Universidade de São Paulo
Instituto de Física de São Carlos

High Performance Programming-Parallel

Éverton Luís Mendes da Silva (10728171)

Contents

1 Resume

Currently, there are several applications in which, given a matrix A, the eigenvalues and eigenvectors are relevant information to solve the problem. With this in mind, there are several methods to find these values, however, most of them do not take into account the limitation of current computers in dealing with large memories and processing speed of current CPUs(approximately 10^8 iterations per second).As an example, if we have a problem with a large number of variables(N), algorithms that have a complexity of N^2(N$\rightarrow \infty$, $\Theta(N^2)$) are unfeasible for solution due to the high processing time. Therefore, in this project, the representation of a graph as a weighted adjacency matrix was discarded, thus the multiplication of a matrix by a vector was implemented with an weighted adjacency list(weighted linked list). Finally, the power iteration method for matrix multiplication was implemented in two ways: sequential and parallel.

2 Theoretical Introduction

The multiplication of a matrix by a vector can be done in two ways, one with matrix representation and the other with an adjacency list, as below:

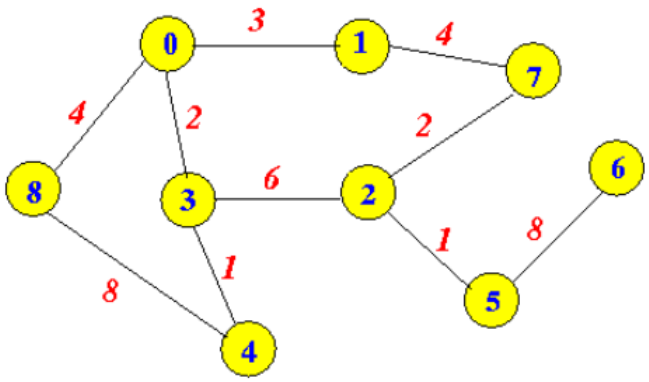

Figure 1: Graph

	0	1	2	3	4	5	6	7	8
0	0	3	0	2	0	0	0	0	4
1	3	0	0	0	0	0	0	4	0
2	0	0	0	6	0	1	0	2	0
3	2	0	6	0	1	0	0	0	0
4	0	0	0	1	0	0	0	0	8
5	0	0	1	0	0	0	8	0	0
6	0	0	0	0	0	8	0	0	0
7	0	4	2	0	0	0	0	0	0
8	4	0	0	0	8	0	0	0	0

Figure 2: Adjacency Matrix

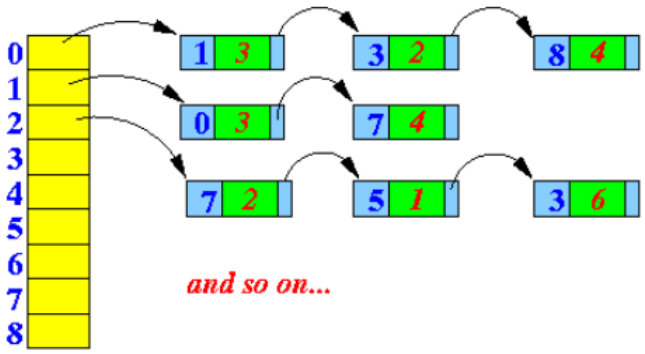

Figure 3: Adjacency List

3 Description of files

This project was divided into three main files that will be presented below. The first consists of a file, named 'AdjList', which reads a file in Pajek format and creates an adjacency list with the respective values. In addition, we have two more files to perform the power iteration method, one sequential and the other parallel.

3.1 Code AdjList

```c
#include <stdio.h>
#include <stdlib.h>

struct Graph
{
    struct Node *head[1];
};

struct Edge
{
    int i, j;
    double weight;
};

struct Node
{
    int dest;
    double weight;
    struct Node *next;
};

struct Graph *createGraph(struct Edge edges[], int n, int n_vertexs)
{
    struct Graph *graph = (struct Graph *)malloc(sizeof(struct Graph) + sizeof(struct Node) * (n_vertexs - 1));

    for (int i = 0; i < n_vertexs; i++)
    {
        graph->head[i] = NULL;
    }

    for (int i = 0; i < n; i++)
    {
        int src = edges[i].i;
        int dest = edges[i].j;
        double weight = edges[i].weight;

        struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
        newNode->dest = dest;
        newNode->weight = weight;

        newNode->next = graph->head[src];

        graph->head[src] = newNode;
    }
    return graph;
}
```

```
47
48 void printGraph(struct Graph *graph, int n_vertexs)
49 {
50     int i;
51     for (i = 0; i < n_vertexs; i++)
52     {
53         struct Node *ptr = graph->head[i];
54         while (ptr != NULL)
55         {
56             printf("%d    > %d ", i, ptr->dest);
57             ptr = ptr->next;
58         }
59
60         printf("\n");
61     }
62 }
63
64 struct File_data
65 {
66     int n_vertexs;
67     struct Graph *graph;
68 };
69
70 struct File_data *ReadPajek(char *filename)
71 {
72     FILE *fp;
73     fp = fopen(filename, "r");
74
75     int n_vertexs, n_edges;
76     int i_element, j_element;
77     double weight;
78
79     if (fscanf(fp, "%d", &n_vertexs))
80     {
81     }
82     if (fscanf(fp, "%d ", &n_edges))
83     {
84     }
85
86     struct Edge *Edges = malloc(2 * n_edges * sizeof(struct
   Edge));
87
88     int count_equal_ij = 0;
89     for (int edge = 0; edge < n_edges; edge++)
90     {
91         if (fscanf(fp, "%d %d %lf", &i_element, &j_element, &
   weight))
92         {
93         }
```

```
94
95          if (i_element != j_element)
96          {
97              Edges[2 * edge - count_equal_ij].i = i_element;
98              Edges[2 * edge - count_equal_ij].j = j_element;
99              Edges[2 * edge - count_equal_ij].weight = weight;
100             Edges[2 * edge + 1 - count_equal_ij].i =
    j_element;
101             Edges[2 * edge + 1 - count_equal_ij].j =
    i_element;
102             Edges[2 * edge + 1 - count_equal_ij].weight =
    weight;
103         }
104         else
105         {
106             Edges[2 * edge - count_equal_ij].i = i_element;
107             Edges[2 * edge - count_equal_ij].j = j_element;
108             Edges[2 * edge - count_equal_ij].weight = weight;
109             count_equal_ij += 1;
110         }
111     }
112
113     fclose(fp);
114
115     struct Graph *graph = createGraph(Edges, 2 * n_edges -
    count_equal_ij, n_vertexs);
116
117     struct File_data *file_data = malloc(sizeof(int) + sizeof
    (struct Graph) + sizeof(struct Node) * (n_vertexs - 1));
118     file_data->n_vertexs = n_vertexs;
119     file_data->graph = graph;
120
121     return file_data;
122 }
```

Listing 1: Read file and Adjacency List Representation

3.2 Code FindEigen - Sequential

```
1  #include "AdjListT.c"
2  #include <math.h>
3  #include <string.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <sys/time.h>
7  #include <unistd.h>
8
9  void read_arguments_or_abort(int argc, char *argv[]);
10 double normalize_vec(int n_vertexs, double *vector);
```

```
11 void mat_mult_AdjList(struct Graph *graph, double *vector,
       double *new_vector, int n_vertexs);
12 void printfvector(double *vector, int n_vertexs);
13 void cleanVector(double *vector, int n_vertexs);
14 double mult_pointers(double num1, double num2);
15 void copy_vec(double *vector, double *new_vector, int
       n_vertexs);
16
17 int main(int argc, char *argv[])
18 {
19     read_arguments_or_abort(argc, argv);
20     char *input_filename = argv[1];
21
22     double precision;
23     sscanf(argv[2], "%lf", &precision);
24     char *output_filename = argv[3];
25
26     struct File_data *file_data = ReadPajek(input_filename);
27
28     double *vec = (double *)malloc(file_data->n_vertexs *
       sizeof(double));
29     double *new_vec = (double *)malloc(file_data->n_vertexs *
       sizeof(double));
30
31     for (int i = 0; i < file_data->n_vertexs; i++)
32     {
33         vec[i] = rand() / (RAND_MAX + 1.0);
34         if (rand() / (RAND_MAX + 1.0) >= 0.5)
35         {
36             vec[i] *= -1;
37         }
38     }
39
40     int stop_iter = 0;
41     double norm_vec, new_norm_vec;
42
43     norm_vec = normalize_vec(file_data->n_vertexs, vec);
44
45     struct timeval t1, t2;
46     gettimeofday(&t1, NULL);
47     while (stop_iter < 3)
48     {
49         cleanVector(new_vec, file_data->n_vertexs);
50         mat_mult_AdjList(file_data->graph, vec, new_vec,
       file_data->n_vertexs);
51         new_norm_vec = normalize_vec(file_data->n_vertexs,
       new_vec);
52         copy_vec(vec, new_vec, file_data->n_vertexs);
53
```

```
54          if (fabs(new_norm_vec - norm_vec) / new_norm_vec <
    precision)
55          {
56              stop_iter += 1;
57          }
58          else
59          {
60              stop_iter = 0;
61          }
62
63          norm_vec = new_norm_vec;
64      }
65      gettimeofday(&t2, NULL);
66
67      printf("It took %.17lf milliseconds.\n", (t2.tv_sec - t1.
    tv_sec) + (t2.tv_usec - t1.tv_usec) / 1e6);
68
69      FILE *output_file;
70      output_file = fopen(output_filename, "w");
71      fprintf(output_file, "%lf\n", new_norm_vec);
72      fprintf(output_file, "%d\n", file_data->n_vertexs);
73
74      for (int i = 0; i < file_data->n_vertexs; i++)
75      {
76          fprintf(output_file, "%lf\n", vec[i]);
77      }
78
79      fclose(output_file);
80
81      cleanVector(new_vec, file_data->n_vertexs);
82      mat_mult_AdjList(file_data->graph, vec, new_vec,
    file_data->n_vertexs);
83      new_norm_vec = normalize_vec(file_data->n_vertexs,
    new_vec);
84
85      if (fabs(new_norm_vec - norm_vec) / new_norm_vec <
    precision)
86      {
87          printf("The Method works well\n");
88      }
89      else
90      {
91          printf("The Method don't work so well\n");
92          printf("method: %.17lf precision: %.17lf\n", fabs(
    new_norm_vec - norm_vec) / new_norm_vec, precision);
93      }
94
95      FILE *time_record_file;
96      char timefilename[100] = "time_";
```

```
97      strcat(timefilename, input_filename);
98      printf("%s\n", timefilename);
99      time_record_file = fopen(timefilename, "a+");
100     fprintf(time_record_file, "%.10lf\n", (t2.tv_sec - t1.
    tv_sec) + (t2.tv_usec - t1.tv_usec) / 1e6);
101     fclose(time_record_file);
102
103     return 0;
104 }
105
106 void read_arguments_or_abort(int argc, char *argv[])
107 {
108     if (argc != 4)
109     {
110         fprintf(stderr, "Usage: %s <number of elements> <
    number of arrays>\n",
111                 argv[0]);
112         exit(505);
113     }
114 }
115
116 double normalize_vec(int n_vertexs, double *vector)
117 {
118
119     double sum_elements = 0;
120     for (int i = 0; i < n_vertexs; i++)
121     {
122         sum_elements += pow(vector[i], 2);
123     }
124
125     for (int i = 0; i < n_vertexs; i++)
126     {
127         vector[i] /= sqrt(sum_elements);
128     }
129
130     return sqrt(sum_elements);
131 }
132
133 void mat_mult_AdjList(struct Graph *graph, double *vector,
    double *new_vector, int n_vertexs)
134 {
135
136     for (int i = 0; i < n_vertexs; i++)
137     {
138         struct Node *ptr = graph->head[i];
139
140         if (ptr == NULL)
141         {
142         }
```

```
143         else
144         {
145             while (ptr != NULL)
146             {
147                 new_vector[i] += mult_pointers(ptr->weight,
    vector[ptr->dest]);
148                 ptr = ptr->next;
149             }
150         }
151     }
152 }
153
154 void printfvector(double *vector, int n_vertexs)
155 {
156     for (int i = 0; i < n_vertexs; i++)
157     {
158         printf("%lf ", vector[i]);
159     }
160     printf("\n");
161 }
162
163 void cleanVector(double *Clean_vector, int n_vertexs)
164 {
165     double zero = 0;
166     for (int i = 0; i < n_vertexs; i++)
167     {
168         Clean_vector[i] = zero;
169     }
170 }
171
172 double mult_pointers(double num1, double num2)
173 {
174     double aux1 = num1;
175     double aux2 = num2;
176     double mult_value = aux1 * aux2;
177     return mult_value;
178 }
179
180 void copy_vec(double *vector, double *new_vector, int
    n_vertexs)
181 {
182     for (int i = 0; i < n_vertexs; i++)
183     {
184         double aux = new_vector[i];
185         vector[i] = aux;
186     }
187 }
```

Listing 2: power iteratrion - Sequential

3.3 Code FindEigen_omp - Parallel

```c
#include "AdjListT.c"
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>

void read_arguments_or_abort(int argc, char *argv[]);
double normalize_vec(int n_vertexs, double *vector);
void mat_mult_AdjList(struct Graph *graph, double *vector,
    double *new_vector, int n_vertexs);
void printfvector(double *vector, int n_vertexs);
void cleanVector(double *vector, int n_vertexs);
double mult_pointers(double num1, double num2);
void copy_vec(double *vector, double *new_vector, int
    n_vertexs);

int main(int argc, char *argv[])
{

    read_arguments_or_abort(argc, argv);
    char *input_filename = argv[1];

    double precision;
    sscanf(argv[2], "%lf", &precision);

    char *output_filename = argv[3];

    struct File_data *file_data = ReadPajek(input_filename);

    double *vec = (double *)malloc(file_data->n_vertexs *
    sizeof(double));
    double *new_vec = (double *)malloc(file_data->n_vertexs *
    sizeof(double));

    for (int i = 0; i < file_data->n_vertexs; i++)
    {
        vec[i] = rand() / (RAND_MAX + 1.0);
        if (rand() / (RAND_MAX + 1.0) >= 0.5)
        {
            vec[i] *= -1;
        }
    }

    int stop_iter = 0;
    double norm_vec, new_norm_vec;
```

```
44
45      norm_vec = normalize_vec(file_data->n_vertexs, vec);
46
47      struct timeval t1, t2;
48      gettimeofday(&t1, NULL);
49      while (stop_iter < 3)
50      {
51
52          cleanVector(new_vec, file_data->n_vertexs);
53          mat_mult_AdjList(file_data->graph, vec, new_vec,
    file_data->n_vertexs);
54          new_norm_vec = normalize_vec(file_data->n_vertexs,
    new_vec);
55          copy_vec(vec, new_vec, file_data->n_vertexs);
56
57          if (fabs(new_norm_vec - norm_vec) / new_norm_vec <
    precision)
58          {
59              stop_iter += 1;
60          }
61          else
62          {
63              stop_iter = 0;
64          }
65
66          norm_vec = new_norm_vec;
67      }
68      gettimeofday(&t2, NULL);
69
70      printf("It took %.17lf milliseconds.\n", (t2.tv_sec - t1.
    tv_sec) + (t2.tv_usec - t1.tv_usec) / 1e6);
71
72      FILE *output_file;
73      output_file = fopen(output_filename, "w");
74      fprintf(output_file, "%lf\n", new_norm_vec);
75      fprintf(output_file, "%d\n", file_data->n_vertexs);
76
77      for (int i = 0; i < file_data->n_vertexs; i++)
78      {
79          fprintf(output_file, "%lf\n", vec[i]);
80      }
81
82      fclose(output_file);
83
84      cleanVector(new_vec, file_data->n_vertexs);
85      mat_mult_AdjList(file_data->graph, vec, new_vec,
    file_data->n_vertexs);
86      new_norm_vec = normalize_vec(file_data->n_vertexs,
    new_vec);
```

```
87
88      if (fabs(new_norm_vec - norm_vec) / new_norm_vec <
        precision)
89      {
90          printf("The Method works well\n");
91      }
92      else
93      {
94          printf("The Method don't work so well\n");
95          printf("method: %.17lf precision: %.17lf\n", fabs(
        new_norm_vec - norm_vec) / new_norm_vec, precision);
96      }
97
98      FILE *time_record_file;
99      char timefilename[100] = "time_omp_";
100     strcat(timefilename, input_filename);
101     printf("%s\n", timefilename);
102     time_record_file = fopen(timefilename, "a+");
103     fprintf(time_record_file, "%.10lf\n", (t2.tv_sec - t1.
        tv_sec) + (t2.tv_usec - t1.tv_usec) / 1e6);
104     fclose(time_record_file);
105     return 0;
106 }
107
108 void read_arguments_or_abort(int argc, char *argv[])
109 {
110     if (argc != 4)
111     {
112         fprintf(stderr, "Usage: %s <number of elements> <
        number of arrays>\n",
113                 argv[0]);
114         exit(505);
115     }
116 }
117
118 double normalize_vec(int n_vertexs, double *vector)
119 {
120
121     double sum_elements = 0;
122 #pragma omp parallel for default(none) shared(vector,
        n_vertexs) reduction(+ \
123
                     : sum_elements) schedule(static)
124     for (int i = 0; i < n_vertexs; i++)
125     {
126         sum_elements += pow(vector[i], 2);
127     }
128
129 #pragma omp parallel for default(none) shared(vector,
```

```
    n_vertexs, sum_elements) schedule(static)
    for (int i = 0; i < n_vertexs; i++)
    {
        vector[i] /= sqrt(sum_elements);
    }
    return sqrt(sum_elements);
}

void mat_mult_AdjList(struct Graph *graph, double *vector,
    double *new_vector, int n_vertexs)
{

#pragma omp parallel for default(none) shared(new_vector,
    n_vertexs, vector, graph) schedule(dynamic)
    for (int i = 0; i < n_vertexs; i++)
    {
        struct Node *ptr = graph->head[i];
        if (ptr == NULL)
        {
        }
        else
        {
            while (ptr != NULL)
            {
                new_vector[i] += mult_pointers(ptr->weight,
    vector[ptr->dest]);
                ptr = ptr->next;
            }
        }
    }
}

void printfvector(double *vector, int n_vertexs)
{
    for (int i = 0; i < n_vertexs; i++)
    {
        printf("%lf ", vector[i]);
    }
    printf("\n");
}

void cleanVector(double *Clean_vector, int n_vertexs)
{
    double zero = 0;
#pragma omp parallel for default(none) shared(Clean_vector,
    n_vertexs, zero) schedule(static)
    for (int i = 0; i < n_vertexs; i++)
    {
        Clean_vector[i] = zero;
```

```
174     }
175 }
176
177 double mult_pointers(double num1, double num2)
178 {
179     double aux1 = num1;
180     double aux2 = num2;
181     double mult_value = aux1 * aux2;
182     return mult_value;
183 }
184
185 void copy_vec(double *vector, double *new_vector, int
        n_vertexs)
186 {
187 #pragma omp parallel for default(none) shared(new_vector,
        vector, n_vertexs) schedule(static)
188     for (int i = 0; i < n_vertexs; i++)
189     {
190         double aux = new_vector[i];
191         vector[i] = aux;
192     }
193 }
```

Listing 3: power iteratrion - Parallel

4 Performance analysis

For the performance analysis, the test files available in the compiled file powerit_test.tar.gz were used. With this in mind, error bar plots were performed to verify the processing time of each code.

4.1 Small

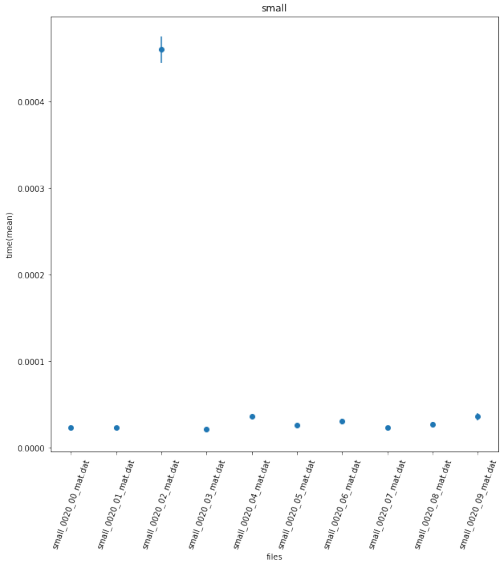

Figure 4: Sequential

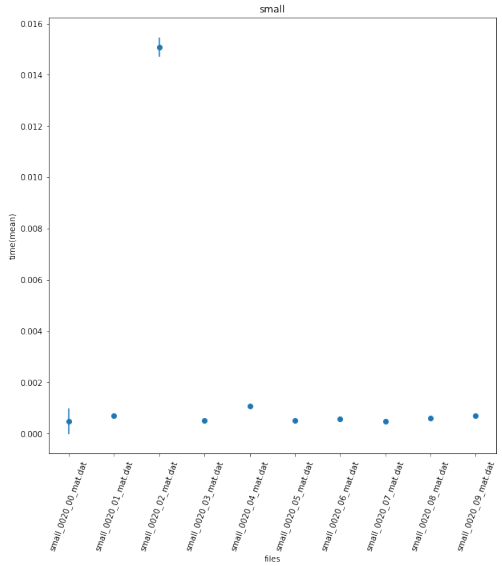

Figure 5: Parallel

4.2 Medium

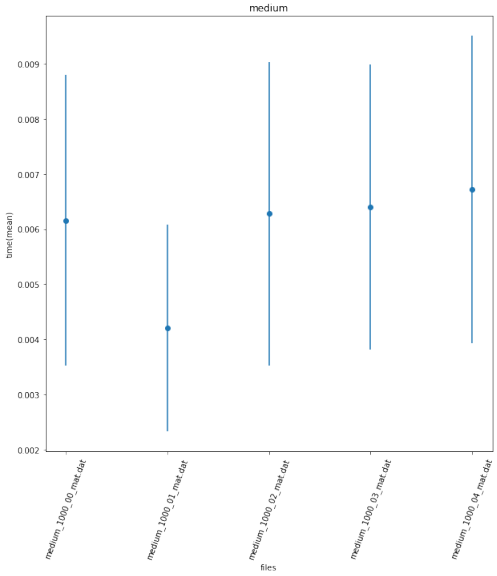

Figure 6: Sequential

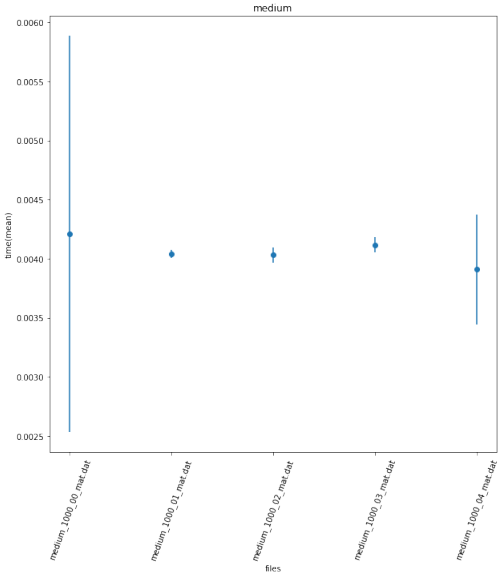

Figure 7: Parallel

4.3 Large

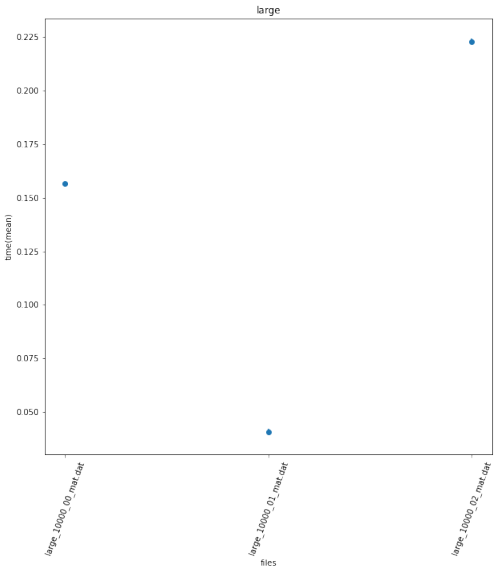

Figure 8: Sequential

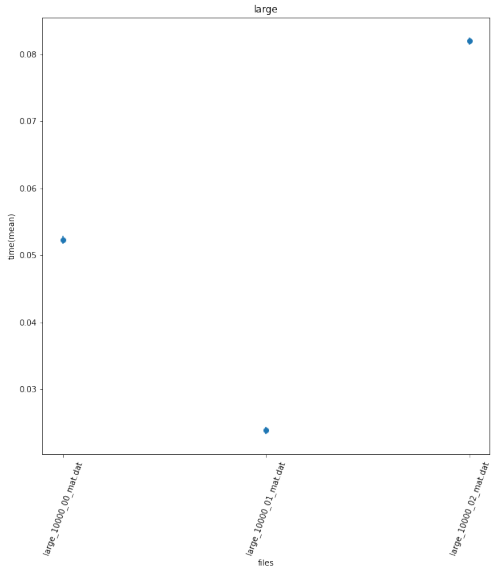

Figure 9: Parallel

4.4 Huge

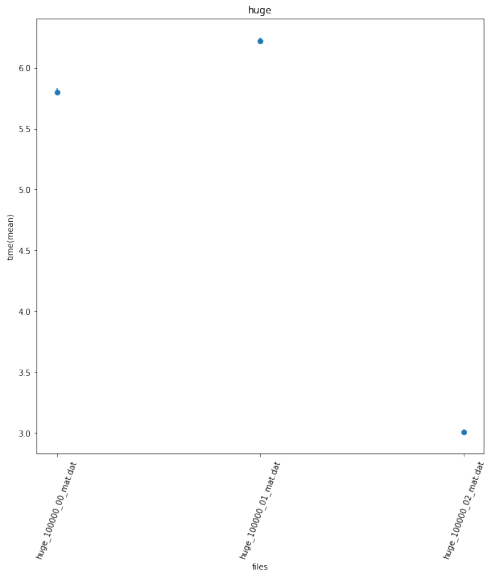

Figure 10: Sequential

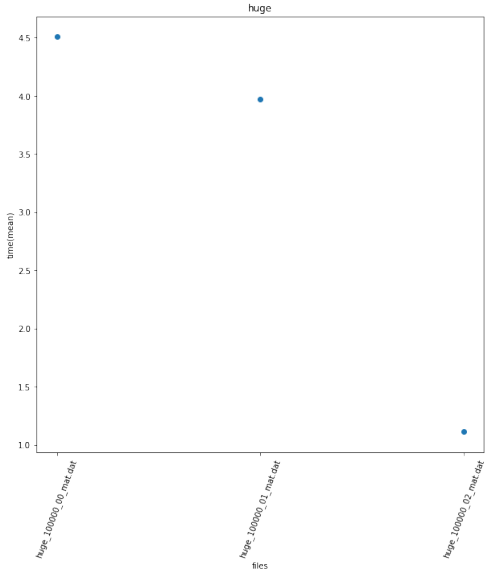

Figure 11: Parallel

4.5 All size files

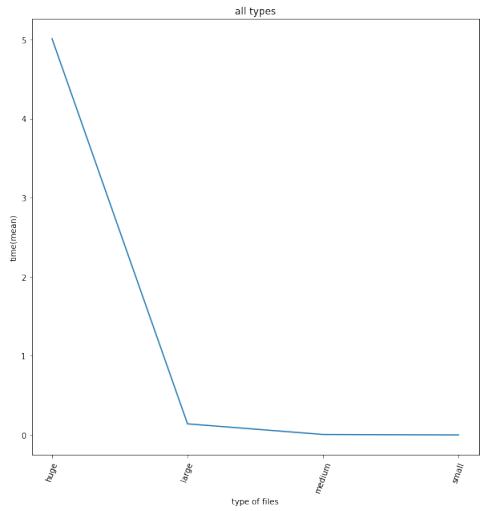

Figure 12: Mean of each type file - Sequential

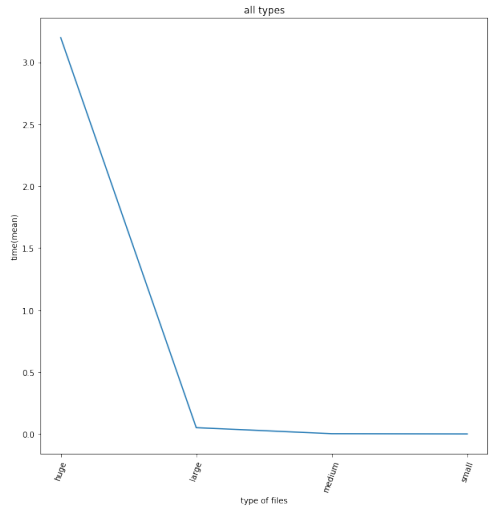

Figure 13: Mean of each type file - Parallel

5 Reference

[1] Links with the images of Graph and Adjancecy List
[2] da Silva, Éverton Luís Mendes. Codes from this project