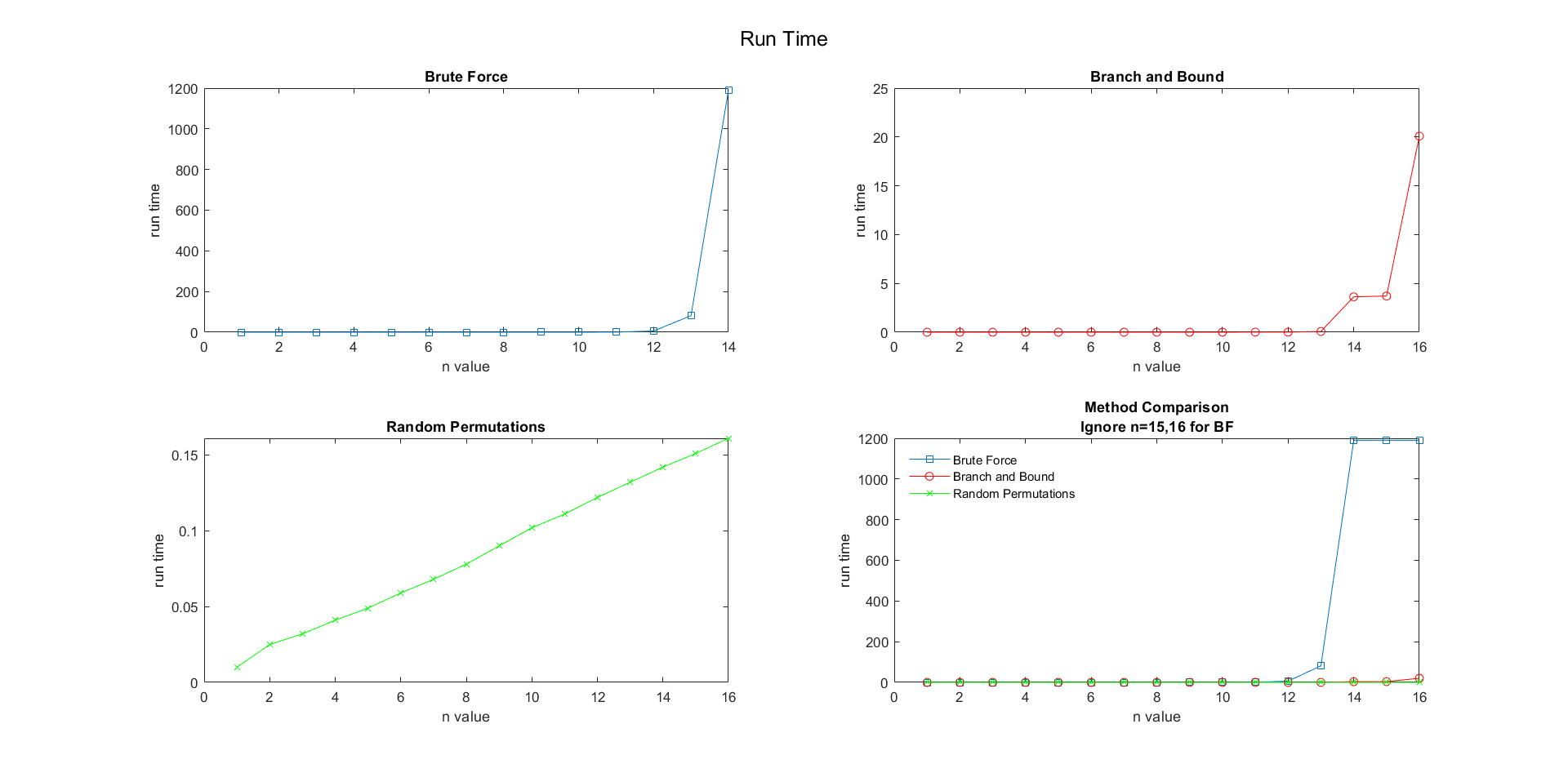
Por fim, é importante mencionar que o Assignment Problem pode ser associado a *Tropical Geometry* ou *Max-plus Algebra,* onde o custo mínimo total para cada caso é de facto o determinante trópico de uma matriz ***X***(***n*** por ***n***) :

**

Adaptado de [https://personal-homepages.mis.mpg.de](https://personal-homepages.mis.mpg.de/michalek/may22.pdf)

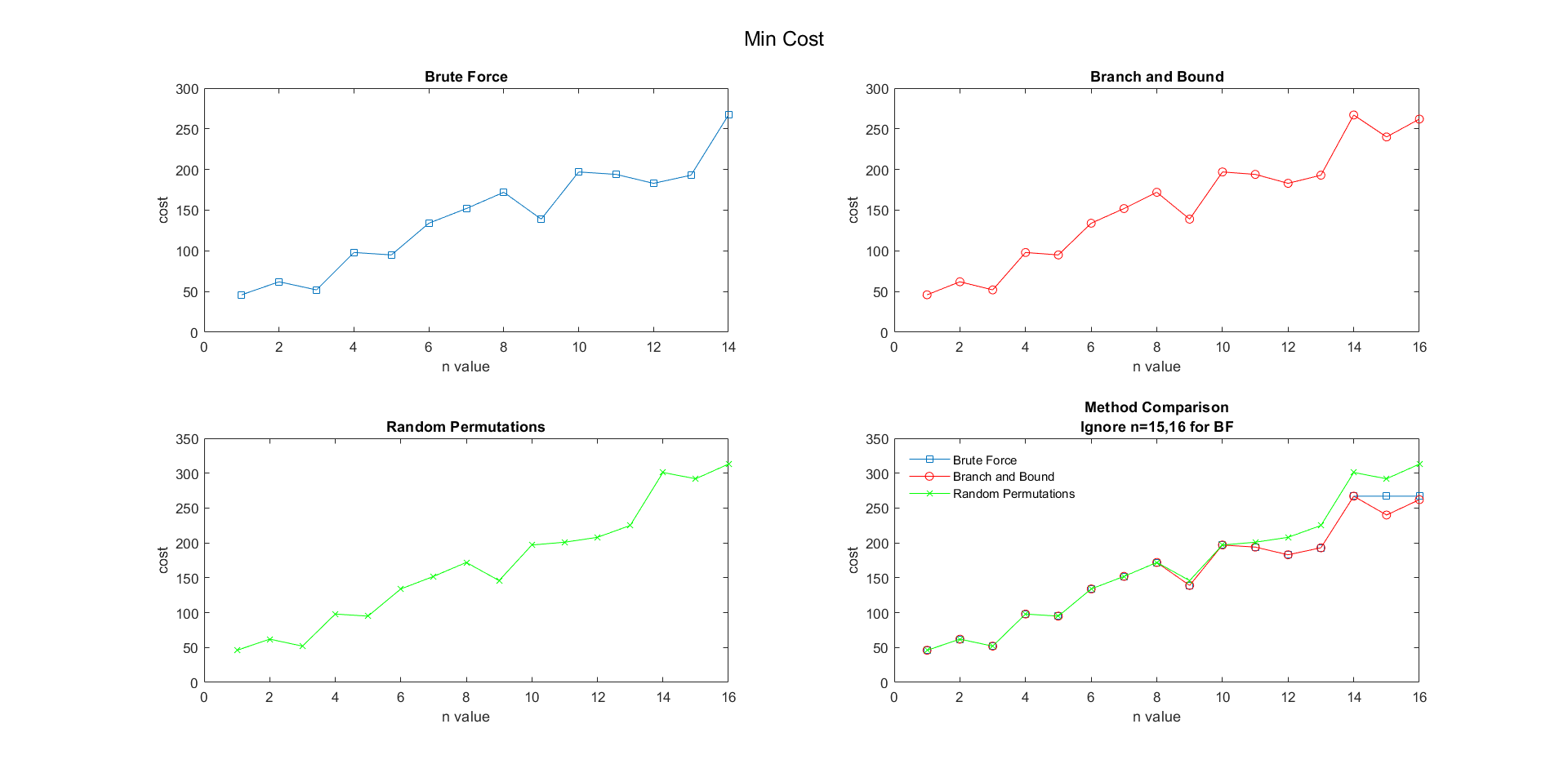
1. Resultados Obtidos
   1. Comparação de eficiência dos métodos

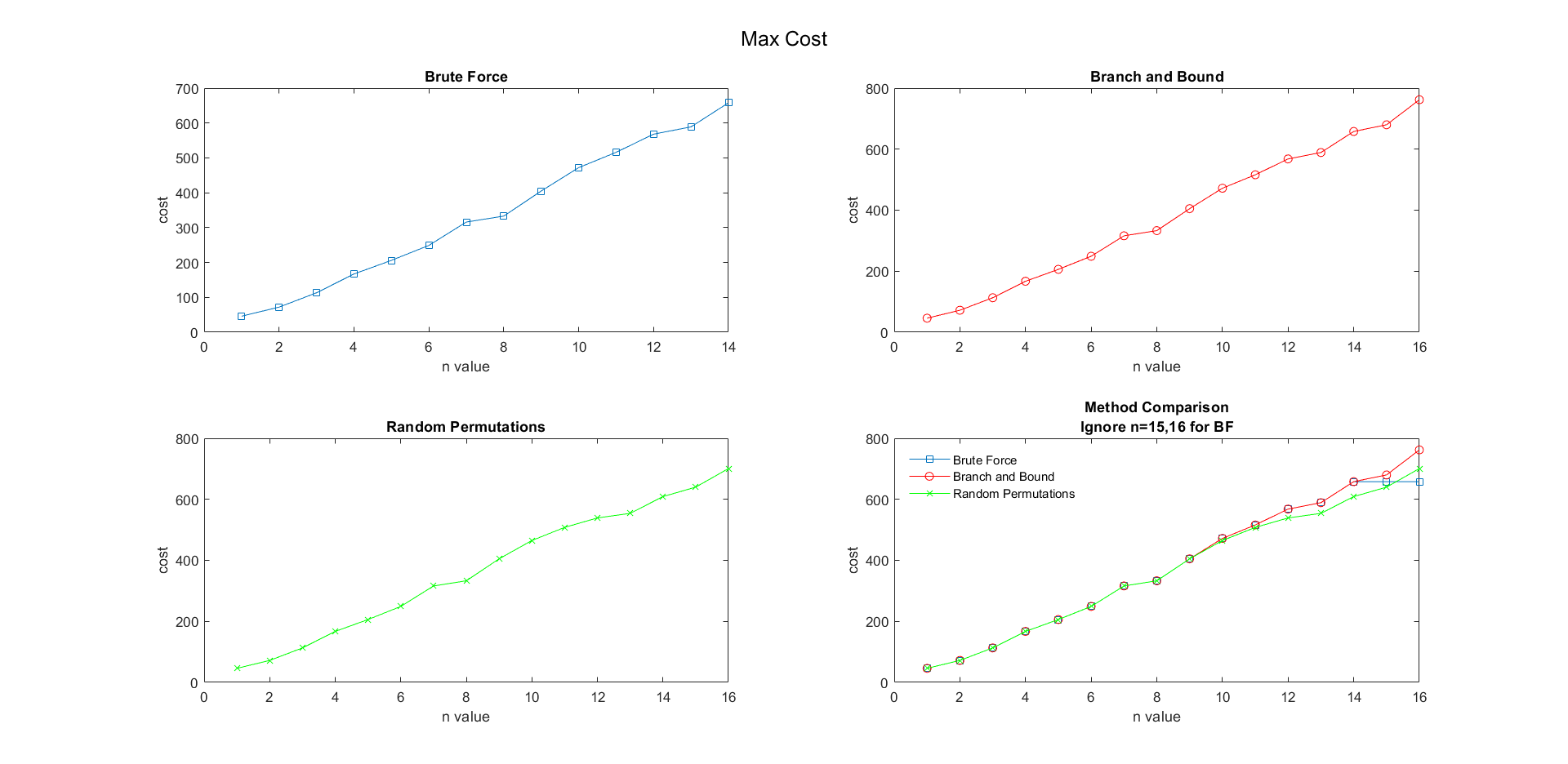
Deve-se notar que os seguintes resultados de Run Time e dos custos mínimos e máximos foram efetuados usando a seed 92952.



Como é possível observar nos gráficos de *run time*(s), confirma-se que o método de *“Brute Force”*é de longe o que se torna mais demorado com o aumento de ***n***. Para ***n***=14 já tem um *run time* de cerca de 1200s, ou seja, 20 minutos, enquanto que o *“Branch and Bound”* demora menos de 5 segundos. Confirma-se ainda que o gráfico do *“Brute Force”* tem de facto a forma aproximada de **n\*n!.**

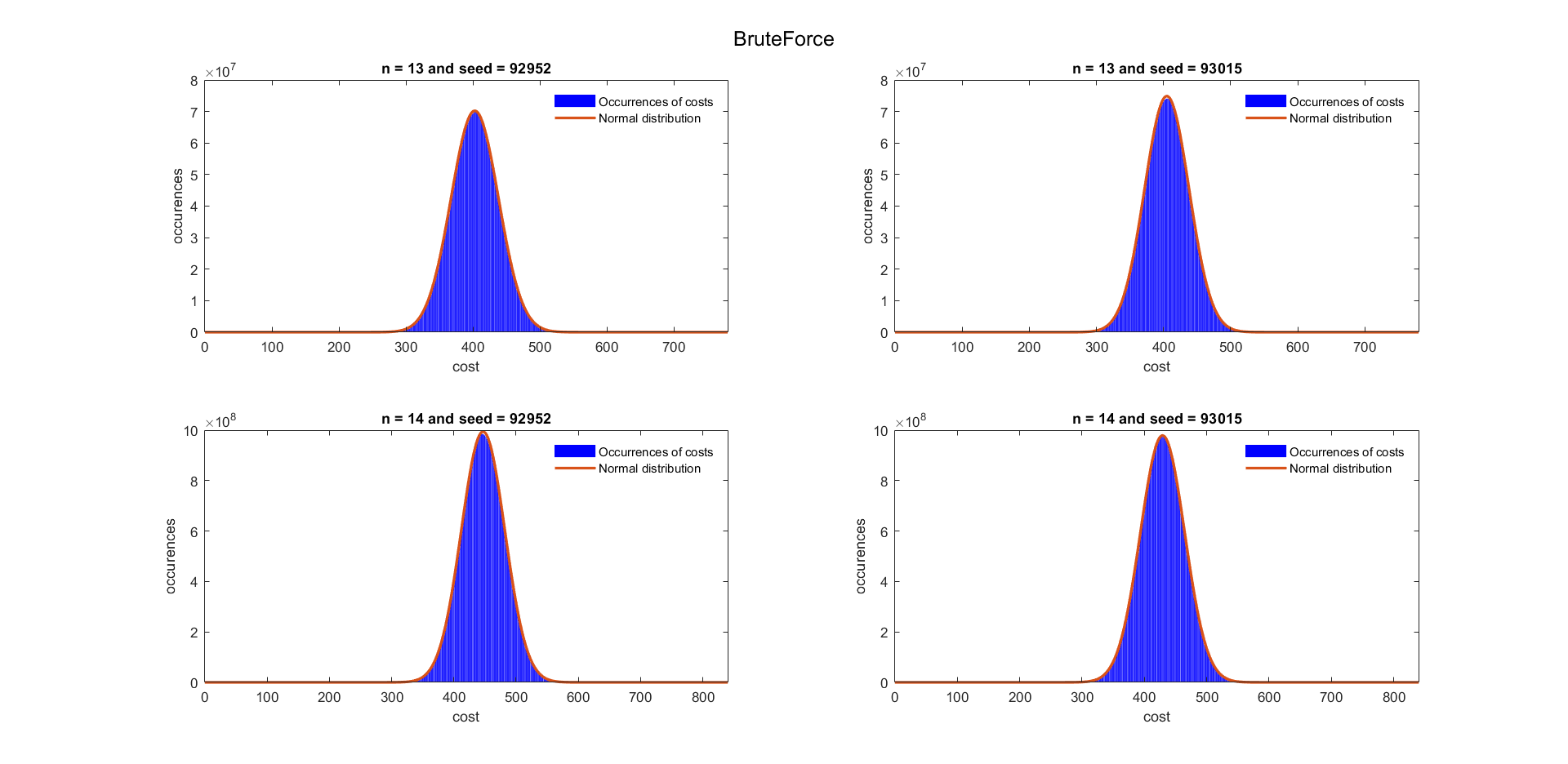
Analisando o gráfico das *“Random Permutations”* denota-se que se assemelha em forma com uma função de ordem n, e é como esperado a que se mantem com um *run time* mais baixo á medida que o valor de ***n*** aumenta.



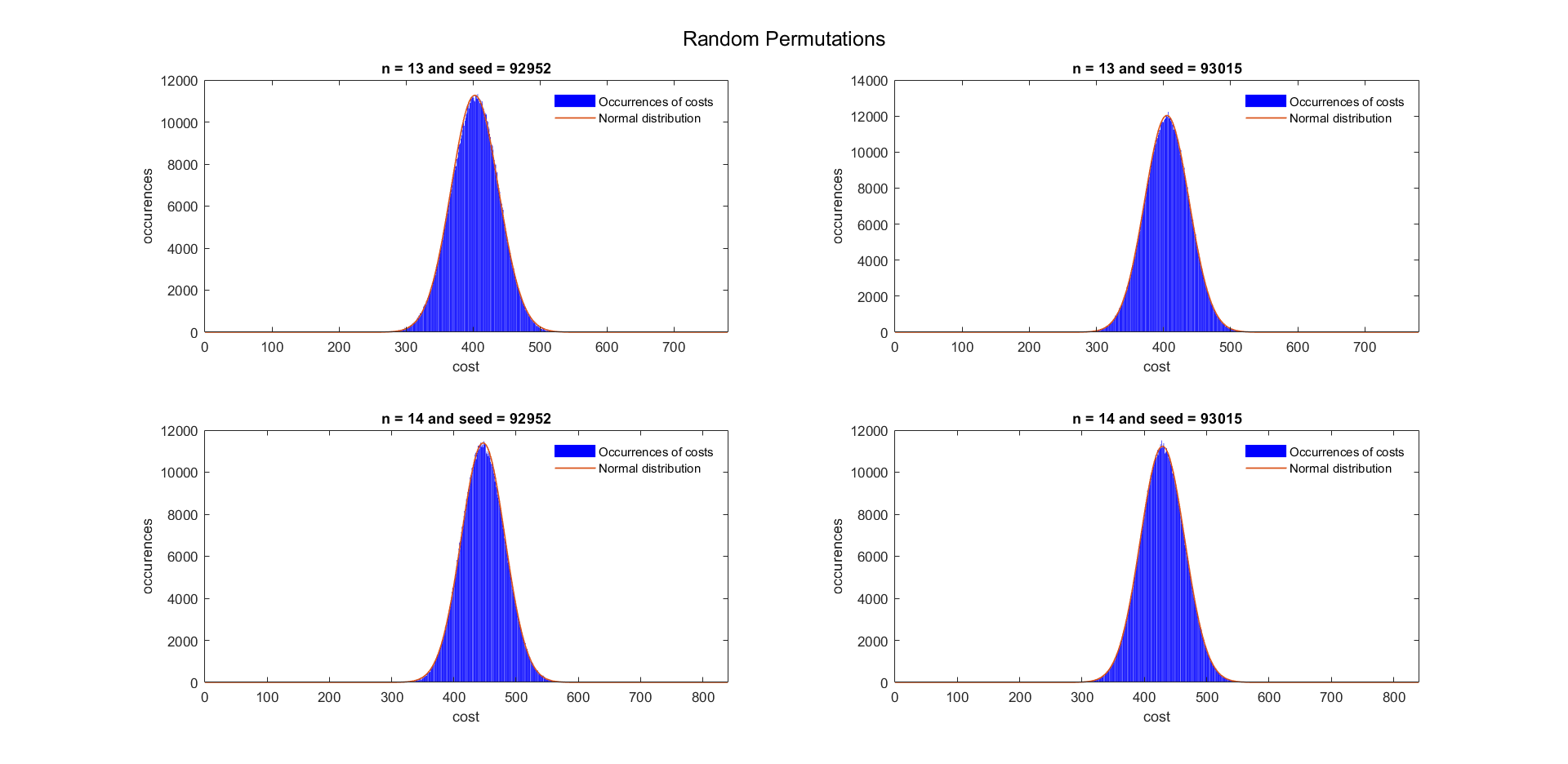


Nos gráficos acima, estão representados os custos mínimos e máximos calculados pelos diversos métodos com o crescer de ***n***. Confirma-se que sendo os valores encontrados pelo *“Brute Force”* os verdadeiros, o *“Branch and Bound”* encontra precisamente os mesmos valores. Acontece ainda o afastamento dos valores encontrados pelas *“Random Permutations”* dos valores reais “fugindo” a estes cada vez mais á medida que o ***n*** aumenta.

* 1. Análise de Histogramas



Um gráfico interessante de analisar é o histograma do número de ocorrências de cada custo possível para um dado n (neste caso quando igual a 13 e 14). Principalmente para valores de n superior onde existe um numero considerável de permutações, o histograma tende para seguir a forma de uma distribuição normal, praticamente coincidindo com a mesma.



No caso das *“Random Permutations”* observa-se algo semelhante exceto que a forma se torna menos uniforme, “fugindo” mais á Gaussiana, isto era expectável visto que se trata de uma amostra muito menor de permutações ao invés de todas as hipóteses.

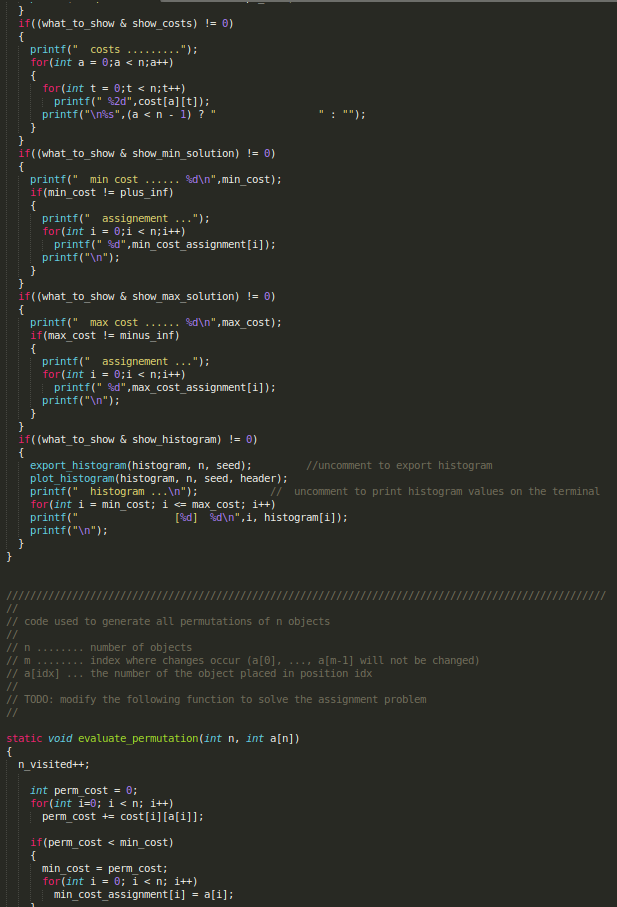
1. Código Desenvolvido
   1. Assignment.c

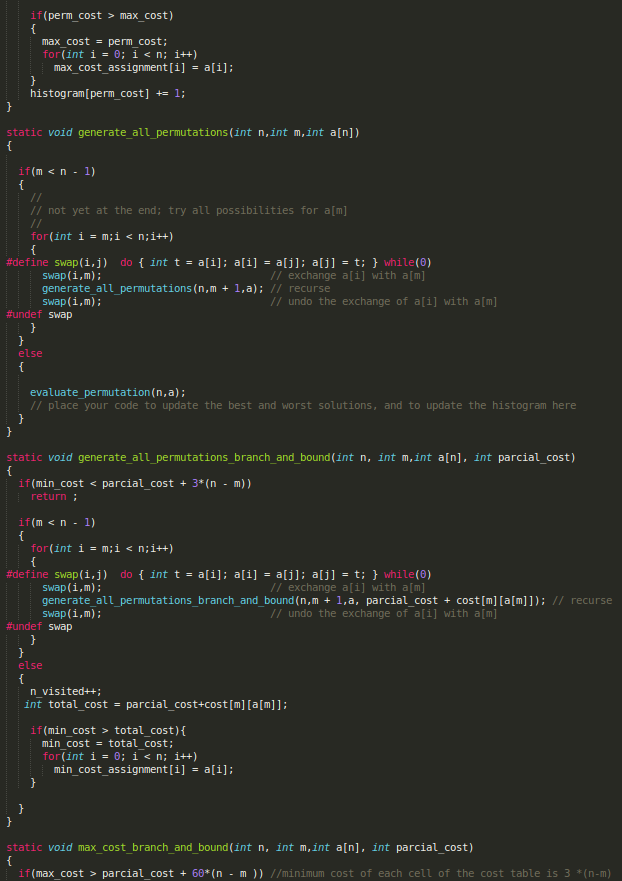
A maior parte deste código foi nos disponibilizado numa das aulas práticas ,sendo a nossa base do projeto, e cabendo a nós completar a função *generate\_all\_permutations* (de forma a obter o custo máximo e mínimo para cada n e guardar os dados para conseguir obter histogramas), assim como criar a função *generate\_all\_permutations\_branch\_bound* para implementação do *branch and bound,* e chamar a função *random\_permutation* que já tinha sido disponibilizada *.* Como forma de compactar o código foi criada a função *evaluate\_permutation*.Além disto  foi gerada a função *export\_histogram* como forma de poderem ser exportados, para o *Matlab,* os dados dos histogramas e gráficos analisados anteriormente neste Relatório. Por fim, para que fosse possível gerar automaticamente os histogramas pretendidos assim que fosse corrido o código recorremos ao *GnuPlot (*função *plot\_histogram*).

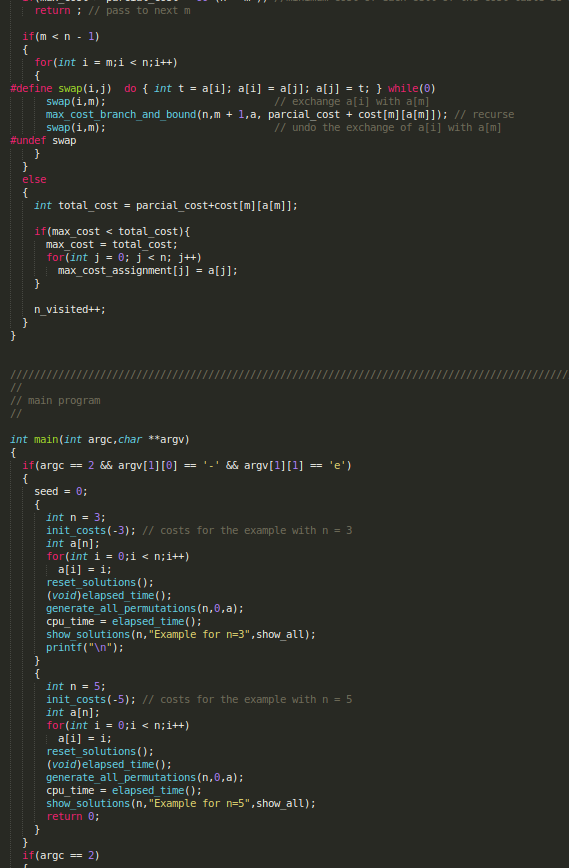
**

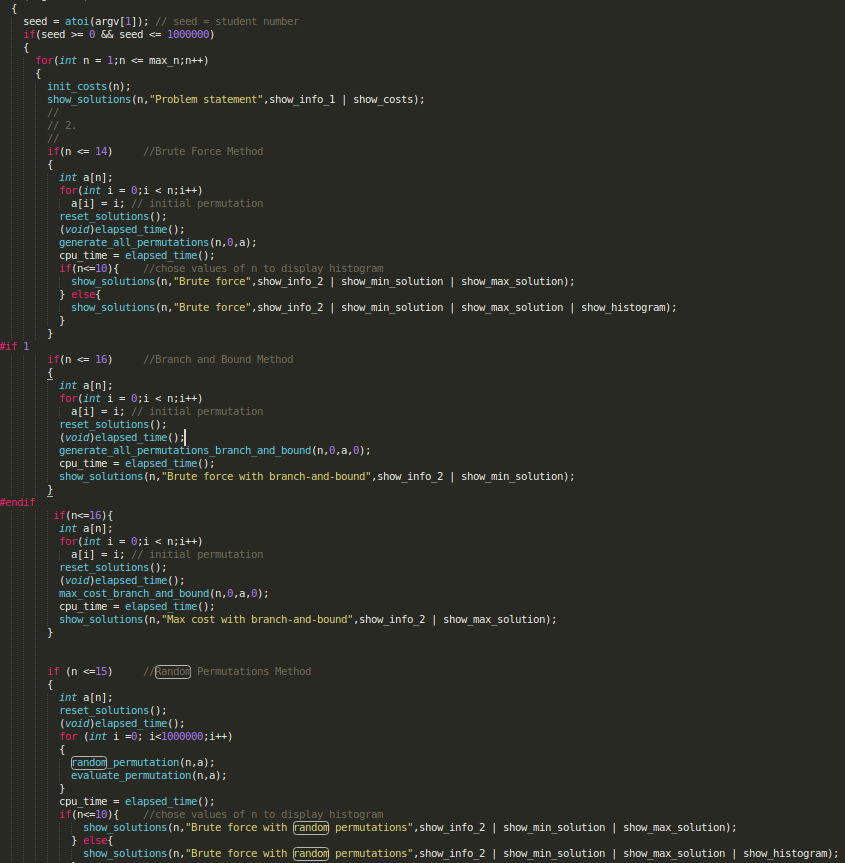
**

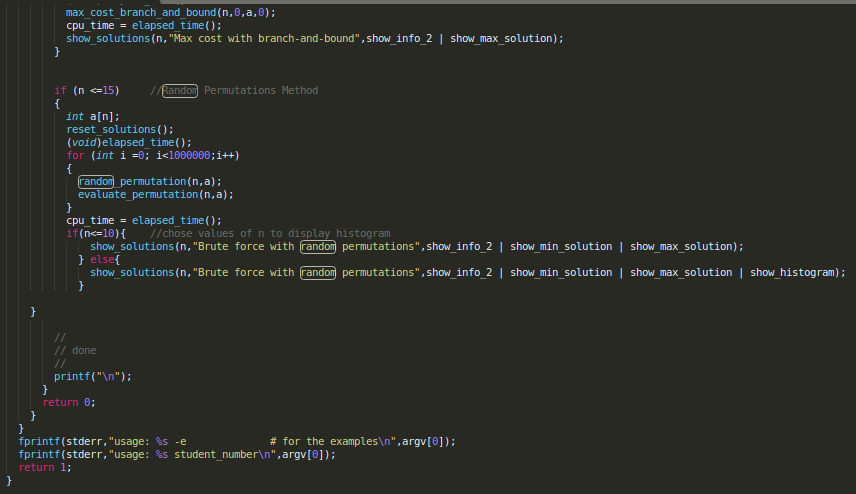
**

**

**

**

**

**

* 1. Código Matlab

