Projekt ZPR

Analiza widmowa sekwencji DNA

Krzysztof Grzyb
Tomasz Zieliński

15 stycznia 2013

1 Opis projektu

Temat

Obliczanie i wizualizacja widma transformaty Fouriera dla nietypowych danych np. sekwencji DNA. Sekwencję DNA możemy traktować jak napis, a ten z kolei, po zamianie na liczby, jako ciąg próbek na podstawie których można policzyć transformatę.

Wymagania funkcjonalne

Program jest uruchamiany z linii poleceń. Jako argument podawana jest nazwa pliku zawierającego sekwencje DNA. Dla każdej z sekwencji DNA wykonywana jest transformata Fouriera. Wyniki są wyświetlane na wykresie.

Format danych wejściowych

Pliki z danymi są zapisywane w różnych formatach z rozszerzeniem .dat. Parser sam rozpoznaje format pliku.

- **Typ 1**
  Plik typu tekstowego. Pierwszy wiersz zawiera jedną liczbę dodatnią - określa ona pozycję, na której następuje rozdział między egzonem a intronem. Następne wiersze występują w parach. Pierwszy mówi czy sekwencja DNA jest poprawna (1 poprawna, 0 niepoprawna), a drugi zawiera sekwencję DNA. Przykładowy plik z dwiema sekwencjami:

  ```
  7
  1
  CTCCGAAGTAGGATT
  1
  TCAGAAGGTGAGGGC
  ```

- **Typ 2**
  Plik typu tekstowego. Może składać się z wielu sekwencji DNA. Składnia sekwencji jest następująca: Najpierw występuje etykieta, a w następnej lini dane. Każda sekwencja musi posiadać 4 etykiety ,tóre będą sprawdzane:
  - Len - długość sekwencji DNA
  - Introns - w tych zakresach sekwencji DNA znajdują się introny
  - Exons - w tych zakresachsekwencji DNA znajdują się Introny

– Data - sekwencja DNA

Przykładowy plik:

```
>Seq 0 Len:
250
5UTR

Intergenic

Introns
10 55 100 130 150 170
Exons
56 99 180 200
3UTR

Data
AAGCTTATTATCTCTCCTTGACTCTCATCCGAGCTATCTTCTTCCACAT
CTCTCTCGTTCCTCGGCGCGAACCTCTCGCTTCTTCTCCTCTTACTCCGATTGAACGATTCCGGATCT
```

- **Inne formaty**
  Dodatkowo struktura programu umożliwia dodanie nowych obsługiwanych formatów bez ingerencji w achitekturę.

Interfejs programu i format danych wyjściowych

W przypadku nierozpoznanego formatu wejścia zwracany jest komunikat o błędzie, a program przechodzi do przetwarzania kolejnego pliku wejściowego.
Po sparsowaniu danych program przechodzi do wyliczania widma FFT dla poprawnie sparsowanych sekwencji. Następnie wyświetlane jest okno główne programu i ładowana lista sekwencji widm możliwych do obejrzenia (wyświetlanie: [numer sekwencji] [długość sekwencji]).
Program oferuje:

- Po zaznaczeniu sekwencji i naciśnięciu przycisku Chart program wyświetla w nowym oknie wykres widma danej sekwencji.
- Po zaznaczeniu sekwencji i naciśnięciu przycisku Save program zapisuje wykres wybranej sekwencji do pliku `spectrum.bmp` w katalogu z programem (Uwaga: w razie istnienia pliku nadpisuje go)
- Po naciśnieciu Save All wszystkie sekwencje są zapisywane do plików `.bmp`
- Istnieje możliwość wyświetlania/zapisywania wielu sekwencji na jednym wykresie po zaznaczeniu checkbox'a.

Po wykonaniu tranformaty Fouriera wyświetlana jest lista widm poszczególnych sekwencji. Użytkownik może wybrać dowolną z sekwencji i wyświetlić jej widmo. Podawane są podstawowe dane, takie jak liczba i długość sekwencji.

2 Rozwiązania programowe

Struktura programu

Program składa się z trzech głównych modułów:

Przetwarzanie plików wejściowych (Klasa `Parser`)

Moduł wczytuje dane z pliku, następnie rozpoznaje, jakiego typu dane zostały wczytane i przetwarza je do formatu używanego w programie, na którym możliwe jest zastosowanie FFT.

Nadrzędną klasą odpowiedzialną za parsowanie plików jest ParserFactory. Jest ona zaimplementowana jako Singleton i przechowuje kolekcję obiektów typu Parser. Poszczególne parsery dziedziczą po klasie abstrakcyjnej Parser i dokonuja rzeczywistego przekształcenia wejściowego ciągu znaków na wyjściowe próbki – wartości liczbowe typu double.

W pierwotnym zamierzeniu klasa Parserfactory miała tworzyć odpowiedni parser dla każdego pliku wejściowego. Jednak wydedukowanie formatu pliku a priori jest trudne, zwłaszcza dla formatów testowych – oba korzystają z rozszerzenia `.dat`. Odczytanie informacji o pliku w takim wypadku jest równoznaczne z przeprowadzeniem przynajmniej częściowego parsowania, zatem przyjęto inne rozwiązanie – każdy z parserów po kolei próbuje wczytać dany plik. Pozwala to również na dodanie kolejnych parserów w przyszłości bez zmiany istniejącego kodu.
Gdy któremukolwiek z parserów uda się przeprowadzić wczytywanie do końca, oznacza to sukces, natomiast gdy żadnemu z parserów to się nie uda – porażkę. Odpowiedni komunikat jest drukowany na standardowe wyjście.

Właściwe parsery – ParserSplice i ParserFullEx – wykorzystują bibliotekę `Boost:::Spirit`, w szczególności jej fragment Qi przeznaczony właśnie do tworzenia modularnych parserów. Każdy z parserów dziedziczy po klasie `Qi::grammar` i może służyć do budowy większych parserów za pomocą przeciążonego operatora `<<`. Dzięki temu na przykład możliwe jest wczytanie liczby typu `unsigned`, następnie pary tych liczb, ciągu par i w końcu całego formatu FullExOr.
Wczytywane dane są umieszczane bezpośrednio w strukturach lub też w wektorach struktur. Struktury danych tworzą hierarchię odpowiadającą hierarchii parserów. Wczytywanie danych bezpośrednio do struktur o dość dowolnym typie zdefiniownaym przez użytkownika (np. `FullExData`) jest możliwe dzięki makro `BOOST_FUSION_ADAPT_STRUCT`, które łączy pole struktury z jej typem. Parser dla odpowiedniej struktury umieszcza kolejno wczytane fragmenty w odpowiadających im polach.

Użyte parsery są restrykcyjne pod względem składni, to znaczy wymagają istnienia także pól, które nie są wczytywane do struktur – informacji pomocniczych nie będących przedmiotem zainteresowania, które możnaby pominąć. Zaliczają się do nich także m.in. znaki nowej linii obecne w pliku. Dodanie zbyt wielu opcjonalnych wyrazów do parsera praktycznie uniemożliwiało jego debugowanie, zanim zaczął działać. Dla pewności więc parsowany jest pełny format pliku.

Konwersja pomiędzy próbkami DNA (actg) i typem rzeczywistym (`double`) odbywa się dzięki funkcji convertACTG. Został tu wykorzystany algorytm `std::transform` i `boost::bind`. Para AC odpowiada przeciwnym wartościom, natomiast para TG wnosi pewną wartość stałą, dzięki czemu odpowiadające im widmo jest różne.

Obliczanie transformaty Fouriera (Klasa `Analyzer`)

Obliczaniem fft zajmuje się prosta klasa Analyzer, która krzysta z biblioteki GSL, a dokładniej jej fragmentu gsl_ftt aby dla wejściowego wektora próbek $x\{n\}$ wyliczyć FFT i zwrócić moduł widma $|X\{k\}|$.

epswidth=0width=0.9classnamed$_e$*xception*\endcsname

Biblioteka GSL operuje na niskopoziomowych strukturach danych POD, a połączenie ich ze strukturami wyspokopoziomowymi (np. klasą `Spectrum`) okazało się problematyczne. Funkcje bezpośredniego kopiowania pamięci między oboma rodzajami struktur niosą ryzyko wycieków czy przekroczenia zakresu, zatem wymagały dodatkowych starań, aby to ryzyko zminimalizować. Dodatkowo ewentualne błędy zasygnalizuje komunikat o wyjątku występującym w tej sekcji.

Wizualizacja wyników (Klasa `Viewer`)

Klasa Viewer jest abstrakcyjną klasą dostarczającą interfejs dla klas, które chciałyby wizualizować widma FFT. Dziedziczy po niej klasa `ChartViewer`, wyświetlająca widmo w postaci wykresu. Z kolei po klasie `ChartViewer` dziedziczy klasa `BMPFileViewer`, rozszerzająca możliwości swojej klasy bazowej o zapis wykresu do pliku.
Taki schemat umożliwia łatwe rozszerzanie funkcjonalności przez dodanie nowych klas. Będą one musiały jedynie implementować interfejs `Viewer`. W przypadku wykorzystania możliwości zapisu do pliku wystarczy użyć interfejsu `FileViewer`.

Hierarchia wyjątków

W programie została zaimplementowana hierarchia wyjątków pozwalająca reagować na nietypowe zdarzenia.

Styl kodowania

Zgodny z tymi wytycznymi

Wykorzystane biblioteki

- GSL (FFT)
- QT (interfejs graficzny)
- Boost (FOREACH, smart_ptr, assign, bind, lexical_cast, inne)
- QCustomPlot (Widget do Qt wyświetlający wykres transformaty)

Przenośność kodu

Kod był pisany w sposób niezależny od platformy. Testy zostły przeprowadzone na dwóch systemach: Linux i Windows. Do kodu źródłowego dołączony jest plik makefile (wraz z plikami, które dołącza) pozwalający na kompilację na systemach Linux i Windows.

3 Utrzymanie i rozszerzalność projektu

Możliwość rozszerzania funkcjonalności

Kod programu będzie można łatwo rozszerzać o nowe funkcjonalności:

- Obsługiwanie dodatkowych formatów wejściowych
- Inny sposób wizualizacji wyników