Identyfikacja różnic w strukturze 2D RNA

Katarzyna Iwanów
Ewelina Kurek

Cel projektu

Pierwszym celem projektu było napisanie skryptu, który na podstawie danych z PDB znajduje redundantne łańcuchy RNA, czyli takie o identycznej sekwencji nukleotydów. Drugim celem było stworzenie algorytmów do porównania struktury drugorzędowej tych łańcuchów. Ostatnim celem pracy była analiza otrzymanych wyników.

Wyszukiwanie redundantnych łańcuchów

Pierwszym krokiem było pobranie struktur trzeciorzędowych cząsteczek RNA ze strony bazy PDB. Następnie tworzyliśmy listę wszystkich łańcuchów o zadanej długości pochodzących z pobranych molekuł. Kolejnym krokiem było porównywanie łańcuchów i doprowadziło nas to do otrzymania listy, której elementami były listy łańcuchów o tej samej sekwencji nukleotydowej. Dokładny opis użytych funkcji znajduje się w dodatku.

Otrzymywanie struktury drugorzędowej.

Dla każdego łańcucha z danego klastra należało otrzymać strukturę drugorzędową. W tym celu skorzystano ze strony http://rnapdbee.cs.put.poznan.pl/, która zwraca strukturę drugorzędową dla wybranej cząsteczki RNA na podstawie pliku pdb. Ponieważ nie jest możliwe otrzymanie struktury drugorzędowe w sposób zautomatyzowany, interesujące wyniki należało pobrać ręcznie. Wobec tego niemożliwa okazała się analiza wszystkich klastrów, jednak nawet kilkanaście klastrów wystarczyło, by wyciągnąć interesujące hipotezy dotyczące struktury drugorzędowej łańcuchów redundantnych.

Analiza i porównywanie struktur drugorzędowych

Dla każdego łańcucha, na podstawie dot-bracket zostało otrzymane drzewo opisujące cząsteczkę. Drzewo w sposób schematyczny opisuje strukturę drugorzędową. Każdy wierzchołek posiada etykietę n oraz, jeśli n>1, n-1 dzieci. Odpowiada to sytuacji, gdy pętla rzędu n jest połączona helisami z n-1 pętlami. Oczywiście, n-tą helisą jest połączona z pętlą, która odpowiada jej rodzicowi na drzewie. Opis algorytmu, który został użyty do otrzymania drzewa znajduje się w dodatku.

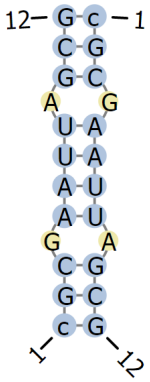
Zdefiniowano dwie funkcje, które zwracają odległość między drzewami, to znaczy im bardziej podobne są drzewa, tym mniejsza jest wartość tej funkcji. Dokładny opis funkcji znajduje się w dodatku.

Ponieważ wiele klastrów zawierało więcej niż dwa łańcuchy, napisano algorytm, który dla listy łańcuchów, zwraca drzewo podobieństwa. Idea algorytmu jest bardzo podobna do algorytmu najbliższego sąsiada stosowanego w filogenetyce. Dokładny opis znajduje się w dodatku.

Analiza wyników

Na podstawie przeanalizowanych wyników można wysunąć hipotezę, że jednym z najczęstszych przypadków było zaliczenie do klastra dwóch (lub więcej) łańcuchów tej samej cząsteczki – niemal połowa wszystkich klastrów. Często żaden z tych łańcuchów nie miał oddziaływań wewnętrznych, ale oddziaływał z drugim łańcuchem. Taki typ oddziaływań nie jest przedmiotem zainteresowania tego projektu. Warte jest jednak odnotowania, jak wiele cząsteczek jest złożonych z identycznych łańcuchów i wzajemnie komlementarnych na wszystkich lub na większości pozycji.

Przykład:
[['157d', <Chain id=B>], ['157d', <Chain id=A>]]



>strand_A
cGCGAAUUAGCG
(((.(((((.(((

>strand_B
cGCGAAUUAGCG
)))).)))))

Drugim typem klastra jest klaster z łańcuchami, które nie mają żadnych oddziaływań wewnętrznych – to znaczy oddziałujących jednie z innymi łańcuchami. Takich oddziaływań nie uwzględniamy w dalszej analizie.

Kolejnym typem klastra jest klaster, w którym znalazły się dwa łańcuchy o istotnie różnych strukturach drugorzędowych. Dla takich przykładów policzono liczbę pętli każdego rzędu oraz obliczono odległość w obu pseudometrykach.

Przykład:
W cząsteczkach o munerach indentyfikacyjnych 1cql oraz 1cq5 łańcuchy A miały identyczną sekwencję, jednak struktura drugorzędowa jest inna.

_1cql_A="((((((..(.)((((..((((((..))))))..).)))..)))))))"

_1cq5_A="(((((...(..((((..((((((..))))))..).)))))..))))))"

Obliczono liczbę pętli poszczególnych rzędów. Wynik działania programu:

_1cql_A ma 2 pętli rzędu 1
_1cq5_A ma 1 pętli rzędu 1
_1cql_A ma 2 pętli rzędu 2
_1cq5_A ma 4 pętli rzędu 2
_1cql_A ma 1 pętli rzędu 3
_1cq5_A ma 0 pętli rzędu 3

Ponadto obliczono odległości w obu pseudometrykch. Wyniosły one odpowiednio 9.30232558139535 oraz 2.449489742783178.

Najciekawszym typem klastrów są klastry, w których znalazło sie wiele łańcuchów o różnych strukturach drugorzędowych. Dla takich klastrów można uzyskać drzewo obrazujące stopień podobieństwa między strukturami.

Przykład:

Cząsteczki o numerach 1f78, 1h7h, 1f6x, 1f7h mają identyczne sekwencje, ale jedna z nich różni się strukturą drugorzędową.

Wejście:

```
_1f78_A=".((((.(((((.....)))))))))."
_1f7f_A=".((((.(((((.....)))))))))."
_1f6x_A=".((((.(((((.....)))))))))."
_1f7h_A=".((((.(.(.(.....).).)))))."

lista=[_1f78_A, _1f7f_A,_1f6x_A, _1f7h_A]

s = drzewo(lista, ["_1f78_A", "_1f7f_A","_1f6x_A", "_1f7h_A"], 1)
print(s[1])

s = drzewo(lista, ["_1f78_A", "_1f7f_A","_1f6x_A", "_1f7h_A"], 2)
print(s[1])
```

Wyjście:

```
[['_1f7h_A', ['_1f6x_A', ['_1f78_A', '_1f7f_A']]]]

[['_1f7h_A', ['_1f6x_A', ['_1f78_A', '_1f7f_A']]]]
```

Jak widać, obie funkcje dały to samo drzewo. Jest ono zgodne ze stanem faktycznym – to znaczy pokazuje ono, że łańcuch cząsteczki 1f7h jest różny od pozostałych łańcuchów.

Przykład 2.

Łańcuchy A cząsteczek 1f79, 1f7g, 1f71, 1f6z mają identyczną sekwencję, ale ich struktury drugorzędwe są inne, co zobrazowano na drzewie podobieństwa.

Wejście:

```
_1f79_A=".((((.(((((....)))))))))."
_1f7g_A="((((((.(((((....))))))))))"
```

```
_1f71_A=".(((((.(((((....))))))))))."
_1f6z_A="..((((.(((.(....).))))))).."

lista=[_1f79_A, _1f7g_A,_1f71_A, _1f6z_A]
s = drzewo(lista, ["_1f79_A", "_1f7g_A","_1f71_A", "_1f6z_A"], 1)
print(s[1])
s = drzewo(lista, ["_1f79_A", "_1f7g_A","_1f71_A", "_1f6z_A"], 2)
print(s[1])
```

Wyjście:

```
[['_1f6z_A', ['_1f7g_A', ['_1f79_A', '_1f71_A']]]]
[['_1f6z_A', ['_1f71_A', ['_1f79_A', '_1f7g_A']]]]
```

Jak widać, pierwsze drzewo jest bardziej zgodne z rzeczywistym podobieństwem struktur drugorzędowych. Nieodpowiednie działanie drugiej funkcji jest związane z tym, że liczba pętli poszczególnych rzędów jest identyczna w obu cząsteczkach, zatem drzewo jest tworzone niejako losowo.

Dodatek - opis funkcji

rozbicieNaLancuchy(a, b, sciezka) – dwa pierwsze argumenty to końce przedziału, w którym ma się zawierać długośc łańcucha (liczba reszt). Trzeci argument, to ścieżka do pliku, w którym opisana jest cząsteczka RNA. Funkcja uzupełnia listę listaLancuchow, której elementami są wszystkie łańcuchy o długości zawierającej się w przedziale [a,b].

przeszukiwanie() - funkcja uzupełnia listę redundantnych łańcuchów. Pobiera z listy łańcuch i tworzy listę redundantnych z nim elementów, do czego używa operacji porownanieLancuchow.

porownanieLancuchow(c1, c2) – funkcja zwraca 1 jeśli łańcuchy c1 i c2 mają taką samą sekwencję i zwraca 0 w przeciwnym przypadku.

ileKlastrow(n) – funkcja zwraca liczbę list z redundantnymi łańcuchami, które mają liczność n.

ileKlastrowZJednej() - funkcja zwraca liczbę list z redundantnymi łańcuchami, których elementy pochodzą z jednej cząsteczki RNA.

ileKlastrowZRoznych() - funkcja zwraca liczbę list z redundantnymi łańcuchami, których elementy pochodzą z różnych cząsteczek RNA.

koniec(napis)
argumentem funkcji jest string opisujący strukturę. Funkcja zwraca 1, jeśli string nie zawiera nawiasów, co odpowiada temu, że nie ma żadnych oddziaływań

zamień(napis, drzewoarg) - dla stringa opisującego strukturę drugorzędową I dotychczas stworzonego drzewa struktury, funkcja oblicza kolejny poziom drzewa. Przy pierwszym wywołaniu zlicza liście (spinki), następnie kolejne pętle.

uprość(napis) - dla stringa opisującego strukturę drugorzędową funkcja wywołuje zamień tak długo, aż opisane zostanie całe drzewo. Zwraca listę złożoną z uprzędzonego napisu, żądanego drzewa oraz pomocniczą tablicę.

ilePetli(string, rzad) - dla string opisującego strukturę drugorzędową oraz zadanego rzędu zwraca liczę pętli rzędu rzad w łańcuchu.

metryka1(s1, s2) - Zwraca procent liczby niezgodności w dot-bracketach dla dwóch sekwencji. Przyjmuje wartości od 0 (identyczne struktury 2-rzędowe) do 100.

metryka2(s1, s2) - Średnią kwadratową różnic w liczbie pętli poszczególnych rzędów.

d(x, y, par) - W zależności od parametru par używa metryki1 lub metryki2 do obliczenia odległości między strukturami drugorzędowymi.

sąsiad(lista, listaB, par) - dla listy struktur drugorzędowych oraz ich nazw znajduje elementy z listy o najmniejszej odległości względem metryki danej parametrem par oraz łączy te elementy w jedno poddrzewo.

drzewo(lista, listaB, par) - Wywołuje rekurencyjnie funkcję sąsiad aż do uzyskania pełnego drzewa podobieństwa. Zwraca to drzewo z wierzchołkami będącymi sekwencją dot-bracket oraz drzewo z wierzchołkami będącymi nazwami łańcuchów dla większej przejrzystości.