Search for new radio pulsars via coincidence in the Einstein@Home and Cornell result databases

Tjark Miener

04.12.2014

Contents

Contents

Contents

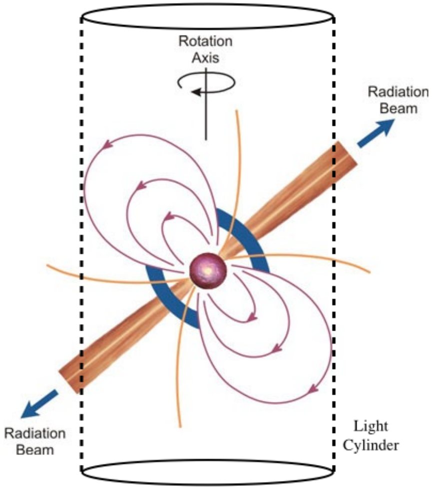
Basic facts about neutron stars and radio pulsars

- **Radio pulsars** are rapidly rotating, highly magnetized neutron stars.
- The remains of a supernova evolve to three different objects, depending on the core mass m_{core} of the original star (solar mass $M_{\odot} := 1.989 \cdot 10^{30}\,\mathrm{kg}$):
  - white dwarf ($m_{core} < 1.5 M_{\odot}$)
  - **neutron star** ($1.5 M_{\odot} < m_{core} < 3 M_{\odot}$)
  - black hole ($3 M_{\odot} < m_{core}$)

Basic facts about neutron stars and radio pulsars

- About 2400 pulsars have been found after the first pulsar (B1919+21) discovered in 1967 by J. Bell and A. Hewish.
- The Hulse-Taylor binary pulsar (B1913+16) discovered in 1974. Analysis of this pulsar system provides indirect evidence for gravitational waves according to the general theory of relativity.

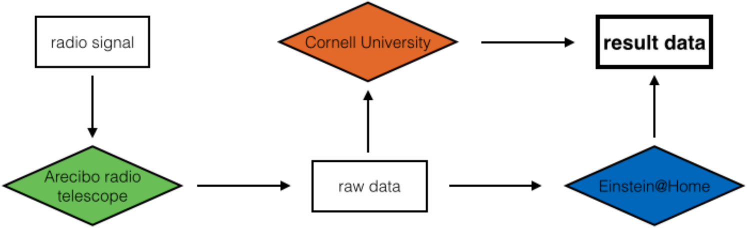
Radio pulsars



Arecibo radio telescope

- Puerto Rico
- 305 m radio telescope (world's largest single-aperture telescope)

Processing the data



Einstein@Home

- Volunteer distributed computing project launched at the American Association for the Advancement of Science meeting on 2005 February 19.
- Volunteers from 193 countries allocate idle time from their computers.
- Raw data from the LIGO gravitational-wave detectors, the Arecibo radio telescope and the Fermi gamma-ray satellite is processed.
- Further information: *The Einstein@Home search for radio pulsars and PSR J2007+2722 Discovery (B. Allen 2013)*

Data structures

The **Cornell** database (five columns) is merged with the **Einstein@Home** database (six columns):

1. Candidate numeration
2. Candidate ID
3. Right ascension in J2000 epoch
4. Declination in J2000 epoch
5. Angular frequency
6. Significance

Example of the data

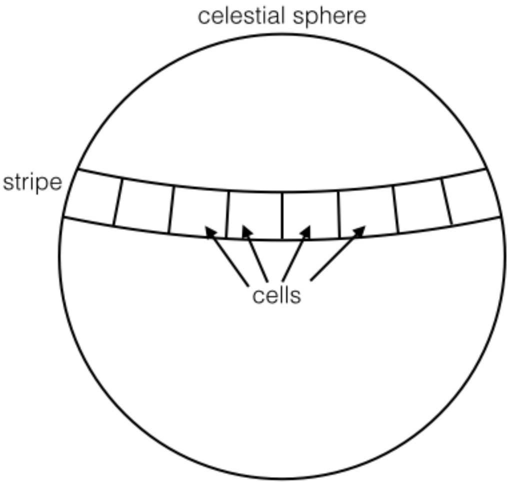
Einstein@Home candidate:

Nr	ID	RAJ	DecJ	f	s
-1	20090317.G35.41-02.95.N.b0s0g0.00000_14944	19.106888888888875	0.917583333333335	59.8496523770 42	24.2328

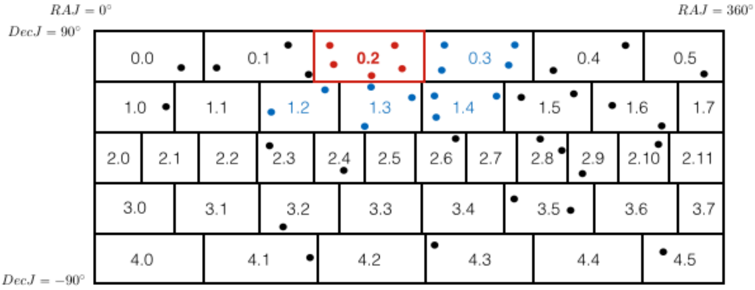
Cornell candidate:

Nr	ID	RAJ	DecJ	f	s
1	20110112.G193.10-03.16.N.b3.00000_1	90.56429833	16.03619522	0.837937759123	

General approach



Example of comparing candidates



Explanation of the code

The code is created in the programming language **C**. Instead of showing the whole code, only important parts of the code are explained:

- Six C standard libraries are used:

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

- Gobal running indices:

```c
int i,j,k,l,m;
```

Explanation of the code

- A structure called **Skypoint** is utilized:

```
#define Maximum_scan 64

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

  double phi;
  double theta;

  int p_cell;
  int t_cell;

  double frequency;
  float significance;
};
```

Explanation of the code

- In the following the number of cells in each stripe is calculated:

```
#define Stripestheta 3001
#define Stripesphi (2*Stripestheta)

int num_in_stripe[Stripestheta];

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

Explanation of the code

- The cells are computed, using the function *cellcomputing()*, which uses *num_in_stripe[]*:

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

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

Explanation of the code

- Open the file with the data:

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

if (!input_candidate) {
  printf("Error! File %s not found\n", argv[1]);
  exit(4);
}
```

- Dynamic memory allocation:

```
struct Skypoint *list=NULL;

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

Explanation of the code

- The inputfile is inserted in the structure Skypoint:

```
#define Allocate_blocksize 1024
#define End_of_first_list 1265589

int skypoints_read=0;
int skypoints_allocated=0;

while (true) {

  //reallocating memory if necessary
  if (skypoints_read==skypoints_allocated) {
    skypoints_allocated += Allocate_blocksize;
    my_Allocate(skypoints_allocated);
  }
```

Explanation of the code

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

Explanation of the code

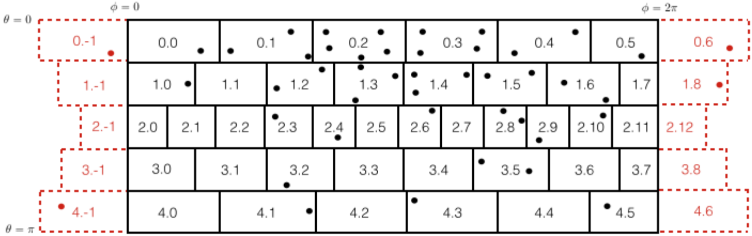
```
list[skypoints_read].phi=(list[
    skypoints_read].phi*15.0)*2.0*M_PI
    /360.0;
list[skypoints_read].theta=(90.0-list[
    skypoints_read].theta)*2.0*M_PI/360.0;

//check the data ranges, exit if error
if (list[skypoints_read].phi < 0 ||
     list[skypoints_read].phi > 2*M_PI ||
     list[skypoints_read].theta < 0 ||
     list[skypoints_read].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,
       list[skypoints_read].phi, list[
       skypoints_read].theta);
   exit(5);
}
```

Explanation of the code

```
            cellcomputing(list+skypoints_read);
            skypoints_read++;

            } else break;

        } else {
        //scan the elements of the second part of the
            file
        ...
} //closing of the while-loop

if (i!=EOF) {
   fprintf(stderr, "Error! Problem reading %s at line
        %d (number:%d). fscanf() returned %d\n", argv
        [1], skypoints_read, list[skypoints_read].
        number, i);
   exit(6);
}
```

Explanation of the code

- The procedure of creating shadow points:



Explanation of the code

- The list is sorted by the function *qsort()* of <stdlib.h>, which utilizes a quicksort algorithm:

```
qsort (list ,skypoints_read ,sizeof(struct
    Skypoint), compare);
```

Explanation of the code

- The fourth parameter in *qsort()* is a function, which orders two elements of the list:

```
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;
}
```

Explanation of the code

- The distance between two points on the celestial sphere ($r = 1$):

$$\cos\gamma = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} \sin\theta_1 \cos\phi_1 \\ \sin\theta_1 \sin\phi_1 \\ \cos\theta_1 \end{pmatrix} \cdot \begin{pmatrix} \sin\theta_2 \cos\phi_2 \\ \sin\theta_2 \sin\phi_2 \\ \cos\theta_2 \end{pmatrix}$$

$$\gamma = \arccos(\sin\theta_1 \cdot \sin\theta_2 \cdot \cos(\phi_1 - \phi_2) + \cos\theta_1 \cdot \cos\theta_2)$$

Explanation of the code

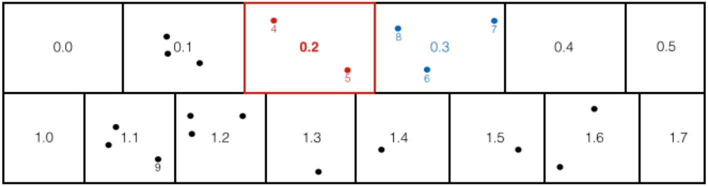
```
double angle(struct Skypoint *a,
    struct Skypoint *b) {
  double x = sin(a->theta)*sin(b->theta)*cos(
    (a->phi)-(b->phi))+cos(a->theta)*cos(b->theta);
  if (x > 1.0) {
      x = 1.0;
  }
  return acos(x)*(180.0/M_PI);
}
```

Explanation of the code

- The output (*Output_Einstein@home_Cornell.txt*) is created with the function *print_output()* and possesses ten columns with the following characterizing information:
  1. CandidateID (Einstein@home)
  2. CandidateID (Cornell)
  3. Frequency in *Hz* (Einstein@home)
  4. Frequency in *Hz* (Cornell)
  5. Azimuthal angle ϕ in *rad* (Einstein@home)
  6. Polar angle θ in *rad* (Einstein@home)
  7. Azimuthal angle ϕ in *rad* (Cornell)
  8. Polar angle θ in *rad* (Cornell)
  9. Significance (Einstein@home)
  10. Distance γ in *degrees* between both candidates

Explanation of the code

- The comparison between candidates of the same stripe:



Explanation of the code

```
int target, runner;
double gamma;

for (target=0; target<skypoints_read; target++) {

  //only take the non shadow point
  if (list[target].phi>=0 && list[target].phi<=2.0*
      M_PI) {

    for (runner=target+1; runner<skypoints_read;
        runner++) {

      //target and runner are in the same cell
      if (list[target].p_cell==list[runner].p_cell
          && list[target].t_cell==list[runner].
          t_cell) {
        gamma=angle(&list[target],&list[runner]);
        print_output(target,runner,gamma);
```

Explanation of the code

```
        //runner are in the next cell
        } else if(list[target].p_cell==((list[runner].
            p_cell)-1) && list[target].t_cell==list[
            runner].t_cell) {
          gamma=angle(&list[target],&list[runner]);
          print_output(target,runner,gamma);

        //runner are too far away -> break
        } else break;
      }
    }
}
```

Explanation of the code

- The following notes are needed for the comparison with candidates of the stripe below:
  - The function *close_enough()*, which detects whether or not a comparison between the targetcell and the underlying runnercell is needed. The idea of the function is to look at the corners of those cells and to see if any two corner points of different cells are close to each other.
  - The initialization of the first target, which have to compare to a candidates of the stripe below.
  - The first point in each stripe:

Explanation of the code

```
#define Empty -12345

int first_point_in_stripe[Stripestheta+1];
int last_stripe;

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

last_stripe=first_point_in_stripe[list[0].t_cell]=0;
for (i=1; i<skypoints_read; i++) {
  if (list[i].t_cell != last_stripe) {
    last_stripe=list[i].t_cell;
    first_point_in_stripe[list[i].t_cell]=i;
  }
}
```

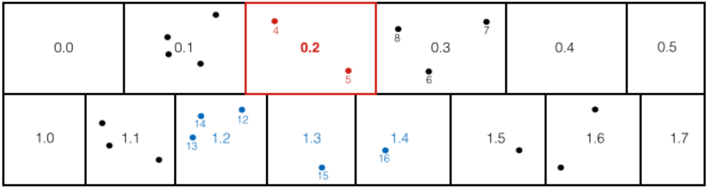
Explanation of the code

- The comparison with candidates of the stripe below:

```
for (target=i+1; target<skypoints_read; target
    ++) {

  //only take the non shadow point
  if (list[target].phi>=0 && list[target].phi
      <=2.0*M_PI) {

    //if the stripe below is empty, take the
        next point in the for loop and check
    if (first_point_in_stripe[((list[target].
        t_cell)+1)]==Empty) continue;
```

Explanation of the code

```
//Part 1
//the target stays in the same cell (no update
    of the runner_below_begin and
    runner_below_end)
```

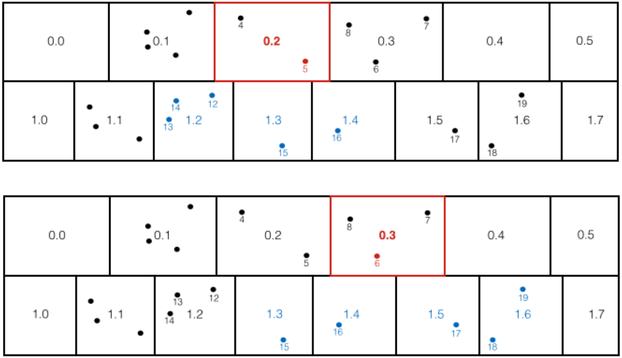


Explanation of the code

```
if (list[target].p_cell==list[target-1].p_cell
    && list[target].t_cell==list[target-1].
    t_cell) {
  //if no point in the stripe below is
      close_enough(), take the next point in the
       For-loop and check!
  if (check_point==Empty) continue;
  //compute gamma of the points in the cell
      below which are close_enough() to the
      target cell
  for (runner=runner_below_begin; runner<=
      runner_below_end; runner++) {
   gamma=angle(&list[target],&list[runner]);
   print_output(target,runner,gamma);
  }
```

Explanation of the code

```
//Part 2
//the target moves to a different cell (update
    of the runner_below_begin and
    runner_below_end)
```



Explanation of the code

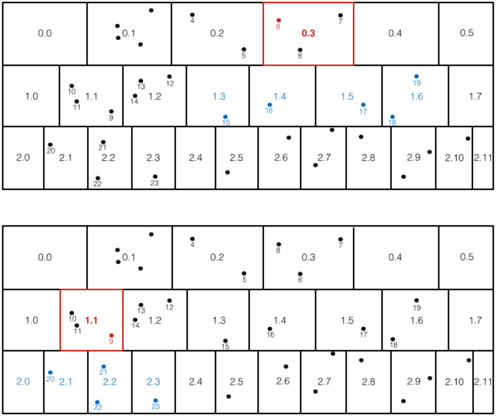
```
} else if (list[target].p_cell!=list[target-1].
      p_cell && list[target].t_cell==list[target
      -1].t_cell) {
   //check if any point in the stripe below is
         close_enough()
   for (j=first_point_in_stripe[((list[target].
         t_cell)+1)]; list[j].t_cell==list[target].
         t_cell+1; j++) {
      if (close_enough(&list[target],&list[j])) {
         runner_below_begin=j;
         check_point=j;
         break;
      } else check_point=Empty;
   }
```

Explanation of the code

```
//if no point in the stripe below is
    close_enough(), take the next point in the
    for loop and check!
if (check_point==Empty) continue;
//check how much points in the stripe below are
    close_enough()
for (k=runner_below_begin; list[k].t_cell==(list
    [target].t_cell+1); k++) {
  if (close_enough(&list[target],&list[k])) {
    runner_below_end=k;
  } else break;
}
//compute gamma of the points in the stripe
    below which are close_enough() to the target
     cell
for (runner=runner_below_begin; runner<=
    runner_below_end; runner++) {
  gamma=angle(&list[target],&list[runner]);
  print_output(target,runner,gamma);
}
```

Explanation of the code

```
//Part 3
//the target moves to a different stripe (update
    of the runner_below_begin and
    runner_below_end)
```



Explanation of the code

```
} else {
  //check if any point in the stripe below is
      close_enough()
  for (l=first_point_in_stripe[((list[target].
      t_cell)+1)]; list[l].t_cell==(list[target
      ].t_cell+1); l++) {
    if (close_enough(&list[target],&list[l])) {
      runner_below_begin=l;
      check_point=l;
      break;
    } else check_point=Empty;
  }
  //if no point in the stripe below is
      close_enough(), take the next point in the
       for loop and check!
  if (check_point==Empty) continue;
```

Explanation of the code

```
        //check how much points in the stripe below
            are close_enough()
        for (m=runner_below_begin; list[m].t_cell==
            list[target].t_cell+1; m++) {
          if (close_enough(&list[target],&list[m])) {
            runner_below_end=m;
          } else break;
        }
        //compute gamma of the points in the stripe
            below which are close_enough() to the
            target cell
        for (runner=runner_below_begin; runner<=
            runner_below_end; runner++) {
          gamma=angle(&list[target],&list[runner]);
          print_output(target,runner,gamma);
        }
      }
    }
}
```

Further processing of the output

Def.: "**Best candidates**" appear in a cluster of coincidences between candidates, whose data set's gap is at least one year, and the candidates come from different databases.

Identification of previously discovered pulsars

- Pulsar **J1935+2025** (F0=12.4815688643; RAJD=293.92475; DecJD=20.42781)
  Einstein@home-candidates:
  - 20090630.G56.12-00.09.N.b6s0g0.00000_3115 (F0=12.479955; RAJD=293.94924; DecJD=20.44774)
  - 20130522.G56.11-00.23.N.b2s0g0.00000_3115 (F0=12.484811; RAJD=294.01839; DecJD=20.47810)

  Cornell-candidates:
  - 20130522.G56.11-00.23.N.b2.00000_2841259 (F0=12.480759; RAJD=294.01902; DecJD=20.47805)
  - 20090630.G56.12-00.09.N.b6.00000_2143915 (F0=12.480714; RAJD=293.94918; DecJD=20.44768)

Identification of previously discovered pulsars

- Pulsar **B1855+02** (F0=2.40486905991; RAJD=284.43184; DecJD=2.21142)
- Pulsar **B1913+10** (F0=2.47189777002; RAJD=288.87493; DecJD=10.16213)
- Pulsar **B1914+13** (F0=3.5480820974; RAJD=289.24446; DecJD=13.21389)
- Pulsar **B1929+20** (F0=3.72831885206; RAJD=293.03343; DecJD=20.34623)

Abstract

- 24 clusters with "best" candidates were found
  - 5 clusters matches with previously discovered pulsars
  - 7 clusters are obviously RFI
  - 12 clusters remain after the complex filter process
- These 12 clusters were given to collaborators of Allen to have a closer look at the structure of their signal. The result is that all clusters are different types of RFIs.
- Since new pulsars haven't been discovered, this project seems to be a good start for further studies, which may follow.

Thank you for your attention.