# Contact

**Category:** Anomaly Review Bored

Billy Bob says he’s the best orbit designer there’s ever been. He designed an orbit with python skyfield that gets 230 minutes of contact on our ground station network. Can you beat him?

### Prompt

Reachable at: nc contact.quals2023-kah5Aiv9.satellitesabove.me 5300

\_|\_|\_| \_|\_| \_| \_| \_|\_|\_|\_|\_| \_|\_| \_|\_|\_| \_|\_|\_|\_|\_|  
 \_| \_| \_| \_|\_| \_| \_| \_| \_| \_| \_|  
 \_| \_| \_| \_| \_| \_| \_| \_|\_|\_|\_| \_| \_|  
 \_| \_| \_| \_| \_|\_| \_| \_| \_| \_| \_|  
 \_|\_|\_| \_|\_| \_| \_| \_| \_| \_| \_|\_|\_| \_|  
  
  
 Billy Bob says he's the best orbit designer there's ever been. He designed an orbit with python skyfield that gets 230 minutes of contact on our ground station network.  
 Can you beat him?  
  
 Ground stations are located across the United States at these WGS-84 coordinates:  
 Name Lat (deg) Long (deg) Alt (m)  
 Cape Canaveral 28.40 -80.61 27  
 Cape Cod 41.70 -70.03 9  
 Anchorage 61.21 -149.90 40  
 Vandenberg 34.76 -120.52 122  
 Denver 39.74 -104.98 1594  
  
 Contact is established at 15 degrees above the horizon and with one ground station at a time.  
 Our link budget supports a range of up to 6,000 km.  
 Between 1 Apr 2023 00:00:00.000 UTC and 1 Apr 2023 08:00:00.000 UTC, get more hours of contact than Billy Bob.  
  
 Good luck!  
  
  
Provide your TLE Line 2 parameters.  
Inclination (deg):  
RAAN (deg):  
Eccentricity (x10^-7):  
Argument of perigee (deg):  
Mean anomaly (deg):  
Mean motion (revs/day):

## Description

Positional data about 5 ground stations around the United Stated were given:

Name Lat (deg) Long (deg) Alt (m)  
Cape Canaveral 28.40 -80.61 27  
Cape Cod 41.70 -70.03 9  
Anchorage 61.21 -149.90 40  
Vandenberg 34.76 -120.52 122  
Denver 39.74 -104.98 1594

The challenge asked for six parameters (Inclination, RAAN, Eccentricity, Argument of perigee, Mean anomaly and Mean motion) to be used for a satellite orbiting around the Earth, such that the satellite would get >230 minutes of contact with the ground station network.

The challenge then gives more details about the scenario of the challenge.

* Contact is established at 15 degrees above the horizon and with one ground station at a time.
* The link budget supports a range of up to 6,000 km.
* The time of the orbit to be considered is between 1 Apr 2023 00:00:00.000 UTC and 1 Apr 2023 08:00:00.000 UTC

## Solution

### Initial idea

As a first idea, we thought of solving the challenge by understanding the problem and developing a well developed and justified solution.

Well, that didn’t happen: here is how we solved this task.

### First tries

When the challenge was released, some members of our team started sending to the remote connection random values or random patterns for the six parameters requested: some tries were worse and some better…

After a few tries, we came up with:

Inclination (deg): 45  
RAAN (deg): 45  
Eccentricity (x10^-7): 0  
Argument of perigee (deg): 45  
Mean anomaly (deg): 45  
Mean motion (revs/day): 15

That gave 61 minutes of contact time. Following with:

Inclination (deg): 12  
RAAN (deg): 10  
Eccentricity (x10^-7): 10  
Argument of perigee (deg): 10  
Mean anomaly (deg): 10  
Mean motion (revs/day): 10

Which gave a grand total of 105 minutes.

Having some great starting points, almost halfway through the value requested by the challenge, we started to write an **A\* search** using the values in minutes returned by the remote challenge as a heuristic.

The motivation behind the A\* algorithm was to try to slowly improve our best solutions by generating random variations and hoping to get better contact times.

The algorithm works by keeping a **heap** (a queue with the “best” element always at the top) with a lot of inputs to try. Every cycle, the top element of the heap is popped out and is sent to the remote server. After checking the result in minutes with the best current result, the script generates random variations of that input, and pushes the variations back into the **heap**, weighting them by the result received by the server.

The variation of the inputs is really basic, something like:

rnd = random.randint(1, 63)  
new\_tuple = [inclination, raan, eccentricity, arg\_perigee, mean\_anomaly, mean\_motion]  
for i, bit in enumerate(bin(rnd)[2:].zfill(6)):  
 new\_tuple[i] = new\_tuple[i] + int(bit) \* random.randint(-500, 500) \* DELTA

We first generate a “mask” to choose which elements to change, and then we add some random values to the elements chosen.

### Improving the results

We got lucky and, while still sending random stuff by hand, we got to:

Inclination (deg): 60  
RAAN (deg): 10  
Eccentricity (x10^-7): 10  
Argument of perigee (deg): 10  
Mean anomaly (deg): 30  
Mean motion (revs/day): 10

Which gave an impressive 171 minutes of contact time.

We ran the script starting with that input and (after adjusting the eccentricity by hand) slowly improved the results, getting timings like 178, …, 199, … and 211 with:

Inclination: 115.831300  
RAAN: 187.375700  
Eccentricity: 1290114.660000  
Arg Perigee: 230.202200  
Mean Anomaly: 159.298200  
Mean Motion: 9.654654

Improving from here was getting difficult, and we started writing a local simulator that would act similar to the remote server, to speed up the A\* search (in the end the simulator wasn’t always correct, so we left the scripts running).

Lowering the DELTA in the script, we got some better inputs, such as:

Inclination: 114.881300  
RAAN: 175.965700  
Eccentricity: 1290122.490000  
Arg Perigee: 236.002200  
Mean Anomaly: 171.928200  
Mean Motion: 9.404654

With 213 minutes of contact, and:

Provide your TLE Line 2 parameters.  
Inclination: 107.641800  
RAAN: 175.370600  
Eccentricity: 1290126.607700  
Arg Perigee: 234.187200  
Mean Anomaly: 170.385800  
Mean Motion: 8.307954

With 214 minutes of contact.

### Final

With a little help from the local tests, we got to the input:

Inclination: 117.791300  
RAAN: 180.615700  
Eccentricity: 2050128.000000  
Arg Perigee: 239.192200  
Mean Anomaly: 175.058200  
Mean Motion: 9.404654

Which gave a score of 225, really close! From here, we got to 226 and 227 with the remote script, until one of our teammates started changing stuff by hand and got the input:

Inclination: 112  
RAAN: 175  
Eccentricity: 23000000  
Arg Perigee: 238  
Mean Anomaly: 150  
Mean Motion: 9.9

Which got a enough time of contact!

## Solution script

One of the (multiple) heuristics solve scripts:

from pwn import \*  
from skyfield.api import \*  
from datetime import datetime  
from heapq import \*  
import random  
import itertools  
  
PREV\_SCORE = 170  
DELTA = 0.01  
  
pq = []  
heapify(pq)  
  
# here are some random checkpoints from which we started  
# heappush(pq, (-PREV\_SCORE, 60, 10, 10, 10, 30, 10))  
# heappush(pq, (-PREV\_SCORE, 60, 10, 1000000, 10, 30, 10))  
heappush(pq, (-215.000000, 111.621200, 172.495300, 1290108.581600, 239.212900, 153.050600, 9.571854))  
# heappush(pq, (-175.000000, 58.700000, 10.000000, 999992.700000, 9.740000, 28.000000, 14.510000))  
  
while True:  
 # connection stuff  
 r = remote("contact.quals2023-kah5Aiv9.satellitesabove.me", 5300)  
 r.sendlineafter(b"Ticket please:", b"TICKET")  
  
 # read best entry  
 score, inclination, raan, eccentricity, arg\_perigee, mean\_anomaly, mean\_motion = heappop(pq)  
 print("OLD\_SCORE: %f" % score)  
 print("Inclination: %f" % inclination)  
 print("RAAN: %f" % raan)  
 print("Eccentricity: %f" % eccentricity)  
 print("Arg Perigee: %f" % arg\_perigee)  
 print("Mean Anomaly: %f" % mean\_anomaly)  
 print("Mean Motion: %f" % mean\_motion)  
  
 r.sendlineafter(b"Inclination (deg):", str(inclination).encode())  
 r.sendlineafter(b"RAAN (deg):", str(raan).encode())  
 r.sendlineafter(b"Eccentricity (x10^-7):", str(eccentricity).encode())  
 r.sendlineafter(b"Argument of perigee (deg):", str(arg\_perigee).encode())  
 r.sendlineafter(b"Mean anomaly (deg):", str(mean\_anomaly).encode())  
 r.sendlineafter(b"Mean motion (revs/day):", str(mean\_motion).encode())  
  
 # read new score  
 try:  
 r.recvuntil(b"Your orbit achieved ")  
 line = r.recvline(False).split(b" ", 1)[0]  
 except EOFError:  
 print("BRUCIA!")  
 continue  
  
 new\_score = -int(line)  
 print("New Score: %d" % new\_score)  
  
 # generate random variations, we also used a version of the script which generated 100 variations and didn't use a mask to choose which parameters to change  
 # also we had a version with local testing and multithreading :)  
 for i in range(10):  
 rnd = random.randint(1, 63)  
 new\_tuple = [inclination, raan, eccentricity, arg\_perigee, mean\_anomaly, mean\_motion]  
 for i, bit in enumerate(bin(rnd)[2:].zfill(6)):  
 new\_tuple[i] = new\_tuple[i] + int(bit) \* random.randint(-500, 500) \* DELTA  
  
 heappush(pq, tuple([new\_score] + new\_tuple))  
  
 r.close()

The local simulation script:

from skyfield.api import load, wgs84  
import IPython  
  
'''  
Name Lat (deg) Long (deg) Alt (m)  
Cape Canaveral 28.40 -80.61 27  
Cape Cod 41.70 -70.03 9  
Anchorage 61.21 -149.90 40  
Vandenberg 34.76 -120.52 122  
Denver 39.74 -104.98 1594  
'''  
  
eph = load("de421.bsp")  
earth = eph['Earth']  
  
stations = {  
 "Cape Canaveral": wgs84.latlon(28.40, -80.61, 27),  
 "Cape Cod": wgs84.latlon(41.70, -70.03, 9),  
 "Anchorage": wgs84.latlon(61.21, -149.90, 40),  
 "Vandenberg": wgs84.latlon(34.76, -120.52, 122),  
 "Denver": wgs84.latlon(39.74, -104.98, 1594),  
}  
  
stations\_eph = {  
 "Cape Canaveral": wgs84.latlon(28.40, -80.61, 27) + earth,  
 "Cape Cod": wgs84.latlon(41.70, -70.03, 9) + earth,  
 "Anchorage": wgs84.latlon(61.21, -149.90, 40) + earth,  
 "Vandenberg": wgs84.latlon(34.76, -120.52, 122) + earth,  
 "Denver": wgs84.latlon(39.74, -104.98, 1594) + earth,  
}  
  
secs = {}  
  
ts = load.timescale()  
t0 = ts.utc(2023, 4, 1)  
t1 = ts.utc(2023, 4, 1, 8)  
sat = load.tle\_file("tle.txt")[0]  
secs = 0;  
t = t0  
for \_ in range (8 \* 60):  
 t = t + 1/(24\*60)  
 for k in stations.keys():  
 difference = sat - stations[k]  
 topocentric = difference.at(t)  
 alt, az, distance = topocentric.altaz()  
 if distance.km < 300:  
 print("BRUCIA")  
 exit()  
 if alt.degrees > 15 and distance.km < 6000:  
 secs += 1  
 print(f"{k}: Alt: {alt}, Dist: {distance.km}")  
 break  
  
print(f"Total sec: {secs}")