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

1.1. – 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

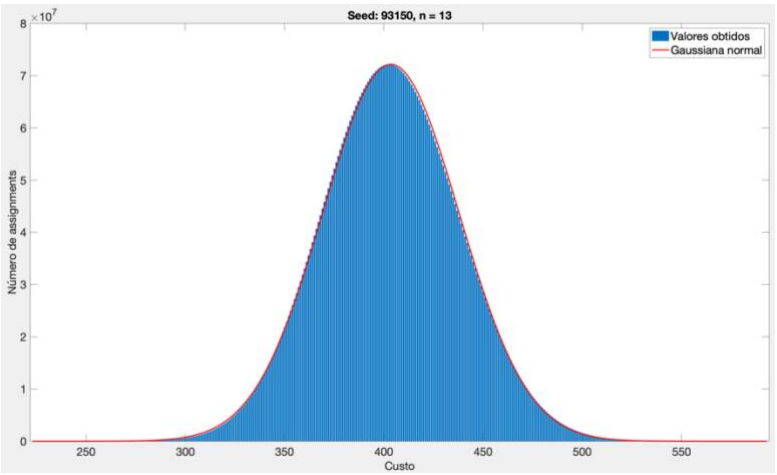
Para este algoritmo, vamos escolher aleatoriamente um valor elevado de permutações e calcular a permutação com o custo mínimo. Basicamente, para cada permutação escolhida calculamos o custo e verificamos se é menor que o mínimo que tínhamos e, se for, substituímos o custo mínimo e o seu assignment correspondente.

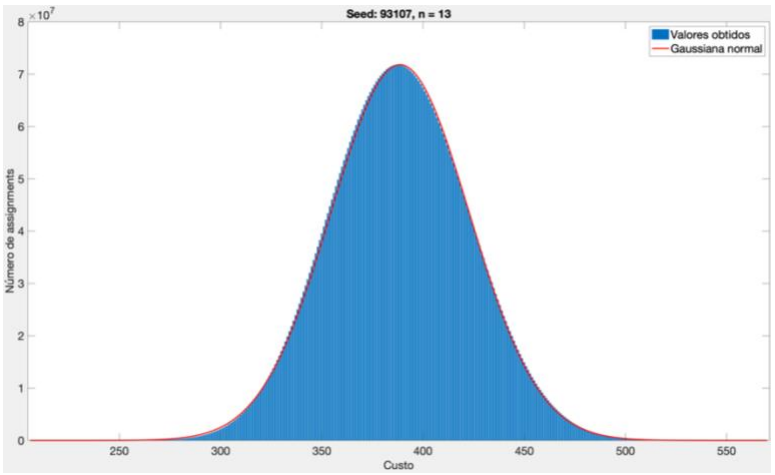
No código, usamos a função (dada pelo professor) random_permutation que retorna uma permutação aleatória e calculamos, para 1 milhão de permutações aleatórias, o custo mínimo e máximo e os seus assignments correspondentes.

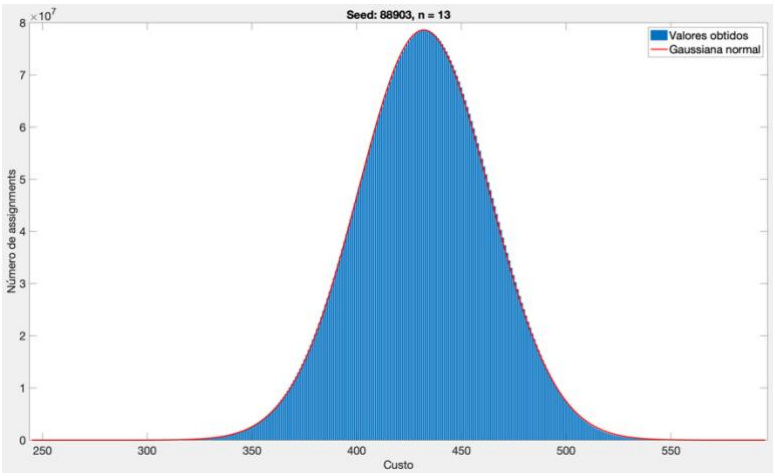
3.

3.1 – Resultados

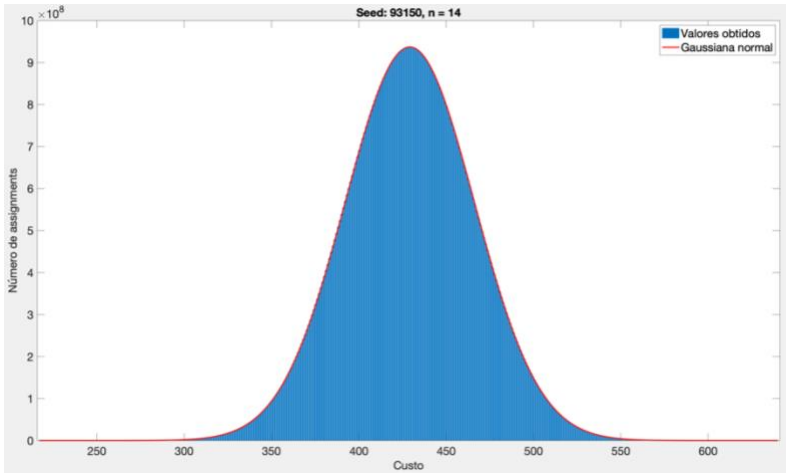
Histogramas e Gaussianas para cada número mecanográfico com n=13:

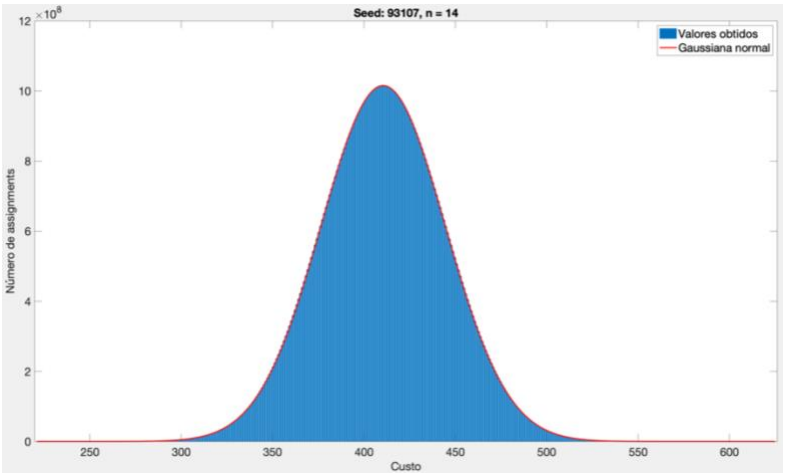


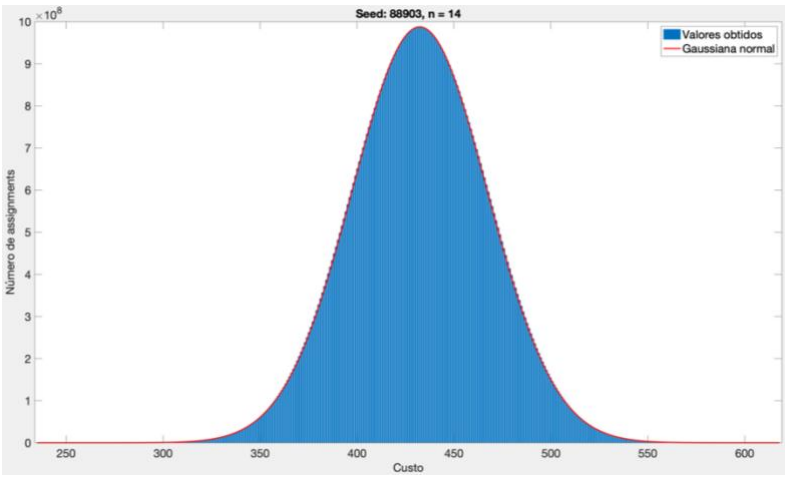




Histogramas e Gaussianas para cada número mecanográfico com n=14:

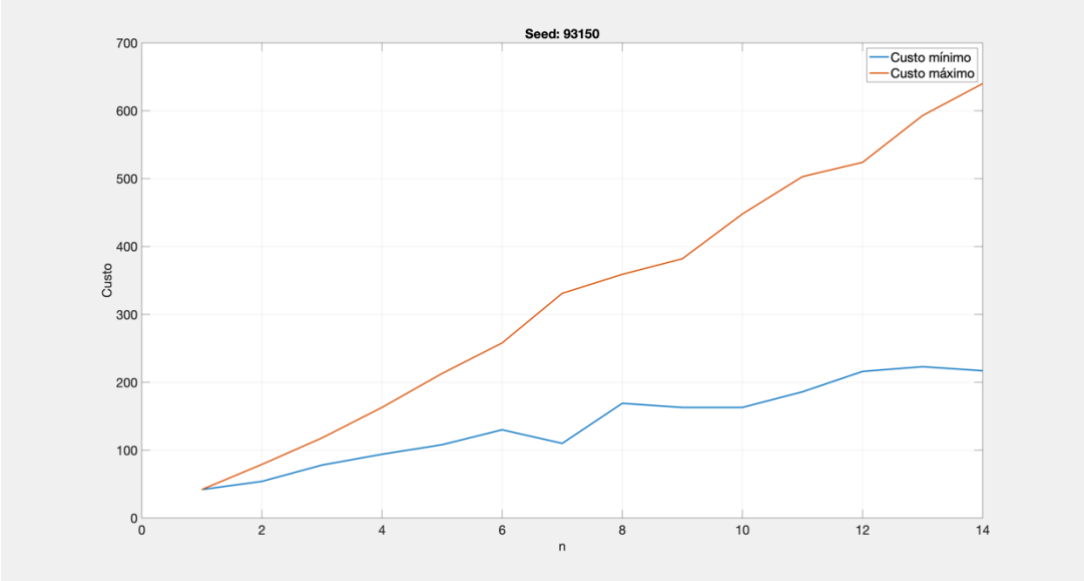


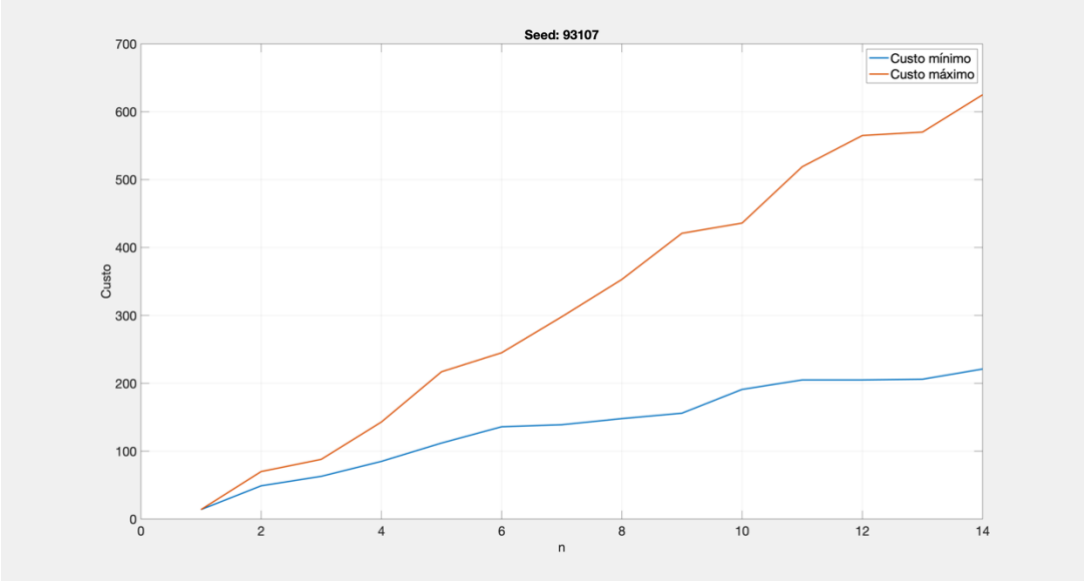


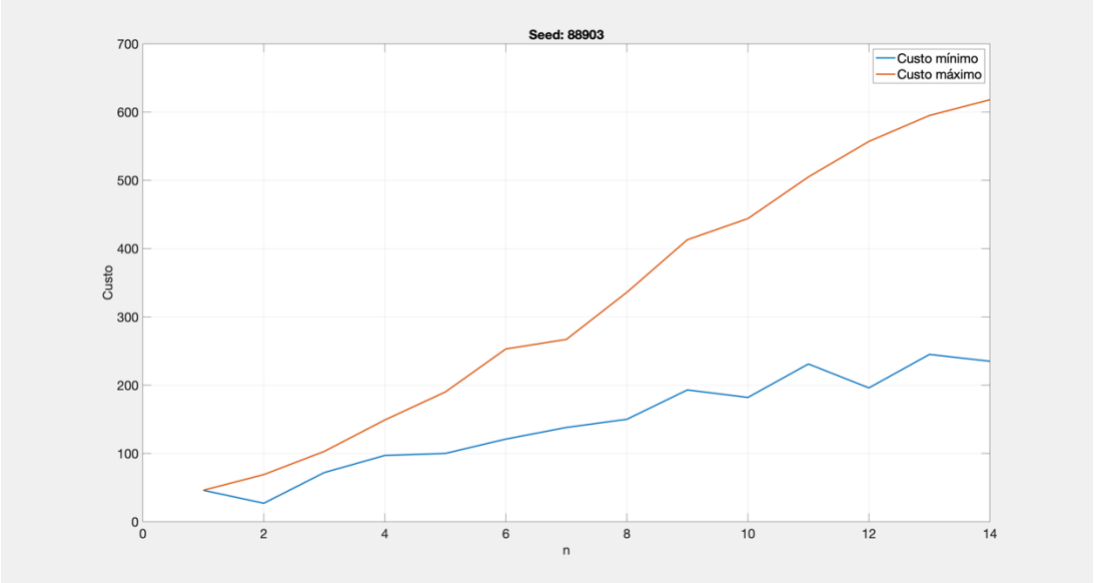


Como se pode verificar, os histogramas do número de assignments em função do custo estão muito semelhantes aos valores teóricos (representados pelas Gaussianas normais).

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

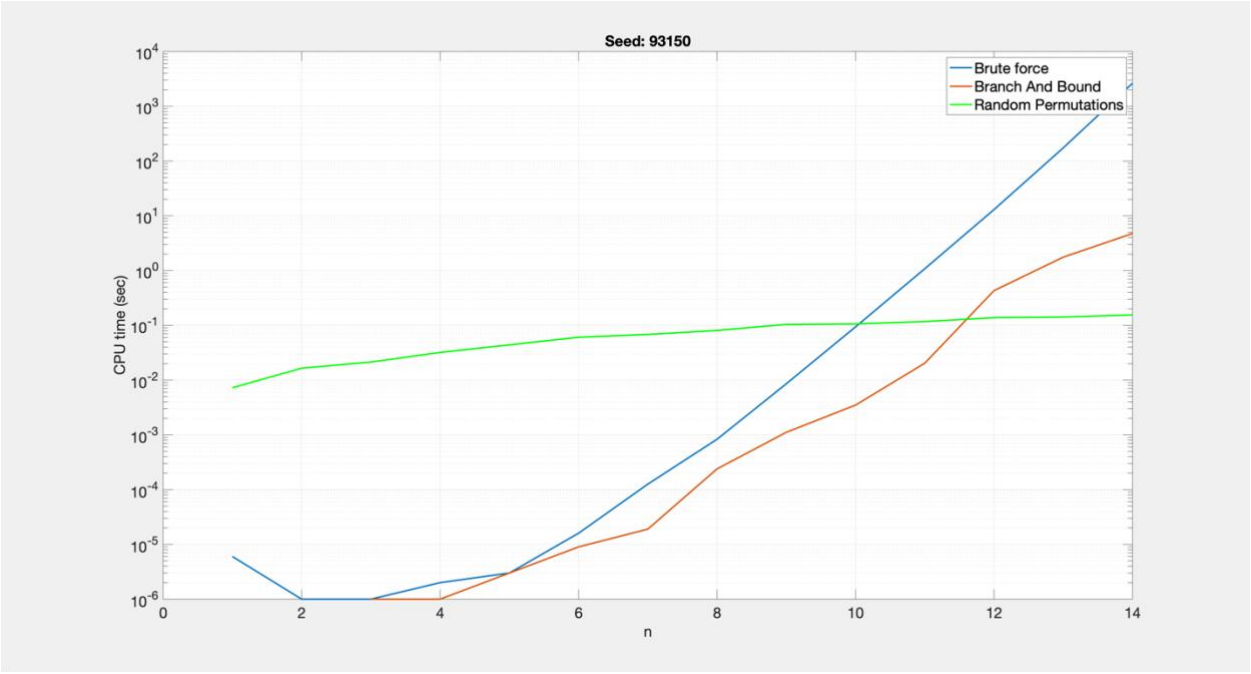


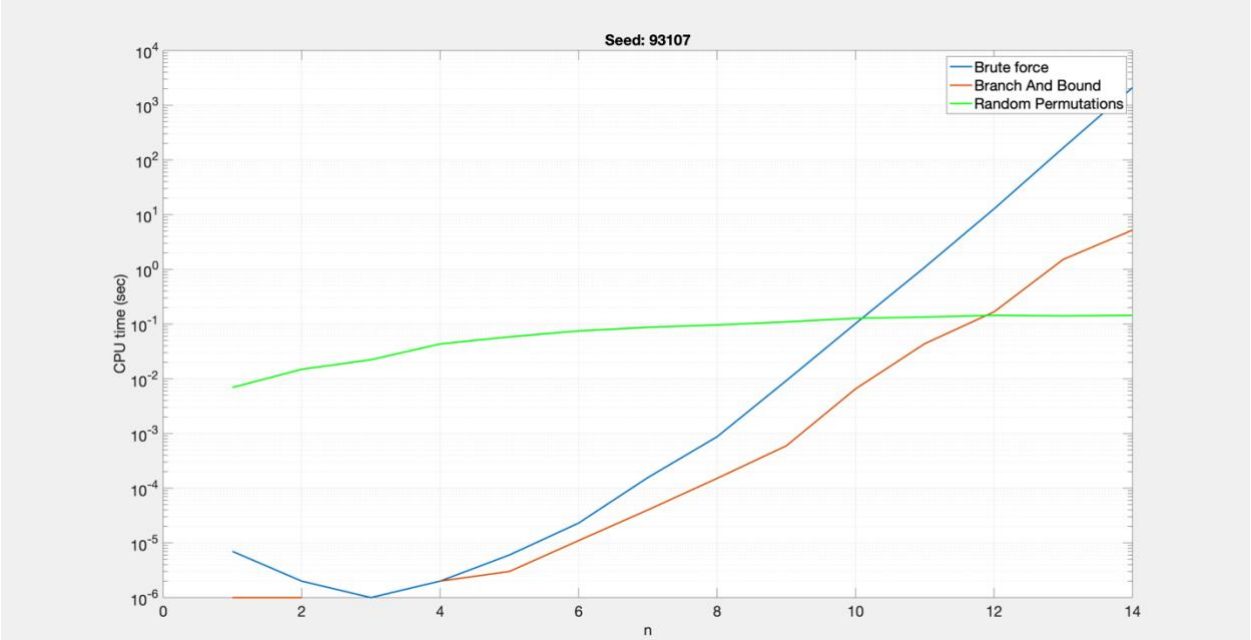


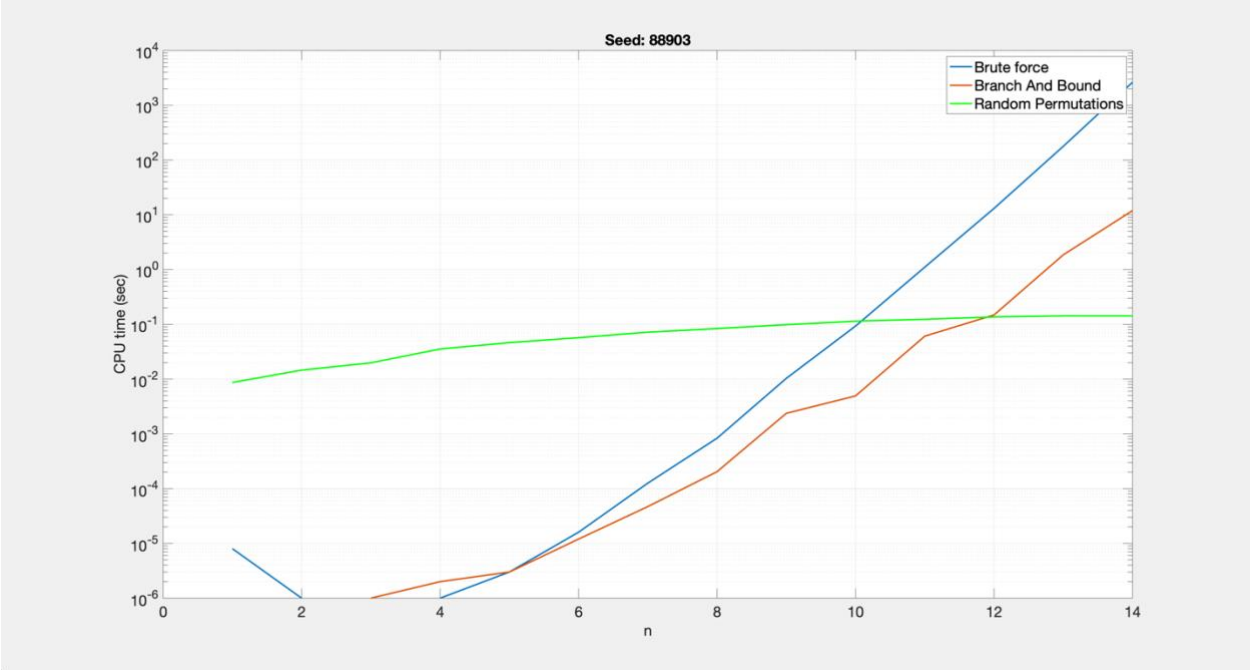


Quanto aos custos mínimos e custos máximos, quanto maior for o n, mais diferença há entre eles, como seria de esperar. É também visível que a função do custo máximo cresce muito mais rapidamente que a do custo mínimo.

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).

```c
////////////////////////////////////////////////////////////////////////////////////////
//
// AED, 2019/2020
//
// Pedro Bastos 93150
// Eduardo Santos 93107
// José Vaz 88903
//
// Brute-force solution of the assignment problem (https://en.wikipedia.org/wiki/Assignment_problem)
//
// Compile with "cc -Wall -O2 assignment.c -lm" or equivalent
//
// In the assignment problem we will solve here we have n agents and n tasks; assigning agent
//   a
// to task
//   t
// costs
//   cost[a][t]
// The goal of the problem is to assign one agent to each task such that the total cost is minimized
// The total cost is the sum of the costs
//
// Things to do:
//   0. (mandatory)
//      Place the student numbers and names at the top of this file   [DONE]
//   1. (highly recommended)
//      Read and understand this code [DONE]
//   2. (mandatory)
//      Modify the function generate_all_permutations to solve the assignment problem [DONE]
//      Compute the best and worst solutions for all problems with sizes n=2,...,14 and for each
//      student number of the group
//   3. (mandatory)
//      Calculate and display an histogram of the number of occurrences of each cost [DONE]
//      Does it follow approximately a normal distribution?
//      Note that the maximum possible cost is n * t_range
//   4. (optional)
//      For each problem size, and each student number of the group, generate one million (or more!)
//      random permutations and compute the best and worst solutions found in this way; compare
//      these solutions with the ones found in item 2
//      Compare the histogram computed in item 3 with the histogram computed using the random
//      permutations
//   5. (optional)
//      Try to improve the execution time of the program (use the branch-and-bound technique)
//   6. (optional)
//      Surprise us, by doing something more!
//   7. (mandatory)
//      Write a report explaining what you did and presenting your results
//

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

//#define NDEBUG  // uncomment to skip disable asserts (makes the code slightly faster)
#include <assert.h>


////////////////////////////////////////////////////////////////////////////////////////
//
// problem data
//
// max_n ........ maximum problem size
// cost[a][t] ... cost of assigning agent a to task t
//


//
// if your compiler complains about srandom() and random(), replace #if 0 by #if 1
//
#if 0
# define srandom srand
# define random  rand
#endif

#define max_n    32          // do not change this (maximum number of agents, and tasks)
#define range    20          // do not change this (for the pseudo-random generation of costs)
#define t_range  (3 * range)  // do not change this (maximum cost of an assignment)

static int cost[max_n][max_n];
static int seed; // place a student number here!

static void init_costs(int n)
{
  if(n == -3)
  { // special case (example for n=3)
    cost[0][0] = 3; cost[0][1] = 8; cost[0][2] = 6;
    cost[1][0] = 4; cost[1][1] = 7; cost[1][2] = 5;
    cost[2][0] = 5; cost[2][1] = 7; cost[2][2] = 5;
    return;
  }
  if(n == -5)
  { // special case (example for n=5)
    cost[0][0] = 27; cost[0][1] = 27; cost[0][2] = 25; cost[0][3] = 41; cost[0][4] = 24;
    cost[1][0] = 28; cost[1][1] = 26; cost[1][2] = 47; cost[1][3] = 38; cost[1][4] = 21;
    cost[2][0] = 22; cost[2][1] = 48; cost[2][2] = 26; cost[2][3] = 14; cost[2][4] = 24;
    cost[3][0] = 32; cost[3][1] = 31; cost[3][2] =  9; cost[3][3] = 41; cost[3][4] = 36;
    cost[4][0] = 24; cost[4][1] = 34; cost[4][2] = 30; cost[4][3] = 35; cost[4][4] = 45;
    return;
  }
  assert(n >= 1 && n <= max_n);
  srandom((unsigned int)seed * (unsigned int)max_n + (unsigned int)n);
  for(int a = 0;a < n;a++)
    for(int t = 0;t < n;t++)
      cost[a][t] = 3 + (random() % range) + (random() % range) + (random() % range); // [3,3*range]
}

```

```
105
106 ////////////////////////////////////////////////////////////////////////////////////////////
107 //
108 // code to measure the elapsed time used by a program fragment (an almost copy of elapsed_time.h)
109 //
110 // use as follows:
111 //
112 //   (void)elapsed_time();
113 //   // put your code to be time measured here
114 //   dt = elapsed_time();
115 //   // put morecode to be time measured here
116 //   dt = elapsed_time();
117 //
118 // elapsed_time() measures the CPU time between consecutive calls
119 //
120
121 #if defined(__linux__) || defined(__APPLE__)
122
123 //
124 // GNU/Linux and MacOS code to measure elapsed time
125 //
126
127 #include <time.h>
128
129 static double elapsed_time(void)
130 {
131   static struct timespec last_time,current_time;
132
133   last_time = current_time;
134   if(clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&current_time) != 0)
135     return -1.0; // clock_gettime() failed!!!
136   return            ((double)current_time.tv_sec - (double)last_time.tv_sec)
137         + 1.0e-9 * ((double)current_time.tv_nsec - (double)last_time.tv_nsec);
138 }
139
140 #endif
141
142 #if defined(_MSC_VER) || defined(_WIN32) || defined(_WIN64)
143
144 //
145 // Microsoft Windows code to measure elapsed time
146 //
147
148 #include <windows.h>
149
150 static double elapsed_time(void)
151 {
152   static LARGE_INTEGER frequency,last_time,current_time;
153   static int first_time = 1;
154
155   if(first_time != 0)
156   {
157     QueryPerformanceFrequency(&frequency);
158     first_time = 0;
159   }
160   last_time = current_time;
161   QueryPerformanceCounter(&current_time);
162   return (double)(current_time.QuadPart - last_time.QuadPart) / (double)frequency.QuadPart;
163 }
164
165 #endif
166
167
168 ////////////////////////////////////////////////////////////////////////////////////////////
169 //
170 // function to generate a pseudo-random permutation
171 //
172
173 void random_permutation(int n,int t[n])
174 {
175   assert(n >= 1 && n <= 1000000);
176   for(int i = 0;i < n;i++)
177     t[i] = i;
178   for(int i = n - 1;i > 0;i--)
179   {
180     int j = (int)floor((double)(i + 1) * (double)random() / (1.0 + (double)RAND_MAX)); // range 0..i
181     assert(j >= 0 && j <= i);
182     int k = t[i];
183     t[i] = t[j];
184     t[j] = k;
185
186   }
187 }
188
189
190 ////////////////////////////////////////////////////////////////////////////////////////////
191 //
192 // place to store best and worst solutions (also code to print them)
193 //
194
195 static int min_cost,min_cost_assignment[max_n]; // smallest cost information
196 static int max_cost,max_cost_assignment[max_n]; // largest cost information
197 static long n_visited; // number of permutations visited (examined)
198 static int histogram[max_n*t_range];// place your histogram global variable here
199 static double cpu_time;
200
201 #define minus_inf  -1000000000  // a very small integer
202 #define plus_inf   +1000000000  // a very large integer
203
204 #include <string.h>
205
206 static void reset_solutions(void)
207 {
208   min_cost = plus_inf;
```

```
  max_cost = minus_inf;
  n_visited = 0l;
  memset(histogram, 0, max_n*t_range*sizeof(histogram[0])); // place your histogram initialization code here
  cpu_time = 0.0;
}

static void reset_solutions_without_histogram(void)
{
  min_cost = plus_inf;
  max_cost = minus_inf;
  n_visited = 0l;
  cpu_time = 0.0;
}

#define show_info_1        (1 << 0)
#define show_info_2        (1 << 1)
#define show_costs         (1 << 2)
#define show_min_solution  (1 << 3)
#define show_max_solution  (1 << 4)
#define show_histogram     (1 << 5)
#define show_all           (0xFFFF)

static void show_solutions(int n,char *header,int what_to_show)
{
  printf("%s\n",header);
  if((what_to_show & show_info_1) != 0)
  {
    printf("  seed ......... %d\n",seed);
    printf("  n ............ %d\n",n);
  }
  if((what_to_show & show_info_2) != 0)
  {
    printf("  visited ....... %ld\n",n_visited);
    printf("  cpu time ...... %.3fs\n",cpu_time);
  }
  if((what_to_show & show_costs) != 0)
  {
    printf("  costs .........");
    for(int a = 0;a < n;a++)
    {
      for(int t = 0;t < n;t++)
        printf(" %2d",cost[a][t]);
      printf("\n%s",(a < n - 1) ? "                " : "");
    }
    printf("\n");
  }
  if((what_to_show & show_min_solution) != 0)
  {
    printf("  min cost ...... %d\n",min_cost);
    if(min_cost != plus_inf)
    {
      printf("  assignement ...");
      for(int i = 0;i < n;i++)
        printf(" %d",min_cost_assignment[i]);
      printf("\n");
    }
  }
  if((what_to_show & show_max_solution) != 0)
  {
    printf("  max cost ...... %d\n",max_cost);
    if(max_cost != minus_inf)
    {
      printf("  assignement ...");
      for(int i = 0;i < n;i++)
        printf(" %d",max_cost_assignment[i]);
      printf("\n");
    }
    printf("\n");
  }
  if((what_to_show & show_histogram) != 0)
  {
    printf("  histogram  .... ");
    for(int i=min_cost; i <= max_cost; i++){
      printf("%s", (i != min_cost) ? "                " : ""); // para o print ficar alinhado
      printf("[%d] = %d",i, histogram[i]);
      printf("\n");
    }
  }
}




int custo_permutacao(int n, int assignment[n]) //calcula e retorna o custo de uma permutação, dado um assignment de tamanho n
{
  int custo = 0;

  for(int i=0;i <n; i++){
    custo += cost[i][assignment[i]]; //para cada linha (cost[i]) adiciona à variável 'custo' o valor de assignment[i]
  }

  return custo;
}


void write_file(int * numOcorrencias, char *nomeFicheiro){ // escreve num ficheiro os valores do histograma

    FILE *fp;

    char *output = nomeFicheiro;
    int tamanho = sizeof(histogram)/sizeof(histogram[0]);

    int tamanho2 = max_cost - min_cost; // tamanho do array dos custos

```

```
    int custos[tamanho2];
     for(int k=0; k <= tamanho2; k++){
       custos[k] = min_cost + k; // cria o array dos custos desde o mínimo custo até ao máximo custo
     }

     int numOcorrencias_semZeros[tamanho2];
     int indice = 0;
     for(int i=0; i <= tamanho; i++){
       if(numOcorrencias[i] != 0 || (numOcorrencias[i] == 0 && i >min_cost && i < max_cost)){ //retira os zeros do histograma que estão fora do intervalo de custos
         numOcorrencias_semZeros[indice] = numOcorrencias[i];
         indice++;
       }
     }


     fp=fopen(output,"w");


     for(int n=0;n <= tamanho2 ;n++) {

        fprintf(fp,"%d\t%d\n", custos[n], numOcorrencias_semZeros[n]);    // escreve os 2 arrays no ficheiro, separados por tab
     }



     fclose(fp);
}

void write_file_custos(int * array, char *nomeFicheiro){ // escreve no ficheiro o array dos custos mínimos/ máximos

     FILE *fp;

     char *output = nomeFicheiro;

     int n;

     fp=fopen(output,"w");

     for(n=0;n < 14;n++) {

        fprintf(fp,"%d\n", array[n]);

     }

     fclose(fp);


}

void write_file_cputime(double * array, char *nomeFicheiro){ // escreve no ficheiro  o array dos tempos de execução dos algoritmos

     FILE *fp;

     char *output = nomeFicheiro;

     int n;

     fp=fopen(output,"w");

     for(n=0;n < 14;n++) {

        fprintf(fp,"%f\n", array[n]);

     }

     fclose(fp);


}

////////////////////////////////////////////////////////////////////////////////////////////////
//
// code used to generate all permutations of n objects
//
// n ........ number of objects
// m ........ index where changes occur (a[0], ..., a[m-1] will not be changed)
// a[idx] ... the number of the object placed in position idx
//
// TODO: modify the following function to solve the assignment problem
//



static void generate_all_permutations(int n,int m,int a[n])
{
  if(m < n - 1)
  {
    //
    // not yet at the end; try all possibilities for a[m]
    //
    for(int i = m;i < n;i++)
    {
#define swap(i,j)  do { int t = a[i]; a[i] = a[j]; a[j] = t; } while(0)
      swap(i,m);                          // exchange a[i] with a[m]
      generate_all_permutations(n,m + 1,a); // recurse
      swap(i,m);                          // undo the exchange of a[i] with a[m]
#undef swap
    }

  }
  else
```

```
  {
    //
    // visit the permutation (TODO: change this ...)
    //
    n_visited++;



    if(custo_permutacao(n, a) < min_cost){
      min_cost = custo_permutacao(n, a);

    for(int i=0; i<n; i++){
      min_cost_assignment[i] = a[i]; // coloca o assignment correspondente ao mínimo custo
    }
  }

    if(custo_permutacao(n, a) > max_cost){
      max_cost = custo_permutacao(n, a);

    for(int i=0; i<n; i++){
      max_cost_assignment[i] = a[i]; // same as minimum cost but for the maximum cost

    }
  }

    histogram[custo_permutacao(n, a)]+=1; // incrementa 1 ao histogram[cost]

  }

}

static void generate_all_permutations_branch_and_bound(int n, int m, int a[n], int custo_parcial){
   if (min_cost < custo_parcial + 3*(m-n + 3)) // pra cada custo parcial, soma o custo mínimo das restantes e se já for maior que o custo mínimo atual,
    return;                                    // dá return e passa para a permutação seguinte
  if(m < n - 1)
  {
    //
    // not yet at the end; try all possibilities for a[m]
    //
    for(int i = m;i < n;i++)
    {

#define swap(i,j)  do { int t = a[i]; a[i] = a[j]; a[j] = t; } while(0)
      swap(i,m);
                                // exchange a[i] with a[m]
      generate_all_permutations_branch_and_bound(n,m + 1,a, custo_parcial + cost[m][a[m]]); // recurse
      swap(i,m);                            // undo the exchange of a[i] with a[m]
#undef swap
    }
  }
  else
  {
    //
    // visit the permutation (TODO: change this ...)
    //

    int custo_total = custo_parcial + cost[m][a[m]];

    if(custo_total < min_cost){
      min_cost = custo_total;

    for(int i=0; i<n; i++){
      min_cost_assignment[i] = a[i];
    }
  }
  n_visited++;

  }

}




////////////////////////////////////////////////////////////////////////////////////////////////
//
// main program
//

int main(int argc,char **argv)
{
  if(argc == 2 && argv[1][0] == '-' && argv[1][1] == 'e')
  {
    seed = 0;
    {
      int n = 3;
      init_costs(-3); // costs for the example with n = 3
      int a[n];
      for(int i = 0;i < n;i++)
        a[i] = i;
      reset_solutions();
      (void)elapsed_time();
      generate_all_permutations(n,0,a);
      cpu_time = elapsed_time();
      show_solutions(n,"Example for n=3",show_all);
      printf("\n");
    }
    {
      int n = 5;
      init_costs(-5); // costs for the example with n = 5
      int a[n];
      for(int i = 0;i < n;i++)
        a[i] = i;
```

```
      reset_solutions();
      (void)elapsed_time();
      generate_all_permutations(n,0,a);
      cpu_time = elapsed_time();
      show_solutions(n,"Example for n=5",show_all);
      return 0;
    }
  }
  if(argc == 2)
  {
    seed = atoi(argv[1]); // seed = student number
    if(seed >= 0 && seed <= 1000000)
    {
      int custos_minimos[14];
      int custos_maximos[14];
      double cputime_bruteforce[14];
      double cputime_BnB[14];
      double cputime_RandomPerm[14];
      for(int n = 1;n <= 14;n++)
      {
        init_costs(n);
        show_solutions(n,"Problem statement",show_info_1 | show_costs);
        //
        // 2.
        //

        if(n <= 14) // use a smaller limit here while developing your code
        {
          int a[n];
          for(int i = 0;i < n;i++)
            a[i] = i; // initial permutation
          reset_solutions();
          (void)elapsed_time();
          generate_all_permutations(n,0,a);
          cpu_time = elapsed_time();
          show_solutions(n,"Brute force",show_info_2 | show_min_solution | show_max_solution | show_histogram);

        }
        if(n==13) {
          char dest[50] = "n_13_";
          strcat(dest, argv[1]);
          strcat(dest, ".txt");

          write_file(histogram, dest); // escreve no ficheiro para o nmec passado como argumento, para n = 13

        }
        if(n==14){
          char dest[50] = "n_14_";
          strcat(dest, argv[1]);
          strcat(dest, ".txt");

          write_file(histogram, dest); // escreve no ficheiro para o nmec passado como argumento, para n = 14

        }
        custos_minimos[n-1] = min_cost; // coloca no array de custos_minimos o custo mínimo para cada n
        custos_maximos[n-1] = max_cost; // coloca no array de custos_maximos o custo máximo para cada n
        cputime_bruteforce[n-1] = cpu_time; // coloca no array de cputime_bruteforce o tempo de execução do brute force para cada n



        // place here your code that solves the problem with branch-and-bound
        //
#if 1
        if(n <= 14) // use a smaller limit here while developing your code
        {
          int a[n];
          for(int i = 0;i < n;i++)
            a[i] = i; // initial permutation
          reset_solutions_without_histogram();
          (void)elapsed_time();
          generate_all_permutations_branch_and_bound(n,0,a,0);
          cpu_time = elapsed_time();
          show_solutions(n,"Brute force with branch-and-bound",show_info_2 | show_min_solution);
          printf("\n");
        }
        cputime_BnB[n-1] = cpu_time; // coloca no array de cputime_BnB o tempo de execução do branch and bound para cada n

#endif
        //
        // place here your code that generates the random permutations
        //
        if(n <= 14){
          int a[n];

          reset_solutions_without_histogram();
          (void)elapsed_time();

          for(int i=1; i<1000000; i++){

            random_permutation(n,a);
            if(custo_permutacao(n, a) < min_cost){
              min_cost = custo_permutacao(n, a);

            for(int i=0; i<n; i++){
              min_cost_assignment[i] = a[i]; // places the assignment with the minimum cost for each n
              }
            }

            if(custo_permutacao(n, a) > max_cost){
              max_cost = custo_permutacao(n, a);

            for(int i=0; i<n; i++){
              max_cost_assignment[i] = a[i]; // same as minimum cost but for the maximum cost
              }
```

```
625             }
626           }
627           cpu_time = elapsed_time();
628           show_solutions(n,"Brute force Random Permutations",show_info_2 | show_min_solution | show_max_solution);
629
630           }
631           cputime_RandomPerm[n-1] = cpu_time;    // coloca no array de cputime_RandomPerm o tempo de execução do random permutations para cada n
632
633
634
635         //
636         // ...
637         //
638
639         //
640         // done
641         //
642         printf("\n");
643       }
644       char dest[1024];
645       memset(dest, 0, sizeof(dest)); //coloca o array vazio
646       strcat(dest, argv[1]); // para ir buscar o nmec passado como argumento
647       write_file_custos(custos_minimos, strcat(dest,"_custos_minimos.txt")); //escreve no ficheiro os custos minimos para o nmec passado como argumento
648
649       memset(dest, 0, sizeof(dest)); //coloca o array vazio
650       strcat(dest, argv[1]); // para ir buscar o nmec passado como argumento
651       write_file_custos(custos_maximos, strcat(dest,"_custos_maximos.txt")); //escreve no ficheiro os custos maximos para o nmec passado como argumento
652
653       memset(dest, 0, sizeof(dest)); //coloca o array vazio
654       strcat(dest, argv[1]); // para ir buscar o nmec passado como argumento
655       write_file_cputime(cputime_bruteforce, strcat(dest,"_cputime_bf.txt")); //escreve no ficheiro os tempos de execução do brute force para o nmec passado como argumento
656
657       memset(dest, 0, sizeof(dest)); //coloca o array vazio
658       strcat(dest, argv[1]); // para ir buscar o nmec passado como argumento
659       write_file_cputime(cputime_BnB, strcat(dest,"_cputime_BnB.txt")); //escreve no ficheiro os tempos de execução do branch and bound para o nmec passado como argumento
660
661       memset(dest, 0, sizeof(dest)); //coloca o array vazio
662       strcat(dest, argv[1]); // para ir buscar o nmec passado como argumento
663       write_file_cputime(cputime_RandomPerm, strcat(dest,"_cputime_RandomPerm.txt")); //escreve no ficheiro os tempos de execução do random permutations para o nmec passado como ar
664       return 0;
665     }
666
667   }
668   fprintf(stderr,"usage: %s -e             # for the examples\n",argv[0]);
669   fprintf(stderr,"usage: %s student_number\n",argv[0]);
670   return 1;
671 }
672
```

4.2 – Código implementado em Matlab (comentado com explicações)

```
%% Histograma para n = 13

% 93150
fileID = fopen('n_13_93150.txt');
mydata = textscan(fileID, '%f%f'); %separa as duas colunas por tab
md1 = mydata{1,1}; %coluna 1 (custos)
md2 = mydata{1,2}; %coluna 2 (número de ocorrências)
fclose(fileID);

figure(1);
bar(md1, md2); %histograma
hold on;

%calculo da Gaussiana normal
media = sum(md2.*md1)/sum(md2);
variancia = (sum(md2.*(md1-media).^2))/sum(md2);
desvio = sqrt(variancia);
x= md1;
y = exp((-(x-media).^2)/(2*desvio^2)) * max(md2);

plot(x,y, 'r', 'LineWidth', 2); %plot da Gaussiana
set(gca,'FontSize',20);
xlabel('Custo', 'FontSize', 20);
ylabel('Número de assignments' , 'FontSize', 20);
title('Seed: 93150, n = 13', 'FontSize', 20);
legend('Valores obtidos', 'Gaussiana normal', 'FontSize', 20);
hold off;

% 93107
fileID = fopen('n_13_93107.txt');
mydata = textscan(fileID, '%f%f'); %separa as duas colunas por tab
md1 = mydata{1,1}; %coluna 1 (custos)
md2 = mydata{1,2}; %coluna 2 (número de ocorrências)
fclose(fileID);

figure(2);
bar(md1, md2);
hold on;

%calculo da Gaussiana normal
media = sum(md2.*md1)/sum(md2);
```

```
variancia = (sum(md2.*(md1-media).^2))/sum(md2);
desvio = sqrt(variancia);
x= md1;
y = exp((-(x-media).^2)/(2*desvio^2)) * max(md2);

plot(x,y, 'r', 'LineWidth', 2); %plot da Gaussiana
set(gca,'FontSize',20);
xlabel('Custo', 'FontSize', 20);
ylabel('Número de assignments' , 'FontSize', 20);
title('Seed: 93107, n = 13', 'FontSize', 20);
legend('Valores obtidos', 'Gaussiana normal', 'FontSize', 20);
hold off;

% 88903
fileID = fopen('n_13_88903.txt');
mydata = textscan(fileID, '%f%f'); %separa as duas colunas por tab
md1 = mydata{1,1}; %coluna 1 (custos)
md2 = mydata{1,2}; %coluna 2 (número de ocorrências)
fclose(fileID);

figure(3);
bar(md1, md2);
hold on;

%calculo da Gaussiana normal
media = sum(md2.*md1)/sum(md2);
variancia = (sum(md2.*(md1-media).^2))/sum(md2);
desvio = sqrt(variancia);
x= md1;
y = exp((-(x-media).^2)/(2*desvio^2)) * max(md2);

plot(x,y, 'r', 'LineWidth', 2); %plot da Gaussiana
set(gca,'FontSize',20);
xlabel('Custo', 'FontSize', 20);
ylabel('Número de assignments' , 'FontSize', 20);
title('Seed: 88903, n = 13', 'FontSize', 20);
legend('Valores obtidos', 'Gaussiana normal', 'FontSize', 20);
hold off;

%% Histograma para n = 14

% 93150
fileID = fopen('n_14_93150.txt');
mydata = textscan(fileID, '%f%f'); %separa as duas colunas por tab
md1 = mydata{1,1}; %coluna 1 (custos)
md2 = mydata{1,2}; %coluna 2 (número de ocorrências)
fclose(fileID);

figure(4);
bar(md1, md2);
hold on;

%calculo da Gaussiana normal
media = sum(md2.*md1)/sum(md2);
variancia = (sum(md2.*(md1-media).^2))/sum(md2);
desvio = sqrt(variancia);
x= md1;
y = exp((-(x-media).^2)/(2*desvio^2)) * max(md2);

plot(x,y, 'r', 'LineWidth', 2); %plot da Gaussiana
set(gca,'FontSize',20);
xlabel('Custo', 'FontSize', 20);
ylabel('Número de assignments' , 'FontSize', 20);
title('Seed: 93150, n = 14', 'FontSize', 20);
legend('Valores obtidos', 'Gaussiana normal', 'FontSize', 20);
hold off;

% 93107
fileID = fopen('n_14_93107.txt');
mydata = textscan(fileID, '%f%f'); %separa as duas colunas por tab
md1 = mydata{1,1}; %coluna 1 (custos)
md2 = mydata{1,2}; %coluna 2 (número de ocorrências)
fclose(fileID);

figure(5);
bar(md1, md2);
hold on;

%calculo da Gaussiana normal
media = sum(md2.*md1)/sum(md2);
variancia = (sum(md2.*(md1-media).^2))/sum(md2);
desvio = sqrt(variancia);
```

```
    x= md1;
    y = exp((-(x-media).^2)/(2*desvio^2)) * max(md2);

    plot(x,y, 'r', 'LineWidth', 2); %plot da Gaussiana
    set(gca,'FontSize',20);
    xlabel('Custo', 'FontSize', 20);
    ylabel('Número de assignments' , 'FontSize', 20);
    title('Seed: 93107, n = 14', 'FontSize', 20);
    legend('Valores obtidos', 'Gaussiana normal', 'FontSize', 20);
    hold off;

    % 88903
    fileID = fopen('n_14_88903.txt');
    mydata = textscan(fileID, '%f%f'); %separa as duas colunas por tab
    md1 = mydata{1,1}; %coluna 1 (custos)
    md2 = mydata{1,2}; %coluna 2 (número de ocorrências)
    fclose(fileID);

    figure(6);
    bar(md1, md2);
    hold on;

    %calculo da Gaussiana normal
    media = sum(md2.*md1)/sum(md2);
    variancia = (sum(md2.*(md1-media).^2))/sum(md2);
    desvio = sqrt(variancia);
    x= md1;
    y = exp((-(x-media).^2)/(2*desvio^2)) * max(md2);

    plot(x,y, 'r', 'LineWidth', 2); %plot da Gaussiana
    set(gca,'FontSize',20);
    xlabel('Custo', 'FontSize', 20);
    ylabel('Número de assignments' , 'FontSize', 20);
    title('Seed: 88903, n = 14', 'FontSize', 20);
    legend('Valores obtidos', 'Gaussiana normal', 'FontSize', 20);
    hold off;

    %% Gráfico que compara custos mínimos com custos máximos

    % 93150
    f = fopen('93150_custos_minimos.txt', 'r');
    formatSpec = '%f';
    custos_minimos = fscanf(f, formatSpec);
    fclose(f);

    f = fopen('93150_custos_maximos.txt', 'r');
    formatSpec = '%f';
    custos_maximos = fscanf(f, formatSpec);
    fclose(f);

    figure(7);
    n = [1:14]; %valores de n
    ylim([0 max(custos_maximos)]);
    plot(n , custos_minimos, n, custos_maximos, 'LineWidth', 2); %plot dos custos mínimos e máximos
    set(gca,'FontSize',20);
    legend('Custo mínimo', 'Custo máximo', 'FontSize', 20);
    xlabel('n', 'FontSize', 20);
    ylabel('Custo', 'FontSize', 20);
    title('Seed: 93150', 'FontSize', 20);
    grid;

    % 93107
    f = fopen('93107_custos_minimos.txt', 'r');
    formatSpec = '%f';
    custos_minimos = fscanf(f, formatSpec);
    fclose(f);

    f = fopen('93107_custos_maximos.txt', 'r');
    formatSpec = '%f';
    custos_maximos = fscanf(f, formatSpec);
    fclose(f);

    figure(8);
    n = [1:14]; %valores de n
    ylim([0 max(custos_maximos)]);
    plot(n , custos_minimos, n, custos_maximos, 'LineWidth', 2); %plot dos custos mínimos e máximos
    set(gca,'FontSize',20);
    legend('Custo mínimo', 'Custo máximo', 'FontSize', 20);
    xlabel('n', 'FontSize', 20);
    ylabel('Custo', 'FontSize', 20);
    title('Seed: 93107', 'FontSize', 20);
    grid;
```

```
206
207     % 88903
208 -   f = fopen('88903_custos_minimos.txt', 'r');
209 -   formatSpec = '%f';
210 -   custos_minimos = fscanf(f, formatSpec);
211 -   fclose(f);
212
213 -   f = fopen('88903_custos_maximos.txt', 'r');
214 -   formatSpec = '%f';
215 -   custos_maximos = fscanf(f, formatSpec);
216 -   fclose(f);
217
218 -   figure(9);
219 -   n = [1:14]; %valores de n
220 -   ylim([0 max(custos_maximos)]);
221 -   plot(n , custos_minimos, n, custos_maximos, 'LineWidth', 2); %plot dos custos mínimos e máximos
222 -   set(gca,'FontSize',20);
223 -   legend('Custo mínimo', 'Custo máximo', 'FontSize', 20);
224 -   xlabel('n', 'FontSize', 20);
225 -   ylabel('Custo', 'FontSize', 20);
226 -   title('Seed: 88903', 'FontSize', 20);
227 -   grid;
228
229     %% Tempo de execução dos3 algoritmos
230
231     %93150
232 -   f = fopen('93150_cputime_bf.txt', 'r');
233 -   formatSpec = '%f';
234 -   cpu_time_bf = fscanf(f, formatSpec);
235 -   fclose(f);
236
237 -   f = fopen('93150_cputime_BnB.txt', 'r');
238 -   formatSpec = '%f';
239 -   cpu_time_bnb = fscanf(f, formatSpec);
240 -   fclose(f);
241
242 -   f = fopen('93150_cputime_RandomPerm.txt', 'r');
243 -   formatSpec = '%f';
244 -   cpu_time_randomperm = fscanf(f, formatSpec);
245 -   fclose(f);
246
247 -   figure(10);
248 -   n = [1:14]; %valores de n
249     %igual a plot mas escala de y é logarítmica
250 -   semilogy(n , cpu_time_bf, n, cpu_time_bnb, n, cpu_time_randomperm, 'g', 'LineWidth', 2 );
251 -   set(gca,'FontSize',20);
252 -   legend('Brute force', 'Branch And Bound', 'Random Permutations','FontSize',20);
253 -   xlabel('n','FontSize',20);
254 -   ylabel('CPU time (sec)','FontSize',20);
255 -   title('Seed: 93150','FontSize',20);
256 -   grid;
257
258     %93107
259 -   f = fopen('93107_cputime_bf.txt', 'r');
260 -   formatSpec = '%f';
261 -   cpu_time_bf = fscanf(f, formatSpec);
262 -   fclose(f);
263
264 -   f = fopen('93107_cputime_BnB.txt', 'r');
265 -   formatSpec = '%f';
266 -   cpu_time_bnb = fscanf(f, formatSpec);
267 -   fclose(f);
268
269 -   f = fopen('93107_cputime_RandomPerm.txt', 'r');
270 -   formatSpec = '%f';
271 -   cpu_time_randomperm = fscanf(f, formatSpec);
272 -   fclose(f);
273
274 -   figure(11);
275 -   n = [1:14];%valores de n
276     %igual a plot mas escala de y é logarítmica
277 -   semilogy(n , cpu_time_bf, n, cpu_time_bnb, n, cpu_time_randomperm, 'g', 'LineWidth', 2 );
278 -   set(gca,'FontSize',20);
279 -   legend('Brute force', 'Branch And Bound', 'Random Permutations','FontSize',20);
280 -   xlabel('n','FontSize',20);
281 -   ylabel('CPU time (sec)','FontSize',20);
282 -   title('Seed: 93107','FontSize',20);
283 -   grid;
284
285     %88903
286 -   f = fopen('88903_cputime_bf.txt', 'r');
287 -   formatSpec = '%f';
```

```
288 -   cpu_time_bf = fscanf(f, formatSpec);
289 -   fclose(f);
290
291 -   f = fopen('88903_cputime_BnB.txt', 'r');
292 -   formatSpec = '%f';
293 -   cpu_time_bnb = fscanf(f, formatSpec);
294 -   fclose(f);
295
296 -   f = fopen('88903_cputime_RandomPerm.txt', 'r');
297 -   formatSpec = '%f';
298 -   cpu_time_randomperm = fscanf(f, formatSpec);
299 -   fclose(f);
300
301 -   figure(12);
302 -   n = [1:14];%valores de n
303     %igual a plot mas escala de y é logarítmica
304 -   semilogy(n , cpu_time_bf, n, cpu_time_bnb, n, cpu_time_randomperm, 'g', 'LineWidth', 2 );
305 -   set(gca,'FontSize',20);
306 -   legend('Brute force', 'Branch And Bound', 'Random Permutations','FontSize',20);
307 -   xlabel('n','FontSize',20);
308 -   ylabel('CPU time (sec)','FontSize',20);
309 -   title('Seed: 88903','FontSize',20);
310 -   grid;
311
```

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.