universidade de aveiro

Speed Run

Relatório
08/12/2022

Tiago Sousa Fonseca (107266) -
Tomás Sousa Fonseca (107245) -
Beatriz Ferreira (107214) -

Universidade de Aveiro
Algoritmos e Estruturas de Dados

universidade de aveiro

Índice

Introdução 3

Métodos Utilizados 4

Solução Dada Inicialmente.....…….....4

Métodos Desenvolvidos.…….....5

Solução Mais Eficiente.....…….....6

Resultados 7

Solução Dada Inicialmente.....…….....7

Solução Desenvolvida…….....10

Comentários 13

Apêndice 14

universidade de aveiro

Introdução

Uma rua é subdividida em segmentos com aproximadamente o mesmo comprimento. Cada segmento tem um limite de velocidade associado. A velocidade é medida pelo número de segmentos de estrada que um carro é capaz de avançar num único movimento (***move_number***). Em cada movimento (***move_number***), o carro pode:

1. **Travar**: reduzir a velocidade por 1 (**new_speed = *speed - 1***);
2. **Manter**: manter a velocidade (***new_speed = speed***);
3. **Acelerar**: aumentar a velocidade por 1 (***new_speed = speed + 1***).

No entanto, este só pode avançar para a próxima posição (***position***) se não exceder o limite de velocidade (***speed_limit***) de cada segmento por onde passa, sendo que na estrada o limite de velocidade máximo é 9 (***_max_road_speed = 9***) e o limite mínimo é 2 (***_min_road_speed = 2***).

O carro inicia a viagem no primeiro segmento da estrada, com velocidade 0 (***speed = 0***). Ao chegar ao último segmento, o carro deve possuir velocidade 1 (***speed = 1***), para que o mesmo consiga reduzir a sua velocidade para 0 (***speed = 0***) e consequentemente parar.

Com este problema, definimos o objetivo de determinar o número mínimo de movimentos (***move_number***) necessários para atingir a posição final (***final_position***).

Métodos Utilizados

Solução Dada Inicialmente (speed_run):

É feito o uso da estrutura ***solution_t***, que contem o número de movimentos (***n_moves***), e um **array** com as posições da estrada (***positions[1 + _max_road_size]***). A estrutura é usada para guardar a solução válida calculada (***solution_1***) ao longo da determinação da melhor solução (***solution_1_best***).

Nesta solução é chamada a função ***solve1(int final_position)***, tendo como argumento a última posição (***final_position***), ou seja, o tamanho da estrada. Por sua vez, esta função começa a registar o tempo para a determinação de uma solução (***solution_1_elapsed_time***), define o melhor número de movimentos (***solution_1_best.n_moves***) para um valor impossível e chama a função ***solution_1_recursion(int move_number, int position, int speed, int final_position)*** com os argumentos: ***move_number = 0, position = 0, speed = 0, final_position.***

Nesta função recursiva, é registado o esforço (***effort/count***) e a posição (***position***) de cada movimento (***move_number***), assim como também é verificado a existência de uma solução válida (***position == final_position && speed == 1***), e se for o caso, verifica se é uma melhor solução do que aquela que já se encontra guardada (***solution_1_bes***t), com base nos movimentos feitos (***move_number***). Caso não seja uma solução válida, chamámos até três vezes a função recursiva ***solution_1_recursion(int move_number, int position, int speed, int final_position***, sendo a primeira com uma velocidade decrementada (**travar**) (***new_speed = speed - 1***), a segunda mantendo a velocidade (**manter**) (***new_speed = speed***) e a terceira com uma velocidade incrementada (**acelerar**) (***new_speed = speed + 1***), sendo que para cada chamada incrementamos o movimento (***move_number + 1***) e a posição (***position + new_speed***). O chamamento da função só é feito se a nova velocidade (***new_speed***) não infringir os limites de velocidade de todos os segmentos envolvidos no próximo movimento.

Assim a função ***solution_1_recursion(int move_number, int position, int speed, int final_position)*** determina a melhor solução (guardando em ***solution_1_best***), sendo que no final da sua execução, a função ***solve1(int final_position)*** acaba de registar o tempo para a determinação de uma solução (***solution_1_elapsed_time***).

Métodos desenvolvidos:

- Consultar o *array* ***solution_1_best.positios[move_number]*** para ver se já existe uma melhor solução do que aquela que estamos a calcular (comparando se para o mesmo número de movimentos (***move_number***), temos uma maior posição guardada, do que aquela da posição da solução a ser calculada);

- Verificar se o número de movimentos (***move_number***) é maior do que o número de movimentos da melhor solução guardada (***solution_1_best.n_moves)***, e caso o seja (como já temos uma melhor solução), desistimos da solução que estamos a calcular;

- Verificar sempre se a ***speed*** está dentro dos parâmetros aceitáveis;

- Verificar sempre se a decrementação consecutiva e continua da velocidade nos permite parar de forma segura, e não exceder o tamanho da estrada, usando o número de segmentos que nos faltam percorrer como forma de comparação, caso seja detetada a transposição da estrada, desistimos do desenvolvimento da solução que está a ser calculada atualmente;

- Apurar se, para um dado movimento, a velocidade adquirida pelo carro não excede a velocidade permitida em cada um dos segmentos por onde o carro passa, até ter a opção de alterar a sua velocidade. Caso o carro esteja em excesso de velocidade em pelo menos um segmento de estrada, desistimos da solução que está atualmente a ser calculada;

- Caso todas as condições anteriormente referidas sejam cumpridas, então o carro encontra-se de acordo com todos os parâmetros aceitáveis para que seja possível continuar a determinação de um caminho alternativo (que mais tarde pode tornar-se mais eficiente, ou menos eficiente, sendo que todas as condições voltam a ser verificadas para cada movimento que o carro faz).

Solução Mais Eficiente

Ao longo da determinação de uma solução para o problema dado, são encontradas várias que podem ser usadas, e que, cumprindo todas as regras, permitem-nos chegar ao destino. No entanto, há que ser feita a seleção da melhor solução, tendo em conta a eficiência (dada pelo número de movimentos do carro).

Assim, desde o início da determinação das soluções para o problema dado, e para cada um dos movimentos feitos, é verificada a existência de uma melhor solução que se encontra guardada em ***solution_1_best*** (derivada da estrutura ***solution_t***).

Caso a solução atualmente a ser desenvolvida, e até ao momento da verificação, seja mais eficiente do que aquela guardada, continuamos a desenvolvê-la, de forma a explorar a existência de uma solução ainda melhor.

Resultados

Solução Dada Inicialmente

Com uma solução ineficiente como esta, a complexidade revelou ser um verdadeiro problema, tendo aumentado exponencialmente os tempos de execução em função do tamanho da estrada.

Assim, esta solução só permite resolver o problema para tamanhos de estrada menores ou iguais a 55. Na Figura 1 podem ser observados estes dados, assim como uma expressão matemática aproximada, que traduz o tempo de execução em função do tamanho da estrada.

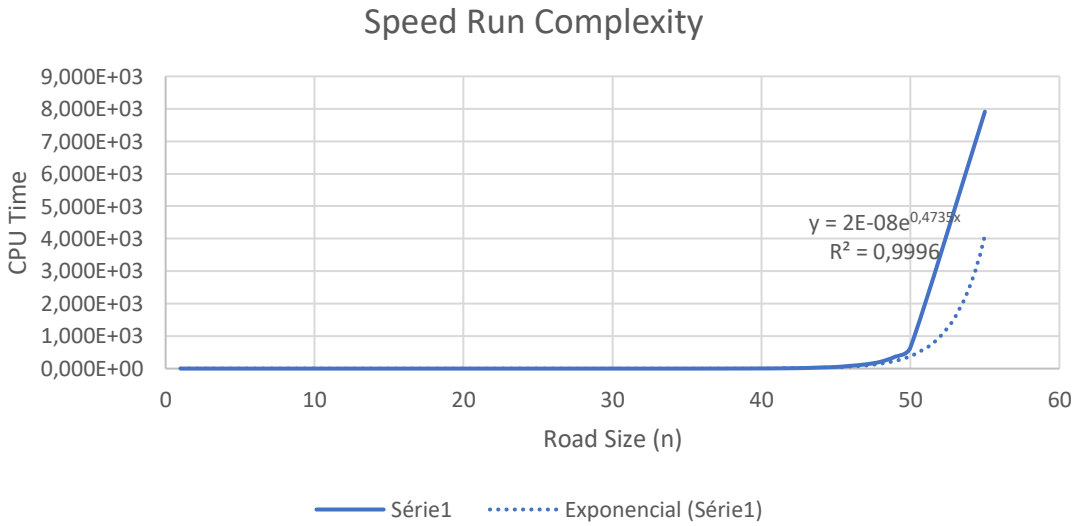

Figura 1. Complexidade Speed Run

No entanto, uma previsão do tempo de execução para uma estrada com tamanho 800 (Figura 2), revela que a solução mais eficiente levaria a imenso tempo a ser encontrada, o que não é ideal para um algoritmo generalizado para todos os tamanhos de estrada.

Em suma, a reformulação do algoritmo revelou ser algo critico para a resolução do problema num tempo de execução significativamente menor.

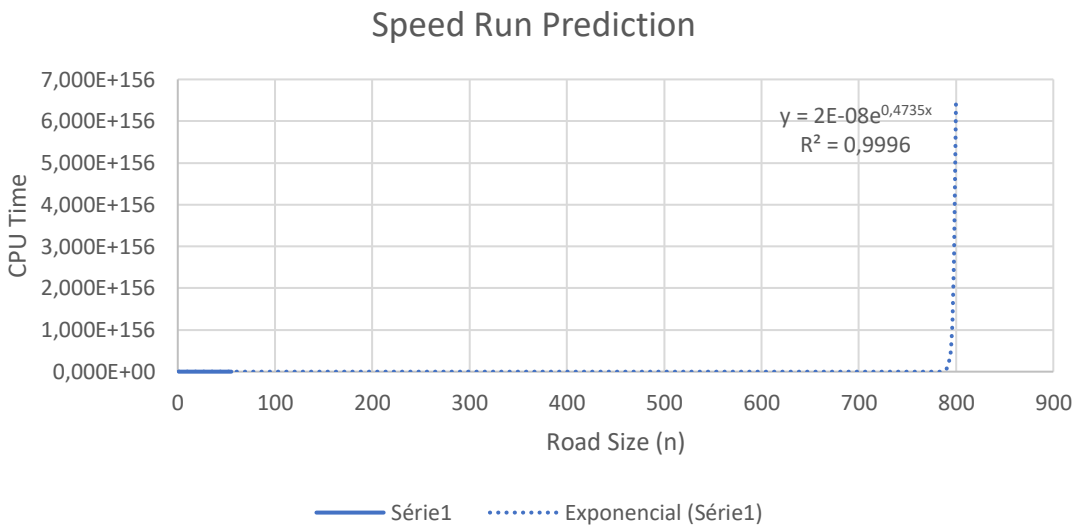

Figura 2. Speed Run Prediction

Na Figura 3 é apresentado o resultado da solução inicialmente dada, para o número mecanográfico 107266. Como se observa, o tempo de execução para uma estrada de tamanho 50 é menor do que o tempo de execução médio para uma estrada do mesmo tamanho (sendo, no entanto, bastante alto), este tempo está diretamente associado ao esforço necessário para determinar o melhor caminho.

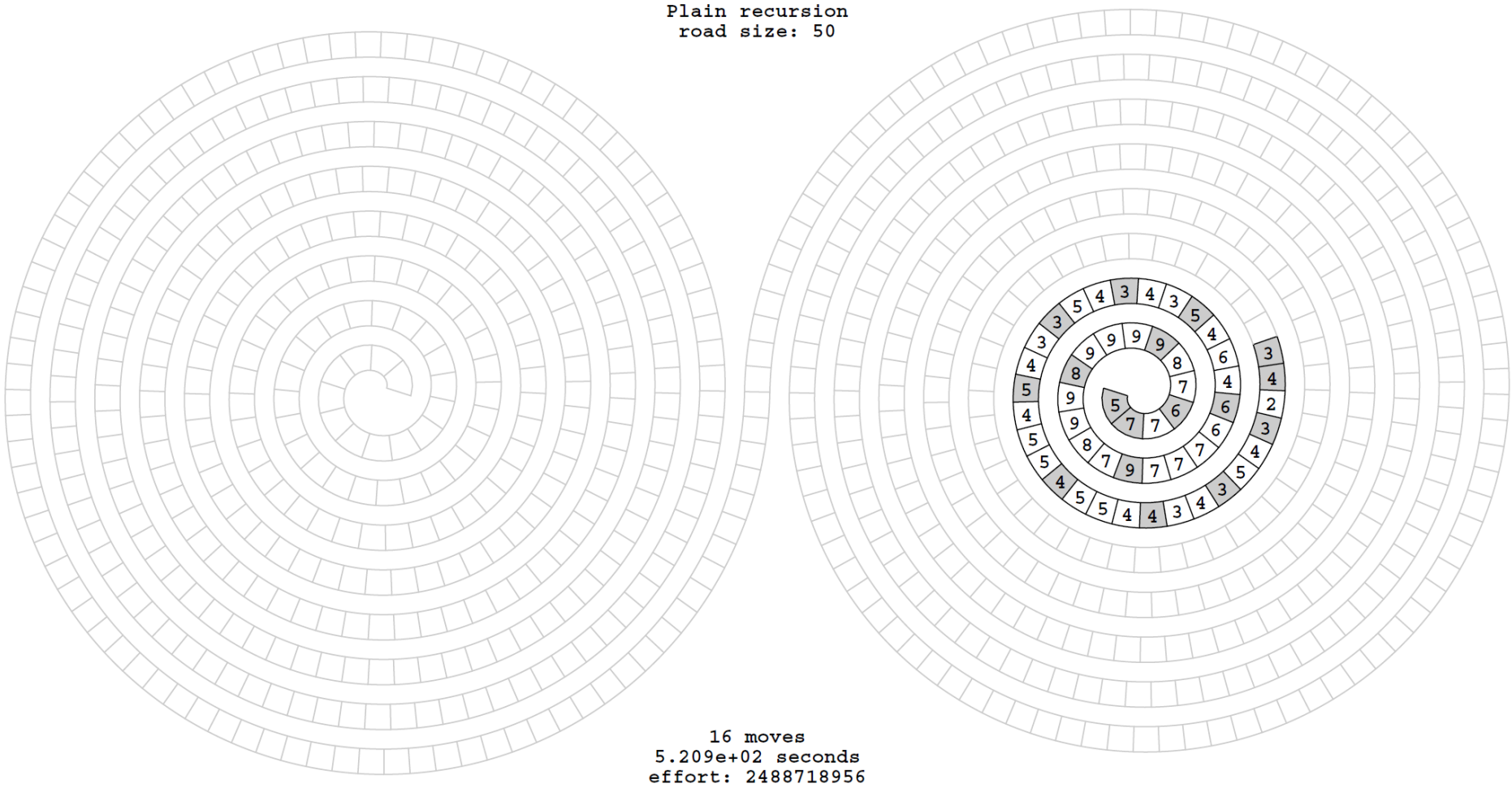

Figura 3. Resultados Speed Run para o número mecanográfico 107266

Solução Desenvolvida

Com base na solução inicialmente dada, foi desenvolvida uma nova solução. Esta solução tem uma complexidade significativamente mais reduzida do que a fornecida inicialmente, aplicando métodos baseados na realidade (métodos mencionados no capítulo Métodos Utilizados), e que revelam ser eficazes na identificação de caminhos inválidos com uma maior antecedência.

Assim, a complexidade passou de exponencial para linear, tendo sido observados tempos de execução notoriamente menores, como também um esforço reduzido na resolução do problema em questão, o que permitiu atingir uma estrada de tamanho 800 em microssegundos (Figura 4).

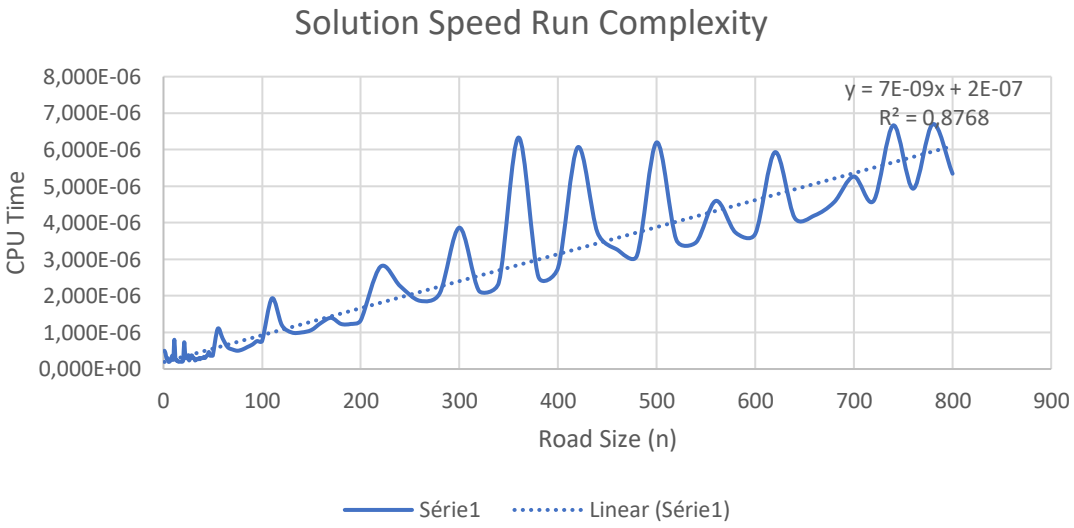

Figura 4. Solution Speed Run Complexity

Uma comparação da complexidade da solução dada inicialmente (***Speed Run***) com a complexidade da solução desenvolvida (***Solution Speed Run***), demonstra a significante redução do tempo de execução em função do tamanho da estrada (Figura 5), sendo assim demonstrada a eficácia dos métodos aplicados para resolver o problema, independentemente estrada do tamanho da estrada.

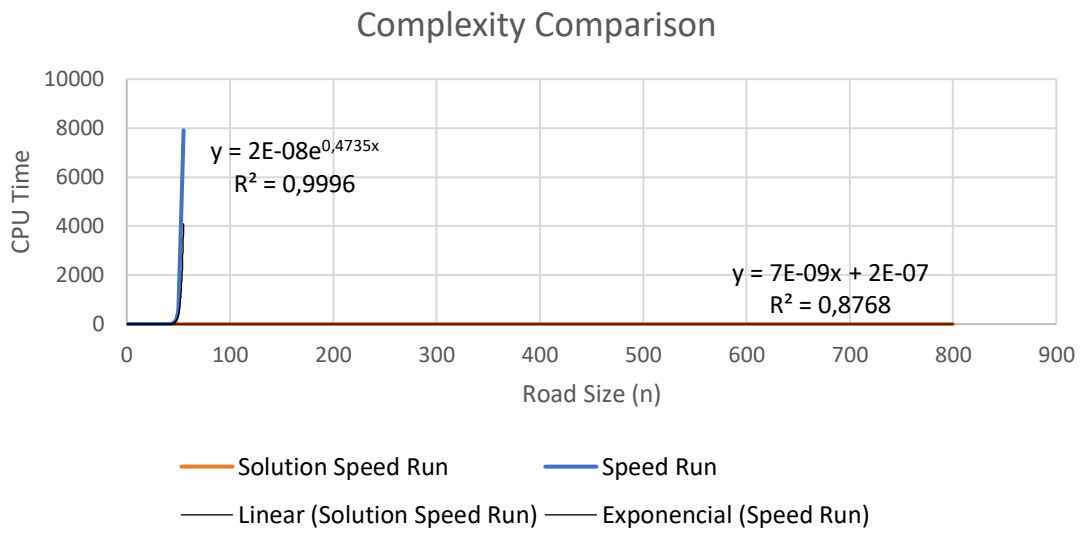

Figura 5. Speed Run Complexity vs Solution Speed Run Complexity

Na Figura 6 é apresentado o resultado da solução desenvolvida, para o número mecanográfico 107245. Como se observa, foi possível obter uma solução para uma estrada de tamanho 800 em microssegundos, tendo sido reduzido o esforço necessário para chegar a uma solução.

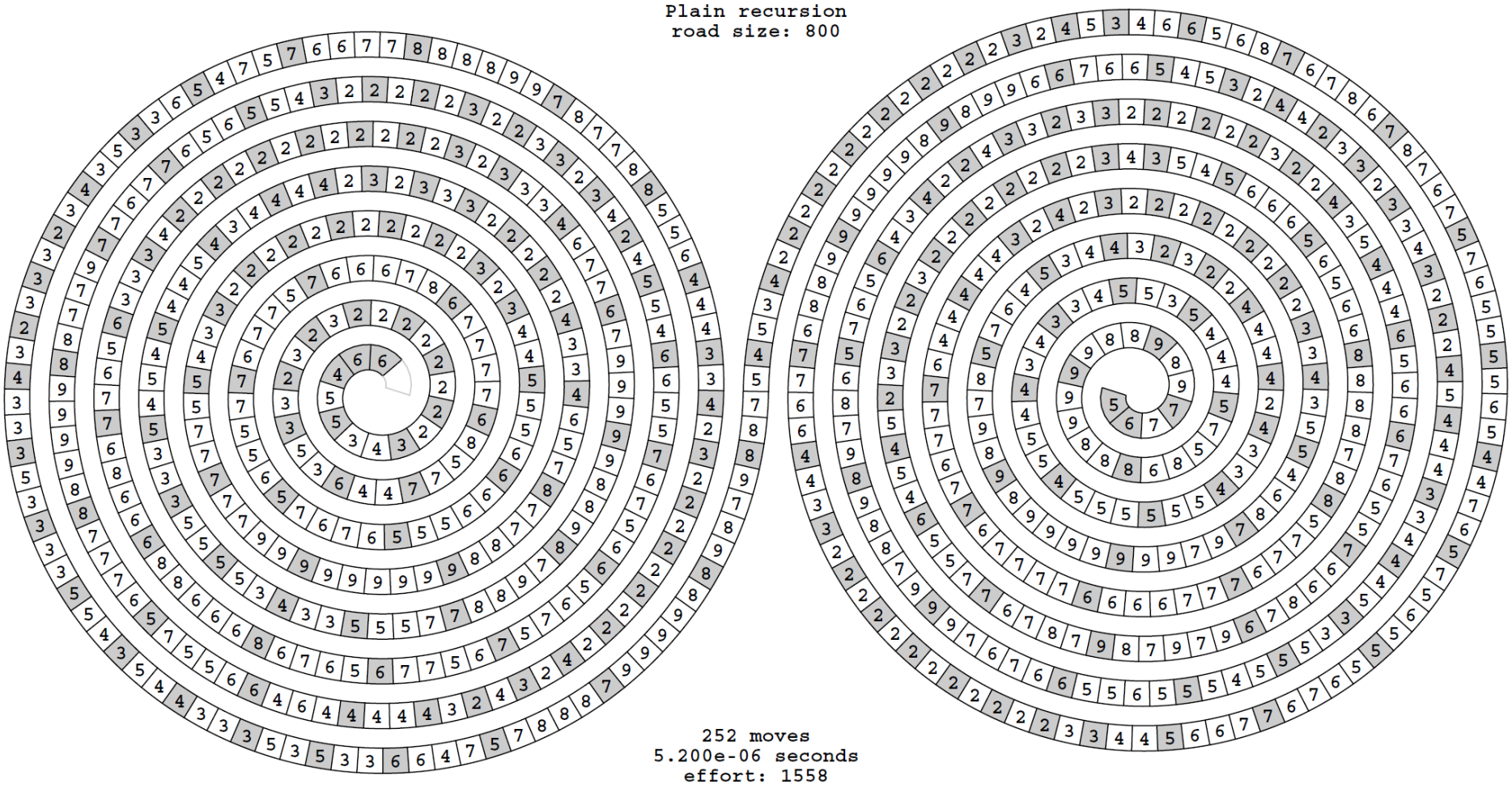

Figura 6. Resultados Solution Speed Run para o número mecanográfico 107245

Comentários

Todos os dados apresentados acima são baseados na média dos tempos de execução em função do tamanho da estrada para cada um dos números mecanográficos dos autores.

O esforço apresentado na solução desenvolvida é relativamente alto para o resultado apresentado. Este fenómeno pode ser justificado pela exploração momentânea de caminhos inválidos para uma nova solução, caminhos estes que instantaneamente deixaram de ser exploradas, mas que acabam por ser contabilizados.

Assim, e tendo em conta a rápida desistência por parte do algoritmo em prosseguir a exploração destes caminhos erráticos, este esforço não traduz de forma direta a exploração feita, podendo antes ser visto como um número que representa o caminho explorado combinado com desvios breves (é expectável que o verdadeiro esforço seja de 518 para uma estrada de tamanho 800).

Os dados e resultados de cada um dos autores, assim como a sua exaustiva análise, podem ser encontrados aqui.

Todo o código usado pode ser encontrado aqui.

universidade de aveiro

Apêndice

solution_speed_run.c

```
//
// AED, December 2022 (107266, 107245, 107214)
//
// First practical assignement (speed run)
//
// Compile using either
//   cc -Wall -O2 -D_use_zlib_=0 solution_speed_run.c -lm
// or
//   cc -Wall -O2 -D_use_zlib_=1 solution_speed_run.c -lm -lz
//
// Place your student numbers and names here
//   N.Mec. 107266  Name: Tiago Fonseca
//   N.Mec. 107245  Name: Tomás Fonseca
//   N.Mec. 107214  Name: Beatriz Ferreira
//



//
// static configuration
//

#define _max_road_size_  800  // the maximum problem size
#define _min_road_speed_   2  // must not be smaller than 1, shouldnot be smaller than 2
#define _max_road_speed_   9  // must not be larger than 9 (only because of the PDF figure)



//
// include files --- as this is a small project, we include the PDF generation code directly from
make_custom_pdf.c
//
```

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "../P02/elapsed_time.h"
#include "make_custom_pdf.c"


//
// road stuff
//

static int max_road_speed[1 + _max_road_size_]; // positions 0.._max_road_size_

static void init_road_speeds(void)
{
  double speed;
  int i;

  for(i = 0;i <= _max_road_size_;i++)
  {
    speed = (double)_max_road_speed_ * (0.55 + 0.30 * sin(0.11 * (double)i) + 0.10 * sin(0.17 * (double)i + 1.0) + 0.15 * sin(0.19 * (double)i));
    max_road_speed[i] = (int)floor(0.5 + speed) + (int)((unsigned int)rand() % 3u) - 1;
    if(max_road_speed[i] < _min_road_speed_)
      max_road_speed[i] = _min_road_speed_;
    if(max_road_speed[i] > _max_road_speed_)
      max_road_speed[i] = _max_road_speed_;
  }
}


//
// description of a 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)
```

```
}
solution_t;



//
// the (very inefficient) recursive solution given to the students
//

static solution_t solution_1,solution_1_best;
static double solution_1_elapsed_time; // time it took to solve the problem
static unsigned long solution_1_count; // effort dispended solving the problem


static boolean solution_1_recursion(int move_number,int position,int speed,int final_position) // first
arguments (0, 0, 0, final_position)
{
  int speed_soma = 0;

  solution_1.positions[move_number] = position;
  solution_1_count++;

  if (speed == 1 && position == final_position){
    if(move_number < solution_1_best.n_moves)
    {
      solution_1_best = solution_1;
      solution_1_best.n_moves = move_number;
    }
    return TRUE;
  }

  if (solution_1_best.positions[move_number] > position){
    return FALSE;
  }


  if (position == final_position){
    return FALSE;
  }

  if(move_number > solution_1_best.n_moves){
```

```
    return FALSE;
  }

  if(speed > _max_road_speed_ || speed > max_road_speed[position] || speed < 0 || (speed < 1 && position
>0)  ){
    return FALSE;
  }




  int positions_left = final_position - position;

  for(int i = speed; i >=1; i--){
      speed_soma += i;
      if (speed > max_road_speed[position+i]){
          return FALSE;
      }
  }

  if(speed_soma > positions_left){
    return FALSE;
  }

  solution_1_recursion(move_number + 1, position + speed, speed+1, final_position);
  solution_1_recursion(move_number + 1, position + speed, speed, final_position);
  solution_1_recursion(move_number + 1, position + speed, speed-1, final_position);

  return FALSE;
}

static void solve_1(int final_position) // we pass the final position of the road as an argument
{
  if(final_position < 1 || final_position > _max_road_size_) // verify that the final position is valid
  {
    fprintf(stderr,"solve_1: bad final_position\n");
    exit(1);
  }
  solution_1_elapsed_time = cpu_time(); // start the timer
  solution_1_count = 0ul; // reset the effort counter
  solution_1_best.n_moves = final_position + 100; // set the best solution to an impossible value
```

```
  solution_1_recursion(0,0,0,final_position); // call the recursive function
(move_number,position,speed,final_position)
  solution_1_elapsed_time = cpu_time() - solution_1_elapsed_time; // stop the timer and calculate the elapsed
time
}


//
// example of the slides
//

static void example(void)
{
  int i,final_position;

  srand(0xAED2022);
  init_road_speeds();
  final_position = 30;
  solve_1(final_position);
  make_custom_pdf_file("example.pdf",final_position,&max_road_speed[0],solution_1_best.n_moves,&solution_1_bes
t.positions[0],solution_1_elapsed_time,solution_1_count,"Plain recursion");
  printf("mad road speeds:");
  for(i = 0;i <= final_position;i++)
    printf(" %d",max_road_speed[i]);
  printf("\n");
  printf("positions:");
  for(i = 0;i <= solution_1_best.n_moves;i++)
    printf(" %d",solution_1_best.positions[i]);
  printf("\n");
}



//
// 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_b
est.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("                                |");
    }


    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_
}
```