lab4_Alicja_Augustyniak_bioinfstrukturRNA

May 19, 2025

Wypisze potencjalne pary G-C na podstawie odległości atomowych (3.0)

Wypisze dodatkowo potencjalne pary A-U oraz G-U na podstawie odległości atomowych (3.5)

[52]:
```
import sys
import os
import glob
from Bio.PDB import PDBParser

def find_gc(structure):

    d1_min, d1_max = 2.67, 3.19 # G:N1 - C:N3
    d2_min, d2_max = 2.44, 3.20 # G:N2 - C:O2
    d3_min, d3_max = 2.61, 3.31 # G:O6 - C:N4
    d4_min, d4_max = 2.53, 3.30 # A:N1 - U-N3
    d5_min, d5_max = 2.57, 3.47 # A:N6 - U:O4
    d6_min, d6_max = 2.29, 3.48 # G:N1 - U:O2
    d7_min, d7_max = 2.39, 3.48 # G:O6 - U:N3

    G_residues = []
    C_residues = []
    A_residues = []
    U_residues = []
    for model in structure:
        for chain in model:
            for res in chain:
                name = res.get_resname().strip()
                if name == 'G':
                    G_residues.append(res)
                elif name == 'U':
                    U_residues.append(res)
                elif name == 'A':
                    A_residues.append(res)
                elif name == 'C':
                    C_residues.append(res)

    results = []
    for g in G_residues:
```

```
        for c in C_residues:
            try:
                d1 = g['N1'] - c['N3']
                d2 = g['N2'] - c['O2']
                d3 = g['O6'] - c['N4']
            except KeyError:
                continue
            if d1_min <= d1 <= d1_max and d2_min <= d2 <= d2_max and d3_min <=␣
 ↪d3 <= d3_max:
                results.append((g, c))

    for a in A_residues:
        for u in U_residues:
            try:
                d4 = a['N1'] - u['N3']
                d5 = a['N6'] - u['O4']
            except KeyError:
                continue
            if d4_min <= d4 <= d4_max and d5_min <= d5 <= d5_max:
                results.append((a,u))

    for g in G_residues:
        for u in U_residues:
            try:
                d6 = g['N1'] - u['O2']
                d7 = g['O6'] - u['N3']
            except KeyError:
                continue
            if d6_min <= d6 <= d6_max and d7_min <= d7 <= d7_max:
                results.append((g,u))
    return results

if __name__ == '__main__':
    parser = PDBParser(QUIET=True)
    pdb_files = glob.glob(os.path.join('.', '*.pdb'))
    if not pdb_files:
        print("Brak")
        sys.exit(0)

    total_pairs = 0
    for pdb_file in pdb_files:
        basename = os.path.splitext(os.path.basename(pdb_file))[0]
        try:
            structure = parser.get_structure(basename, pdb_file)
        except Exception as e:
            print(f"Nie można wczytać {pdb_file}: {e}")
            continue
```

```
        raw = find_gc(structure)


        chains_in_pairs = { r.get_parent().id for pair in raw for r in pair }

        pairs = []
        for r1, r2 in raw:
            ch1, rn1, num1 = r1.get_parent().id, r1.get_resname().strip(), r1.
↪id[1]
            ch2, rn2, num2 = r2.get_parent().id, r2.get_resname().strip(), r2.
↪id[1]
            if ch1 == ch2 and abs(num1 - num2) == 1:
                continue

            if {'A','B'}.issubset(chains_in_pairs):
                if {ch1, ch2} != {'A','B'}:
                    continue
                if ch1 == 'A':
                    left, right = (ch1, rn1, num1), (ch2, rn2, num2)
                else:
                    left, right = (ch2, rn2, num2), (ch1, rn1, num1)

            else:
                if num1 <= num2:
                    left, right = (ch1, rn1, num1), (ch2, rn2, num2)
                else:
                    left, right = (ch2, rn2, num2), (ch1, rn1, num1)

            pairs.append((left, right))

        pairs.sort(key=lambda x: x[0][2])


        if pairs:
            out_name = f"{basename}_pairs.txt"
            with open(out_name, 'w') as out_file:
                for (l_ch, l_rn, l_num), (r_ch, r_rn, r_num) in pairs:
                    out_file.write(f"{l_ch}:{l_rn}{l_num} - {r_ch}:
↪{r_rn}{r_num}\n")
            print(f"Plik: {pdb_file} zapisano {len(pairs)} par do {out_name}")
            total_pairs += len(pairs)

    print(f"zapisano {total_pairs} par.")
```

```
Plik: ./rp10.pdb zapisano 3 par do rp10_pairs.txt
Plik: ./rp12.pdb zapisano 37 par do rp12_pairs.txt
```

```
Plik: ./rp09.pdb zapisano 19 par do rp09_pairs.txt
Plik: ./rp21.pdb zapisano 9 par do rp21_pairs.txt
Plik: ./rp03.pdb zapisano 23 par do rp03_pairs.txt
Plik: ./rp14_free.pdb zapisano 10 par do rp14_free_pairs.txt
Plik: ./rp04.pdb zapisano 31 par do rp04_pairs.txt
Plik: ./rp01.pdb zapisano 19 par do rp01_pairs.txt
Plik: ./rp24.pdb zapisano 40 par do rp24_pairs.txt
Plik: ./rp06.pdb zapisano 27 par do rp06_pairs.txt
Plik: ./rp13.pdb zapisano 21 par do rp13_pairs.txt
Plik: ./rp15.pdb zapisano 10 par do rp15_pairs.txt
Plik: ./rp02.pdb zapisano 4 par do rp02_pairs.txt
Plik: ./rp14_bound.pdb zapisano 17 par do rp14_bound_pairs.txt
Plik: ./rp20.pdb zapisano 11 par do rp20_pairs.txt
Plik: ./rp17.pdb zapisano 15 par do rp17_pairs.txt
Plik: ./rp08.pdb zapisano 29 par do rp08_pairs.txt
Plik: ./rp11.pdb zapisano 21 par do rp11_pairs.txt
Plik: ./rp18.pdb zapisano 20 par do rp18_pairs.txt
Plik: ./rp19.pdb zapisano 16 par do rp19_pairs.txt
Plik: ./rp05.pdb zapisano 45 par do rp05_pairs.txt
zapisano 427 par.
```

Sprawdzi każdą potencjalną parę pod względem wartości kąta między płaszczyznami zasad azotowych (4.0)

Jw. dla odległości punktu od płaszczyzny (4.5)

[1]:
```python
import sys
import os
import glob
import numpy as np
from Bio.PDB import PDBParser

CS_MIN = 0.8
DIST_MAX = 5.73


def find_gc(structure):

    d1_min, d1_max = 2.67, 3.19  # G:N1 - C:N3
    d2_min, d2_max = 2.44, 3.20  # G:N2 - C:O2
    d3_min, d3_max = 2.61, 3.31  # G:O6 - C:N4
    d4_min, d4_max = 2.53, 3.30  # A:N1 - U:N3
    d5_min, d5_max = 2.57, 3.47  # A:N6 - U:O4
    d6_min, d6_max = 2.29, 3.48  # G:N1 - U:O2
    d7_min, d7_max = 2.39, 3.48  # G:O6 - U:N3

    G_res = []
    C_res = []
```

```
    A_res = []
    U_res = []
    for model in structure:
        for chain in model:
            for res in chain:
                name = res.get_resname().strip()
                if name == 'G':   G_res.append(res)
                elif name == 'C': C_res.append(res)
                elif name == 'A': A_res.append(res)
                elif name == 'U': U_res.append(res)

    raw = []
    for g in G_res:
        for c in C_res:
            try:
                d1 = g['N1'] - c['N3']
                d2 = g['N2'] - c['O2']
                d3 = g['O6'] - c['N4']
            except KeyError:
                continue
            if d1_min <= d1 <= d1_max and d2_min <= d2 <= d2_max and d3_min <=␣
 ↪d3 <= d3_max:
                raw.append((g, c))
    for a in A_res:
        for u in U_res:
            try:
                d4 = a['N1'] - u['N3']
                d5 = a['N6'] - u['O4']
            except KeyError:
                continue
            if d4_min <= d4 <= d4_max and d5_min <= d5 <= d5_max:
                raw.append((a, u))
    for g in G_res:
        for u in U_res:
            try:
                d6 = g['N1'] - u['O2']
                d7 = g['O6'] - u['N3']
            except KeyError:
                continue
            if d6_min <= d6 <= d6_max and d7_min <= d7 <= d7_max:
                raw.append((g, u))
    return raw


def calculate_cos_norm(resR, resY):
    try:
        C2R = resR['C2'].get_vector().get_array()
```

```
        C4R = resR['C4'].get_vector().get_array()
        C6R = resR['C6'].get_vector().get_array()
        C2Y = resY['C2'].get_vector().get_array()
        C4Y = resY['C4'].get_vector().get_array()
        C6Y = resY['C6'].get_vector().get_array()
    except KeyError:
        return None
    # wektory
    v11 = C6R - C4R
    v12 = C2R - C4R
    v21 = C4Y - C2Y
    v22 = C6Y - C2Y
    # normalne
    nR = np.cross(v11, v12)
    nY = np.cross(v21, v22)
    normR = np.linalg.norm(nR)
    normY = np.linalg.norm(nY)
    if normR == 0 or normY == 0:
        return None
    cs = np.dot(nR, nY) / (normR * normY)
    return float(cs)


def calculate_dist(resR, resY):
    try:
        C2R = resR['C2'].get_vector().get_array()
        C4R = resR['C4'].get_vector().get_array()
        C6R = resR['C6'].get_vector().get_array()
        C6Y = resY['C6'].get_vector().get_array()
    except KeyError:
        return None
    # wektory R
    v11 = C6R - C4R
    v12 = C2R - C4R
    # normalna do płaszczyzny R
    nR = np.cross(v11, v12)
    normR = np.linalg.norm(nR)
    if normR == 0:
        return None
    # wektor od punktu leżącego w płaszczyźnie
    v2 = C6Y - C4R
    dist = abs(np.dot(nR, v2)) / normR
    return float(dist)


if __name__ == '__main__':
    parser = PDBParser(QUIET=True)
```

```
    pdb_files = glob.glob(os.path.join('.', '*.pdb'))
    if not pdb_files:
        print("Brak")
        sys.exit(0)

    total_pairs = 0
    for pdb_file in pdb_files:
        basename = os.path.splitext(os.path.basename(pdb_file))[0]
        try:
            structure = parser.get_structure(basename, pdb_file)
        except Exception as e:
            print(f"Nie można wczytać {pdb_file}: {e}")
            continue

        raw = find_gc(structure)
        cs_filtered = []
        for r1, r2 in raw:
            res1, res2 = r1.get_resname().strip(), r2.get_resname().strip()
            if res1 in ('G','A') and res2 in ('C','U'):
                R, Y = r1, r2
            elif res2 in ('G','A') and res1 in ('C','U'):
                R, Y = r2, r1
            else:
                continue
            cs = calculate_cos_norm(R, Y)
            if cs is not None and cs >= CS_MIN:
                cs_filtered.append((r1, r2, cs))

        final_pairs = []
        for r1, r2, cs in cs_filtered:
            res1, res2 = r1.get_resname().strip(), r2.get_resname().strip()
            if res1 in ('G','A') and res2 in ('C','U'):
                R, Y = r1, r2
            else:
                R, Y = r2, r1
            dist = calculate_dist(R, Y)
            if dist is not None and dist <= DIST_MAX:
                final_pairs.append((r1, r2, cs, dist))

        chains_in_pairs = { r.get_parent().id for (r, *_ ) in final_pairs for r␣
↪in (r,) }

        pairs_out = []
        for r1, r2, cs, dist in final_pairs:
            ch1, rn1, num1 = r1.get_parent().id, r1.get_resname().strip(), r1.
↪id[1]
```

```
            ch2, rn2, num2 = r2.get_parent().id, r2.get_resname().strip(), r2.
↪id[1]
            if ch1 == ch2 and abs(num1 - num2) == 1:
                continue
            if {'A','B'}.issubset(chains_in_pairs):
                if {ch1, ch2} != {'A','B'}:
                    continue
                if ch1 == 'A':
                    left, right = (ch1,rn1,num1,cs,dist), (ch2,rn2,num2,cs,dist)
                else:
                    left, right = (ch2,rn2,num2,cs,dist), (ch1,rn1,num1,cs,dist)
            else:
                if num1 <= num2:
                    left, right = (ch1,rn1,num1,cs,dist), (ch2,rn2,num2,cs,dist)
                else:
                    left, right = (ch2,rn2,num2,cs,dist), (ch1,rn1,num1,cs,dist)
            pairs_out.append((left, right))

        pairs_out.sort(key=lambda x: x[0][2])

        if pairs_out:
            out_name = f"{basename}_pairs.txt"
            with open(out_name, 'w') as out_file:
                for (l_ch,l_rn,l_num,l_cs,l_dist),␣
↪(r_ch,r_rn,r_num,r_cs,r_dist) in pairs_out:
                    out_file.write(f"{l_ch}:{l_rn}{l_num} - {r_ch}:
↪{r_rn}{r_num}  cs={l_cs:.3f} dist={l_dist:.3f}\n")
            print(f"Plik: {pdb_file} zapisano {len(pairs_out)} par (cs >=␣
↪{CS_MIN} i dist <= {DIST_MAX}) do {out_name}")
            total_pairs += len(pairs_out)

    print(f"zapisano {total_pairs} par.")
```

```
Plik: ./rp10.pdb zapisano 3 par (cs >= 0.8 i dist <= 5.73) do rp10_pairs.txt
Plik: ./rp12.pdb zapisano 36 par (cs >= 0.8 i dist <= 5.73) do rp12_pairs.txt
Plik: ./rp09.pdb zapisano 19 par (cs >= 0.8 i dist <= 5.73) do rp09_pairs.txt
Plik: ./rp21.pdb zapisano 9 par (cs >= 0.8 i dist <= 5.73) do rp21_pairs.txt
Plik: ./rp03.pdb zapisano 23 par (cs >= 0.8 i dist <= 5.73) do rp03_pairs.txt
Plik: ./rp14_free.pdb zapisano 10 par (cs >= 0.8 i dist <= 5.73) do
rp14_free_pairs.txt
Plik: ./rp04.pdb zapisano 30 par (cs >= 0.8 i dist <= 5.73) do rp04_pairs.txt
Plik: ./rp01.pdb zapisano 19 par (cs >= 0.8 i dist <= 5.73) do rp01_pairs.txt
Plik: ./rp24.pdb zapisano 40 par (cs >= 0.8 i dist <= 5.73) do rp24_pairs.txt
Plik: ./rp06.pdb zapisano 27 par (cs >= 0.8 i dist <= 5.73) do rp06_pairs.txt
Plik: ./rp13.pdb zapisano 21 par (cs >= 0.8 i dist <= 5.73) do rp13_pairs.txt
Plik: ./rp15.pdb zapisano 10 par (cs >= 0.8 i dist <= 5.73) do rp15_pairs.txt
Plik: ./rp02.pdb zapisano 4 par (cs >= 0.8 i dist <= 5.73) do rp02_pairs.txt
```

```
Plik: ./rp14_bound.pdb zapisano 17 par (cs >= 0.8 i dist <= 5.73) do
rp14_bound_pairs.txt
Plik: ./rp20.pdb zapisano 11 par (cs >= 0.8 i dist <= 5.73) do rp20_pairs.txt
Plik: ./rp17.pdb zapisano 15 par (cs >= 0.8 i dist <= 5.73) do rp17_pairs.txt
Plik: ./rp08.pdb zapisano 29 par (cs >= 0.8 i dist <= 5.73) do rp08_pairs.txt
Plik: ./rp11.pdb zapisano 21 par (cs >= 0.8 i dist <= 5.73) do rp11_pairs.txt
Plik: ./rp18.pdb zapisano 20 par (cs >= 0.8 i dist <= 5.73) do rp18_pairs.txt
Plik: ./rp19.pdb zapisano 16 par (cs >= 0.8 i dist <= 5.73) do rp19_pairs.txt
Plik: ./rp05.pdb zapisano 44 par (cs >= 0.8 i dist <= 5.73) do rp05_pairs.txt
zapisano 424 par.
```