

Relatório Projeto 1

Speed Run

Algoritmos e Estruturas de Dados

Das turmas P08 e TP1 dos professores:

Prof. Tomás Oliveira e Silva
Prof. Joaquim Madeira

Nome: Diogo Almeida NºMec: 108902 (33,33%)
Nome: Gonçalo Ferreira NºMec: 107853 (33,33%)
Nome: Tomás Matos NºMec: 108624 (33,33%)

5 de Dezembro de 2022



Introdução ao Tema	3
O problema	3
Primeiras abordagens	4
Análise do código dado	4
Elapsed_time.h	4
speed_run.c	5
makefile e make_custom_pdf.c	7
Apresentação dos Métodos de Solução	8
Análise do código desenvolvido	9
Solve 2 inicial melhorada	9
Solve 3 while dynamic	11
Solve 4 dynamic	16
Solve 5 while	18
Resultados Obtidos	24
Solve 1 inicial	24
Gráfico Matlab solve inicial	24
Solve 2 inicial melhorada	25
PC1	25
PC2	26
PC3	27
Gráfico Matlab Solve 2	28
Solve 3 while dynamic	29
PC1	29
PC2	30
PC3	31
Gráfico Matlab Solve 3	32
Solve 4 dynamic	33
PC1	33
PC2	34
PC3	35
Gráfico Matlab Solve 4	36
Gráfico comparação de entradas nas funções recursivas:	36
Solve 5 while	37
PC1	37
PC2	38
PC3	39
Gráfico Matlab Solve 5	40
Caracol Exemplo (PC2)	41
Gráfico com todas as soluções:	43
Gráfico apenas com as soluções melhoradas:	43
Apêndice	44
Código C	44
Solve 2 inicial melhorada	44
Solve 3 while dynamic	45
Solve 4 dynamic	49
Solve 5 while	52
Main	56
Código Matlab	60
Conclusão	64



Introdução ao Tema

O problema

Imaginemos uma estrada que está subdividida por segmentos de estrada que têm aproximadamente todos o mesmo tamanho, e cada um desses segmentos da estrada possui, tal como qualquer estrada normal, um limite de velocidade, que precisa de ser respeitado. Nesta estrada, a velocidade é medida pela quantidade de segmentos que têm capacidade de avançar com apenas movimento. Falando em movimentos, em cada um deles o carro pode acelerar, manter a velocidade ou diminuir, que em termos práticos significa percorrer um segmento a mais do que na última movimento, manter o número de segmentos da estrada percorridos ou diminuir este número em uma unidade, respectivamente. De notar que em nenhuma das situações citadas a velocidade não aumenta ou diminui mais de uma unidade desde a última iteração, já que isso não pode acontecer.

Tendo em conta todas estas informações e que a velocidade no primeiro segmento da estrada e no último tem que ser obrigatoriamente um, o objetivo do trabalho é passar pela estrada com o **menor número de movimentos possível**.

Para demonstrar, segue o seguinte exemplo de uma estrada dividida em 30 segmentos iguais

5	5	6	6	8	7	8	9	8	9	8	9	9	8	9	7	7	6	6	6	5	6	5	4	4	3	5	3	3	3	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	[26]	[27]	[28]	[29]	[30]

Tendo estas estradas as velocidades máximas que podemos ver na imagem, a maneira mais rápida de percorrer esta estrada, conforme as regras impostas no enunciado, é a seguinte:

1

5	5	6	6	8	7	8	9	8	9	8	9	9	8	9	7	7	6	6	6	5	6	5	4	4	3	5	3	3	3	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	[26]	[27]	[28]	[29]	[30]

¹ Na imagem, os quadrados cinzentos assinalam aqueles quadrados que simbolizam posições do carro.



Primeiras abordagens

Para além do problema em si, o trabalho também vem com dicas e programas a que temos acesso para nos ajudar na resolução, e que temos de considerar a quando de realizar uma análise inicial do problema em si.

E a primeira coisa que pensamos em fazer mal nos começamos a debater sobre o assunto, foi ler e analisar o código dado pelo professor no ficheiro que fazia referência ao trabalho

Análise do código dado

Elapsed_time.h

```

1  //
2  // Tomás Oliveira e Silva, AED, October 2021
3  //
4  // code to measure the elapsed time used by a program fragment
5  //
6  // use as follows:
7  //
8  //   double t1 = cpu_time();
9  //   // put your code to be time measured here
10 //   double t2 = cpu_time();
11 //   printf("elapsed time: %.6f seconds\n",t2 - t1);
12 //
13
14
15 #if defined(__linux__) || defined(__APPLE__)
16
17 //
18 // GNU/Linux and MacOS code to measure elapsed time
19 //
20
21 #include <time.h>
22
23 double cpu_time(void)
24 {
25     struct timespec current_time;
26
27     if(clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&current_time) != 0) // the first argument could also be CLOCK_REALTIME
28     | return -1.0; // clock_gettime() failed!!!
29     return (double)current_time.tv_sec + 1.0e-9 * (double)current_time.tv_nsec;
30 }
31
32 #endif
33
34
35 #if defined(_MSC_VER) || defined(_WIN32) || defined(_WIN64)
36
37 //
38 // Microsoft Windows code to measure elapsed time
39 //
40
41 #include <windows.h>
42
43 double cpu_time(void)
44 {
45     static LARGE_INTEGER frequency;
46     static int first_time = 1;
47     LARGE_INTEGER current_time;
48
49     if(first_time != 0)
50     {
51         QueryPerformanceFrequency(&frequency);
52         first_time = 0;
53     }
54     QueryPerformanceCounter(&current_time);
55     return (double)current_time.QuadPart / (double)frequency.QuadPart;
56 }
57
58 #endif

```



O “header file” **elapsed_time.h** é um ficheiro composto por 2 funções **cpu_time()** que têm como objetivo medir o tempo que o programa demorou a correr o programa que executamos, que vai ser bastante útil para saber qual das soluções é a mais rápida.

De notar que ambas as funções funcionam de maneira similar, apenas com a diferença de leitura dos valores para sistemas operativos diferentes.

speed_run.c

Depois de termos os comentários que o professor realizou no código que ele mesmo escreveu, analisamos o restante do código e fizemos as seguintes apreciações:

```

// 78 static void solution_1_recursion(int move_number,int position,int speed,int final_position)
79 {
80     int i,new_speed;
81
82     // record move
83     solution_1_count++;
84     solution_1.positions[move_number] = position;
85     // is it a solution?
86     if(position == final_position && speed == 1)
87     {
88         // is it a better solution?
89         if(move_number < solution_1_best.n_moves)
90         {
91             solution_1_best = solution_1;
92             solution_1_best.n_moves = move_number;
93         }
94         return;
95     }
96     // no, try all legal speeds
97     for(new_speed = speed - 1;new_speed <= speed + 1;new_speed++)
98     {
99         if(new_speed >= 1 && new_speed <= _max_road_speed_ && position + new_speed <= final_position)
100        {
101            for(i = 0;i <= new_speed && new_speed <= max_road_speed[position + i];i++)
102            {
103                if(i > new_speed)
104                    solution_1_recursion(move_number + 1,position + new_speed,new_speed,final_position);
105            }
106        }
107    static void solution_1(int final_position)

```

A solução recorre a uma função recursiva, ou seja, ela chama-se a si mesma várias vezes até chegar ao final da estrada. Já dentro da função **solution_1_recursion**, o código das linhas 83 e 84 é necessário para guardar o movimento feito em variáveis, ou seja, aumenta-se o valor do **solution_1_count** em mais 1, que é uma variável declarada em cima e que faz a contagem da quantidade de vezes que a função entra na recursiva, que serve maioritariamente para efeitos mais tardios de estatística.

Já na linha 84 é guardada a posição em que o carro se encontra, numa lista de 801 elementos (sendo 800 a maior quantidade de subdivisões que a estrada pode ter e é somado 1, tendo em conta que a primeira posição do array é obrigatoriamente 0). A lista **positions** recebe como index o número de movimentos já realizados na situação e é diferente para cada maneira de resolver, por isso faz parte da classe construída em cima que tem o nome de **solution_t** no código, que tem o objetivo de deixar criar 2 instâncias dessa classe, que vão ser usadas para saber qual das maneiras de resolução é efetuada com recurso ao menor número de movimentos, para isso a necessidade das linhas 86 a 94.

Verificam inicialmente se o programa já chegou ao fim da estrada e com velocidade 1, e se tal se suceder, entramos dentro de outra condição que vai avaliar se o maneira de resolução descrita é efetivamente melhor (ou seja, possui menos movimentos do que a anteriormente

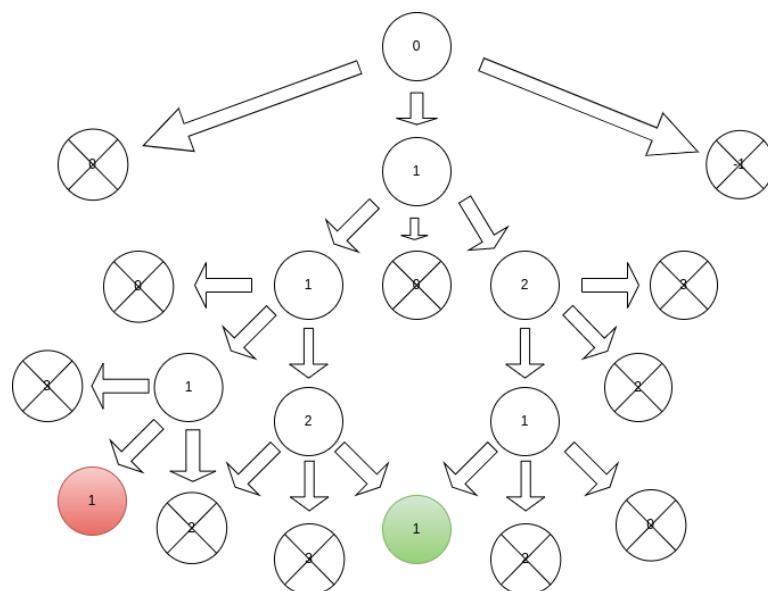


considerada como “melhor solução”). Independente desta ser ou não a melhor solução, depois de entrar aqui, a tentativa de solução termina, uma vez que já não tem qualquer tipo de utilidade. Caso seja, a melhor solução passa para a instância de `solution_t`, `solution_1_best`.

Depois de explicarmos o que acontece quando as diferentes possíveis soluções geradas e como é feita a escolha da melhor solução possível, vamos ver como é que as diferentes soluções são criadas.

Na linha 97, podemos ver um `for loop` que usa uma variável criada no início da recursiva, e que ainda não tinha sido usada, `new_speed`, variável essa vai correr todas as possibilidades que pode tomar a próxima velocidade, ou seja, a velocidade da última iteração menos um, mantendo a última velocidade ou somando 1, respectivamente nesta ordem. Depois cada uma destas velocidades vai passar por uma quantidade de verificações para saber se podem ou não continuar na estrada, verificando se a velocidade está no intervalo [1 , 9] e se quando somamos a `new_speed` à posição atual esse valor ultrapassa a posição final, coisa que não pode acontecer. Na linha seguinte vemos um `for loop` que percorre todos os valores intermediários (ou seja, todos os valores entre a posição atual e a soma da posição com a nova velocidade) e ascendendo ao valor com o mesmo índice na lista das velocidades máximas a que é permitido circular naquele segmento de estrada, verifica se em cada um dos segmentos percorridos a velocidade máxima permitida é excedida. Caso as velocidades estejam todas de acordo com a velocidade máxima, então pode ser chamada uma nova recursiva usando essa nova velocidade guardada em `new_speed`.

Assim, fizemos um gráfico que exemplifica como o código dado chega à posição final 5, por exemplo.



Como podemos ver, 3 caminhos conseguem chegar à posição final sem ficarem para trás, mas apenas os que terminam no círculo verde são as melhores possibilidades, tendo em conta que o caminho que termina no círculo vermelho possui um movimento a mais que os outros.



Aqui podemos ver os resultados produzidos pelo programa, e percebemos logo que não consegue chegar aos 800 segmentos de estradas

n	sol	count	cpu	time
1	1	2	3.352e-06	
2	2	3	1.341e-06	
3	3	5	1.465e-06	
4	3	8	1.926e-06	
5	4	13	2.213e-06	
6	4	22	2.909e-06	
7	5	36	3.199e-06	
8	5	60	4.473e-06	
9	5	100	6.228e-06	
10	6	167	8.438e-06	
11	6	279	4.788e-06	
12	6	465	5.535e-06	
13	7	777	8.434e-06	
14	7	1297	1.345e-05	
15	7	2165	2.212e-05	
16	7	3614	3.612e-05	
17	8	6031	5.974e-05	
18	8	10065	9.888e-05	
19	8	16795	1.642e-04	
20	8	28011	2.938e-04	
21	9	46724	3.906e-04	
22	9	77940	6.452e-04	
23	9	130018	1.172e-03	
24	9	216884	1.773e-03	
25	9	361795	3.177e-03	
26	10	599912	5.688e-03	
27	10	997122	9.386e-03	
28	10	1659747	1.416e-02	
29	10	2761495	2.542e-02	
30	11	4599395	3.821e-02	
31	11	7702564	6.732e-02	
32	11	12803795	1.030e-01	
33	12	21306293	1.816e-01	
34	12	35512666	2.786e-01	
35	12	59117851	4.643e-01	
36	13	98492776	7.820e-01	
37	13	164114436	1.295e+00	
38	13	273408602	2.152e+00	
39	14	455591670	3.585e+00	
40	14	759109266	5.866e+00	
41	14	1264877251	9.764e+00	
42	15	2118524472	1.631e+01	
43	15	3540919788	2.746e+01	
44	15	5911111247	4.567e+01	
[OBJ]	45	9871346606	7.680e+01	

makefile e make_custom_pdf.c

Estes 2 ficheiros fazem referência à criação de documentos que ajudam a demonstrar os resultados dos diferentes algoritmos.



Apresentação dos Métodos de Solução

Brute Force

O termo Brute Force é muitas vezes utilizado para contextos de ataques onde se força a entrada em algum sistema, site, Este consiste em testar todas as combinações de um certo problema, normalmente senhas ou códigos secretos.

Para a solução que nos foi disponibilizada para a resolução deste problema pensamos que o método que esteve presente na sua realização foi o **Brute Force**.

Durante a interpretação do código referido, reparamos que apesar de este ter algumas condições estas são indispensáveis para a resolução do problema, esta continua a percorrer imensas combinações de velocidades possíveis que são facilmente evitáveis. Desta maneira, este é um método que suporta muita abrangência tornando-se menos eficaz e computacionalmente muito complexo, tanto que para resolver para uma estrada com 45 blocos, o PC2 demorou 76,80 segundos e o programa chamou a função 9.871.346.606 vezes.

Clever Brute Force

O Clever Brute Force é uma versão mais “inteligente” da Brute Force, uma vez que é uma versão otimizada da Brute Force.

No começo da realização da nossa solução Solve 2 (antes de a tornarmos dinâmica) utilizamos este método, aumentando os critérios de seleção e com apenas uma condição, aumentamos a eficácia do programa fornecido e conseguimos fazer a solução percorrer para uma estrada de 800 blocos com valores na ordem dos 10 elevado a -5, -6.

Dynamic programming

Na maioria das nossas soluções utilizamos este método, a programação dinâmica é uma técnica que divide um problema em vários subproblemas menores e que após os resolver armazena os resultados e utiliza-os no problema original. Em todas as nossas soluções dinâmicas guardamos num array "último_incremento" os valores do número de movimentos, a posição e a velocidade da estrada na última vez que a velocidade aumentou numa determinada estrada. Vamos utilizar estes valores guardados na estrada seguinte.

Se tivéssemos que dividir no problema que nos foi proposto no problema original e nos subproblemas menores, diríamos que o problema maior seria a estrada com 800 blocos e os menores todas as estradas com o número de blocos inferior a 800.



Análise do código desenvolvido

Solve 2 inicial melhorada

Como já podemos verificar, código inicialmente dado não corre até aos 800 segmentos devido à pouca eficiência que apresenta, ou seja, o programa gera imensas soluções que vão funcionar, porque andam sempre a uma velocidade muito baixa e essa velocidade nunca vai ser barrada pela velocidade máxima de cada segmento. Essas soluções podem até ser melhores quando falamos de números mais baixos de segmentação, mas à medida que os segmentos vão aumentando, vamos começar a formular mais e mais soluções com altos números de movimentos, por isso, fizemos 2 alterações que melhoraram bastante a eficiência do programa inicialmente feito, que permite que ele corra mais segmentos do que o inicial, isto antes de a tornarmos dinâmica.

```

151     if(position == final_position && speed == 1)
152     {
153         // is it a better solution?
154         if(move_number < solution_2_best.n_moves)
155         {
156             solution_2_best = solution_2;
157             solution_2_best.n_moves = move_number;
158             solution_2_last = solution_2_best;
159         }
160         return 1;
161     }
162     if(solution_2_best.positions[move_number] > solution_2.positions[move_number]){
163         return 0;
164     }
165     // no, try all legal speeds
166     for(new_speed = speed + 1;new_speed >= speed - 1;new_speed--) {
167         if(new_speed >= 1 && new_speed <= _max_road_speed_ && position + new_speed <= final_position)
168         {
169             for(i = 0;i <= new_speed && new_speed <= max_road_speed[position + i];i++)
170             ;
171             if(i > new_speed){
172                 int verify = solution_2_recursion(move_number + 1,position + new_speed,new_speed,final_position);
173                 if (new_speed - speed == 1){
174                     solution_2_last.ultimo_incremneto[0] = move_number + 1;
175                     solution_2_last.ultimo_incremneto[1] = position + new_speed;
176                     solution_2_last.ultimo_incremneto[2] = new_speed;
177                 }
178                 if (verify == 1)
179                 {
180                     return 1;
181                 }
182             }
183         }
184     }
185     return 0;
186 }

```

A primeira alteração é visível na linha 162, onde vemos uma condição que muda muita coisa. No programa inicial todas as possíveis soluções, mesmo que com muitos movimentos, corriam até ao fim, e isso era um fator que causava muito pouca eficiência ao programa. Por isso, aquela condição trata de verificar se a “melhor solução” estaria mais avançada (ou seja, se já teria percorrido mais segmentos da estrada) do que a solução que está a ser construída, para o mesmo número de movimentos. Se sim, então ela pega na solução que está a ser construída, já que ambas conservam as regras exigidas pela



estrada, e a solução que está a ser construída já não vai ser aquela que possui o menor número de movimentos possíveis.

Depois disso, a segunda mudança passa por mudar a maneira como são geradas as velocidades a usar na construção de novas soluções. Na maneira enviada pelo professor, as velocidades são testadas da menor para a maior, ou seja, primeiro, testam-se as velocidades mais baixas para andar na estrada, e depois só depois as mais altas são testadas. Se trocarmos a maneira como o **for loop** da linha 166 gera, podemos começar a testar logo as velocidades mais altas, obtendo logo uma solução mais eficiente e colocando a condição da linha 162 a trabalhar, porque assim, quando as soluções de velocidades mais baixas começarem a ser executadas, na sua maioria vão entrar nessa condição e poupar tempo de execução ao programa , uma vez que não vão ter que correr na sua totalidade.

Depois de explicadas as alterações que atuam na eficiência do programa, decidimos aplicar o conceito de **dynamic programming** , para tornar o programa mais eficiente

Para colocar esta abordagem de programação na solução explicada, foi preciso fazer algumas mudanças, como por exemplo: criar uma nova instância chamada de **solution_2_last** , que vai guardar os valores necessários do último incremento da melhor solução para aquela quantidade específica de segmentos, onde o próximo valor de segmentos vai começar a correr a partir desses valores. Para isso também precisamos de criar um array específico para guardar os valores na estrutura inicial, que serão a posição, a velocidade e o número de movimentos naquele específico caso, para depois podermos invocar a função com esses valores.

Também mudamos a função recursiva para retornar valores inteiros, valores esses que nos ajudam a perceber se a função correu com sucesso ou não. Quando retorna valor 1, significa que foi teve sucesso, caso retorne 0, é suposto parar. Usamos esse valor no programa para verificar se é possível executar o próximo movimento com sucesso.

```
static solution_t solution_2,solution_2_best,solution_2_last;
static double solution_2_elapsed_time; // time it took to solve the problem
static unsigned long solution_2_count; // effort dispended solving the problem
```

```
--> 189 static void solve_2(int final_position)
190 {
191     if(final_position < 1 || final_position > _max_road_size_)
192     {
193         fprintf(stderr,"solve_1: bad final_position\n");
194         exit(1);
195     }
196     solution_2_elapsed_time = cpu_time();
197     solution_2_count = 0ul;
198     solution_2_best.n_moves = final_position + 100;
199     solution_2_recursion(solution_2_last.ultimo_incremeno[0],solution_2_last.ultimo_incremeno[1],solution_2_last.ultimo_incremeno[2],final_position);
200     solution_2_elapsed_time = cpu_time() - solution_2_elapsed_time;
201 }
```



Solve 3 while dynamic

Primeiramente, depois de uma análise do problema dado e dos resultados que fomos obtendo, reparámos que quando a velocidade é incrementada em 1 numa casa todos os movimentos anteriores a esse são os melhores possíveis para todas as soluções seguintes. Isso quer dizer que podemos salvar a posição, a velocidade e o número de moves na casa quando é realizado um incremento de velocidade pois, todas as casas anteriores, não iram alterar mais para todas soluções seguintes.

Então para salvarmos esses valores começámos por adicionar um novo array (`ultimo_incremendo`) ao objeto Solution:

```
typedef struct
{
    int n_moves; // the number of moves (the number of positions is one more than the number of moves)
    int positions[1 + _max_road_size_]; // the positions (the first one must be zero)
    int ultimo_incremendo[3]; //array para guardar o valor da posição, da velocidade e do n. de movimentos do ultimo incremento
}
solution_t;
```

Pode também usar se um array externo e uma função que irá atualizar esse array:

```
//Utilizando array externo e ponteiros
int useful[3];
void ultimo_incremendo_Update(int *useful,int move,int position, int speed){
    useful[0] = move;
    useful[1] = position;
    useful[2] = speed;
}
```

Chamada da função:

```
solution_3_while_dynamic(solution_3.ultimo_incremendo[0],solution_3.ultimo_incremendo[1],solution_3.ultimo_incremendo[2],final_position); //Move_number,position,speed
```

São dados como argumentos o move_number, a velocidade e a posição do último incremento da solução anterior e a final_position da solução atual a calcular.

Variáveis:

```
int new_speed = solution_3.ultimo_incremendo[2];
int new_speed_test;
int position_test;
int variavel_teste;
int position_test_anterior;
int teste_max_speed;
int moves = solution_3.ultimo_incremendo[0];
position = solution_3.ultimo_incremendo[1];
speed = solution_3.ultimo_incremendo[2];
```



Primeiro são criadas variáveis que irão ser usadas para realizar os testes da velocidade e dos speed limits durante a execução da função.

```
while(position<final_position){ // Executa enquanto a posição não for a final_position
    solution_3_count++; // Conta o número de iterações
    speed = new_speed; // Atribui a speed o valor de new_speed
    variavel_teste = 1; // Variável que controla a entrada nos if's seguintes
    position_test = position; // Atribui a position_test o valor de position

    teste_max_speed = 1; // Variável que controla se a speed ultrapassa o max_road_speed
```

É depois iniciado o while que irá executar de acordo com o valor da position que quando for igual ao valor da final_position quer dizer que já calculou todos os valores para a solução atual e irá parar a execução.

É colocado um count que irá contar o número de vezes que o while irá ser executado para cada solução sendo esse o “effort” da solução.

É atribuído a speed o valor de new_speed que será calculado a cada execução do while.

Atribuímos a duas variáveis de teste o valor 1 para meios de controlo de situações futuras.

É também atribuído o valor da posição atual à position_test que irá ser usado no cálculo das velocidades.

```
if(final_position==1){
    moves = 1;
    solution_3.positions[moves] = 1;
    solution_3_best.positions[moves] = 1;
    solution_3.ultimo_incremento[0] = moves;
    solution_3.ultimo_incremento[1] = 1;
    solution_3.ultimo_incremento[2] = 1;

    break;
}
```

Quando a final_position é 1 nunca varia e por isso é colocado já esse caso de parte. A velocidade na casa 0 é 0. É feito um incremento na velocidade para 1 e acaba na position 1 com velocidade 1, realizando assim apenas 1 move. É então guardado nos arrays das posições a casa 1 ficando assim com [0,1].

É salvo também no array do último incremento os valores da speed, da position e do número de movimentos pois para as soluções seguintes estas duas casas não irão alterar mais podendo assim começar a calcular a partir destes valores.



Caso a final_position não seja 1 então:

```

else{
if(variavel_teste == 1 && speed<_max_road_speed_){ // Se speed for menor que o max_road_speed
position_test = position;
for(new_speed_test = speed +1;new_speed_test >= 1; new_speed_test--){ //Loop desde speed+1 até 1
position_test_anterior = position_test; //Atribui a position_test_anterior o valor de position_test
position_test = position_test + new_speed_test; //Atribui a position_test o valor de position_test + new_speed_test
for(int i = position_test_anterior; i <= position_test; i++){ //Testa max_speed desde posição inicial+1 ate posição depois do incremento de velocidade
if(max_road_speed[i]<new_speed_test){ //Se max_speed for menor que new_speed
teste_max_speed = 0; //Pode dar break;
break;
}
}
if(teste_max_speed == 0){break;} //Se max_speed for menor que new_speed pode dar break no loop
}
if(position_test<final_position && teste_max_speed == 1){ //Se a posição for menor que a final_position e se max_speed for maior que new_speed
new_speed = speed +1; //Atribui a new_speed o valor de speed +1
moves++; //Incrementa o número de movimentos
variavel_teste = 0; //Pode começar uma nova iteração no while
solution_3.ultimo_increm[0] = moves; //Atribui a solution_3.ultimo_increm[0] o valor de moves
solution_3.ultimo_increm[1] = position + new_speed; //Atribui a solution_3.ultimo_increm[1] o valor de position + new_speed
solution_3.ultimo_increm[2] = new_speed; //Atribui a solution_3.ultimo_increm[2] o valor de new_speed
}
}

```

Começamos por verificar se a variável_teste está a 1 e se o speed é menor que a max road speed pois o máximo de velocidade é 9. Caso respeite as condições vai começar por atribuir a uma variável new_speed_test o valor de speed + 1 para testar se a velocidade pode ser incrementada. Vai ser feito um loop desde o valor de new_speed_test até 1 atualizando sempre as variáveis position_test_anterior e position_test fazendo com que position_test_anterior receba sempre o valor da posição anterior e o position_test o da posição seguinte de acordo com os valores de new_speed_test para testar se ao incrementarmos o valor de speed se iremos ter casas suficientes para diminuir a velocidade sem que ultrapasse a final_position e de modo a que entre as position_test_anterior e a position_test a velocidade nunca seja superior ao speed limit. Para isso é criado um For que irá percorrer as posições desde a position_test_anterior e a position_test de modo a verificar que o new_speed_test não ultrapassa o valor do max_road_speed nessas casas. Caso em alguma casa desses intervalos até o new_speed_test ser 1 a velocidade for superior ao max_road_speed dessa casa a variável teste_max_speed é colocada a 0 de modo a que a velocidade não seja incrementada e são parados os For's.

Por outro lado, caso todas as velocidades respeitem os Max_road_speed e no final do loop a posição final (position_test) for menor ou igual à final_position isso quer dizer que a velocidade pode ser incrementada nessa position pois irá ter casas suficientes para decrementar a velocidade e chegar à final_position com speed = 1. E por isso a nova velocidade na próxima iteração do while (new_speed) vai ser o speed + 1. O valor dos moves também é incrementado em 1. Como a velocidade sofreu um incremento então vamos atualizar os valores no array do ultimo_increm com o novo valor dos moves, com o valor da nova posição que será position mais o novo valor da velocidade (position + new_speed) e com o novo valor da velocidade (new_speed). A variável_teste é colocada a 0 para que não sejam feitos os teste para manter a velocidade ou para diminuir fazendo assim o while começar o cálculo para a nova posição.



```

if(variavel_teste==1 && speed<= _max_road_speed_){
    position_test = position;
    for(new_speed_test = speed;new_speed_test >= 1; new_speed_test--){ //Loop desde speed até 1 (Manter speed)
        position_test_anterior = position_test;
        position_test = position_test + new_speed_test;
        for(int i = position_test_anterior; i <= position_test; i++){ //Testa max_speed no intervalo das posições
            if(max_road_speed[i]<new_speed_test){
                teste_max_speed = 0; //Pode dar break;
                break;
            }
        }
        if(teste_max_speed == 0){break;}
    }
    if (teste_max_speed == 0) //Caso não possa manter o speed a única opção é decrementar o speed
    {
        new_speed = speed - 1;
        moves++;
        variavel_teste = 0;
    }
    else if(position_test<=final_position && teste_max_speed == 1){
        new_speed = speed;
        moves++;
        variavel_teste = 0;
    }
}

```

Caso não pudesse aumentar a velocidade então vai executar este If onde irá ser executado o mesmo loop visto anteriormente mas new_speed_test vai do valor de speed até 1. Vai testar se pode manter a velocidade. Caso alguma casa entre as posições position_test_anterior e a position_test durante os valores de new_speed_test não respeite a velocidade máxima, max_road_speed então a valor teste_max_speed vai ser colocada a 0. Irá parar o For e como a velocidade não pode ser mantida então a única opção restante é diminuir a velocidade e por isso entra no if:

```

if (teste_max_speed == 0) //Caso não possa manter o speed a única opção
é decrementar o speed
{
    new_speed = speed - 1;
    moves++;
    variavel_teste = 0;
}

```

O new_speed vai ser speed - 1 (decrementa a velocidade), incrementamos o número de moves e a variavel_teste é colocada a 0 para ser iniciada uma nova iteração no while.

Caso contrário, se todas as condições forem bem sucedidas para o loop mantendo o speed então colocamos o new_speed = speed, incrementamos os moves e a variavel_teste colocada a 0.



```
if(variavel_teste==1){  
    position_test = position;  
    for(new_speed_test = speed-1;new_speed_test >= 1; new_speed_test--){ //Desce speed  
        position_test = position_test + new_speed_test;  
    }  
    new_speed = speed -1;  
    moves ++;  
    variavel_teste = 0;  
}
```

Caso depois de todos os passos anteriores a variavel_teste ainda tiver a 1 é porque a única opção é diminuir a velocidade. Então atribuímos a new_speed o valor speed - 1, incrementamos os moves e variavel_teste é colocada a 0.

```
if(new_speed == 0){  
    position = position + speed;  
}else{  
    position = position + new_speed;  
    solution_3.positions[moves] = position;  
    solution_3_best.positions[moves] = position;  
}  
}
```

Aqui vemos que se o new_speed é 0 é porque o decrementámos o valor do speed e por isso para haver um avanço na posição dizemos que a posição não é a soma da posição + new_speed mas sim posição + speed (1). Caso contrário a posição seguinte irá ser a soma da posição anterior mais o novo speed (new_speed). Depois atualizamos o array das positions com a posição calculada nesta iteração no index do valor dos moves.



Solve 4 dynamic

Nesta solução usámos o array com os valores do último incremento de speed como descrito na solução 3.

```
int position_teste,position_anterior; //Variáveis que irão receber valores de posições para testar se a speed é válida
int acabou = 0; //Variável de controlo
int speed_valido=1; //Variável de controlo para saber se a speed é válida
solution_4_count++;
position_teste = position;
solution_4.positions[move_number] = position;
solution_4_best.positions[move_number] = position;
```

Primeiro criamos variáveis que nos vão ser úteis durante a execução da função recursiva. solution_4_count irá contar o effort, cada vez que a função recursiva for executada este valor irá incrementar. A cada execução o valor da position atual irá ser guardado no array das positions no index do valor do move_number atual.

```
if(position==final_position){
    solution_4_best.n_moves = move_number;
    return;
}
```

A função começa por verificar se a position atual for a final_position então é porque chegou ao final e guarda o número atual de move_number que irá ser o valor ideal de movimentos.

```
else{
    for(int new_speed = speed + 1; new_speed >= speed - 1; new_speed--){ //3 possibilidades de speed, incrementa, mantém ou decrementa
        speed_valido = 1; //Variável de controlo para saber se a speed é válida
        if(new_speed<=max_road_speed_){ //Testa se a speed é válida
            position_teste = position; //Reinicia a posição de teste
            for(int new_speed_teste = new_speed;new_speed_teste>=1; new_speed_teste--){ //Loop desde new_speed até 1
                position_anterior = position_teste; //Guarda a posição anterior
                position_teste = position_teste + new_speed_teste; //Incrementa a posição de teste

                for(int i = position_anterior;i<=position_teste;i++){ //Testa max speed
                    if(max_road_speed[i]<new_speed_teste){
                        speed_valido = 0;
                        break;
                    }
                }

                if(position_teste>final_position || acabou == 1 || speed_valido == 0){break;}
            }

            if(new_speed_teste == 1 && position_teste<=final_position && speed_valido == 1){ //Speed válida entra na função com novo speed e nova posição
                acabou = 1;
                if(new_speed - speed == 1){ // Deu incremento no speed
                    // ultimo_incremento_update(&useful,move_number + 1,position + new_speed, new_speed); //Com Array externo
                    solution_4.ultimo_incremento[0] = move_number + 1;
                    solution_4.ultimo_incremento[1] = position + new_speed;
                    solution_4.ultimo_incremento[2] = new_speed;
                }
                solution_4_dynamic(move_number + 1,position + new_speed, new_speed,final_position);
            }
        }
    }
}
```

Caso contrário começamos por criar um For em que a variável new_speed irá receber as 3 opções possíveis. Primeiro vai ser speed + 1, se esta falhar então vai receber speed e se mesmo assim não for possível manter o speed então irá receber o valor de speed - 1. Vão ser realizados os mesmos testes para cada um destes valores mas caso o primeiro caso (speed + 1) seja logo válido o loop irá parar e vai ser visto para a posição seguinte. O primeiro ponto é :

```
if(new_speed<=max_road_speed_){ //Testa se a speed é válida
    position_teste = position; //Reinicia a posição de teste
```



A new_speed nunca pode ser maior que 9 (Max_road_speed) caso contrário irá avançar logo para a próxima iteração do primeiro For.

```

for(int new_speed_teste = new_speed;new_speed_teste >= 1; new_speed_teste--) //Loop desde new_speed até 1
    position_anterior = position_teste; //Guarda a posição anterior
    position_teste = position_teste + new_speed_teste; //Incrementa a posição de teste

    for(int i = position_anterior;i<=position_teste;i++){ //Testa max speed
        if(max_road_speed[i]<new_speed_teste){
            speed_valido = 0;
            break;
        }
    }

    if(position_teste>final_position || acabou == 1 || speed_valido == 0){break;}

    if(new_speed_teste == 1 && position_teste<=final_position && speed_valido == 1){ //Speed válida entra na função com novo speed e nova posição
        acabou = 1;
        if(new_speed - speed == 1){ // Deu incremento no speed
            // ultimo_incremento_Update(&useful,move_number + 1,position + new_speed, new_speed); //Com Array externo
            solution_4.ultimo_incremento[0] = move_number + 1;
            solution_4.ultimo_incremento[1] = position + new_speed;
            solution_4.ultimo_incremento[2] = new_speed;
        }
        solution_4_dynamic(move_number + 1,position + new_speed, new_speed,final_position);
    }
}

```

Vai ser feito um loop desde o valor de new_speed até 1 atualizando sempre as variáveis position_test_anterior e position_test fazendo com que position_test_anterior receba sempre o valor da posição anterior e o position_test o da posição seguinte de acordo com os valores de new_speed_test. Assim conseguimos verificar se com o valor de new_speed irá ser possível decrementar a velocidade para se chegar à final position com speed = 1 e se ao realizar este decremento a velocidade de cada casa respeita os valores de max_road_speed nos intervalos das posições de teste. Caso a velocidade em alguma casa neste decremento de teste não respeitar o speed limit então a variável speed_valido é colocada a 0 e é parado o loop fazendo com que o new_speed receba a próxima opção possível e teste novamente. Caso a position_test ultrapasse o valor da final_position quer dizer que a velocidade está muito elevada e então também irá parar o loop para realizar o teste com a nova velocidade possível.

Quando a new_speed_teste chega a 1 e todas as condições anteriores foram cumpridas com sucesso é porque o new_speed encontrou um valor válido e vai assim chamar novamente a função com os novos valores de move_number, a posição seguinte irá ser a position somado com o novo valor da velocidade (new_speed) e a nova velocidade.

Faz-se ainda um teste antes, caso a nova velocidade (new_speed) menos a velocidade atual (speed) seja 1 é porque a velocidade sofreu um incremento e por isso o array dinâmico é atualizado com os novos valores.



Solve 5 while

Nesta tentativa menos eficaz, começamos por pensar primeiro em tentar resolver o problema sem o efeito das velocidades máximas, e só depois colocar as validações necessárias para os limites de velocidade.

Para facilitar a resolução desta solução, criámos uma função que nos vai ajudar a simular as descidas:

```
static int descida(int nr){                                     //função que soma os numeros todos do argumento até 1
    int sum=0;
    for (int i = nr; i >=1; i--)
    {
        sum = sum + i;
    }
    return sum;
}
```

Versão sem limites de velocidade

Começamos por inicializar as nossas variáveis número de movimentos, posição, velocidade e colocamos logo no array que vai conter as diversas posições por onde vamos passar (solution_5.positions).

```
static void solve_5_while_lento(int final_position){
    int move_number = 0;                                         //o números de movimentos começa em 0
    int position = 0;                                           //a posição começa em 0
    int speed = 0;                                              //a velocidade começa em 0
    solution_5.positions[move_number] = position;             //a posição inicial vai ser 0
```

Continuado o código, criámos um ciclo While que vai iterando sucessivamente e de forma não linear (uma vez que a velocidade vai variar) a posição (int position). Vamos sair do programa quando a nossa posição atual for igual ao tamanho da estrada (800 neste exercício), antes de sair guardamos valores em solution_5_best.

```
while (position <= final_position)
{
    if (position == final_position && speed == 1)
    {
        solution_5_best = solution_5;
        solution_5_best.n_moves = move_number;
        break;
    }
}
```



Se passarmos a condição anterior, continuamos e vamos encontrar primeiramente uma validação para o aumento da velocidade. Sabendo que a velocidade não pode exceder o valor 9, vamos verificar se ainda nos restam casas suficientes para poder aumentar a velocidade e poder reduzir a velocidade para 1. Se não pudermos aumentar, vamos fazer a mesma verificação anterior, mas para manter a velocidade. Não podendo nem aumentar nem manter, temos obrigatoriamente que descer (*).

```

if (descida(speed+1) <= final_position-position && speed+1<10)
{
    speed++;
}
else
{
    if ((final_position - position) >= descida(speed))
    {
        speed=speed;
    }
    else
    {
        speed--;
    }
}

```

Depois de estas condições todas vamos ter a nossa velocidade nova, com isto, deslocamos-nos para a próxima posição (que depende da velocidade), incrementamos o número de movimentos, e guardamos essa nova posição no array solution_5.positions.

```

position = position + speed; //avançamos de posição
move_number++;
solution_5.positions[move_number] = position; //guardamos a posição
solution_5_count++;
}

```

Exemplo de uma estrada com 150 blocos, para a resolução sem limites de velocidade:

A variação da velocidade ao longo da estrada também está exposta no terminal.

Nr Blocos	Velocidades	Nr. movimentos	Count	Tempo
150	123456789999999876654321	25	25	2.410e-06



Versão com implementação dos limites de velocidade

Tendo restrições para os limites de velocidade não podemos aumentar sempre que der para a descer até 1, então dentro do ciclo while adicionamos duas variáveis de estado, que vão definir se a velocidade vai aumentar ou reduzir:

```
while (position <= final_position)
{
    int valAumenta = 1;      //variável inicializada a 1, pois pertende-se aumentar a velocidade sempre que possível
    int valDiminui = 0;      //variável inicializada a 0, para ser 1 quando for necessário diminuir a velocidade

    if (position == final_position && speed == 1)
    {
        solution_5_best = solution_5;
        solution_5_best.n_moves = move_number;
        break;
    }
}
```

De seguida vamos encontrar outra vez as condições para a descida da velocidade (*), dentro do primeiro if vamos inicializar 4 variáveis com a função de simular o aumento e a manutenção da velocidade.

```
if (descida(speed+1) <= final_position-position && speed+1<10)
{
    //-----max speed-----
    int manter_speed = speed;      //simula a velocidade a manter
    int manter_posi = position;    //simula a posição se a velocidade manter
    int new_speed = speed+1;       //simula a velocidade a aumentar
    int new_position = position;   //simula a posição se a velocidade aumentar
}
```

O ciclo for que se encontra na imagem posteriormente vai realizar essas mesmas simulações, esse ciclo vai percorrer da posição atual até à posição que chegaríamos se aumentássemos a velocidade, de seguida a reduzísssemos consecutivamente até 1, ou seja, vamos fazer verificações para as futuras “casas” onde vamos passar.

A primeira condição verifica se aumentando a velocidade (por isso o uso da variável new speed) vamos estar dentro dos limites de velocidade, se não estivermos a nossa variável de teste para o aumento vai ser dada com 0 (valAumenta=0). Para esta simulação ser realista temos que ir diminuindo a velocidade (new_speed), daí a implementação da terceira condição e a necessidade da variável new_position que vai aumentando, mas com a velocidade cada vez menor.



A segunda condição tem um significado semelhante, mas agora vamos nos referir à velocidade sem sofrer alterações (daí o uso de manter_speed), como é lógico se não vamos modificar a velocidade não precisamos de iterar por tantas posições. Se mesmo mantendo a velocidade esta não estiver dentro dos limites vamos ter que diminuir a velocidade (valDiminui=1 valAumenta=0) e saímos do ciclo. Como referido anteriormente para a simulação ser verdadeira temos que ir diminuindo esta velocidade a aumento a posição de teste consoante esta mesma velocidade, para isso temos a quarta condição.

```

for (int i = position; i <= position + descida(speed+1); i++)
{
    if (max_road_speed[i]<new_speed)
    {
        valAumenta = 0;
    }
    if (max_road_speed[i]<manter_speed && i<=position + descida(speed))
    {
        valDiminui = 1;
        valAumenta = 0;
        break;
    }

    if ( i == new_position+new_speed)
    {
        new_position = new_position + new_speed;
        new_speed--;
    }
    if ( i == manter_posi+manter_speed && i<=position + descida(speed))
    {
        manter_posi = manter_posi + manter_speed;
        manter_speed--;
    }
}

```

Tendo os variáveis valDiminuir e valAumenta com os valores corretos podemos alterar a nossa variável original da velocidade.

```

if (valAumenta == 1)
{
    speed++;
}else if (valDiminui == 1)
{
    speed--;
}else
{
    speed=speed;
}

//-----max speed-----
}

```



Passamos agora a primeira condição do else (*) para quando a velocidade só pode ou manter ou descer.

Aqui vamos inicializar outra vez as variáveis de simulação (new_speed e new_position) e implementamos um for idêntico aos utilizados anteriormente que vai percorrer as posições da atual até à posição que chegaríamos se reduzíssemos a velocidade consecutivamente.

Se mesmo mantendo a velocidade, esta não respeitar os limites vamos obrigatoriamente ter que diminuir (valDiminuir=1) e saímos do programa. Como referido anteriormente vamos ter que ir incrementando as posições da simulação (new_position) dependentemente da velocidade (new_speed).

```
else
{
    if ((final_position - position) >= descida(speed))
    {

        //-----max speed-----

        int new_speed = speed;
        int new_position = position;
        for (int i = position; i <= position + descida(speed); i++)
        {

            if (max_road_speed[i]<new_speed)
            {
                valDiminui = 1;
                break;
            }

            if ( i == new_position+new_speed)
            {

                new_position = new_position + new_speed;
                new_speed--;
            }
        }
    }
}
```



Como anteriormente tendo o valor correto de valDiminuir podemos alterar nossa variável original da velocidade.

```
if (valDiminui == 1)
{
    speed--;
}
else
{
    speed=speed;
}
//-----max speed-----
}
else
```

Por último, temos o else para quando estamos nos últimos blocos da estrada e temos necessariamente que reduzir a velocidade.

```
else
{
    speed--;
}
```

Tendo a velocidade atualizada dentro de todas as conformidades podemos incrementar a posição original com a velocidade, incrementar o número de movimentos, armazenamos a nova posição no array solution_5.positions e incrementamos o contador de vezes que entramos no ciclo while.

```
position = position + speed;
move_number++;
solution_5.positions[move_number] = position;
solution_5_count++;
}
```



Resultados Obtidos

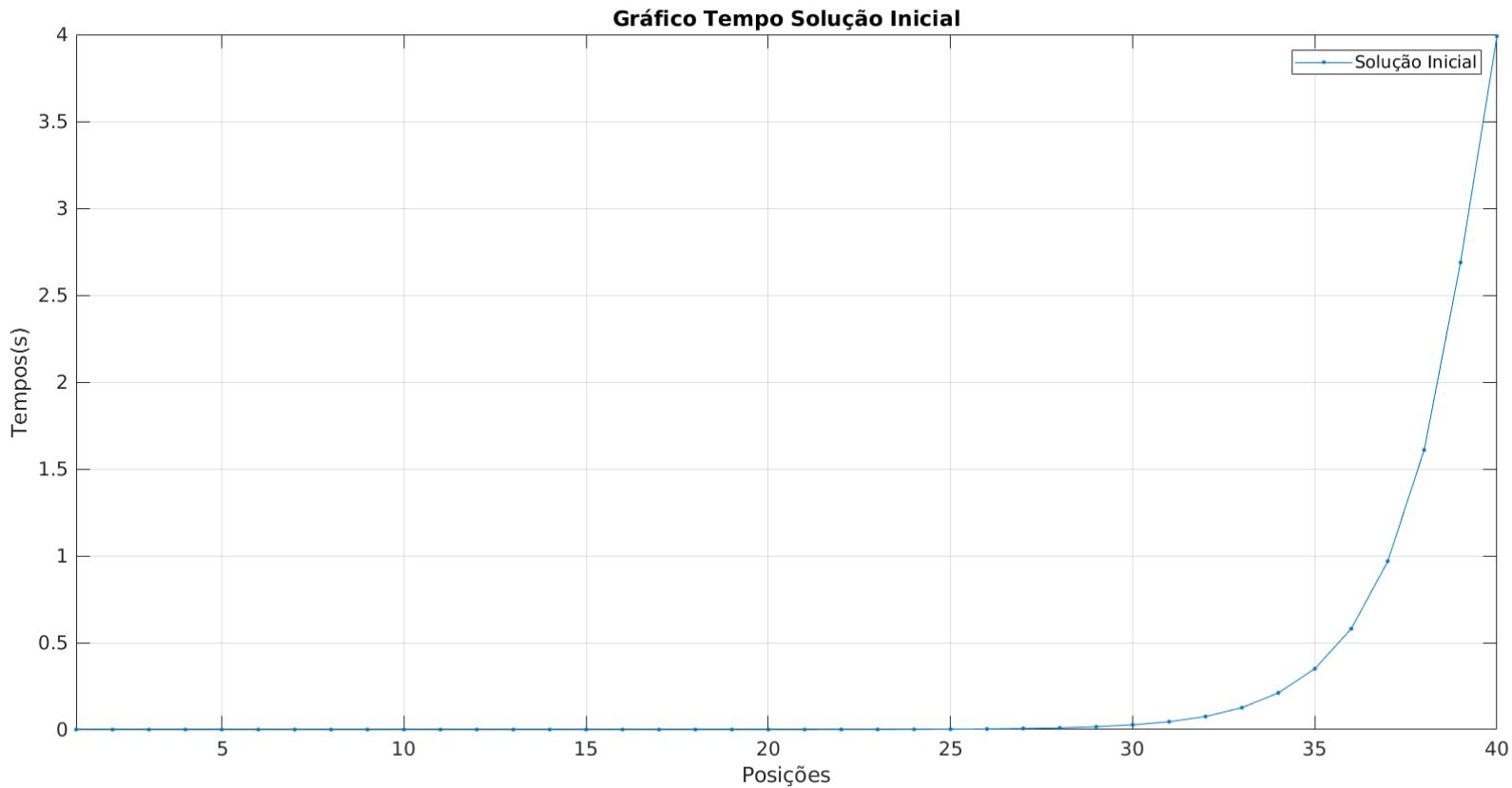
Os resultados obtidos vão depender do CPU de cada computador então para realizar esta parte do relatório fizemos os testes das soluções para cada computador, porém a realização dos gráficos em matlab foi apenas realizada no PC1 Diogo.

Além disso, os resultados obtidos em cada PC foram realizados com o número mecanográfico do utilizador deste (PC1 - 108902; PC2 - 108624; PC3 - 107853).

PC	CPU
PC1 Diogo	AMD Ryzen 7 5800H with Radeon Graphics
PC2 Tomás	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
PC3 Gonçalo	AMD Ryzen 9 5900HS with Radeon Graphics

Solve 1 inicial

Gráfico Matlab solve inicial





Solve 2 inicial melhorada

PC1

n	sol	count	cpu	time
1	1	2	5.620e-06	
2	2	2	1.462e-06	
3	3	5	1.813e-06	
4	3	3	1.343e-06	
5	4	4	1.433e-06	
6	4	6	1.644e-06	
7	5	7	1.704e-06	
8	5	7	1.493e-06	
9	5	7	1.443e-06	
10	6	9	1.823e-06	
11	6	9	9.320e-07	
12	6	6	4.210e-07	
13	7	13	5.210e-07	
14	7	14	5.010e-07	
15	7	13	4.710e-07	
16	7	8	4.000e-07	
17	8	11	4.100e-07	
18	8	17	4.510e-07	
19	8	17	5.010e-07	
20	8	12	4.400e-07	
21	9	15	1.093e-06	
22	9	17	5.510e-07	
23	9	22	6.010e-07	
24	9	18	5.410e-07	
25	9	10	3.810e-07	
26	10	12	4.310e-07	
27	10	12	4.410e-07	
28	10	10	4.210e-07	
29	10	12	4.110e-07	
30	11	13	4.110e-07	
31	11	13	4.100e-07	
32	11	10	3.800e-07	
33	12	11	4.400e-07	
34	12	11	4.000e-07	
35	12	14	4.010e-07	
36	13	15	2.180e-07	
37	13	15	4.000e-07	
38	13	12	3.900e-07	
39	14	13	4.100e-07	
40	14	13	0.000e+00	
41	14	16	1.022e-06	
42	15	18	4.610e-07	
43	15	18	4.510e-07	
44	15	15	4.410e-07	
45	16	18	4.610e-07	
46	16	19	4.710e-07	
47	16	23	4.710e-07	
48	16	18	4.710e-07	
		49	17	0.000e+00
		50	17	3.830e-07
		55	19	1.242e-06
		60	22	5.510e-07
		65	24	5.010e-07
		70	26	6.010e-07
		75	27	6.410e-07
		80	28	7.610e-07
		85	29	8.210e-07
		90	30	9.020e-07
		95	32	6.710e-07
		100	34	5.310e-07
		110	39	1.022e-06
		120	42	5.510e-07
		130	44	6.220e-07
		140	46	5.820e-07
		150	48	6.620e-07
		160	52	4.810e-07
		170	57	4.110e-07
		180	60	1.723e-06
		190	62	1.021e-06
		200	64	1.162e-06
		220	72	1.613e-06
		240	79	1.042e-06
		260	83	1.052e-06
		280	89	9.820e-07
		300	97	9.410e-07
		320	101	9.620e-07
		340	111	1.042e-06
		360	115	9.210e-07
		380	120	1.202e-06
		400	128	1.032e-06
		420	134	5.210e-06
		440	139	1.112e-06
		460	147	1.062e-06
		480	152	1.152e-06
		500	157	1.202e-06
		520	164	1.052e-06
		540	169	1.202e-06
		560	175	8.210e-07
		580	182	1.242e-06
		600	186	1.132e-06
		620	192	1.112e-06
		640	200	1.192e-06
		660	204	1.313e-06
		680	210	3.174e-06
		700	218	4.100e-07
		720	222	1.322e-06
		740	231	1.322e-06
		760	235	1.322e-06
		780	239	1.513e-06
		800	248	1.262e-06



PC2

1	1	2	6.385e-06	50	17	17	7.860e-07
2	2	2	2.612e-06	55	19	17	1.300e-06
3	3	5	2.786e-06	60	21	20	7.470e-07
4	3	3	1.730e-06	65	23	13	6.070e-07
5	4	4	2.074e-06	70	24	8	5.290e-07
6	4	6	2.338e-06	75	26	33	1.027e-06
7	5	7	2.420e-06	80	27	40	1.134e-06
8	5	7	2.533e-06	85	28	40	1.029e-06
9	5	7	2.434e-06	90	29	33	1.044e-06
10	6	9	3.062e-06	95	31	31	9.620e-07
11	6	9	3.718e-06	100	33	31	9.120e-07
12	6	6	1.244e-06	110	38	31	2.253e-06
13	7	13	1.802e-06	120	41	33	9.880e-07
14	7	14	1.720e-06	130	43	38	1.104e-06
15	7	13	1.688e-06	140	45	39	9.580e-07
16	7	8	1.521e-06	150	47	21	7.870e-07
17	8	11	1.525e-06	160	51	16	6.100e-07
18	8	17	1.772e-06	170	56	20	6.030e-07
19	8	17	1.827e-06	180	59	69	2.114e-06
20	8	13	1.642e-06	190	61	70	1.520e-06
21	9	16	2.403e-06	200	63	81	1.804e-06
22	9	18	9.980e-07	220	70	71	2.717e-06
23	9	23	1.187e-06	240	77	85	1.936e-06
24	9	20	1.022e-06	260	81	87	1.906e-06
25	9	12	7.990e-07	280	88	81	1.709e-06
26	10	15	9.750e-07	300	95	86	1.815e-06
27	10	17	8.880e-07	320	99	85	1.716e-06
28	10	15	8.300e-07	340	108	93	1.815e-06
29	10	17	8.580e-07	360	113	89	1.665e-06
30	10	11	6.060e-07	380	118	95	1.816e-06
31	11	12	6.950e-07	400	126	97	1.870e-06
32	11	12	7.480e-07	420	132	97	3.933e-06
33	11	9	6.140e-07	440	137	94	2.175e-06
34	12	10	7.320e-07	460	145	97	1.963e-06
35	12	16	6.800e-07	480	150	107	2.029e-06
36	12	13	5.930e-07	500	157	95	1.758e-06
37	13	14	6.690e-07	520	164	103	1.909e-06
38	13	14	6.540e-07	540	168	118	2.297e-06
39	13	11	5.910e-07	560	176	118	2.454e-06
40	14	12	6.510e-07	580	183	126	2.383e-06
41	14	18	6.880e-07	600	187	127	2.284e-06
42	14	15	6.990e-07	620	193	119	2.145e-06
43	15	17	7.390e-07	640	201	133	2.370e-06
44	15	17	7.600e-07	660	206	134	4.604e-06
45	15	14	5.495e-06	680	212	137	2.410e-06
46	16	16	8.110e-07	700	221	137	2.352e-06
47	16	22	8.880e-07	720	226	142	2.623e-06
48	16	20	8.280e-07	740	235	150	2.613e-06
49	16	16	7.240e-07	760	240	164	2.884e-06
				780	244	146	2.683e-06
				800	253	147	2.575e-06

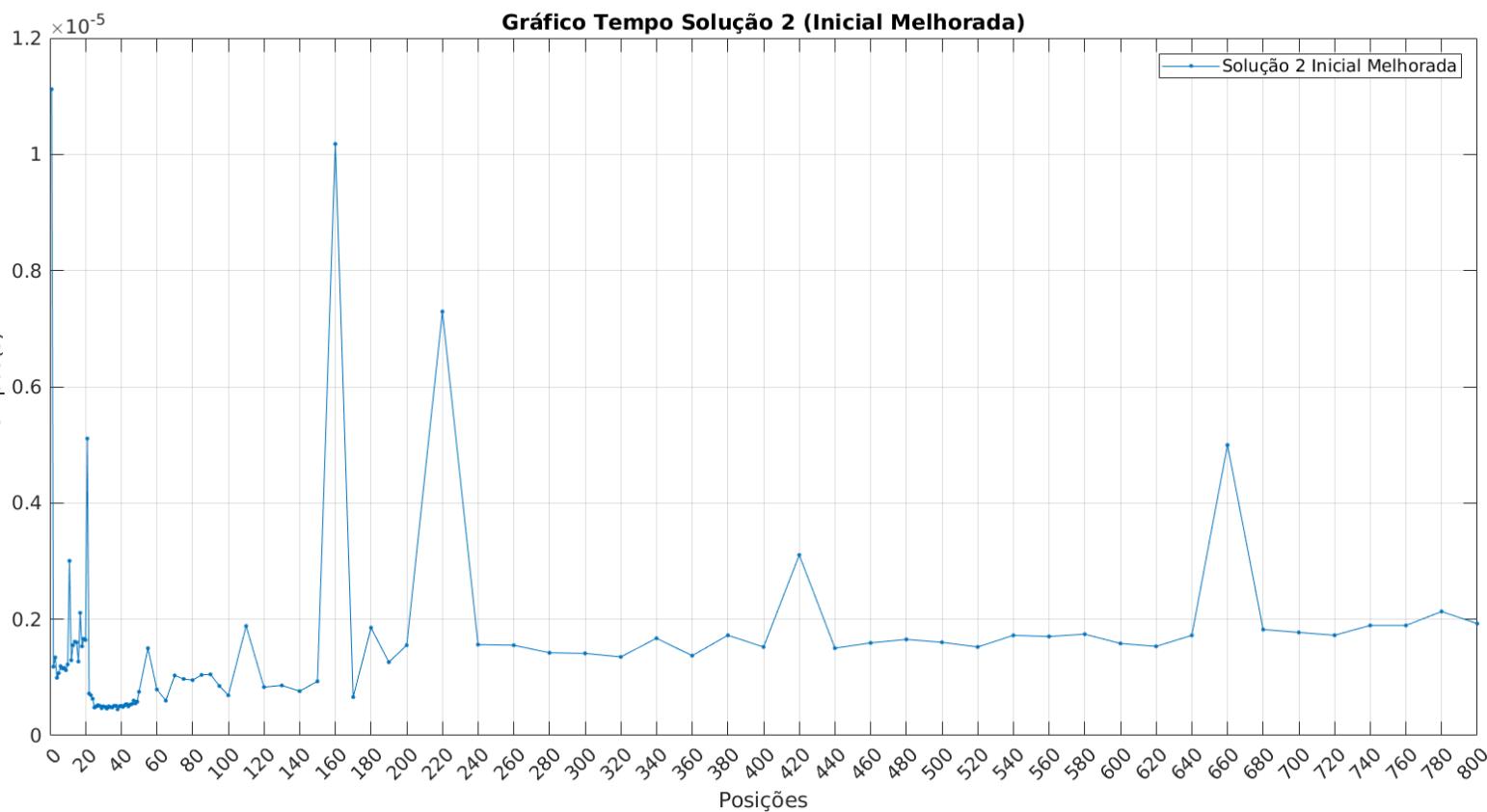


PC3

1	1	2 2.191e-06	47	16	19 0.000e+00
2	2	2 4.200e-07	48	17	20 0.000e+00
3	3	5 4.800e-07	49	17	20 0.000e+00
4	3	3 3.800e-07	50	17	17 4.410e-07
5	4	4 4.300e-07	55	19	17 1.040e-06
6	4	6 4.610e-07	60	22	20 5.600e-07
7	5	7 4.500e-07	65	24	15 5.100e-07
8	5	7 4.300e-07	70	26	33 6.410e-07
9	5	7 4.000e-07	75	27	36 5.800e-07
10	6	9 4.500e-07	80	28	43 7.600e-07
11	6	9 8.800e-07	85	29	44 7.700e-07
12	6	6 3.700e-07	90	30	35 0.000e+00
13	7	13 5.100e-07	95	32	33 6.500e-07
14	7	14 4.700e-07	100	35	28 5.500e-07
15	7	13 4.400e-07	110	39	23 1.290e-06
16	7	8 3.700e-07	120	42	33 5.700e-07
17	8	11 4.200e-07	130	44	39 6.000e-07
18	8	17 4.300e-07	140	46	25 0.000e+00
19	8	17 4.900e-07	150	48	24 6.500e-07
20	8	13 4.400e-07	160	52	16 4.500e-07
21	9	16 3.831e-06	170	57	18 0.000e+00
22	9	18 5.400e-07	180	60	73 1.050e-06
23	9	21 1.171e-06	190	62	84 0.000e+00
24	9	17 5.700e-07	200	64	82 9.300e-07
25	9	10 4.400e-07	220	70	75 4.261e-06
26	10	11 4.300e-07	240	78	90 1.141e-06
27	10	11 4.000e-07	260	82	93 0.000e+00
28	10	8 0.000e+00	280	89	85 0.000e+00
29	11	15 0.000e+00	300	97	90 0.000e+00
30	11	15 4.100e-07	320	102	93 0.000e+00
31	11	12 0.000e+00	340	111	103 0.000e+00
32	12	13 0.000e+00	360	115	90 0.000e+00
33	12	13 0.000e+00	380	120	101 0.000e+00
34	12	10 0.000e+00	400	129	108 0.000e+00
35	13	17 4.500e-08	420	135	120 2.221e-06
36	13	17 4.110e-07	440	140	106 1.130e-06
37	13	14 0.000e+00	460	148	111 1.100e-06
38	14	15 0.000e+00	480	154	127 0.000e+00
39	14	15 3.900e-07	500	160	123 0.000e+00
40	14	12 0.000e+00	520	168	121 0.000e+00
41	15	20 0.000e+00	540	172	142 0.000e+00
42	15	20 4.300e-07	560	179	126 0.000e+00
43	15	17 3.900e-07	580	186	141 0.000e+00
44	16	19 4.600e-07	600	190	134 0.000e+00
45	16	19 0.000e+00	620	197	129 0.000e+00
46	16	17 0.000e+00	640	205	144 0.000e+00
47	16	19 0.000e+00	660	209	127 0.000e+00
48	17	20 0.000e+00	680	216	131 0.000e+00
49	17	20 0.000e+00	700	224	145 0.000e+00
			720	229	144 4.881e-06
			740	238	132 5.350e-07
			760	243	146 1.340e-06
			780	247	145 1.370e-06
			800	256	135 0.000e+00



Gráfico Matlab Solve 2





Solve 3 while dynamic

PC1

n	sol	count	cpu	time			
1	1	1	1.062e-06		49	17	3.400e-07
2	2	1	3.710e-07		50	17	3.410e-07
3	3	2	3.600e-07		55	19	8.710e-07
4	3	2	3.610e-07		60	22	4.800e-07
5	4	2	3.110e-07		65	24	5.010e-07
6	4	2	3.110e-07		70	26	4.910e-07
7	5	3	3.110e-07		75	27	4.210e-07
8	5	3	2.910e-07		80	28	4.610e-07
9	5	3	3.600e-07		85	29	4.910e-07
10	6	3	2.910e-07		90	30	1.840e-07
11	6	3	1.462e-06		95	32	4.210e-07
12	6	3	7.810e-07		100	34	4.310e-07
13	7	4	7.410e-07		110	39	2.435e-06
14	7	4	7.120e-07		120	42	4.210e-07
15	7	4	7.110e-07		130	44	3.010e-07
16	7	4	8.010e-07		140	46	3.410e-07
17	8	4	6.710e-07		150	48	3.810e-07
18	8	4	7.610e-07		160	52	3.300e-07
19	8	4	6.720e-07		170	57	4.810e-07
20	8	4	8.420e-07		180	60	3.210e-07
21	9	5	9.820e-07		190	62	3.210e-07
22	9	5	5.210e-07		200	64	3.110e-07
23	9	5	4.610e-07		220	72	7.720e-07
24	9	5	4.500e-07		240	79	4.510e-07
25	9	5	4.810e-07		260	83	3.510e-07
26	10	5	3.810e-07		280	89	3.610e-07
27	10	5	3.710e-07		300	97	1.092e-06
28	10	5	4.510e-07		320	101	3.200e-07
29	10	5	4.010e-07		340	111	4.810e-07
30	11	6	3.900e-07		360	115	3.010e-07
31	11	6	3.810e-07		380	120	0.0000e+00
32	11	6	0.0000e+00		400	128	3.510e-07
33	12	7	3.710e-07		420	134	1.523e-06
34	12	7	1.930e-07		440	139	5.010e-07
35	12	7	4.210e-07		460	147	5.510e-07
36	13	8	0.0000e+00		480	152	3.400e-07
37	13	8	4.010e-07		500	157	3.210e-07
38	13	8	2.020e-07		520	164	3.810e-07
39	14	9	3.910e-07		540	169	3.710e-07
40	14	9	4.030e-07		560	175	3.710e-07
41	14	9	3.900e-07		580	182	3.910e-07
42	15	10	6.810e-07		600	186	3.700e-07
43	15	10	1.770e-07		620	192	4.010e-07
44	15	10	4.810e-07		640	200	3.600e-07
45	16	11	4.510e-07		660	204	7.500e-08
46	16	11	4.410e-07		680	210	3.600e-07
47	16	11	5.210e-07		700	218	3.800e-07
48	16	11	4.910e-07		720	222	1.960e-07
					740	231	3.310e-07
					760	235	2.450e-07
					780	239	4.010e-07
					800	248	4.310e-07



PC2

1	1	1	2.339e-06
2	2	1	1.298e-06
3	3	2	1.015e-06
4	3	2	1.304e-06
5	4	2	1.157e-06
6	4	2	1.423e-06
7	5	3	1.107e-06
8	5	3	1.026e-06
9	5	3	1.479e-06
10	6	3	1.333e-06
11	6	3	7.530e-07
12	6	3	3.840e-07
13	7	4	3.430e-07
14	7	4	3.750e-07
15	7	4	3.190e-07
16	7	4	4.340e-07
17	8	4	3.880e-07
18	8	4	3.270e-07
19	8	4	3.580e-07
20	8	4	4.000e-07
21	9	5	9.010e-07
22	9	5	4.830e-07
23	9	5	3.700e-07
24	9	5	3.730e-07
25	9	5	4.330e-07
26	10	5	3.870e-07
27	10	5	3.400e-07
28	10	5	3.510e-07
29	10	5	3.680e-07
30	10	5	3.860e-07
31	11	6	2.960e-07
32	11	6	2.830e-07
33	11	6	3.080e-07
34	12	7	3.050e-07
35	12	7	3.140e-07
36	12	7	3.260e-07
37	13	8	3.210e-07
38	13	8	3.810e-07
39	13	8	3.210e-07
40	14	9	3.250e-07
41	14	9	3.400e-07
42	14	9	3.700e-07
43	15	10	3.570e-07
44	15	10	3.880e-07
45	15	10	3.770e-07
46	16	11	4.190e-07
47	16	11	3.960e-07
48	16	11	3.800e-07
49	16	11	4.110e-07

50	17	4	3.110e-07
55	19	6	6.180e-07
60	21	8	3.860e-07
65	23	10	3.950e-07
70	24	5	3.920e-07
75	26	7	4.410e-07
80	27	5	4.120e-07
85	28	6	5.010e-07
90	29	6	5.290e-07
95	31	8	4.470e-07
100	33	10	3.620e-07
110	38	15	1.242e-06
120	41	18	6.450e-07
130	43	6	4.910e-07
140	45	7	5.540e-07
150	47	9	5.170e-07
160	51	13	5.550e-07
170	56	18	5.480e-07
180	59	21	6.110e-07
190	61	6	5.690e-07
200	63	6	5.940e-07
220	70	13	1.030e-06
240	77	19	7.540e-07
260	81	8	5.480e-07
280	88	13	6.150e-07
300	95	14	5.620e-07
320	99	8	4.530e-07
340	108	15	5.010e-07
360	113	7	4.540e-07
380	118	10	6.190e-07
400	126	14	5.700e-07
420	132	20	1.126e-06
440	137	10	6.420e-07
460	145	17	6.130e-07
480	150	16	6.250e-07
500	157	11	4.770e-07
520	164	17	5.370e-07
540	168	7	5.970e-07
560	176	13	6.430e-07
580	183	20	7.400e-07
600	187	9	6.380e-07
620	193	14	5.550e-07
640	201	18	5.410e-07
660	206	9	5.110e-07
680	212	14	5.890e-07
700	221	14	5.140e-07
720	226	9	5.120e-07
740	235	16	5.220e-07
760	240	7	5.320e-07
780	244	9	5.780e-07
800	253	18	5.770e-07



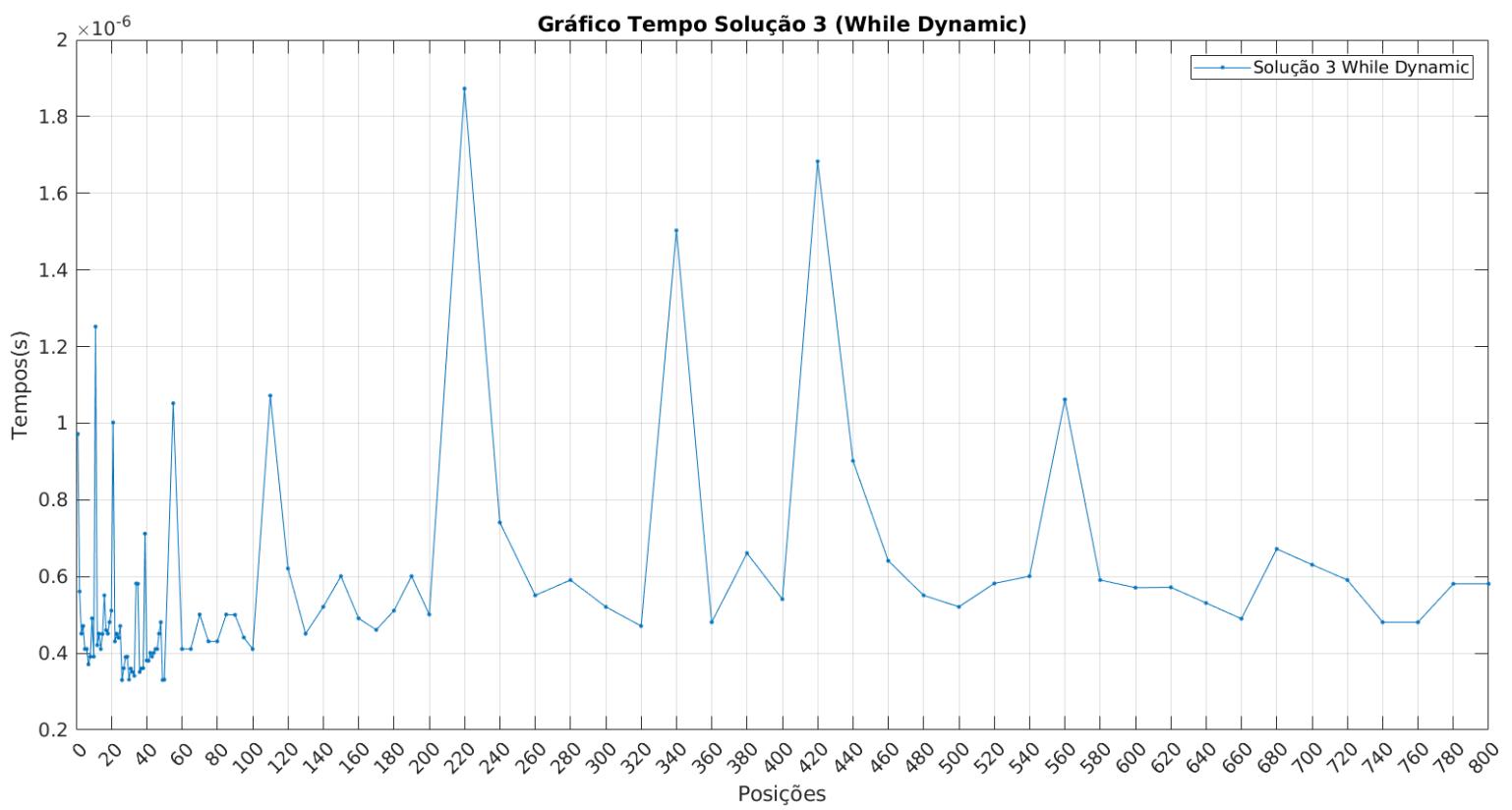
PC3

```
goncalo@goncalo:~/Desktop/Projeto_1
 1   1           1 2.000e-06
 2   2           1 1.270e-06
 3   3           2 1.120e-06
 4   3           2 1.260e-06
 5   4           2 9.300e-07
 6   4           2 9.400e-07
 7   5           3 8.910e-07
 8   5           3 9.000e-07
 9   5           3 1.190e-06
10   6           3 1.010e-06
11   6           3 1.430e-06
12   6           3 7.900e-07
13   7           4 8.300e-07
14   7           4 7.800e-07
15   7           4 9.800e-07
16   7           4 8.600e-07
17   8           4 9.010e-07
18   8           4 8.000e-07
19   8           4 1.060e-06
20   8           4 0.000e+00
21   9           5 8.000e-07
22   9           5 3.310e-07
23   9           5 2.700e-07
24   9           5 2.800e-07
25   9           5 0.000e+00
26  10           5 0.000e+00
27  10           5 1.570e-07
28  10           5 2.300e-07
29  11           6 0.000e+00
30  11           6 0.000e+00
31  11           6 2.300e-07
32  12           7 2.300e-07
33  12           7 0.000e+00
34  12           7 0.000e+00
35  13           8 1.480e-07
36  13           8 0.000e+00
37  13           8 0.000e+00
38  14           9 0.000e+00
39  14           9 2.500e-07
40  14           9 2.900e-07
41  15          10 0.000e+00
42  15          10 0.000e+00
43  15          10 0.000e+00
44  16          11 2.600e-07
45  16          11 0.000e+00
46  16          11 0.000e+00
47  16          11 0.000e+00
48  17           4 2.100e-07
49  17           4 0.000e+00
```

50	17	4 0.000e+00
55	19	6 5.200e-07
60	22	9 3.200e-07
65	24	11 3.500e-07
70	26	13 0.000e+00
75	27	5 0.000e+00
80	28	5 3.000e-07
85	29	6 0.000e+00
90	30	6 0.000e+00
95	32	8 3.100e-07
100	35	11 0.000e+00
110	39	15 0.000e+00
120	42	5 0.000e+00
130	44	6 0.000e+00
140	46	7 0.000e+00
150	48	9 0.000e+00
160	52	13 0.000e+00
170	57	14 0.000e+00
180	60	17 0.000e+00
190	62	6 0.000e+00
200	64	7 0.000e+00
220	70	12 7.400e-07
240	78	20 5.500e-07
260	82	8 3.700e-07
280	89	13 4.700e-07
300	97	21 0.000e+00
320	102	10 0.000e+00
340	111	16 0.000e+00
360	115	8 0.000e+00
380	120	10 0.000e+00
400	129	16 0.000e+00
420	135	22 9.500e-07
440	140	10 3.900e-07
460	148	18 4.200e-07
480	154	11 3.700e-07
500	160	11 3.900e-07
520	168	18 4.100e-07
540	172	8 0.000e+00
560	179	13 0.000e+00
580	186	19 0.000e+00
600	190	8 0.000e+00
620	197	14 0.000e+00
640	205	22 0.000e+00
660	209	8 0.000e+00
680	216	14 0.000e+00
700	224	22 0.000e+00
720	229	10 0.000e+00
740	238	16 0.000e+00
760	243	8 0.000e+00
780	247	10 0.000e+00
800	256	19 0.000e+00



Gráfico Matlab Solve 3





Solve 4 dynamic

PC1

n	sol	count	cpu	time			
1	1	2	3.236e-06		49	17	5 2.600e-07
2	2	2	1.362e-06		50	17	5 2.710e-07
3	3	3	1.383e-06		55	19	7 6.420e-07
4	3	3	1.573e-06		60	22	10 4.200e-07
5	4	3	1.423e-06		65	24	12 3.910e-07
6	4	3	1.453e-06		70	26	14 4.100e-07
7	5	4	1.382e-06		75	27	6 3.210e-07
8	5	4	1.422e-06		80	28	6 3.300e-07
9	5	4	1.643e-06		85	29	7 4.010e-07
10	6	4	1.463e-06		90	30	7 4.110e-07
11	6	4	1.153e-06		95	32	9 4.010e-07
12	6	4	6.210e-07		100	34	11 3.900e-07
13	7	5	5.920e-07		110	39	16 9.110e-07
14	7	5	5.110e-07		120	42	19 6.510e-07
15	7	5	4.910e-07		130	44	7 3.710e-07
16	7	5	6.310e-07		140	46	8 3.810e-07
17	8	5	5.410e-07		150	48	10 5.310e-07
18	8	5	5.210e-07		160	52	11 4.210e-07
19	8	5	5.010e-07		170	57	16 3.900e-07
20	8	5	5.610e-07		180	60	19 5.510e-07
21	9	6	7.310e-07		190	62	7 4.110e-07
22	9	6	4.110e-07		200	64	7 3.910e-07
23	9	6	3.100e-07		220	72	15 8.510e-07
24	9	6	3.410e-07		240	79	21 7.920e-07
25	9	6	3.700e-07		260	83	9 4.510e-07
26	10	6	3.210e-07		280	89	13 5.810e-07
27	10	6	2.710e-07		300	97	16 4.610e-07
28	10	6	3.000e-07		320	101	9 4.310e-07
29	10	6	0.000e+00		340	111	17 4.910e-07
30	11	7	2.910e-07		360	115	8 3.910e-07
31	11	7	2.600e-07		380	120	11 5.510e-07
32	11	7	2.900e-07		400	128	15 3.650e-07
33	12	8	5.800e-08		420	134	20 9.620e-07
34	12	8	2.810e-07		440	139	11 5.310e-07
35	12	8	2.900e-07		460	147	18 5.510e-07
36	13	9	0.000e+00		480	152	9 4.210e-07
37	13	9	2.900e-07		500	157	11 4.510e-07
38	13	9	3.010e-07		520	164	17 4.610e-07
39	14	10	2.910e-07		540	169	12 4.610e-07
40	14	10	2.900e-07		560	175	13 5.010e-07
41	14	10	2.470e-07		580	182	19 5.310e-07
42	15	11	3.200e-07		600	186	9 4.210e-07
43	15	11	3.400e-07		620	192	14 4.810e-07
44	15	11	3.310e-07		640	200	18 5.010e-07
45	16	12	3.610e-07		660	204	9 4.110e-07
46	16	12	3.200e-07		680	210	14 4.910e-07
47	16	12	3.910e-07		700	218	15 4.800e-07
48	16	12	4.510e-07		720	222	9 4.110e-07
					740	231	16 4.110e-07
					760	235	9 3.910e-07
					780	239	10 4.810e-07
					800	248	16 5.110e-07



PC2

1	1	2	2.672e-06	50	17	5	3.990e-07
2	2	2	1.444e-06	55	19	7	8.560e-07
3	3	3	1.662e-06	60	21	9	4.910e-07
4	3	3	1.386e-06	65	23	11	4.860e-07
5	4	3	1.198e-06	70	24	6	4.150e-07
6	4	3	1.521e-06	75	26	8	4.970e-07
7	5	4	1.362e-06	80	27	6	5.350e-07
8	5	4	1.462e-06	85	28	7	5.350e-07
9	5	4	1.814e-06	90	29	7	5.380e-07
10	6	4	1.633e-06	95	31	9	5.450e-07
11	6	4	7.840e-07	100	33	11	5.430e-07
12	6	4	3.810e-07	110	38	16	1.093e-06
13	7	5	5.300e-07	120	41	19	8.690e-07
14	7	5	4.070e-07	130	43	7	6.660e-07
15	7	5	3.970e-07	140	45	8	7.210e-07
16	7	5	4.850e-07	150	47	10	8.260e-07
17	8	5	4.740e-07	160	51	14	8.400e-07
18	8	5	3.710e-07	170	56	19	1.106e-06
19	8	5	4.480e-07	180	59	22	9.870e-07
20	8	5	4.640e-07	190	61	7	7.890e-07
21	9	6	1.797e-06	200	63	7	8.240e-07
22	9	6	5.780e-07	220	70	14	1.412e-06
23	9	6	5.110e-07	240	77	20	1.402e-06
24	9	6	4.860e-07	260	81	9	1.047e-06
25	9	6	4.830e-07	280	88	14	1.101e-06
26	10	6	5.370e-07	300	95	15	9.680e-07
27	10	6	4.540e-07	320	99	9	8.190e-07
28	10	6	4.420e-07	340	108	16	9.480e-07
29	10	6	4.910e-07	360	113	8	8.000e-07
30	10	6	4.880e-07	380	118	11	1.179e-06
31	11	7	3.860e-07	400	126	15	1.057e-06
32	11	7	3.980e-07	420	132	21	2.344e-06
33	11	7	3.620e-07	440	137	11	7.910e-07
34	12	8	4.240e-07	460	145	18	9.440e-07
35	12	8	3.580e-07	480	150	17	9.740e-07
36	12	8	3.390e-07	500	157	12	7.590e-07
37	13	9	3.980e-07	520	164	18	8.510e-07
38	13	9	3.900e-07	540	168	8	8.000e-07
39	13	9	3.780e-07	560	176	14	8.420e-07
40	14	10	3.810e-07	580	183	21	1.034e-06
41	14	10	3.920e-07	600	187	10	7.080e-07
42	14	10	4.880e-07	620	193	15	8.170e-07
43	15	11	4.490e-07	640	201	19	8.710e-07
44	15	11	4.360e-07	660	206	10	6.580e-07
45	15	11	4.480e-07	680	212	15	8.440e-07
46	16	12	4.870e-07	700	221	15	6.600e-07
47	16	12	4.420e-07	720	226	10	7.030e-07
48	16	12	5.540e-07	740	235	17	7.180e-07
49	16	12	5.940e-07	760	240	8	5.900e-07
				780	244	10	8.700e-07
				800	253	19	1.180e-06



PC3

1	1	2 2.431e-06	50	17	5 0.000e+00
2	2	2 1.230e-06	55	19	7 6.810e-07
3	3	3 1.330e-06	60	22	10 4.500e-07
4	3	3 1.270e-06	65	24	12 4.000e-07
5	4	3 1.721e-06	70	26	14 4.710e-07
6	4	3 2.020e-06	75	27	6 0.000e+00
7	5	4 2.260e-06	80	28	6 0.000e+00
8	5	4 2.070e-06	85	29	7 3.900e-07
9	5	4 1.831e-06	90	30	7 3.930e-07
10	6	4 2.020e-06	95	32	9 3.800e-07
11	6	4 1.270e-06	100	35	12 4.000e-07
12	6	4 6.500e-07	110	39	16 1.150e-06
13	7	5 6.900e-07	120	42	6 3.700e-07
14	7	5 0.000e+00	130	44	7 3.800e-07
15	7	5 8.000e-07	140	46	8 3.800e-07
16	7	5 0.000e+00	150	48	10 4.700e-07
17	8	5 9.400e-07	160	52	14 0.000e+00
18	8	5 1.080e-06	170	57	15 0.000e+00
19	8	5 8.400e-07	180	60	18 0.000e+00
20	8	5 1.000e-07	190	62	7 0.000e+00
21	9	6 8.300e-07	200	64	8 0.000e+00
22	9	6 4.600e-07	220	70	13 1.110e-06
23	9	6 3.600e-07	240	78	21 1.080e-06
24	9	6 3.300e-07	260	82	9 6.000e-07
25	9	6 0.000e+00	280	89	14 6.400e-07
26	10	6 0.000e+00	300	97	22 7.700e-07
27	10	6 0.000e+00	320	102	11 6.400e-07
28	10	6 1.510e-07	340	111	17 5.700e-07
29	11	7 0.000e+00	360	115	9 4.600e-07
30	11	7 0.000e+00	380	120	11 6.000e-07
31	11	7 0.000e+00	400	129	17 5.700e-07
32	12	8 2.800e-07	420	135	23 1.680e-06
33	12	8 0.000e+00	440	140	11 7.100e-07
34	12	8 0.000e+00	460	148	19 7.000e-07
35	13	9 0.000e+00	480	154	12 0.000e+00
36	13	9 0.000e+00	500	160	12 0.000e+00
37	13	9 3.000e-07	520	168	19 0.000e+00
38	14	10 0.000e+00	540	172	9 0.000e+00
39	14	10 0.000e+00	560	179	14 0.000e+00
40	14	10 0.000e+00	580	186	20 0.000e+00
41	15	11 0.000e+00	600	190	9 0.000e+00
42	15	11 3.400e-07	620	197	15 0.000e+00
43	15	11 0.000e+00	640	205	23 0.000e+00
44	16	12 0.000e+00	660	209	9 0.000e+00
45	16	12 0.000e+00	680	216	15 0.000e+00
46	16	12 3.600e-07	700	224	23 0.000e+00
47	16	12 0.000e+00	720	229	11 0.000e+00
48	17	5 0.000e+00	740	238	17 0.000e+00
49	17	5 0.000e+00	760	243	9 0.000e+00
50	17	5 0.000e+00	780	247	11 0.000e+00
			800	256	20 0.000e+00



Gráfico Matlab Solve 4

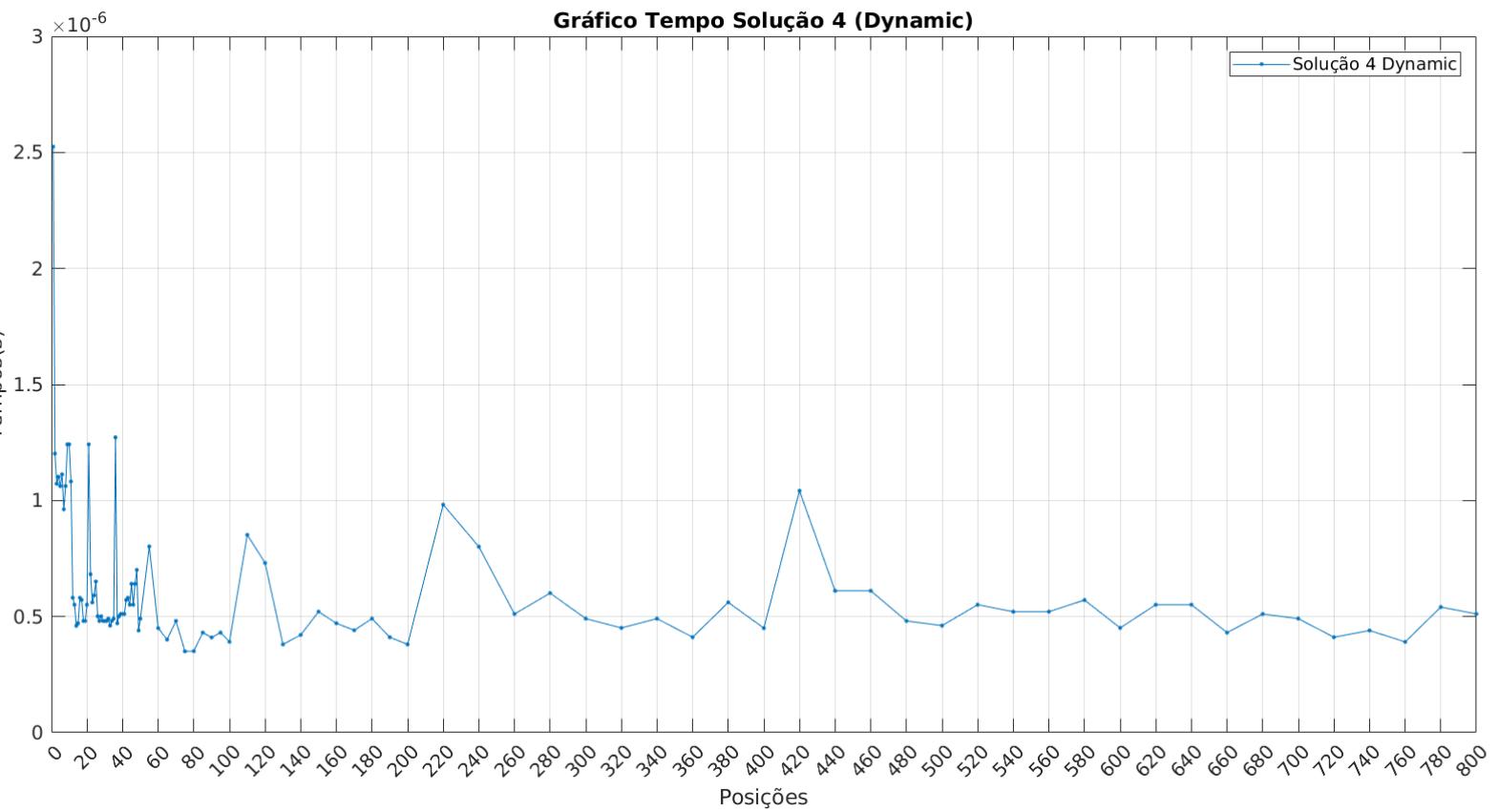
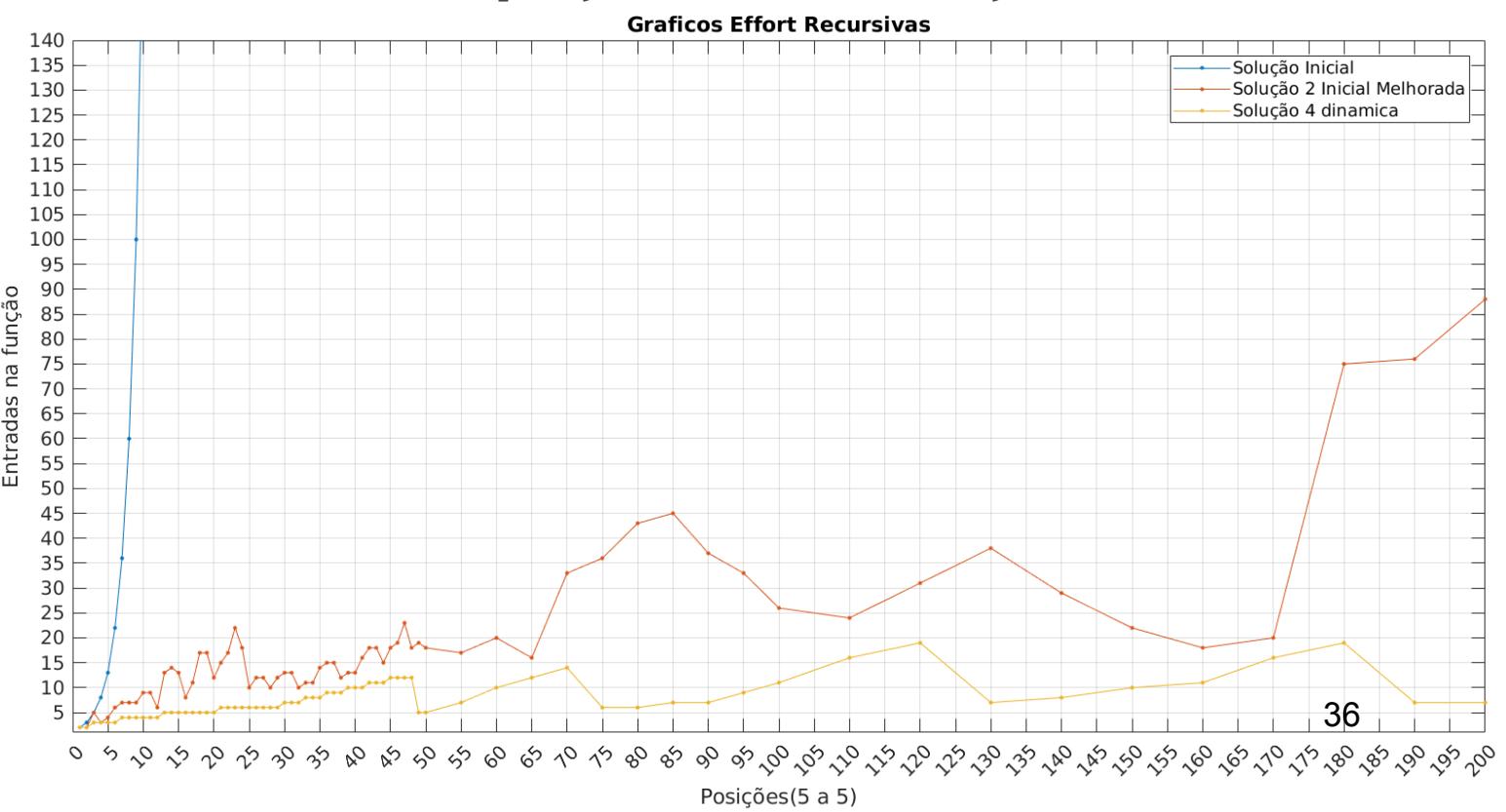


Gráfico comparação de entradas nas funções recursivas:





Solve 5 while

PC1

n	sol	count	cpu	time	n	sol	count	cpu	time
1	1	1	4.038e-06		49	17	17	6.510e-07	
2	2	2	2.164e-06		50	17	17	4.190e-07	
3	3	3	1.673e-06		55	19	19	1.713e-06	
4	3	3	2.174e-06		60	22	22	9.820e-07	
5	4	4	2.084e-06		65	24	24	9.320e-07	
6	4	4	1.834e-06		70	26	26	8.820e-07	
7	5	5	1.834e-06		75	27	27	8.920e-07	
8	5	5	1.723e-06		80	28	28	9.620e-07	
9	5	5	2.315e-06		85	29	29	9.920e-07	
10	6	6	2.535e-06		90	30	30	1.052e-06	
11	6	6	9.910e-07		95	32	32	1.282e-06	
12	6	6	4.910e-07		100	34	34	6.090e-07	
13	7	7	4.300e-07		110	39	39	2.004e-06	
14	7	7	4.200e-07		120	42	42	1.212e-06	
15	7	7	3.700e-07		130	44	44	1.192e-06	
16	7	7	4.810e-07		140	46	46	9.260e-07	
17	8	8	5.010e-07		150	48	48	1.373e-06	
18	8	8	4.110e-07		160	52	52	1.050e-06	
19	8	8	3.710e-07		170	57	57	1.734e-06	
20	8	8	4.410e-07		180	60	60	1.783e-06	
21	9	9	4.127e-06		190	62	62	1.734e-06	
22	9	9	5.810e-07		200	64	64	1.834e-06	
23	9	9	4.610e-07		220	72	72	5.059e-06	
24	9	9	4.310e-07		240	79	79	2.335e-06	
25	9	9	5.910e-07		260	83	83	2.275e-06	
26	10	10	5.610e-07		280	89	89	2.535e-06	
27	10	10	4.410e-07		300	97	97	2.545e-06	
28	10	10	4.410e-07		320	101	101	2.635e-06	
29	10	10	4.310e-07		340	111	111	2.906e-06	
30	11	11	5.610e-07		360	115	115	2.986e-06	
31	11	11	5.510e-07		380	120	120	3.287e-06	
32	11	11	5.410e-07		400	128	128	3.306e-06	
33	12	12	5.510e-07		420	134	134	4.288e-06	
34	12	12	5.610e-07		440	139	139	3.757e-06	
35	12	12	5.310e-07		460	147	147	3.807e-06	
36	13	13	8.510e-07		480	152	152	3.918e-06	
37	13	13	5.970e-07		500	157	157	4.166e-06	
38	13	13	6.910e-07		520	164	164	4.057e-06	
39	14	14	6.420e-07		540	169	169	4.478e-06	
40	14	14	6.010e-07		560	175	175	4.669e-06	
41	14	14	5.910e-07		580	182	182	8.697e-06	
42	15	15	6.220e-07		600	186	186	4.939e-06	
43	15	15	4.970e-07		620	192	192	5.210e-06	
44	15	15	6.110e-07		640	200	200	4.602e-06	
45	16	16	6.120e-07		660	204	204	5.420e-06	
46	16	16	6.520e-07		680	210	210	5.601e-06	
47	16	16	4.230e-07		700	218	218	5.780e-06	
48	16	16	6.410e-07		720	222	222	5.080e-06	
					740	231	231	5.618e-06	
					760	235	235	6.162e-06	
					780	239	239	6.266e-06	
					800	248	248	6.613e-06	



PC2

1	1	1	3.214e-06	50	17	17	9.000e-07
2	2	2	2.142e-06	55	19	19	2.282e-06
3	3	3	1.733e-06	60	21	21	1.267e-06
4	3	3	2.472e-06	65	23	23	1.136e-06
5	4	4	2.042e-06	70	24	24	1.059e-06
6	4	4	2.253e-06	75	26	26	1.108e-06
7	5	5	1.775e-06	80	27	27	1.230e-06
8	5	5	2.243e-06	85	28	28	1.176e-06
9	5	5	2.549e-06	90	29	29	1.310e-06
10	6	6	2.520e-06	95	31	31	1.523e-06
11	6	6	1.516e-06	100	33	33	1.605e-06
12	6	6	9.220e-07	110	38	38	2.821e-06
13	7	7	8.880e-07	120	41	41	2.423e-06
14	7	7	8.420e-07	130	43	43	2.800e-06
15	7	7	8.480e-07	140	45	45	3.511e-06
16	7	7	8.720e-07	150	47	47	3.454e-06
17	8	8	9.250e-07	160	51	51	3.849e-06
18	8	8	8.420e-07	170	56	56	4.255e-06
19	8	8	8.470e-07	180	59	59	3.979e-06
20	8	8	8.740e-07	190	61	61	3.715e-06
21	9	9	1.639e-06	200	63	63	3.762e-06
22	9	9	1.118e-06	220	70	70	6.261e-06
23	9	9	9.330e-07	240	77	77	5.121e-06
24	9	9	7.580e-07	260	81	81	4.821e-06
25	9	9	1.127e-06	280	88	88	5.147e-06
26	10	10	1.064e-06	300	95	95	5.229e-06
27	10	10	9.560e-07	320	99	99	5.291e-06
28	10	10	9.160e-07	340	108	108	5.819e-06
29	10	10	9.830e-07	360	113	113	6.299e-06
30	10	10	1.002e-06	380	118	118	6.353e-06
31	11	11	1.083e-06	400	126	126	6.433e-06
32	11	11	8.900e-07	420	132	132	8.443e-06
33	11	11	1.070e-06	440	137	137	7.584e-06
34	12	12	1.146e-06	460	145	145	7.438e-06
35	12	12	1.119e-06	480	150	150	7.692e-06
36	12	12	5.753e-06	500	157	157	8.510e-06
37	13	13	1.301e-06	520	164	164	7.980e-06
38	13	13	1.198e-06	540	168	168	8.309e-06
39	13	13	1.477e-06	560	176	176	8.651e-06
40	14	14	1.143e-06	580	183	183	8.744e-06
41	14	14	1.589e-06	600	187	187	1.434e-05
42	14	14	1.330e-06	620	193	193	9.651e-06
43	15	15	1.345e-06	640	201	201	9.356e-06
44	15	15	1.305e-06	660	206	206	8.821e-06
45	15	15	1.334e-06	680	212	212	8.996e-06
46	16	16	1.895e-07	700	221	221	9.305e-06
47	16	16	9.280e-07	720	226	226	9.431e-06
48	16	16	8.890e-07	740	235	235	9.714e-06
49	16	16	8.670e-07	760	240	240	9.987e-06
				780	244	244	1.041e-05
				800	253	253	1.059e-05

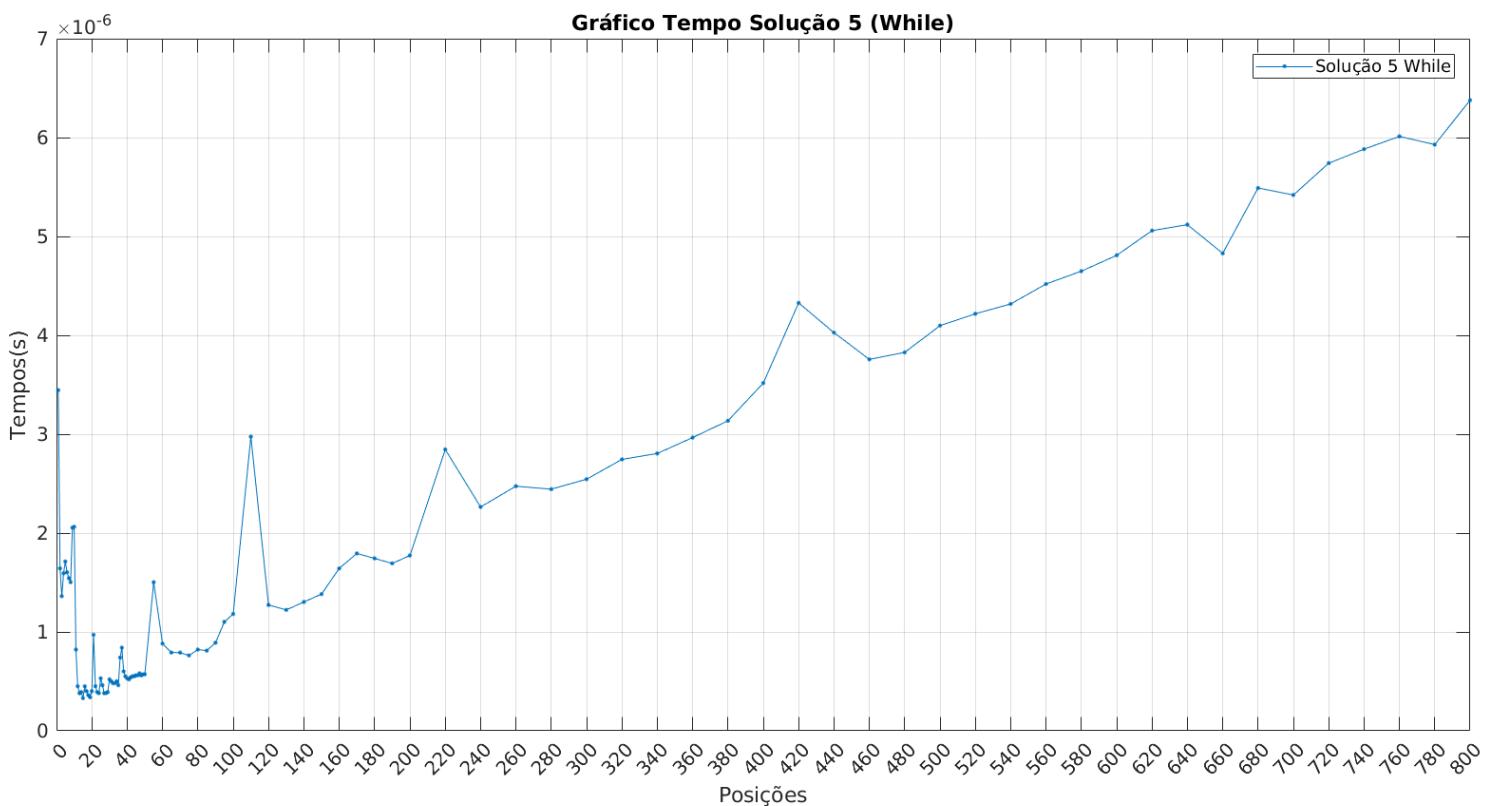


PC3

goncalo@goncalomf:~/Desktop/Projeto_				
1	1	1	5.191e-06	17 0.000e+00
2	2	2	1.020e-06	19 2.361e-06
3	3	3	7.900e-07	22 1.380e-06
4	3	3	9.710e-07	24 0.000e+00
5	4	4	7.400e-07	26 1.120e-06
6	4	4	9.600e-07	27 1.270e-07
7	5	5	8.800e-07	28 1.250e-06
8	5	5	8.400e-07	29 0.000e+00
9	5	5	1.191e-06	30 1.300e-06
10	6	6	1.301e-06	32 2.340e-07
11	6	6	1.691e-06	35 6.655e-06
12	6	6	8.300e-07	39 2.801e-06
13	7	7	0.000e+00	42 1.810e-06
14	7	7	0.000e+00	44 1.800e-06
15	7	7	4.460e-07	46 1.941e-06
16	7	7	0.000e+00	48 0.000e+00
17	8	8	7.600e-07	52 2.630e-07
18	8	8	0.000e+00	57 1.740e-07
19	8	8	4.590e-07	60 2.661e-06
20	8	8	0.000e+00	62 0.000e+00
21	9	9	2.371e-06	64 2.600e-06
22	9	9	1.001e-06	70 3.141e-06
23	9	9	0.000e+00	78 2.250e-06
24	9	9	0.000e+00	82 0.000e+00
25	9	9	0.000e+00	89 0.000e+00
26	10	10	0.000e+00	97 0.000e+00
27	10	10	5.720e-07	102 2.329e-06
28	10	10	0.000e+00	111 1.200e-08
29	11	11	0.000e+00	115 2.870e-06
30	11	11	0.000e+00	120 3.820e-07
31	11	11	0.000e+00	129 3.210e-06
32	12	12	9.000e-07	135 4.251e-06
33	12	12	0.000e+00	140 3.720e-06
34	12	12	0.000e+00	148 3.770e-06
35	13	13	0.000e+00	154 1.321e-06
36	13	13	0.000e+00	160 4.130e-06
37	13	13	0.000e+00	168 8.100e-07
38	14	14	0.000e+00	172 6.610e-07
39	14	14	0.000e+00	179 4.461e-06
40	14	14	0.000e+00	186 4.078e-06
41	15	15	3.470e-07	190 2.824e-06
42	15	15	0.000e+00	197 7.441e-06
43	15	15	0.000e+00	205 1.850e-06
44	16	16	0.000e+00	209 1.859e-06
45	16	16	0.000e+00	216 3.595e-06
46	16	16	0.000e+00	224 1.835e-06
47	16	16	0.000e+00	229 3.988e-06
48	17	17	0.000e+00	238 2.717e-06
49	17	17	0.000e+00	243 3.983e-06
50	17	17	0.000e+00	247 3.641e-06
				256 4.347e-06



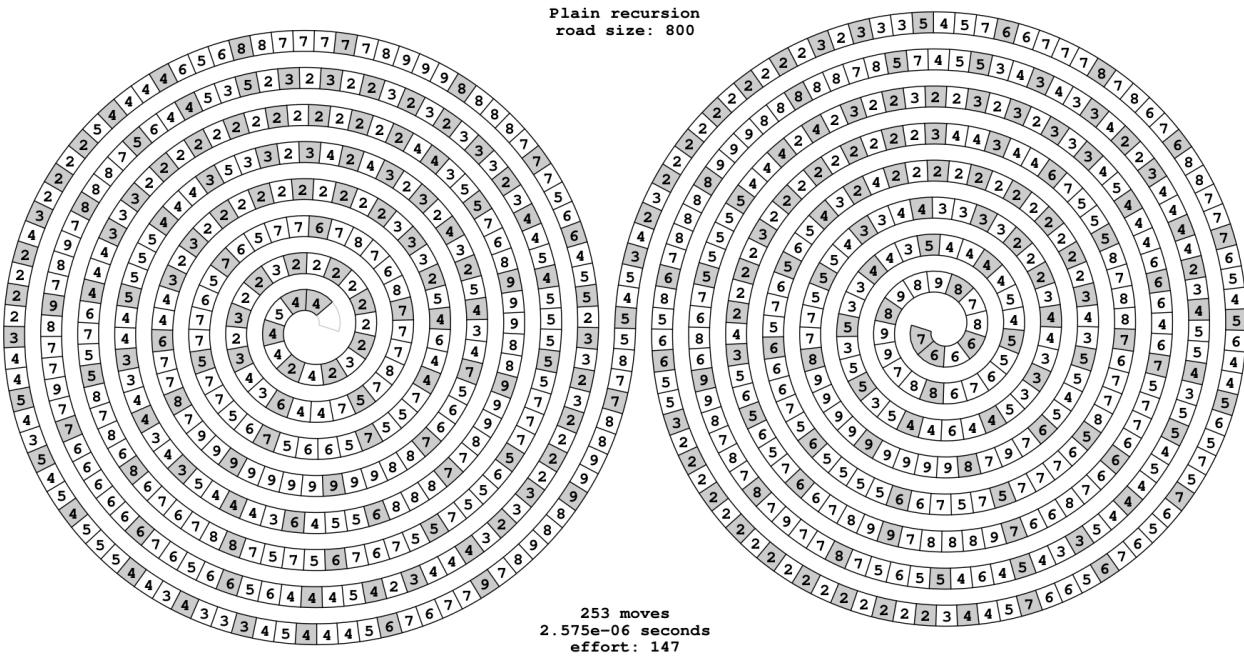
Gráfico Matlab Solve 5



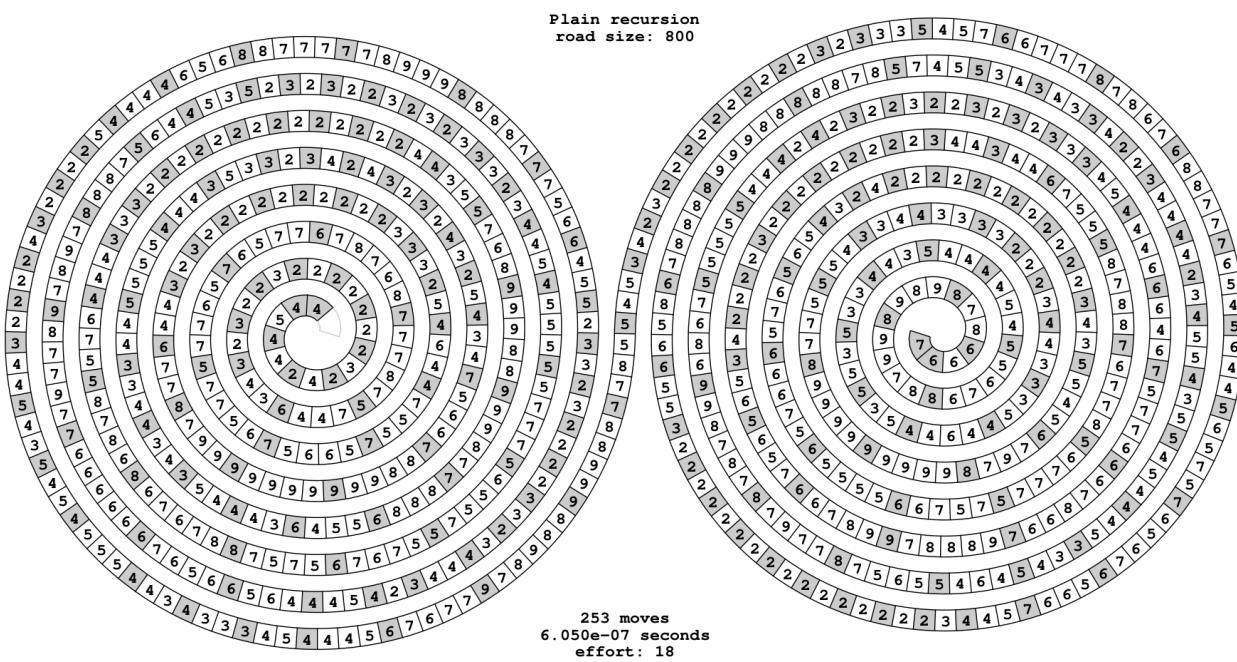


Caracol Exemplo (PC2)

Solve 2 melhorada

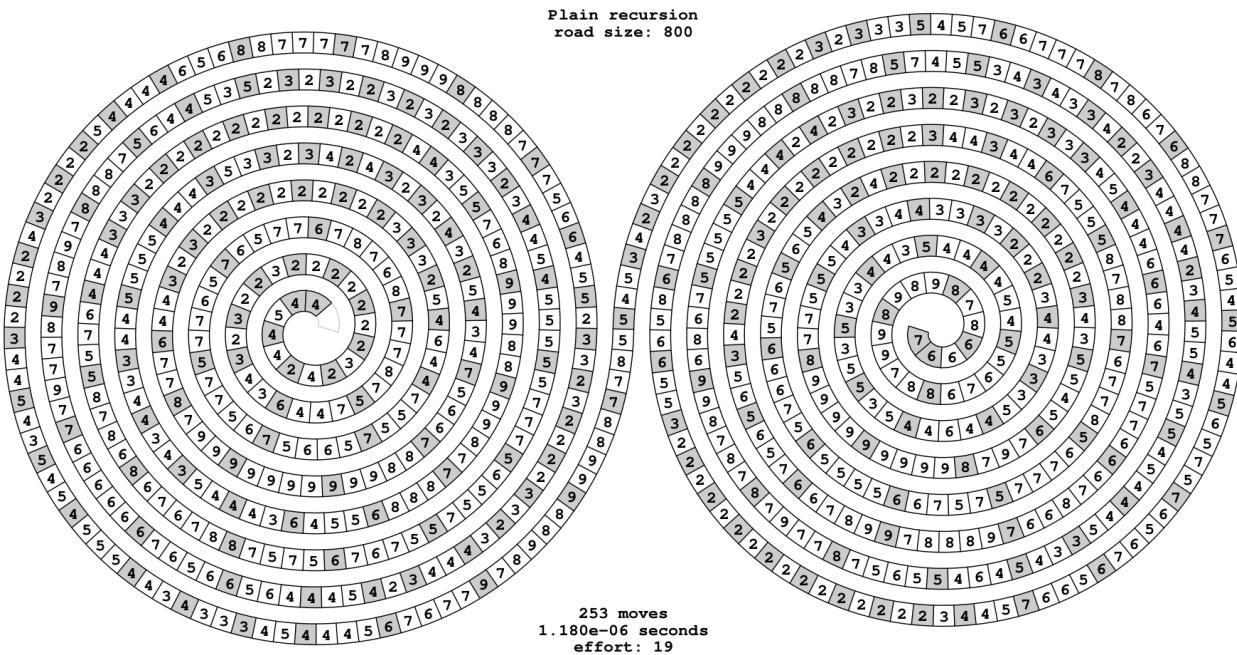


Solve 3 while dynamic





Solve 4 dynamic



Solve 5 while

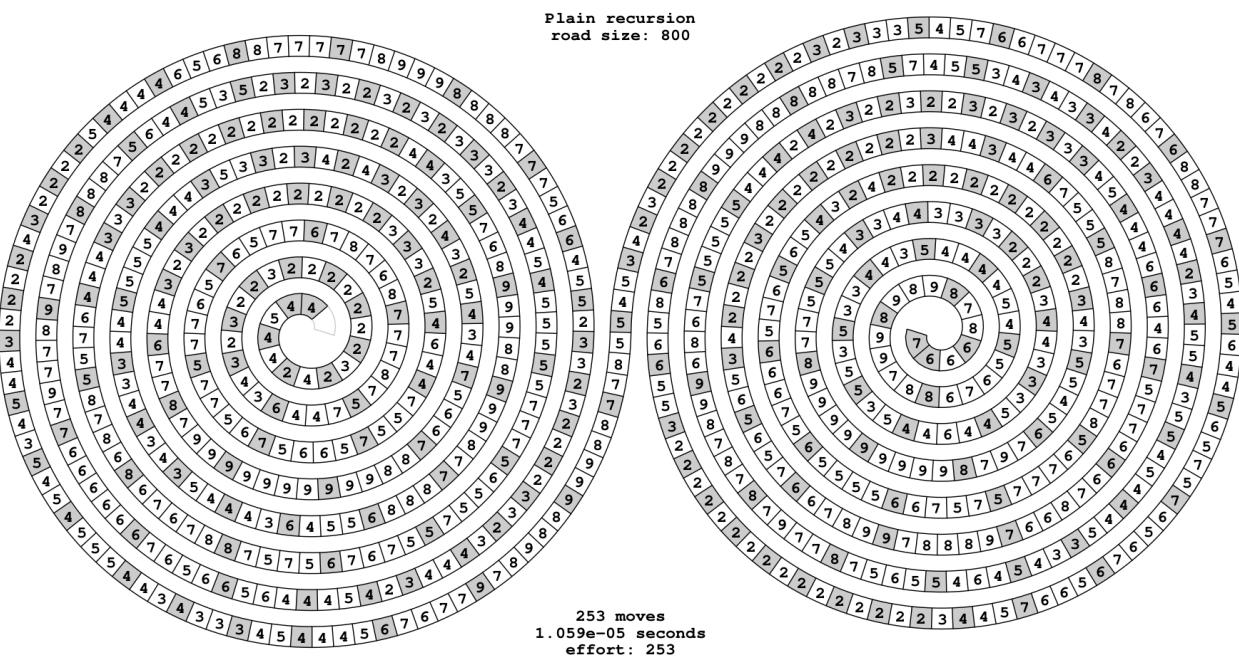




Gráfico com todas as soluções:

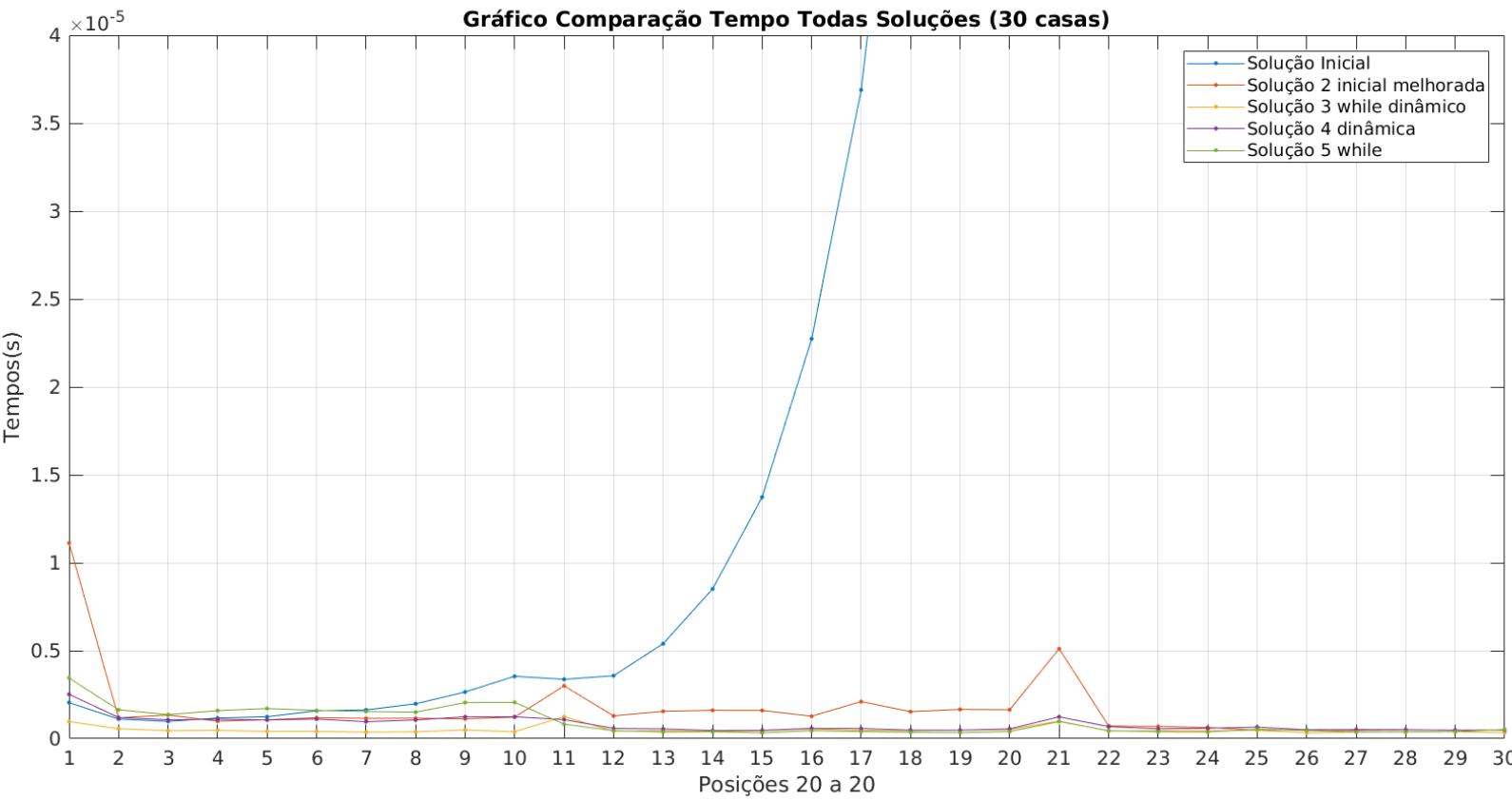
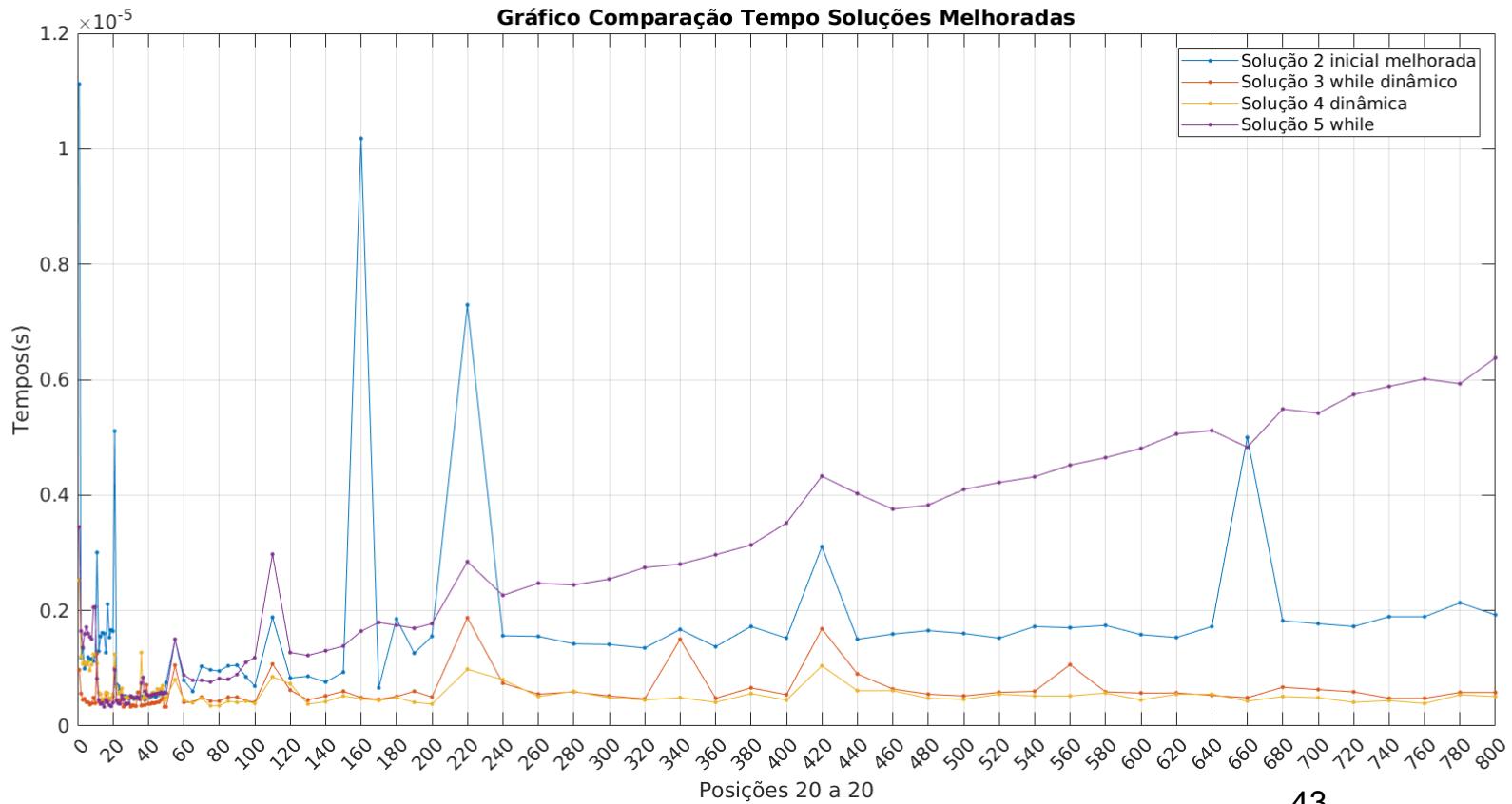


Gráfico apenas com as soluções melhoradas:





Apêndice

Código C

Solve 2 inicial melhorada

```
//Solve 2 (1 melhorada)
static int solution_2_recursion(int move_number,int position,int
speed,int final_position)
{
    int i,new_speed;

    // record move
    solution_2_count++;
    solution_2.positions[move_number] = position;
    // is it a solution?
    if(position == final_position && speed == 1)
    {
        // is it a better solution?
        if(move_number < solution_2_best.n_moves)
        {
            solution_2_best = solution_2;
            solution_2_best.n_moves = move_number;
            solution_2_last = solution_2_best;
        }
        return 1;
    }
    if(solution_2_best.positions[move_number] >
solution_2.positions[move_number]){
        return 0;
    }
    // no, try all legal speeds
    for(new_speed = speed + 1;new_speed >= speed - 1;new_speed--) {
        if(new_speed >= 1 && new_speed <= _max_road_speed_ && position +
new_speed <= final_position)
        {
            for(i = 0;i <= new_speed && new_speed <= max_road_speed[position +
i];i++)
            ;
        }
    }
}
```



```

if(i > new_speed) {
    int verify = solution_2_recursion(move_number + 1, position +
new_speed, new_speed, final_position);
    if (new_speed - speed == 1) {
        solution_2_last.ultimo_incremeno[0] = move_number + 1;
        solution_2_last.ultimo_incremeno[1] = position + new_speed;
        solution_2_last.ultimo_incremeno[2] = new_speed;
    }
    if (verify == 1)
    {
        return 1;
    }
}
}
}
}
return 0;
}

static void solve_2(int final_position)
{
    if(final_position < 1 || final_position > _max_road_size_)
    {
        fprintf(stderr,"solve_1: bad final_position\n");
        exit(1);
    }
    solution_2_elapsed_time = cpu_time();
    solution_2_count = 0ul;
    solution_2_best.n_moves = final_position + 100;

    solution_2_recursion(solution_2_last.ultimo_incremeno[0],solution_2_la-
st.ultimo_incremeno[1],solution_2_last.ultimo_incremeno[2],final_posi-
tion);
    solution_2_elapsed_time = cpu_time() - solution_2_elapsed_time;
}

```

Solve 3 while dynamic

```

//Solve 3 (while)
static void solution_3_while_dynamic(int move_number,int position,int
speed,int final_position)
{

```



```

int new_speed = solution_3.ultimo_incremneto[2];
int new_speed_test;
int position_test;
int variavel_teste;
int position_test_anterior;
int teste_max_speed;
int moves = solution_3.ultimo_incremneto[0];
position = solution_3.ultimo_incremneto[1];
speed = solution_3.ultimo_incremneto[2];

while(position<final_position){ // Executa enquanto a posição não for
a final_position

    solution_3_count++; // Conta o número de iterações
    speed = new_speed; // Atribui a speed o valor de new_speed
    variavel_teste = 1; // Variável que controla a entrada nos if's
    //seguintes
    position_test = position; // Atribui a position_test o valor de
    //position

    teste_max_speed = 1; // Variável que controla se a speed ultrapassa
    //o max_road_speed

    if(final_position==1){
        moves = 1;
        solution_3.positions[moves] = 1;
        solution_3_best.positions[moves] = 1;
        solution_3.ultimo_incremneto[0] = moves;
        solution_3.ultimo_incremneto[1] = 1;
        solution_3.ultimo_incremneto[2] = 1;

        break;
    }
    else{
        if(variavel_teste == 1 && speed<_max_road_speed_){ // Se speed for
        //menor que o max_road_speed
            position_test = position;
            for(new_speed_test = speed +1;new_speed_test >= 1;
            new_speed_test--){ //Loop desde speed+1 até 1
                if(new_speed_test>teste_max_speed){
                    teste_max_speed = new_speed_test;
                }
                if(new_speed_test>speed){
                    speed = new_speed_test;
                }
                if(new_speed_test>position){
                    position = new_speed_test;
                }
                if(new_speed_test>final_position){
                    final_position = new_speed_test;
                }
            }
        }
    }
}

```



```

        position_test_anterior = position_test; //Atribui a
//position_test_anterior o valor de position_test
        position_test = position_test + new_speed_test; //Atribui a
//position_test o valor de position_test + new_speed_test
        for(int i = position_test_anterior; i <= position_test; i++){
//Testa max_speed desde posição inicial+1 ate posição depois do
incremento de velocidade
            if(max_road_speed[i]<new_speed_test){ //Se max_speed for menor
//que new_speed
                teste_max_speed = 0; //Pode dar break;
                break;
            }
        }
        if(teste_max_speed == 0){break;} //Se max_speed for menor que
//new_speed pode dar break no loop
    }
    if(position_test<=final_position && teste_max_speed == 1){ //Se a
//posição for menor que a final_position e se max_speed for maior que
//new_speed
        new_speed = speed +1; //Atribui a new_speed o valor de speed +1
        moves++; //Incrementa o número de movimentos
        variavel_teste = 0; //Pode começar uma nova iteração no while
        solution_3.ultimo_increm[0] = moves; //Atribui a
//solution_3.ultimo_increm[0] o valor de moves
        solution_3.ultimo_increm[1] = position + new_speed;
//Atribui a solution_3.ultimo_increm[1] o valor de position +
//new_speed
        solution_3.ultimo_increm[2] = new_speed; //Atribui a
//solution_3.ultimo_increm[2] o valor de new_speed
    }
}

teste_max_speed = 1; //Variável que controla se a speed ultrapassa o
//max_road_speed

if(variavel_teste==1 && speed<=max_road_speed){
    position_test = position;
    for(new_speed_test = speed;new_speed_test >= 1; new_speed_test--) {
//Loop desde speed até 1 (Manter speed)
        position_test_anterior = position_test;
        position_test = position_test + new_speed_test;
        for(int i = position_test_anterior; i <= position_test; i++){
//Testa max_speed no intervalo das posições
}
}
}

```



```

        if(max_road_speed[i]<new_speed_test) {
            teste_max_speed = 0; //Pode dar break;
            break;
        }
    }
    if(teste_max_speed == 0){break;}
}
if (teste_max_speed == 0) //Caso não possa manter o speed a única
//opção é decrementar o speed
{
    new_speed = speed - 1;
    moves++;
    variavel_teste = 0;
}
else if(position_test<=final_position && teste_max_speed == 1){
    new_speed = speed;
    moves++;
    variavel_teste = 0;
}
}

if(variavel_teste==1){
    position_test = position;
    for(new_speed_test = speed-1;new_speed_test >= 1;
new_speed_test--) { //Desce speed
        position_test = position_test + new_speed_test;
    }
    new_speed = speed -1;
    moves++;
    variavel_teste = 0;
}

if(new_speed == 0){
    position = position + speed;
}else{
    position = position + new_speed;
    solution_3.positions[moves] = position;
    solution_3_best.positions[moves] = position;
}
}
}
}

```



```

solution_3_best.n_moves = moves ;
}

static void solve_3(int final_position)
{
    if(final_position < 1 || final_position > _max_road_size_)
    {
        fprintf(stderr,"solve_2: bad final_position\n");
        exit(1);
    }
    solution_3_elapsed_time = cpu_time();
    solution_3_count = 0ul;
    solution_3_best.n_moves = final_position + 100;

    solution_3_while_dynamic(solution_3.ultimo_incremeno[0],solution_3.ultimo_incremeno[1],solution_3.ultimo_incremeno[2],final_position); //Move number,position,speed
    solution_3_elapsed_time = cpu_time() - solution_3_elapsed_time;
}

```

Solve 4 dynamic

```

typedef struct
{
    int n_moves;                                // the number of moves (the
//number of positions is one more than the number of moves)
    int positions[1 + _max_road_size_]; // the positions (the first one
//must be zero)
    int ultimo_incremeno[3];
}
solution_t;

//Solve 4 (dynamic programing)

//Utilizando array externo e ponteiros
// int useful[3];
// void ultimo_incremeno_Update(int *useful,int move,int position, int
//speed){
//    useful[0] = move;
//    useful[1] = position;

```



```

//    useful[2] = speed;
// }

static void solution_4_dynamic(int move_number,int position,int
speed,int final_position)
{
    int position_teste,position_anterior; //Variáveis que irão receber
valores de posições para testar se a speed é válida
    int acabou = 0; //Variável de controlo
    int speed_valido=1; //Variável de controlo para saber se a speed é
válida
    solution_4_count++;
    position_teste = position;
    solution_4.positions[move_number] = position;
    solution_4_best.positions[move_number] = position;
    if(position==final_position){
        solution_4_best.n_moves = move_number;
        return;
    }
    else{
        for(int new_speed = speed + 1; new_speed >= speed - 1; new_speed--) {
//3 possibilidades de speed, incrementa, mantém ou decrementa
            speed_valido = 1; //Variável de controlo para saber se a speed é
//válida
            if(new_speed<=_max_road_speed_){ //Testa se a speed é válida
                position_teste = position; //Reinicia a posição de teste
                for(int new_speed_teste = new_speed;new_speed_teste >= 1;
new_speed_teste--){ //Loop desde new_speed até 1
                    position_anterior = position_teste; //Guarda a posição
//anterior
                    position_teste = position_teste + new_speed_teste;
//Incrementa a posição de teste

                    for(int i = position_anterior;i<=position_teste;i++){ //Testa
//max speed
                        if(max_road_speed[i]<new_speed_teste){
                            speed_valido = 0;
                            break;
                        }
                    }
                    if(position_teste>final_position || acabou == 1 ||

speed_valido == 0){break;}
                }
            }
        }
    }
}

```



```

        if(new_speed_teste == 1 && position_teste<=final_position &&
speed_valido == 1){ //Speed válida entra na função com novo speed e
//nova posição
    acabou = 1;
    if(new_speed - speed == 1){ // Deu incremento no speed
        // ultimo_incremente_Update(&useful,move_number +
//1,position + new_speed, new_speed); //Com Array externo
        solution_4.ultimo_incremente[0] = move_number + 1;
        solution_4.ultimo_incremente[1] = position + new_speed;
        solution_4.ultimo_incremente[2] = new_speed;
    }
    solution_4_dynamic(move_number + 1,position + new_speed,
new_speed,final_position);
}
}
}
}
}

static void solve_4(int final_position)
{
if(final_position < 1 || final_position > _max_road_size_)
{
    fprintf(stderr,"solve_4: bad final_position\n");
    exit(1);
}
solution_4_elapsed_time = cpu_time();
solution_4_count = 0ul;
solution_4_best.n_moves = final_position + 100;
//solution_4_dynamic(useful[0],useful[1],useful[2],final_position);
//Com Array externo

solution_4_dynamic(solution_4.ultimo_incremente[0],solution_4.ultimo_in
cremente[1],solution_4.ultimo_incremente[2],final_position);
solution_4_elapsed_time = cpu_time() - solution_4_elapsed_time;
}
}

```



Solve 5 while

```
//Solve 5 (while less efficient)
static int descida(int nr) {
    int sum=0;
    for (int i = nr; i >=1; i--)
    {
        sum = sum + i;
    }
    return sum;
}

static void solve_5_while(int final_position){

    int move_number = 0;
    int position = 0;
    int speed = 0;

    solution_5.positions[move_number] = position;
    while (position <= final_position)
    {
        int valAumenta = 1;
        int valDiminui = 0;

        if (position == final_position && speed == 1)
        {
            solution_5_best = solution_5;
            solution_5_best.n_moves = move_number;
            break;
        }

        if (descida(speed+1) <= final_position-position && speed+1<10)
        {
            //-----max speed-----
            int manter_speed = speed;
            int manter_posi = position;
```



```
int new_speed = speed+1;
int new_position = position;

for (int i = position; i <= position + descida(speed+1); i++)
{
    if (max_road_speed[i]<new_speed)
    {
        valAumenta = 0;
    }
    if (max_road_speed[i]<manter_speed && i<=position +
descida(speed))
    {
        valDiminui = 1;
        valAumenta = 0;
        break;
    }

    if ( i == new_position+new_speed)
    {
        new_speed--;
        new_position = new_position + new_speed+1;
    }
    if ( i == manter_posi+manter_speed && i<=position +
descida(speed))
    {

        manter_posi = manter_posi + manter_speed;
        manter_speed--;
    }
}

if (valAumenta == 1)
{
    speed++;
}

else if (valDiminui == 1)
{
    speed--;
}
else
{
    speed=speed;
}
```



```
}

//-----max speed-----


}

else
{
    if ((final_position - position) >= descida(speed))
    {

        //-----max speed-----


        int new_speed = speed;
        int new_position = position;
        for (int i = position; i <= position + descida(speed); i++)
        {

            if (max_road_speed[i]<new_speed)
            {
                valDiminui = 1;
                break;
            }

            if ( i == new_position+new_speed)
            {

                new_position = new_position + new_speed;
                new_speed--;
            }
        }

        if (valDiminui == 1)
        {
            speed--;
        }
        else
        {
            speed=speed;
        }
    }
}
```



```
//-----max speed-----  
  
}  
else  
{  
    speed--;  
}  
}  
  
//printf("%d", speed);  
position = position + speed;  
move_number++;  
solution_5.positions[move_number] = position;  
solution_5_count++;  
}  
  
//printf("\nmove nr:%d", move_number);  
}  
  
  
static void solve_5(int final_position)  
{  
    if(final_position < 1 || final_position > _max_road_size_)  
    {  
        fprintf(stderr,"solve_1: bad final_position\n");  
        exit(1);  
    }  
    solution_5_elapsed_time = cpu_time();  
    solution_5_count = 0ul;  
    solution_5_best.n_moves = final_position + 100;  
    //solution_1_recursion(0,0,0,final_position);  
    solve_5_while(final_position);  
    solution_5_elapsed_time = cpu_time() - solution_5_elapsed_time;  
}
```



Main

```

//  

// main program  

//  
  

int main(int argc,char *argv[argc + 1])  

{  

#define _time_limit_ 3600.0  

    int n_mec,final_position,print_this_one;  

    char file_name[64];  
  

    // generate the example data  

    if(argc == 2 && argv[1][0] == '-' && argv[1][1] == 'e' && argv[1][2] == 'x')  

    {  

        example();  

        return 0;  

    }  

    // initialization  

    n_mec = (argc < 2) ? 0xAED2022 : atoi(argv[1]);  

    srand((unsigned int)n_mec);  

    init_road_speeds();  

    // run all solution methods for all interesting sizes of the problem  

    final_position = 1;  

    solution_1_elapsed_time = 0.0;  

    printf("      + ----- +\n");  

    printf("      |           plain recursion |\n");  

    printf("---- + ----- +----- +\n");  

    printf("  n | sol       count   cpu time |\n");  

    printf("---- +----- +----- +\n");  

    while(final_position <= _max_road_size_/* && final_position <= 20*/)  

    {  

        print_this_one = (final_position == 10 || final_position == 20 ||  

final_position == 50 || final_position == 100 || final_position == 200  

|| final_position == 400 || final_position == 800) ? 1 : 0;  

        printf("%3d ",final_position);  

        // first solution method (very bad)  

        // if(solution_1_elapsed_time < _time_limit_)  

        // {  

        //     solve_1(final_position);  

        //     if(print_this_one != 0)  

        //     {  


```



```

//      sprintf(file_name,"%03d_1.pdf",final_position);
//
make_custom_pdf_file(file_name,final_position,&max_road_speed[0],solution_1_best.n_moves,&solution_1_best.positions[0],solution_1_elapsed_time,solution_1_count,"Plain recursion");
    //
    // printf(" %3d %16lu
%9.3e",solution_1_best.n_moves,solution_1_count,solution_1_elapsed_time);
//
// }
// else
// {
//     solution_1_best.n_moves = -1;
//     //printf("                                | ");
// }

// second solution method (less bad)
// if(solution_2_elapsed_time < _time_limit_)
// {
//     solve_2(final_position);
//     if(print_this_one != 0)
//     {
//         sprintf(file_name,"%03d_2.pdf",final_position);
//     }

make_custom_pdf_file(file_name,final_position,&max_road_speed[0],solution_2_best.n_moves,&solution_2_best.positions[0],solution_2_elapsed_time,solution_2_count,"Plain recursion");
    //
    // printf(" %3d %16lu %9.3e
",solution_2_best.n_moves,solution_2_count,solution_2_elapsed_time);
    //
// }
// else
// {
//     solution_2_best.n_moves = -1;
//     // printf("                                | ");
// }

//third solution method (less bad without recursive)
// if(solution_3_elapsed_time < _time_limit_)
// {
//     solve_3(final_position);

```



```

//      if(print_this_one != 0)
//      {
//          sprintf(file_name,"%03d_3.pdf",final_position);
//      }

make_custom_pdf_file(file_name,final_position,&max_road_speed[0],solution_3_best.n_moves,&solution_3_best.positions[0],solution_3_elapsed_time,solution_3_count,"Plain recursion");
//      }
//      printf(" %3d %16lu %9.3e
",solution_3_best.n_moves,solution_3_count,solution_3_elapsed_time);
//  }

// else
// {
//     solution_3_best.n_moves = -1;
//     // printf("                                | ");
// }

//fourth solution method (good with dynamic)
// if(solution_4_elapsed_time < _time_limit_)
// {
//     solve_4(final_position);
//     if(print_this_one != 0)
//     {
//         sprintf(file_name,"%03d_4.pdf",final_position);
//     }

make_custom_pdf_file(file_name,final_position,&max_road_speed[0],solution_4_best.n_moves,&solution_4_best.positions[0],solution_4_elapsed_time,solution_4_count,"Plain recursion");
//  }
//  printf(" %3d %16lu %9.3e
",solution_4_best.n_moves,solution_4_count,solution_4_elapsed_time);
//  }

// else
// {
//     solution_4_best.n_moves = -1;
//     // printf("                                | ");
// }

//fifth solution method (while loop slow)
// if(solution_5_elapsed_time < _time_limit_)
// {
//     solve_5(final_position);
// }

```



```
//     if(print_this_one != 0)
//     {
//         sprintf(file_name,"%03d_5.pdf",final_position);
//     }
make_custom_pdf_file(file_name,final_position,&max_road_speed[0],solution_5_best.n_moves,&solution_5_best.positions[0],solution_5_elapsed_time,solution_5_count,"Plain recursion");
//     }
//     printf(" %3d %16lu %9.3e
",solution_5_best.n_moves,solution_5_count,solution_5_elapsed_time);
// }
// else
// {
//     solution_5_best.n_moves = -1;
//     // printf("                                | ");
// }

// done

printf("\n");
fflush(stdout);
// new final_position
if(final_position < 50)
    final_position += 1;
else if(final_position < 100)
    final_position += 5;
else if(final_position < 200)
    final_position += 10;
else
    final_position += 20;
}
printf("--- + --- ----- +----- +\n");
return 0;
# undef _time_limit_
}
```



Código Matlab

Gráfico com todas as soluções:

```
Dados_solve_1 = load("ficheiro_solve_1.txt");
Tempos_solve_1 = Dados_solve_1(1:20,4);
Posicoes_solve_1 = Dados_solve_1(1:20,1);

Dados_solve_2 = load("ficheiro_solve_2.txt");
Tempos_solve_2 = Dados_solve_2(1:30,4);
Posicoes_solve_2 = Dados_solve_2(1:30,1);

Dados_solve_3 = load("ficheiro_solve_3.txt");
Tempos_solve_3 = Dados_solve_3(1:30,4);
Posicoes_solve_3 = Dados_solve_3(1:30,1);

Dados_solve_4 = load("ficheiro_solve_4.txt");
Tempos_solve_4 = Dados_solve_4(1:30,4);
Posicoes_solve_4 = Dados_solve_4(1:30,1);

Dados_solve_5 = load("ficheiro_solve_5.txt");
Tempos_solve_5 = Dados_solve_5(1:30,4);
Posicoes_solve_5 = Dados_solve_5(1:30,1);

plot(Posicoes_solve_1,Tempos_solve_1,'.-')
hold on
plot(Posicoes_solve_2,Tempos_solve_2,'.-')
plot(Posicoes_solve_3,Tempos_solve_3,'.-')
plot(Posicoes_solve_4,Tempos_solve_4,'.-')
plot(Posicoes_solve_5,Tempos_solve_5,'.-')
hold off
legend('Solução Inicial','Solução 2 inicial melhorada','Solução 3 while dinâmico','Solução 4 dinâmica','Solução 5 while')
xlabel('Posições 20 a 20')
ylabel('Tempos(s)')
xticks(0:1:30)
ylim([0 4e-05])
xlim([1 30])
grid on
title('Gráfico Comparação Tempo Todas Soluções (30 casas)')
```

Gráfico com as soluções melhoradas:

```
Dados_solve_2 = load("ficheiro_solve_2.txt");
Tempos_solve_2 = Dados_solve_2(:,4);
Posicoes_solve_2 = Dados_solve_2(:,1);

Dados_solve_3 = load("ficheiro_solve_3.txt");
Tempos_solve_3 = Dados_solve_3(:,4);
Posicoes_solve_3 = Dados_solve_3(:,1);

Dados_solve_4 = load("ficheiro_solve_4.txt");
Tempos_solve_4 = Dados_solve_4(:,4);
Posicoes_solve_4 = Dados_solve_4(:,1);

Dados_solve_5 = load("ficheiro_solve_5.txt");
Tempos_solve_5 = Dados_solve_5(:,4);
Posicoes_solve_5 = Dados_solve_5(:,1);

plot(Posicoes_solve_2,Tempos_solve_2,'.-')
hold on
plot(Posicoes_solve_3,Tempos_solve_3,'.-')
plot(Posicoes_solve_4,Tempos_solve_4,'.-')
plot(Posicoes_solve_5,Tempos_solve_5,'.-')
hold off
legend('Solução 2 inicial melhorada','Solução 3 while dinâmico','Solução 4 dinâmica','Solução 5 while')
xlabel('Posições 20 a 20')
ylabel('Tempos(s)')
xticks(0:20:800)
grid on
title('Gráfico Comparação Tempo Soluções Melhoradas')
```



Gráfico com a solução inicial:

```
Dados_solve_1 = load("ficheiro_solve_1.txt");
Tempos_solve_1 = Dados_solve_1(:,4);
Posicoes_solve_1 = Dados_solve_1(:,1);

plot(Posicoes_solve_1, Tempos_solve_1, '.-')

legend('Solução Inicial')
xlabel('Posições')
ylabel('Tempos(s)')
grid on
title('Gráfico Tempo Solução Inicial')
xlim([1 40])
```

Gráfico com a solução inicial melhorada (2):

```
Dados_solve_2=load("ficheiro_solve2.txt");
Tempos_2 = Dados_solve_2(:,4);
Posicoes_2 = Dados_solve_2(:,1);

plot(Posicoes_2, Tempos_2, '.-')

xlabel('Posições(20 a 20)')
ylabel('Tempo(s)')
xticks(0:20:800)
grid on
legend('Solução 2 Inicial Melhorada')
title('Grafico solucao 2 (inicial melhorada)')
```



Gráfico com a solução 3 while:

```
Dados_solve_3 = load("ficheiro_solve_3.txt");
Tempos_solve_3 = Dados_solve_3(:,4);
Posicoes_solve_3 = Dados_solve_3(:,1);

plot(Posicoes_solve_3, Tempos_solve_3, '.-')

legend('Solução 3 While Dynamic')
xlabel('Posições')
ylabel('Tempos(s)')
grid on
title('Gráfico Tempo | Solução 3 (While Dynamic)')
xticks(0:20:800)
```

Gráfico com a solução 4 dynamic programming:

```
Dados_solve_4 = load("ficheiro_solve_4.txt");
Tempos_solve_4 = Dados_solve_4(:,4);
Posicoes_solve_4 = Dados_solve_4(:,1);

plot(Posicoes_solve_4, Tempos_solve_4, '.-')

legend('Solução 4 Dynamic')
xlabel('Posições')
ylabel('Tempos(s)')
grid on
title('Gráfico Tempo Solução 4 (Dynamic)')
xticks(0:20:800)
```



Gráfico com a solução 5 while:

```
Dados_solve_5 = load("ficheiro_solve_5.txt");
Tempos_solve_5 = Dados_solve_5(:,4);
Posicoes_solve_5 = Dados_solve_5(:,1);

plot(Posicoes_solve_5, Tempos_solve_5, '.-')
legend('Solução 5 While')
xlabel('Posições')
ylabel('Tempos (s)')
grid on
title('Gráfico Tempo Solução 5 (While)')
xticks(0:20:800)
```

Gráfico de entradas nas funções recursivas:



```
Dados_solve_Inicial = load("ficheiro_solve_1.txt");
Dados_solve_Inicial_Melhorada = load("ficheiro_solve_2.txt");
Dados_solve_Dinamica = load("ficheiro_solve_4.txt");

Effort_solve_inicial = Dados_solve_Inicial(1:10,3);
Posicoes_solve_inicial = Dados_solve_Inicial(1:10,1);

Effort_solve_inicial_melhorada = Dados_solve_Inicial_Melhorada(1:70,3);
Posicoes_solve_inicial_melhorada = Dados_solve_Inicial_Melhorada(1:70,1);

Effort_solve_dinamica = Dados_solve_Dinamica(1:70,3);
Posicoes_solve_dinamica = Dados_solve_Dinamica(1:70,1);

plot(Posicoes_solve_inicial,Effort_solve_inicial,'.-')
hold on
plot(Posicoes_solve_inicial_melhorada,Effort_solve_inicial_melhorada,'.-')
plot(Posicoes_solve_dinamica,Effort_solve_dinamica,'.-')

xlabel('Posições(5 a 5)')
ylabel('Entradas na função')
xticks(0:5:200)
yticks(0:5:140)
ylim([1 140])
grid on
legend('Solução Inicial','Solução 2 Inicial Melhorada','Solução 4 dinamica')
title('Graficos Effort Recursivas')
```



Conclusão

Com a realização deste trabalho , percebemos a dificuldade de pesquisar, perceber e aplicar algoritmos e maneiras diferentes para resolver um problema específico, que neste caso resume-se a uma subdivisão de estradas.

Aprender como aplicar diferentes tipos de programação e analisar código enriqueceu também o nosso conhecimento na linguagem C.

Explicar o código ajudou-nos a melhorar as nossas capacidades de comunicação e trabalho de equipa.

Com isto, concluímos por final que , na nossa perspetiva, este trabalho foi realizado de maneira bastante célebre e acrescentamos bastantes conhecimentos aos nossos portefólios de conhecimento sobre algoritmia.