Leibniz Universität Hannover

Further investigation

Optimize the bachelor degree project
"Search for new radio pulsars via coincidence in the Einstein@Home and Cornell result databases"

Author:
Tjark Miener 2860000

Supervisor:
Prof. Dr. Bruce Allen

June 20, 2016

Abstract

The task of this investigation is to optimize the bachelor degree project *"Search for new radio pulsars via coincidence in the Einstein@Home and Cornell result databases"*. The goal is still to find new radio pulsars via coincidence in the Einstein@Home and Cornell result databases, but in a faster and more efficent way. Therefore, the databases are stored separately in two linear lists. Merely coincidences between candidates from different databases are relevant for the project. Basically, the new algorithm skips all comparisons of candidates from the same database. This decreases the runtime from roughly 22 h to 15 min (MacBook Pro, 2,66 GHz Intel Core 2 Duo, 4 GB 1067 MHz DDR3).

1 Introduction

The following report is a continuation of the bachelor degree project *"Search for new radio pulsars via coincidence in the Einstein@Home and Cornell result databases"*. This report only presents improvements of the code. First steps are covered in the bachelor degree project.

The long run time of the code in the bachelor degree project is based on the fact that the two databases are stored in one linear list. There is a huge number of comparisons between two candidates of the same database. These comparisons are totally useless, because in the end only the coincidences between different databases matter. So the main idea of this investigation is to store the two different databases separately in two lists. Figure 1 describes the procedure of the new way of comparing candidates.

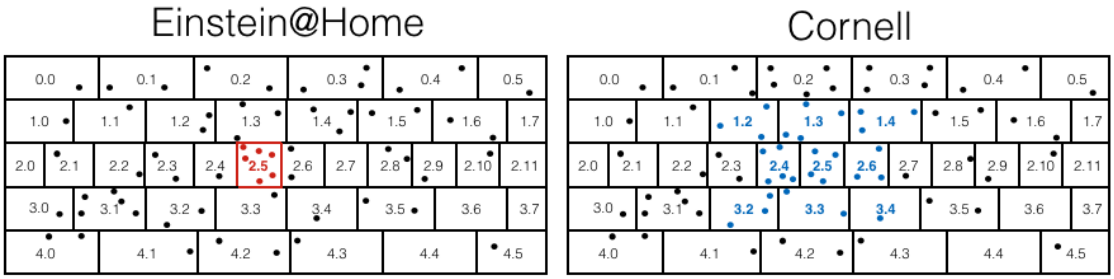

Figure 1: An example of the basic procedure of comparing candidates. The candidates of the two databases, represented as small colored points (black, red and blue), are positioned in the different cells of their appropriate list (database). In contrast of the bachelor degree project the target candidates are located in the einstein list and marked **red**. The runner candidates are located in the cornell list and marked **blue**. The new algorithm will compare the target candidates and the runner candidates. Only comparisons between these two groups of candidates will lead to the required coincidences, because of their similar coordinates.

2 Execution of the idea

2.1 Improvements

Most of the functions and code parts can be reused from the previous code, because the structure of the skypoints is unchanged. The two databases (Einstein@Home and Cornell) are stored separately in two linear lists (einstein_list and cornell_list).

```
struct Skypoint{
  int number;
  char id[Maximum_scan];

  double phi;       //0,...,2*PI
  double theta;     //0,...,PI

  int p_cell;
  int t_cell;

  double frequency;
  float significance;
};
struct Skypoint *einstein_list=NULL;
struct Skypoint *cornell_list=NULL;
```

A dynamic memory allocation is still useful. Whenever a new block of memory is needed in the two lists, the program calls the function *my_Allocate_Einstein()* or *my_Allocate_Cornell()*. The function *realloc()* is of the standard library <stdlib.h>.

```
void my_Allocate_Einstein(int a) {
  einstein_list=realloc(einstein_list,a*sizeof(struct Skypoint));
  if (!einstein_list) {
    printf("Error! No more RAM available, skypoints_max = %d!\n", a);
    exit(1);
  }
}

void my_Allocate_Cornell(int a) {
  cornell_list=realloc(cornell_list,a*sizeof(struct Skypoint));
  if (!cornell_list) {
    printf("Error! No more RAM available, skypoints_max = %d!\n", a);
    exit(2);
  }
}
```

By means of *fscanf()* of <stdio.h>, the input (databases) is inserted in the two different lists of the structure Skypoint. The coordinates of the points are transformed from RAJD and DecJD into phi and theta (declared in radians).

$$\phi = \frac{2\pi}{360} \cdot RAJD$$

$$\theta = \frac{2\pi}{360} \cdot (90 - DecJD)$$

Afterwards, as a verification, it is checked if the coordinates are in the permitted range ($\phi \in [0, \ldots, 2\pi]$ and $\theta \in [0, \ldots, \pi]$). Subsequently, the function *cellcomputing()* is called to compute the cell along with its p_cell and t_cell.

```
#define Allocate_blocksize 1024
#define End_of_first_list 1265589

skypoints_read_Einstein=0;
skypoints_read_Cornell=0;
skypoints_allocated_Einstein=0;
skypoints_allocated_Cornell=0;

while (true) {

  if (skypoints_read_Einstein < End_of_first_list) {

    //reallocating memory if necessary
    if (skypoints_read_Einstein==skypoints_allocated_Einstein) {
      skypoints_allocated_Einstein += Allocate_blocksize;
      my_Allocate_Einstein(skypoints_allocated_Einstein);
    }

    //scan the elements of the first part of the file (Einstein database)
    if (6==(i=fscanf(input_candidate, "%d %s %lf %lf %lf %f",&einstein_list
        [skypoints_read_Einstein].number, einstein_list[
        skypoints_read_Einstein].id,&einstein_list[skypoints_read_Einstein
        ].phi,&einstein_list[skypoints_read_Einstein].theta,&einstein_list[
        skypoints_read_Einstein].frequency,&einstein_list[
        skypoints_read_Einstein].significance))) {

      einstein_list[skypoints_read_Einstein].phi=(einstein_list[
          skypoints_read_Einstein].phi*15.0)*2.0*M_PI/360.0;
      einstein_list[skypoints_read_Einstein].theta=(90.0-einstein_list[
          skypoints_read_Einstein].theta)*2.0*M_PI/360.0;

      //check the data ranges, exit if error
      if (einstein_list[skypoints_read_Einstein].phi < 0 || einstein_list[
          skypoints_read_Einstein].phi > 2*M_PI || einstein_list[
          skypoints_read_Einstein].theta < 0 || einstein_list[
          skypoints_read_Einstein].theta > M_PI) {
        printf("Error! Problem reading %s at line %d. Point (%f,%f) isn't
            in the data range!\n", argv[1],skypoints_read_Einstein,
            einstein_list[skypoints_read_Einstein].phi, einstein_list[
            skypoints_read_Einstein].theta);
        exit(6);
      }

      cellcomputing(einstein_list+skypoints_read_Einstein);
      skypoints_read_Einstein++;

    } else break;

  } else {
```

```
    //reallocating memory if necessary
    if (skypoints_read_Cornell==skypoints_allocated_Cornell) {
      skypoints_allocated_Cornell += Allocate_blocksize;
      my_Allocate_Cornell(skypoints_allocated_Cornell);
    }

    //scan the elements of the secound part of the file (Cornell database)
    if (5==(i=fscanf(input_candidate, "%d %s %lf %lf %lf",&cornell_list[
        skypoints_read_Cornell].number,cornell_list[skypoints_read_Cornell
        ].id,&cornell_list[skypoints_read_Cornell].phi,&cornell_list[
        skypoints_read_Cornell].theta,&cornell_list[skypoints_read_Cornell
        ].frequency))) {

      cornell_list[skypoints_read_Cornell].phi=cornell_list[
          skypoints_read_Cornell].phi*2.0*M_PI/360.0;
      cornell_list[skypoints_read_Cornell].theta=(90.0-cornell_list[
          skypoints_read_Cornell].theta)*2.0*M_PI/360.0;
      cornell_list[skypoints_read_Cornell].significance=0.0;

      //check the data ranges, exit if error
      if (cornell_list[skypoints_read_Cornell].phi < 0 || cornell_list[
          skypoints_read_Cornell].phi > 2*M_PI || cornell_list[
          skypoints_read_Cornell].theta < 0 || cornell_list[
          skypoints_read_Cornell].theta > M_PI) {
        printf("Error! Problem reading %s at line %d. Point (%f,%f) isn't
            in the data range!\n", argv[1],skypoints_read_Cornell,
            cornell_list[skypoints_read_Cornell].phi, cornell_list[
            skypoints_read_Cornell].theta);
        exit(7);
      }

      cellcomputing(cornell_list+skypoints_read_Cornell);
      skypoints_read_Cornell++;

    } else break;

  }
}

if (i!=EOF) {
  if (skypoints_read_Cornell==0) {
    fprintf(stderr, "Error! Problem reading %s at line %d (number:%d).
        fscanf() returned %d\n", argv[1], skypoints_read_Einstein,
        einstein_list[skypoints_read_Einstein].number,i);
    exit(8);
  } else {
    fprintf(stderr, "Error! Problem reading %s at line %d (number:%d).
        fscanf() returned %d\n", argv[1], skypoints_read_Einstein+
        skypoints_read_Cornell, cornell_list[skypoints_read_Cornell].number
        ,i);
    exit(9);
  }
}
```

The border case still remains. If a candidate of the cornell database is located at the beginning or end of a stripe, a so-called shadow-point of the original candidate is generated and added to the cornell list. Since later on candidates/elements of the einstein list will be picked out and compared with the cornell list and not the other way around, the creation of shadow points is only needed in the cornell list.

```
k=j=0;
for (i=0; i<skypoints_read_Cornell; i++) {

  //reallocating memory if necessary
  if ((skypoints_read_Cornell+k)==skypoints_allocated_Cornell && j==1) {
    skypoints_allocated_Cornell += Allocate_blocksize;
    my_Allocate_Cornell(skypoints_allocated_Cornell);
  }

  //adding shadow points on the right
  if (cornell_list[i].p_cell < 2){

    cornell_list[skypoints_read_Cornell+k].phi=((cornell_list[i].phi)+2.0*
        M_PI);
    cornell_list[skypoints_read_Cornell+k].theta=cornell_list[i].theta;
    cornell_list[skypoints_read_Cornell+k].t_cell=cornell_list[i].t_cell;
    cornell_list[skypoints_read_Cornell+k].p_cell=cornell_list[i].p_cell+
        num_in_stripe[cornell_list[i].t_cell];
    cornell_list[skypoints_read_Cornell+k].significance=0.0;
    cornell_list[skypoints_read_Cornell+k].frequency=cornell_list[i].
        frequency;
    cornell_list[skypoints_read_Cornell+k].number=cornell_list[i].number;
    strncpy(cornell_list[skypoints_read_Cornell+k].id, cornell_list[i].id,
        Maximum_scan);

    k++;
    j=1;

  //adding shadow points on the left
  } else if (cornell_list[i].p_cell > num_in_stripe[cornell_list[i].t_cell
      ]-3) {

    cornell_list[skypoints_read_Cornell+k].phi=((cornell_list[i].phi)-2.0*
        M_PI);
    cornell_list[skypoints_read_Cornell+k].theta=cornell_list[i].theta;
    cornell_list[skypoints_read_Cornell+k].t_cell=cornell_list[i].t_cell;
    cornell_list[skypoints_read_Cornell+k].p_cell=cornell_list[i].p_cell-
        num_in_stripe[cornell_list[i].t_cell];
    cornell_list[skypoints_read_Cornell+k].significance=0.0;
    cornell_list[skypoints_read_Cornell+k].frequency=cornell_list[i].
        frequency;
    cornell_list[skypoints_read_Cornell+k].number=cornell_list[i].number;
    strncpy(cornell_list[skypoints_read_Cornell+k].id, cornell_list[i].id,
        Maximum_scan);

    k++;
    j=1;
```

```
  } else {
    j=0;
  }
}

skypoints_read_Cornell += k;
```

The two lists are sorted with the same *compare()*-function as before.

```
qsort (einstein_list,skypoints_read_Einstein,sizeof(struct Skypoint),
    compare);
qsort (cornell_list,skypoints_read_Cornell,sizeof(struct Skypoint), compare
    );
```

It is only important to know the first point in each stripe of the cornell list, due to the same fact mentioned at the shadow points.

```
for (i=0; i<=Stripestheta; i++) {
  first_point_in_stripe[i]=Empty;
}

last_stripe=first_point_in_stripe[cornell_list[0].t_cell]=0;
for (i=0; i<skypoints_read_Cornell; i++) {
  if (cornell_list[i].t_cell != last_stripe) {
    last_stripe=cornell_list[i].t_cell;
    first_point_in_stripe[cornell_list[i].t_cell]=i;
  }
}
```

The *print_output()*-function gets easier, because the databases are stored separately in two linear lists. This function will only operate, if the data set gap is at least one year.
Attention: The first parameter has to be an element of the einstein list and the second parameter has to be an element of the cornell list!

```
void print_output(int i,int j,double g){

  char string1[Maximum_scan];
  char string2[Maximum_scan];

  strncpy(string1, einstein_list[i].id, Maximum_scan);
  strncpy(string2, cornell_list[j].id, Maximum_scan);

  if (fabs(einstein_list[i].frequency-cornell_list[j].frequency) <
      Minimum_frequency){

    /*
    This code was only needed for the file
        BachelorarbeitFurtherInvestigation.c
    This only operates if the two candidates originate from different data
        sets.

    char *ptr1;
    char *ptr2;
```

```
    ptr1 = strtok(string1,"b");
    ptr2 = strtok(string2,"b");

    int result = strcmp (ptr1,ptr2);
    if (result != 0) {
    */

    int result = abs(atoi(strtok(string1,"b"))-atoi(strtok(string2,"b")));
    if (result > 10000) {
      printf("%s %s %f %f %f %f %f %f %f %f\n", einstein_list[i].id,
          cornell_list[j].id,einstein_list[i].frequency,cornell_list[j].
          frequency, einstein_list[i].phi,einstein_list[i].theta,
          cornell_list[j].phi,cornell_list[j].theta, einstein_list[i].
          significance,g);
    }
  }
  return;
}
```

The main body of the code, where the coincidences between the two databases are detected, got some several improvements. First of all the target-loop only runs through the einstein list and the runner-loop only through the cornell list. Thus, there will be no coincidences between candidates of the same database.

The determination of the runner variables divides into three different parts. It is necessary to compare the target (einstein list) with the runners (cornell list) of the same strip (Part 1), the stripe above (Part 2) and the stripe below (Part 3). If the target stays in the cell of the previous target, the previously determined runner variables will be reused. After detecting the coincidences, the rest of the target-loop will be skipped.

```
runner_begin=runner_end=0;
runner_below_begin=runner_below_end=0;
runner_above_begin=runner_above_end=0;
check_point=check_point_below=check_point_above=Empty;

for (target=0; target<skypoints_read_Einstein; target++) {
  //not execute when the target loop start running for the first time
  if(target>0){
    //the target stays in the same cell (no update of the all the runner
        variables)
    if (einstein_list[target].p_cell==einstein_list[target-1].p_cell &&
        einstein_list[target].t_cell==einstein_list[target-1].t_cell) {
      if (check_point!=Empty){
        for (runner=runner_begin; runner<=runner_end; runner++) {
          gamma=angle(&einstein_list[target],&cornell_list[runner]);
          print_output(target,runner,gamma);
        }
      }
      if (check_point_above!=Empty){
        for (runner=runner_above_begin; runner<=runner_above_end; runner++)
            {
          gamma=angle(&einstein_list[target],&cornell_list[runner]);
          print_output(target,runner,gamma);
```

```
        }
      }
      if (check_point_below!=Empty){
        for (runner=runner_below_begin; runner<=runner_below_end; runner++)
            {
          gamma=angle(&einstein_list[target],&cornell_list[runner]);
          print_output(target,runner,gamma);
        }
      }
      continue;
    }
  }

  //Part 1 (Same stripe)

  //let's make sure that the stripe in the cornell list isn't empty
  if (first_point_in_stripe[einstein_list[target].t_cell] != Empty) {

    for (j=first_point_in_stripe[einstein_list[target].t_cell];
        einstein_list[target].t_cell == cornell_list[j].t_cell; j++) {
      if ((einstein_list[target].p_cell==((cornell_list[j].p_cell)+1) &&
          einstein_list[target].t_cell==cornell_list[j].t_cell) || (
          einstein_list[target].p_cell==cornell_list[j].p_cell &&
          einstein_list[target].t_cell==cornell_list[j].t_cell) || (
          einstein_list[target].p_cell==((cornell_list[j].p_cell)-1) &&
          einstein_list[target].t_cell==cornell_list[j].t_cell)) {
        runner_begin = j;
        check_point =j;
        break;
      } else {
        check_point=Empty;
      }
    }

    if(check_point!=Empty){
      for (runner=runner_begin; einstein_list[target].t_cell ==
          cornell_list[runner].t_cell; runner++) {

        if ((einstein_list[target].p_cell==((cornell_list[runner].p_cell)
            +1) || einstein_list[target].p_cell==cornell_list[runner].
            p_cell || einstein_list[target].p_cell==((cornell_list[runner].
            p_cell)-1)) && einstein_list[target].t_cell==cornell_list[
            runner].t_cell) {
          //runner is in the same or neighboring cells
          gamma=angle(&einstein_list[target],&cornell_list[runner]);
          print_output(target,runner,gamma);
          runner_end=runner;
        } else break;
      }
    }
  }

  //Part 2 (Above stripe)
```

```
  //let's make sure we aren't at the first (highest) stripe
  if (einstein_list[target].t_cell != 0){

    //let's make sure that the stripe above in the cornell list isn't empty
    if (first_point_in_stripe[(einstein_list[target].t_cell)-1] != Empty) {

      for (j=first_point_in_stripe[((einstein_list[target].t_cell)-1)]; ((
          einstein_list[target].t_cell)-1) == cornell_list[j].t_cell; j++)
          {
        //Attention: The parameters of the close_enough()-function are
            switched because the cornell-stripe lays above the einstein-
            stripe.
        if (close_enough(&cornell_list[j],&einstein_list[target])) {
          runner_above_begin = j;
          check_point_above=j;
          break;
        } else {
          check_point_above=Empty;
        }
      }

      if(check_point_above!=Empty){
        for (runner=runner_above_begin; ((einstein_list[target].t_cell)-1)
            == cornell_list[runner].t_cell; runner++) {
          //Attention: The parameters of the close_enough()-function are
              switched because the cornell-stripe lays above the einstein-
              stripe.
          if (close_enough(&cornell_list[runner],&einstein_list[target])){
            //runner is in a surrounding cell
            gamma=angle(&einstein_list[target],&cornell_list[runner]);
            print_output(target,runner,gamma);
            runner_above_end=runner;
          } else break;
        }
      }
    }
  }

  //Part 3 (Below stripe)

  //let's make sure that the stripe below in the cornell list isn't empty
  if (first_point_in_stripe[(einstein_list[target].t_cell)+1] != Empty) {

    for (j=first_point_in_stripe[((einstein_list[target].t_cell)+1)]; ((
        einstein_list[target].t_cell)+1) == cornell_list[j].t_cell; j++) {
      if (close_enough(&einstein_list[target],&cornell_list[j])) {
        runner_below_begin = j;
        check_point_below=j;
        break;
      } else {
        check_point_below=Empty;
      }
    }
```

```
    if(check_point_below!=Empty){
      for (runner=runner_below_begin; ((einstein_list[target].t_cell)+1) ==
           cornell_list[runner].t_cell; runner++) {

        if (close_enough(&einstein_list[target],&cornell_list[runner])){
          //runner is in a surrounding cell
          gamma=angle(&einstein_list[target],&cornell_list[runner]);
          print_output(target,runner,gamma);
          runner_below_end=runner;
        } else break;
      }
    }
  }
}
```

2.2 Comparsions of the results

It is important to show that the new output contains at least the coincidences of the old output. Therefore two different files are compiled and executed, as shown in figure 2. In the first file, the two candidates, which require the coincidence condition, originate from different data sets. In the second file, the data set gap of the two candidates is at least one year.

```
dyn-2cfc:FurtherInvestigation Tjark$ gcc -Wall -Wextra -pedantic-errors BachelorarbeitFurtherInvestigation.c
 -o BachelorarbeitFurtherInvestigation
dyn-2cfc:FurtherInvestigation Tjark$ time /Users/Tjark/Desktop/ComputerTjark/Studium/FurtherInvestigation/Ba
chelorarbeitFurtherInvestigation Input_Einstein\@home_Cornell.txt > New_Output_Einstein\@home_Cornell.txt

real    14m11.605s
user    13m23.532s
sys     0m4.806s
dyn-2cfc:FurtherInvestigation Tjark$ gcc -Wall -Wextra -pedantic-errors BachelorarbeitFurtherInvestigation_d
ifferentYear.c -o BachelorarbeitFurtherInvestigation_differentYear
dyn-2cfc:FurtherInvestigation Tjark$ time /Users/Tjark/Desktop/ComputerTjark/Studium/FurtherInvestigation/Ba
chelorarbeitFurtherInvestigation_differentYear Input_Einstein\@home_Cornell.txt > New_Output_Einstein\@home_
Cornell_differentYear.txt

real    13m45.216s
user    12m46.918s
sys     0m5.384s
```

Figure 2: The execution of the files only takes roughly 15 min (MacBook Pro, 2,66 GHz Intel Core 2 Duo, 4 GB 1067 MHz DDR3).

The next step is to compare the two outputs by UNIX commands, as shown in Figure 3. The idea is to add the two files and then use the *uniq* command. The line number of this result must be equal to the line number of the old output.

```
dyn-2cfc:FurtherInvestigation Tjark$ cat New_Output_Einstein\@home_Cornell.txt Old_Output_Einstein\@home_Cor
nell.txt > Output.txt
dyn-2cfc:FurtherInvestigation Tjark$ cat New_Output_Einstein\@home_Cornell_differentYear.txt Old_Output_Eins
tein\@home_Cornell_differentYear.txt > Output_differentYear.txt
dyn-2cfc:FurtherInvestigation Tjark$ sort Output.txt | uniq -d > OutputUnique.txt
dyn-2cfc:FurtherInvestigation Tjark$ sort Output_differentYear.txt | uniq -d > Output_differentYearUnique.tx
t
dyn-2cfc:FurtherInvestigation Tjark$ wc -l Old_Output_Einstein\@home_Cornell.txt
   35384 Old_Output_Einstein@home_Cornell.txt
dyn-2cfc:FurtherInvestigation Tjark$ wc -l OutputUnique.txt
   35384 OutputUnique.txt
dyn-2cfc:FurtherInvestigation Tjark$ wc -l Old_Output_Einstein\@home_Cornell_differentYear.txt
    6163 Old_Output_Einstein@home_Cornell_differentYear.txt
dyn-2cfc:FurtherInvestigation Tjark$ wc -l Output_differentYearUnique.txt
    6163 Output_differentYearUnique.txt
```

Figure 3: Comparing the new and old outputs with each other. After adding the different files with the *cat* command, the *sort* and *uniq -d* command is operated. The *uniq -d* command only prints duplicated lines. So this is why the line number of the results is equal to the line number of the old outputs.

Figure 3 is shown, that the new outputs contain at least the coincidences of the old outputs.

3 Further ideas

3.1 Rating/Judging the output by an algorithm

To be continued...

Appendix

The complete code from section 2:

```
//
//  BachelorarbeitFurtherInvestigation_differentYear.c
//
//  Created by Tjark Miener and Bruce Allen on 31.05.16.
//  Copyright (c) 2016 Tjark Miener and Bruce Allen. All rights reserved.
//

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <math.h>
#include <stdbool.h>
#include <string.h>

#define Allocate_blocksize 1024
#define Stripestheta 3001
#define Stripesphi (2*Stripestheta)
#define Gamma_max (2*M_PI/Stripesphi)
#define Empty -12345
#define End_of_first_list 1265589
#define Minimum_frequency 0.01
#define Maximum_scan 64

int num_in_stripe[Stripestheta];

/*The structure skypoint have two doubles for the position and two ints for
the cell the skypoint is in. Every skypoint has a number (int) and a
    frequence (double).
In addtion, the candidates from the Einstein@Home databases have a
    significance.*/

struct Skypoint{
    int number;
    char id[Maximum_scan];

    double phi;       //0,...,2*PI
    double theta;     //0,...,PI

    int p_cell;
    int t_cell;

    double frequency;
    float significance;
};
struct Skypoint *einstein_list=NULL;
struct Skypoint *cornell_list=NULL;

//function to compute the cells for each skypoint
```

```
void cellcomputing(struct Skypoint *a){

    if ((a->theta)<M_PI) {
        a->t_cell= (int)((a->theta)*Stripestheta/M_PI);
    } else {
        a->t_cell=Stripestheta-1;
    }
    a->p_cell=floor((a->phi)*(num_in_stripe[a->t_cell]-1)/(2.0*M_PI));
    return;
}

//calculate gamma of the skypoints

double angle(struct Skypoint *a,struct Skypoint *b) {
    double x = sin(a->theta)*sin(b->theta)*cos((a->phi)-(b->phi))+cos(a->
        theta)*cos(b->theta);
    if (x > 1.0) {
        x = 1.0;
    }
    return acos(x)*(180.0/M_PI);
}

//this function return if we need to check the candidates of the cells or
    not

int close_enough(struct Skypoint *a, struct Skypoint *b){

    struct Skypoint targetcell_left_bottom;
    struct Skypoint targetcell_right_bottom;
    struct Skypoint runnercell_left_top;
    struct Skypoint runnercell_right_top;

    int p1=a->p_cell;
    int nextp1=(a->p_cell)+1;
    int nextt1=(a->t_cell)+1;

    int p2=b->p_cell;
    int t2=b->t_cell;
    int nextp2=(b->p_cell)+1;

    targetcell_right_bottom.theta=targetcell_left_bottom.theta=nextt1*M_PI/
        Stripestheta;
    targetcell_left_bottom.phi=2*M_PI*p1/num_in_stripe[a->t_cell];
    targetcell_right_bottom.phi=2*M_PI*nextp1/num_in_stripe[a->t_cell];

    runnercell_right_top.theta=runnercell_left_top.theta=t2*M_PI/
        Stripestheta;
    runnercell_left_top.phi=2*M_PI*p2/num_in_stripe[b->t_cell];
    runnercell_right_top.phi=2*M_PI*nextp2/num_in_stripe[b->t_cell];

    double gamma_1=angle(&targetcell_left_bottom,&runnercell_left_top);
    double gamma_2=angle(&targetcell_left_bottom,&runnercell_right_top);
    double gamma_3=angle(&targetcell_right_bottom,&runnercell_left_top);
    double gamma_4=angle(&targetcell_right_bottom,&runnercell_right_top);
```

```
    if (gamma_1<Gamma_max || gamma_2<Gamma_max || gamma_3<Gamma_max ||
        gamma_4<Gamma_max) {
        return 1;
    } else {
        return 0;
    }
}

//dynamic allocation functions

void my_Allocate_Einstein(int a) {
    einstein_list=realloc(einstein_list,a*sizeof(struct Skypoint));
    if (!einstein_list) {
        printf("Error! No more RAM available, skypoints_max = %d!\n", a);
        exit(1);
    }
}

void my_Allocate_Cornell(int a) {
    cornell_list=realloc(cornell_list,a*sizeof(struct Skypoint));
    if (!cornell_list) {
        printf("Error! No more RAM available, skypoints_max = %d!\n", a);
        exit(2);
    }
}

//comparefunktion for the qsort() of <stdlib.h>

int compare(const void *a, const void *b){
    const struct Skypoint *elementa = a;
    const struct Skypoint *elementb = b;

    if (elementa->t_cell > elementb->t_cell) return 1;
    if (elementa->t_cell < elementb->t_cell) return -1;

    if (elementa->p_cell > elementb->p_cell) return 1;
    if (elementa->p_cell < elementb->p_cell) return -1;

    return 0;
}

//print out function

void print_output(int i,int j,double g){

    char string1[Maximum_scan];
    char string2[Maximum_scan];

    strncpy(string1, einstein_list[i].id, Maximum_scan);
    strncpy(string2, cornell_list[j].id, Maximum_scan);

    if (fabs(einstein_list[i].frequency-cornell_list[j].frequency) <
        Minimum_frequency){
```

```
        /*
        This code was only needed for the file
            BachelorarbeitFurtherInvestigation.c
        This only operates if the two candidates originate from different
            data sets.

        char *ptr1;
        char *ptr2;

        ptr1 = strtok(string1,"b");
        ptr2 = strtok(string2,"b");

        int result = strcmp (ptr1,ptr2);
        if (result != 0) {
        */

        int result = abs(atoi(strtok(string1,"b"))-atoi(strtok(string2,"b")
            ));
        if (result > 10000) {
            printf("%s %s %f %f %f %f %f %f %f %f\n",
                einstein_list[i].id,cornell_list[j].id,einstein_list[i].
                    frequency,cornell_list[j].frequency,
                einstein_list[i].phi,einstein_list[i].theta,cornell_list[j
                    ].phi,cornell_list[j].theta,
                einstein_list[i].significance,g);
        }
    }
    return;
}

//main function

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

    int i,j,k;

    int skypoints_read_Einstein;
    int skypoints_read_Cornell;

    int skypoints_allocated_Einstein, skypoints_allocated_Cornell;

    int target;
    int runner, runner_begin, runner_end;
    int runner_below_begin,runner_below_end;
    int runner_above_begin,runner_above_end;
    int check_point,check_point_below,check_point_above;
    int first_point_in_stripe[Stripestheta+1];
    int last_stripe;

    double gamma;

    //check if Stripestheta is odd
    if (Stripestheta%2==0) {
```

```
        printf("Error! Stripestheta = %d is not odd!\n", Stripestheta);
        exit(3);
    }

    //initialisation of num_in_stripe[]

    for (i=0; i<Stripestheta; i++) {
        num_in_stripe[i]=ceil(Stripesphi*sin(M_PI*(i+0.5)/Stripestheta));
    }

    //check if arguments in the command line is right

    if (argc!=2) {
        printf("Error! Missing command line arguments, only found %d not 1\
            n", argc-1);
        exit(4);
    }

    //open the .txt file

    FILE *input_candidate = fopen(argv[1],"r");

    if (!input_candidate) {
        printf("Error! File %s not found\n", argv[1]);
        exit(5);
    }


    skypoints_read_Einstein=0;
    skypoints_read_Cornell=0;
    skypoints_allocated_Einstein=0;
    skypoints_allocated_Cornell=0;

    while (true) {

        if (skypoints_read_Einstein < End_of_first_list) {

            //reallocating memory if necessary
            if (skypoints_read_Einstein==skypoints_allocated_Einstein) {
                skypoints_allocated_Einstein += Allocate_blocksize;
                my_Allocate_Einstein(skypoints_allocated_Einstein);
            }

            //scan the elements of the first part of the file (Einstein
                database)
            if (6==(i=fscanf(input_candidate, "%d %s %lf %lf %lf %f",&
                einstein_list[skypoints_read_Einstein].number,einstein_list
                [skypoints_read_Einstein].id,&einstein_list[
                skypoints_read_Einstein].phi,&einstein_list[
                skypoints_read_Einstein].theta,&einstein_list[
                skypoints_read_Einstein].frequency,&einstein_list[
                skypoints_read_Einstein].significance))) {
```

```
            einstein_list[skypoints_read_Einstein].phi=(einstein_list[
                skypoints_read_Einstein].phi*15.0)*2.0*M_PI/360.0;
            einstein_list[skypoints_read_Einstein].theta=(90.0-
                einstein_list[skypoints_read_Einstein].theta)*2.0*M_PI
                /360.0;

            //check the data ranges, exit if error
            if (einstein_list[skypoints_read_Einstein].phi < 0 ||
                einstein_list[skypoints_read_Einstein].phi > 2*M_PI ||
                einstein_list[skypoints_read_Einstein].theta < 0 ||
                einstein_list[skypoints_read_Einstein].theta > M_PI) {
                printf("Error! Problem reading %s at line %d. Point (%f
                    ,%f) isn't in the data range!\n", argv[1],
                    skypoints_read_Einstein, einstein_list[
                    skypoints_read_Einstein].phi, einstein_list[
                    skypoints_read_Einstein].theta);
                exit(6);
            }

            cellcomputing(einstein_list+skypoints_read_Einstein);
            skypoints_read_Einstein++;

        } else break;

    } else {

        //reallocating memory if necessary
        if (skypoints_read_Cornell==skypoints_allocated_Cornell) {
            skypoints_allocated_Cornell += Allocate_blocksize;
            my_Allocate_Cornell(skypoints_allocated_Cornell);
        }

        //scan the elements of the secound part of the file (Cornell
            database)
        if (5==(i=fscanf(input_candidate, "%d %s %lf %lf %lf",&
            cornell_list[skypoints_read_Cornell].number,cornell_list[
            skypoints_read_Cornell].id,&cornell_list[
            skypoints_read_Cornell].phi,&cornell_list[
            skypoints_read_Cornell].theta,&cornell_list[
            skypoints_read_Cornell].frequency))) {

            cornell_list[skypoints_read_Cornell].phi=cornell_list[
                skypoints_read_Cornell].phi*2.0*M_PI/360.0;
            cornell_list[skypoints_read_Cornell].theta=(90.0-
                cornell_list[skypoints_read_Cornell].theta)*2.0*M_PI
                /360.0;
            cornell_list[skypoints_read_Cornell].significance=0.0;

            //check the data ranges, exit if error
            if (cornell_list[skypoints_read_Cornell].phi < 0 ||
                cornell_list[skypoints_read_Cornell].phi > 2*M_PI ||
                cornell_list[skypoints_read_Cornell].theta < 0 ||
                cornell_list[skypoints_read_Cornell].theta > M_PI) {
```

```
                    printf("Error! Problem reading %s at line %d. Point (%f
                        ,%f) isn't in the data range!\n", argv[1],
                        skypoints_read_Cornell, cornell_list[
                        skypoints_read_Cornell].phi, cornell_list[
                        skypoints_read_Cornell].theta);
                    exit(7);
                }

                cellcomputing(cornell_list+skypoints_read_Cornell);
                skypoints_read_Cornell++;

            } else break;

        }

    }

    if (i!=EOF) {
        if (skypoints_read_Cornell==0) {
            fprintf(stderr, "Error! Problem reading %s at line %d (number:%
                d). fscanf() returned %d\n", argv[1],
                skypoints_read_Einstein, einstein_list[
                skypoints_read_Einstein].number,i);
            exit(8);
        } else {
            fprintf(stderr, "Error! Problem reading %s at line %d (number:%
                d). fscanf() returned %d\n", argv[1],
                skypoints_read_Einstein+skypoints_read_Cornell,
                cornell_list[skypoints_read_Cornell].number,i);
            exit(9);
        }
    }


    //adding shadow points to the Cornell database

    k=j=0;
    for (i=0; i<skypoints_read_Cornell; i++) {

        //reallocating memory if necessary
        if ((skypoints_read_Cornell+k)==skypoints_allocated_Cornell && j
            ==1) {
            skypoints_allocated_Cornell += Allocate_blocksize;
            my_Allocate_Cornell(skypoints_allocated_Cornell);
        }

        //adding shadow points on the right
        if (cornell_list[i].p_cell < 2){

            cornell_list[skypoints_read_Cornell+k].phi=((cornell_list[i].
                phi)+2.0*M_PI);
            cornell_list[skypoints_read_Cornell+k].theta=cornell_list[i].
                theta;
```

```
            cornell_list[skypoints_read_Cornell+k].t_cell=cornell_list[i].
                t_cell;
            cornell_list[skypoints_read_Cornell+k].p_cell=cornell_list[i].
                p_cell+num_in_stripe[cornell_list[i].t_cell];
            cornell_list[skypoints_read_Cornell+k].significance=0.0;
            cornell_list[skypoints_read_Cornell+k].frequency=cornell_list[i
                ].frequency;
            cornell_list[skypoints_read_Cornell+k].number=cornell_list[i].
                number;
            strncpy(cornell_list[skypoints_read_Cornell+k].id, cornell_list
                [i].id, Maximum_scan);

            k++;
            j=1;

        //adding shadow points on the left
        } else if (cornell_list[i].p_cell > num_in_stripe[cornell_list[i].
            t_cell]-3) {

            cornell_list[skypoints_read_Cornell+k].phi=((cornell_list[i].
                phi)-2.0*M_PI);
            cornell_list[skypoints_read_Cornell+k].theta=cornell_list[i].
                theta;
            cornell_list[skypoints_read_Cornell+k].t_cell=cornell_list[i].
                t_cell;
            cornell_list[skypoints_read_Cornell+k].p_cell=cornell_list[i].
                p_cell-num_in_stripe[cornell_list[i].t_cell];
            cornell_list[skypoints_read_Cornell+k].significance=0.0;
            cornell_list[skypoints_read_Cornell+k].frequency=cornell_list[i
                ].frequency;
            cornell_list[skypoints_read_Cornell+k].number=cornell_list[i].
                number;
            strncpy(cornell_list[skypoints_read_Cornell+k].id, cornell_list
                [i].id, Maximum_scan);

            k++;
            j=1;
        } else {
            j=0;
        }

    }

    skypoints_read_Cornell += k;

    //sorting the two lists

    qsort (einstein_list,skypoints_read_Einstein,sizeof(struct Skypoint),
        compare);
    qsort (cornell_list,skypoints_read_Cornell,sizeof(struct Skypoint),
        compare);


    //find the first point in the stripe of the Cornell database
```

```
    for (i=0; i<=Stripestheta; i++) {
        first_point_in_stripe[i]=Empty;
    }

    last_stripe=first_point_in_stripe[cornell_list[0].t_cell]=0;
    for (i=0; i<skypoints_read_Cornell; i++) {
        if (cornell_list[i].t_cell != last_stripe) {
            last_stripe=cornell_list[i].t_cell;
            first_point_in_stripe[cornell_list[i].t_cell]=i;
        }
    }


    //find coincidences between the two databeses

    runner_begin=runner_end=0;
    runner_below_begin=runner_below_end=0;
    runner_above_begin=runner_above_end=0;
    check_point=check_point_below=check_point_above=Empty;

    for (target=0; target<skypoints_read_Einstein; target++) {
        //not execute when the target loop start running for the first time
        if(target>0){
            //the target stays in the same cell (no update of the all the
                runner variables)
            if (einstein_list[target].p_cell==einstein_list[target-1].
                p_cell && einstein_list[target].t_cell==einstein_list[
                target-1].t_cell) {
                if (check_point!=Empty){
                    for (runner=runner_begin; runner<=runner_end; runner++)
                        {
                        gamma=angle(&einstein_list[target],&cornell_list[
                            runner]);
                        print_output(target,runner,gamma);
                    }

                }
                if (check_point_above!=Empty){
                    for (runner=runner_above_begin; runner<=
                        runner_above_end; runner++) {
                        gamma=angle(&einstein_list[target],&cornell_list[
                            runner]);
                        print_output(target,runner,gamma);
                    }

                }
                if (check_point_below!=Empty){
                    for (runner=runner_below_begin; runner<=
                        runner_below_end; runner++) {
                        gamma=angle(&einstein_list[target],&cornell_list[
                            runner]);
                        print_output(target,runner,gamma);
                    }
```

```
            }
            continue;
        }
    }

    //Part 1 (Same stripe)

    //let's make sure that the stripe in the cornell list isn't empty
    if (first_point_in_stripe[einstein_list[target].t_cell] != Empty) {

        for (j=first_point_in_stripe[einstein_list[target].t_cell];
            einstein_list[target].t_cell == cornell_list[j].t_cell; j
            ++) {
            if ((einstein_list[target].p_cell==((cornell_list[j].p_cell
                )+1) && einstein_list[target].t_cell==cornell_list[j].
                t_cell) || (einstein_list[target].p_cell==cornell_list[
                j].p_cell && einstein_list[target].t_cell==cornell_list
                [j].t_cell) || (einstein_list[target].p_cell==((
                cornell_list[j].p_cell)-1) && einstein_list[target].
                t_cell==cornell_list[j].t_cell)) {
                runner_begin = j;
                check_point =j;
                break;
            } else {
                check_point=Empty;
            }
        }

        if(check_point!=Empty){
            for (runner=runner_begin; einstein_list[target].t_cell ==
                cornell_list[runner].t_cell; runner++) {

                if ((einstein_list[target].p_cell==((cornell_list[
                    runner].p_cell)+1) || einstein_list[target].p_cell
                    ==cornell_list[runner].p_cell || einstein_list[
                    target].p_cell==((cornell_list[runner].p_cell)-1))
                    && einstein_list[target].t_cell==cornell_list[
                    runner].t_cell) {
                    //runner is in the same or neighboring cells
                    gamma=angle(&einstein_list[target],&cornell_list[
                        runner]);
                    print_output(target,runner,gamma);
                    runner_end=runner;
                } else break;
            }
        }
    }

    //Part 2 (Above stripe)

    //let's make sure we aren't at the first (highest) stripe
    if (einstein_list[target].t_cell != 0){
```

```
        //let's make sure that the stripe above in the cornell list isn
            't empty
        if (first_point_in_stripe[(einstein_list[target].t_cell)-1] !=
            Empty) {

            for (j=first_point_in_stripe[((einstein_list[target].t_cell
                )-1)]; ((einstein_list[target].t_cell)-1) ==
                cornell_list[j].t_cell; j++) {
                //Attention: The parameters of the close_enough()-
                    function are switched because the cornell-stripe
                    lays above the einstein-stripe.
                if (close_enough(&cornell_list[j],&einstein_list[target
                    ])) {
                    runner_above_begin = j;
                    check_point_above=j;
                    break;
                } else {
                    check_point_above=Empty;
                }
            }

            if(check_point_above!=Empty){
                for (runner=runner_above_begin; ((einstein_list[target
                    ].t_cell)-1) == cornell_list[runner].t_cell; runner
                    ++) {
                    //Attention: The parameters of the close_enough()-
                        function are switched because the cornell-
                        stripe lays above the einstein-stripe.
                    if (close_enough(&cornell_list[runner],&
                        einstein_list[target])){
                        //runner is in a surrounding cell
                        gamma=angle(&einstein_list[target],&
                            cornell_list[runner]);
                        print_output(target,runner,gamma);
                        runner_above_end=runner;
                    } else break;
                }
            }
            }
    }

    //Part 3 (Below stripe)

    //let's make sure that the stripe below in the cornell list isn't
        empty
    if (first_point_in_stripe[(einstein_list[target].t_cell)+1] !=
        Empty) {

        for (j=first_point_in_stripe[((einstein_list[target].t_cell)+1)
            ]; ((einstein_list[target].t_cell)+1) == cornell_list[j].
            t_cell; j++) {
            if (close_enough(&einstein_list[target],&cornell_list[j]))
                {
                runner_below_begin = j;
```

```
                    check_point_below=j;
                    break;
                } else {
                    check_point_below=Empty;
                }
            }

            if(check_point_below!=Empty){
                for (runner=runner_below_begin; ((einstein_list[target].
                    t_cell)+1) == cornell_list[runner].t_cell; runner++) {

                    if (close_enough(&einstein_list[target],&cornell_list[
                        runner])){
                        //runner is in a surrounding cell
                        gamma=angle(&einstein_list[target],&cornell_list[
                            runner]);
                        print_output(target,runner,gamma);
                        runner_below_end=runner;
                    } else break;
                }
            }
        }
    }


    free(einstein_list);
    free(cornell_list);
    fclose(input_candidate);

    return 0;
}
```