Program Assigment - RNA Secondary Structure Prediction

Weize Xu

June 4, 2018

1 About the Program

1.1 Dependency

This program is developed and tested on Linux system, and Python version ≥ 3.4 is required.

1.2 Usage

Open your shell, and type "python main.py –help" to show the usage information, as follow:

```
$ python main.py --help
usage: RNA-2nd-structure-predictor [-h]
                                   [--method [{nussinov,maxstacks,minstackenergy}]]
                                   [--print_max_score]
                                   input

Predict the secondary structure of an RNA sequence.

positional arguments:
  input                 Path to input fasta file.

optional arguments:
  -h, --help            show this help message and exit
  --method [{nussinov,maxstacks,minstackenergy}]
                        Algorithm used for predict.
  --print_max_score, -v
                        Print the max score.
```

2 Algorithms for each Task

2.1 Maximize the number of base pairs

Just use the Nussinov algorithm.

2.1.1 Implementation

See Python module `./nussinov.py`

2.2 Maximize the number of stacking pairs

Solving with a dynamic programming, it's very like the Nussinov algorithm, just a little different:

2.2.1 Algorithm description

Notation:

$S[1..n]$	An RNA sequence with n bases.
$V(i,j)$	The maxmium number of stacking pairs in $S[i,j]$

Basic case:

$$V(i,j) = 0 \textbf{ if } j < j + 3$$

Since, when $j < i + 3$, it can not form a stacked pair. The shortest span length stacking pair looks like this:

```
    i  j
....AAUU....
    (())
```

Recursive case:

1. No stacked pair in position j:

$$V(i,j) = V(i,j-1)$$

2. No stacked pair in position i:

$$V(i,j) = V(i+1,j)$$

3. Position (i,j) form a length h stacking pairs:

$$V(i,j) = \max_{1 \le h \le m} \{V(i+h+1, j-h-1) + h * \prod_{t=0}^{h} \delta(S[i+t], S[j-t])\} \tag{1}$$

Where:

$$m = floor(\frac{i-j+1}{2}) - 1$$

$$\delta(x,y) = 1 \textbf{ if } (x,y) \in P \textbf{ otherwise } 0$$

$$P = \{(a,u),(u,a),(c,g),(g,c),(g,u),(u,g)\}$$

4. Both position i and j form a stacking pair, but (i,j) not form stacking pair:

$$V(i,j) = \max_{i+3 \le k < j} \{V(i,k) + V(k+1,j)\}$$

So, the recursive function is:

$$V(i,j) = \max \begin{cases} V(i,j-1) \\ V(i+1,j) \\ \max\limits_{1 \le h \le m} \{V(i+h+1, j-h-1) + h * \prod\limits_{t=0}^{h} \delta(S[i+t], S[j-t])\} \\ \max\limits_{i+3 \le k < j} \{V(i,k) + V(k+1,j)\} \end{cases} \tag{2}$$

Recover pairs:

We can recover the pairs by backtrack the Matrix, detail see Algorithm 31 and the implementation.

2.2.2 Complexity Analyze

Space used for store the $V(i,j)$ is $O(n^2)$. Time needed to fill the $V(i,j)$ entries is $O(n^2)$, and each $V(i,j)$ entry can be computed in $O(n)$ time. So this algorithm can be solved in $O(n^3)$ time. This complexity is same to the Nussinov algorithm.

2.2.3 Implementation

See Python module `./maxstacks.py`

Algorithm 1: Backtrack Algorithm

```
input  : A filled Matrix V[1..n, 1..n]
output: A set of pairs $P$
procedure backtrack($i$, $j$)
    if $j \leq i + 2$ then
        return
    else if $V(i, j) = V(i, j - 1)$ then
        backtrack($i$, $j$-1)
    else if $V(i, j) = V(i + 1, j)$ then
        backtrack($i$+1, $j$)
    else
        for $k \leftarrow i$ to $j$ do
            if $V(i, j) = V(i, k) + V(k + 1, j)$ then
                backtrack($i$, $k$)
                backtrack($k$+1, $j$)
                return
            else
                $m = floor(\frac{i - k + 1}{2}) - 1$
                for $h \leftarrow m$ to 1 do
                    $AllMatch \leftarrow \prod_{t=0}^{h} \delta(S[i + t], S[j - t])$
                    if $AllMatch \neq 0 \wedge V(i, j) = V(k + h + 1, j - h - 1) + h$ then
                        for $t \leftarrow 0$ to $h$ do
                            $P \cup \{(k + t, j - t)\}$
                        end
                        backtrack($i$, $k$-1)
                        backtrack($k$+$h$+1, $j$-$h$-1)
                        return
                    end
                end
            end
        end
    end
end
backtrack(1,$n$)
```

2.3 Minimize free energy of stacking pairs

2.3.1 Algorithm description

This method is a simpler version of minimum free energy method(MSE), only consider the free energy of stacking pairs. The free energy of stacking pair eS is get from the Turner table:

	A/U	C/G	G/C	U/A	G/U	U/G
A/U	-0.9	-2.2	-2.1	-1.1	-0.6	-1.4
C/G	-2.1	-3.3	-2.4	-2.1	-1.4	-2.1
G/C	-2.4	-3.4	-3.3	-2.2	-1.5	-2.5
U/A	-1.3	-2.4	-2.1	-0.9	-1.0	-1.3
G/U	-1.3	-2.5	-2.1	-1.4	-0.5	$+1.3$
U/G	-1.0	-1.5	-1.4	-0.6	$+0.3$	-0.5

Actually, we just need to make a small modify to the algorithm in the section 2.2. We can just change the equation 1 in recursive case 3 to:

$$V(i,j) = \max_{1 \le h \le m} \{V(i+h+1, j-h-1) - \sum_{t=0}^{h} eS(i+t, j-t) * \prod_{t=0}^{h} \delta(S[i+t], S[j-t])\} \tag{3}$$

Where the $eS(i,j)$ can get from the RNA sequence and the Turner table. Here we maxmized the negative free energy of stacking pairs, it equal to minimize it.

Penalty for short distance pairs:

Actually, this algorithm is not good enough. We can penalize the short distance pairs for get a more closer result, because the short distance pairs have larger energy in general. Here we define a penalty function $P(i,j)$:

$$P(i,j) = \frac{1}{1 + a^{-b((i-j)-c)}}$$

a, b and c are positive constants. Then, put it into the recursive function 3:

$$V(i,j) = \max_{1 \le h \le m} \{V(i+h+1, j-h-1) - \sum_{t=0}^{h} P(i+t, j-t) eS(i+t, j-t) * \prod_{t=0}^{h} \delta(S[i+t], S[j-t])\}$$

2.3.2 Complexity Analyze

Same to section 2.2.2.

2.3.3 Implementation

See Python module `./minstackenergy.py`

The advantage and disadvantage about this method is disscused in section 4.

3 Benchmark on Testing Data

3.1 Run algorithms on the testing data

Firstly run the Nussinov algorithm:

```
$ python main.py test_data/hammerhead_ribozyme.fa --print_max_score --method nussinov
> hammerhead ribozyme (1rmn)
GCGCUCUGAUGAGGCCGCAAGGCCGAAACUGCCGCAAGGCAGUCAGCGC
((()(((()(.(()(()(.(()()...))))).)).)()))(.()).
20
$ python main.py test_data/PDB_00313.fa --print_max_score --method nussinov
> PDB_00313 (E. Coli) CYSTEINYL-TRNA
GGCGCGUUAACAAAGCGGUUAUGUAGCGGAUUGCAAAUCCGUCUAGUCCGGUUCGACUCCGGAACGCGCCUCCA
((()))(()))(..((()(()((((((((((()))()(.))(()))))())))()))(()))))()))).))..
34
$ python main.py test_data/SRP_00004.fa  --print_max_score --method nussinov
> SRP_00004
GGGAGGUUGGUGGUGGACGAGCCACUCGCCAACCGGGUCAGGUCCGGAAGGAAGCAGCCCUAACGAGCCAGGCACGGGUCGCCGUGCC
AGCCUCCCACCUUUU
((((()(((((()((.().())(.)()))))(()))(((.)(((())).((..((.)))().().))))((((((()()))))))(
.)))))).)).))))
45
```

As shown above the max score(maximum pairs) get from the Nussinov algorithm is 20, 34 and 45.

Then the maximum stacking pairs method:

```
$ python main.py test_data/hammerhead_ribozyme.fa --print_max_score --method maxstacks
> hammerhead ribozyme (1rmn)
GCGCUCUGAUGAGGCCGCAAGGCCGAAACUGCCGCAAGGCAGUCAGCGC
(((((.))((((((....)))))....(((((....)))))))))))
13
$ python main.py test_data/PDB_00313.fa --print_max_score --method maxstacks
> PDB_00313 (E. Coli) CYSTEINYL-TRNA
GGCGCGUUAACAAAGCGGUUAUGUAGCGGAUUGCAAAUCCGUCUAGUCCGGUUCGACUCCGGAACGCGCCUCCA
(((((((.(((.(((.))).))).(((((((...)))))))..((((.(()).))))(()))))))))))....
22
$ python main.py test_data/SRP_00004.fa  --print_max_score --method maxstacks
> SRP_00004
GGGAGGUUGGUGGUGGACGAGCCACUCGCCAACCGGGUCAGGUCCGGAAGGAAGCAGCCCUAACGAGCCAGGCACGGGUCGCCGUGCC
AGCCUCCCACCUUUU
(((((((.((((((((.....)))((((((.(((.)))..))(()))(((..((.)))))..)))))))(((((((....)))))))
((.)))).)))))))
29
```

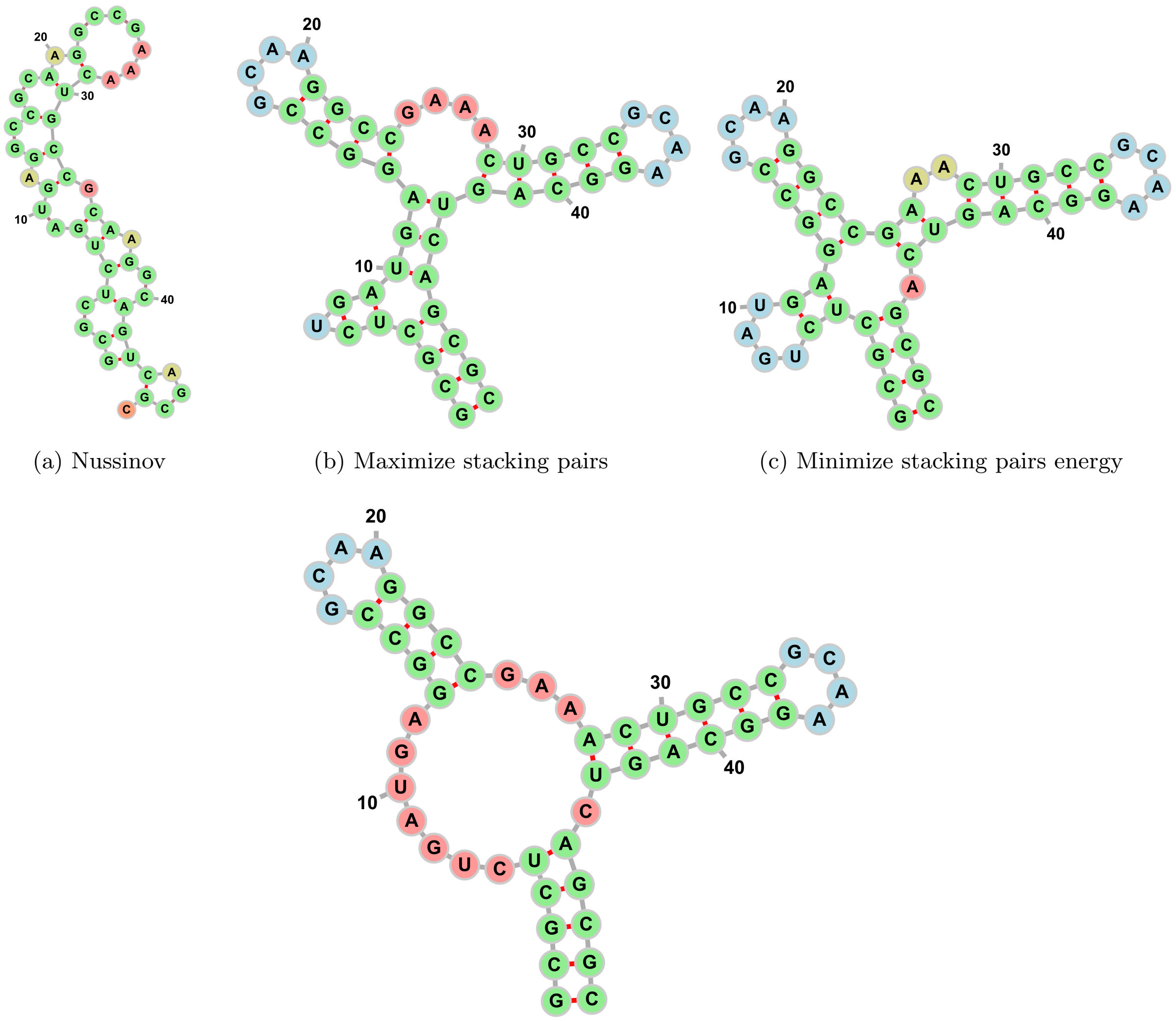
The max stacking pairs of three samples are 13, 22 and 29.

Then run the minimize stacking pairs energy method.

```
$ python main.py test_data/hammerhead_ribozyme.fa --print_max_score --method
minstackenergy
> hammerhead ribozyme (1rmn)
GCGCUCUGAUGAGGCCGCAAGGCCGAAACUGCCGCAAGGCAGUCAGCGC
((((((....))((((....))))((..(((((....))))))).))))
34.47232640482083
$ python main.py test_data/PDB_00313.fa --print_max_score --method minstackenergy
> PDB_00313 (E. Coli) CYSTEINYL-TRNA
GGCGCGUUAACAAAGCGGUUAUGUAGCGGAUUGCAAAUCCGUCUAGUCCGGUUCGACUCCGGAACGCGCCUCCA
(((((((((((((....)).))).(((((((...))))))).)).(((((.(())..)))))))))))))....
47.1048578347625
$ python main.py test_data/SRP_00004.fa  --print_max_score --method minstackenergy
> SRP_00004
GGGAGGUUGGUGGUGGACGAGCCACUCGCCAACCGGGUCAGGUCCGGAAGGAAGCAGCCCUAACGAGCCAGGCACGGGUCGCCGUGCC
AGCCUCCCACCUUUU
(((.((((((.(((((.....)))))..))))))((((((((((....))).((.)))))...))((..(((((((....)))))))
.))..))).)))...
66.78663059444345
```

3.2 Compare with correct structure

Then we can compare the results from each algorithms with the correct structure by visualize them. I have visualized them with forna(`http://rna.tbi.univie.ac.at/forna/`), results is shown in the Fig.1, 2 and 3.



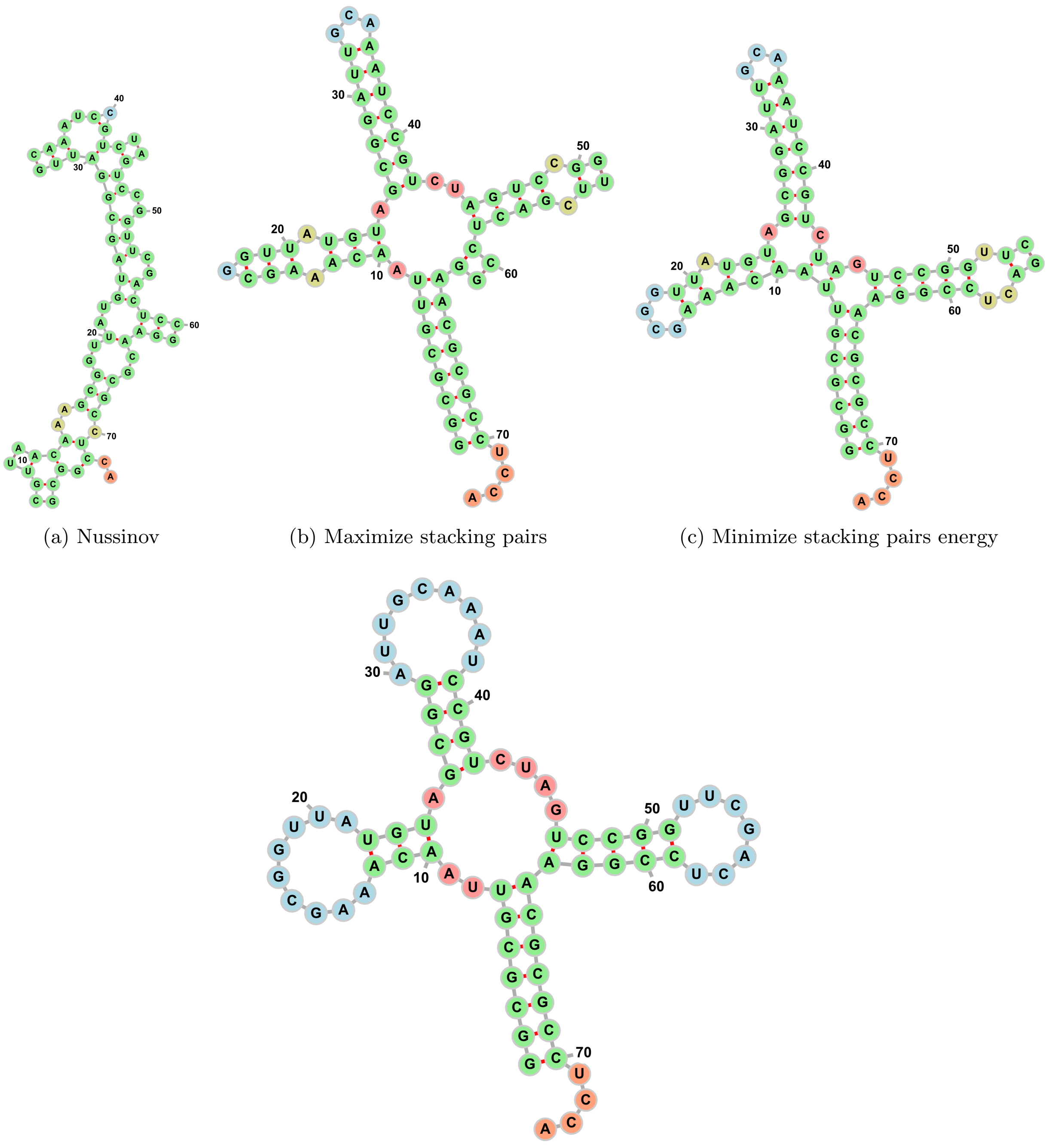
(a) Nussinov

(b) Maximize stacking pairs

(c) Minimize stacking pairs energy

(d) Correct

Figure 1: hammerhead ribozyme



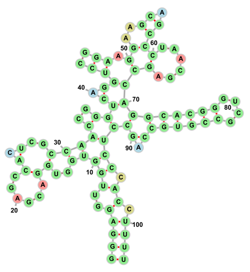
(a) Nussinov

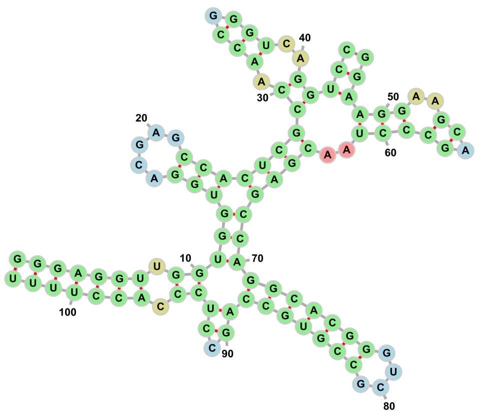
(b) Maximize stacking pairs

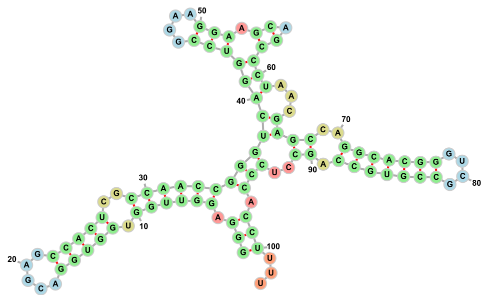
(c) Minimize stacking pairs energy

(d) Correct

Figure 2: PDB_00313


(a) Nussinov


(b) Maximize stacking pairs


(c) Minimize stacking pairs energy

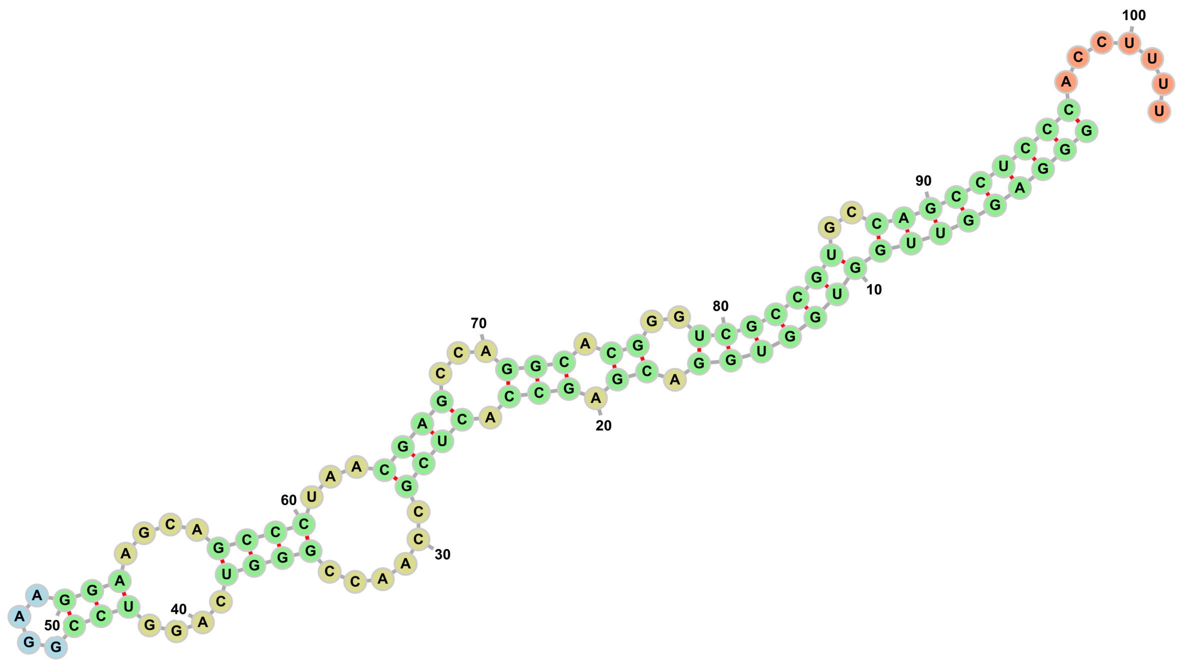

(d) Correct

Figure 3: SRP_00004

3.3 Time consuming

I tested the time consuming of each algorithm, result is shown in Fig.4. From this result, we can see although these three algorithms has same time complexity, but the time consuming are different. Nussinov is fastest, other two algorithms is similar. This is because these two algoritms have an additional loop when fill the matrix and traceback compare to the Nussinov.

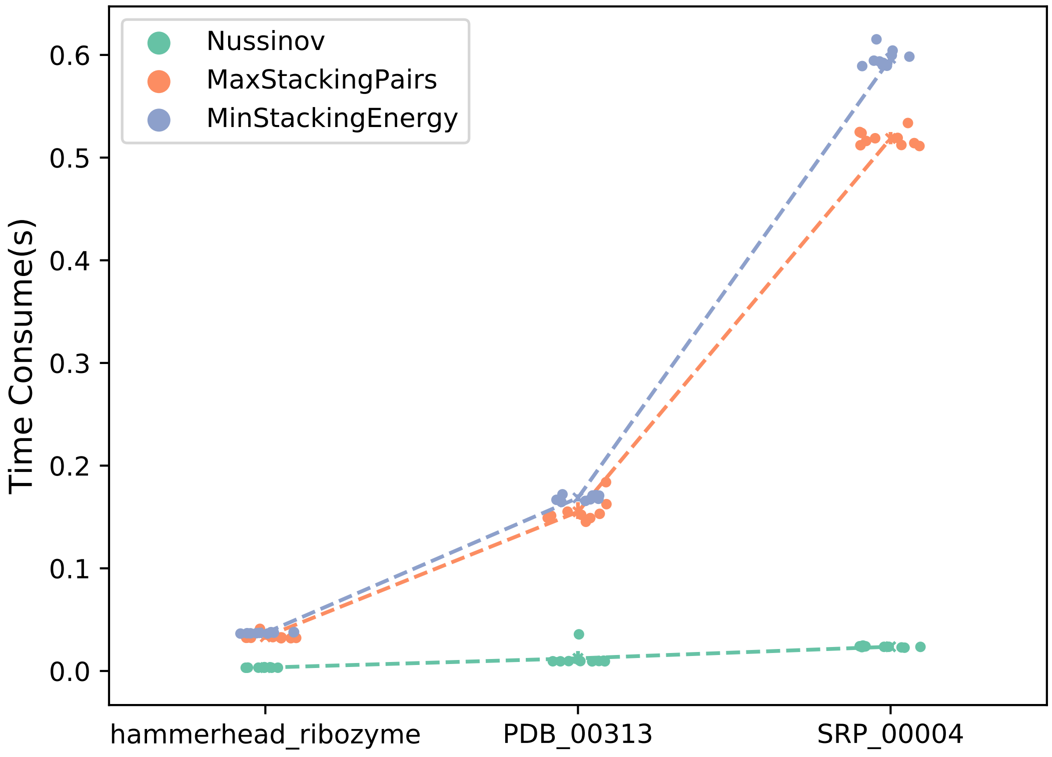


Figure 4: Time consuming of each algorithm

4 Discussion

From the results in the section 3.2, we can see that. 1. All these three algorithms can not predict the correct structure. I'm soo sorry. 2. Maximize stacking pairs and minimize stacking pairs energy methods produce the similar results. 3. Penalty for the short distance pairs can prohibit the formation of some incorrect short distance pairs.

The advantages of my score function is consider the real energy of each stacking pairs instead of just count the numbers. And I have considered the fact that, short distance pair is more unstable than long distance pairs, and this point indeed improved the results of prediction. But my score function is not considered the energy of other secondary structures in addition to stacking pairs. So it can not predict the structure correctly. In addition the backtrack algorithm which I used can not trace all possible structures. Last, this method also can not predict the pseudo-knot structure.

At the same time, this implementation is also not good enough. At least two point can be improved: 1. Python interpreter is not fast, if implement algorithm with C it will be much more faster. 2. Actually, in this implementation, the matrix for store scores only a half useful, I can use hash table or sparse matrix to improve the memory usage.