

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Informatyki

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Inżynieria Systemów Informatycznych

Porównanie architektur systemów
do badania pokrycia genomu

Marek Moraczyński

Numer albumu 261473

promotor
prof. dr hab. inż. Jan Mulawka

WARSZAWA 2018

Porównanie architektur systemów do badania pokrycia genomu

STRESZCZENIE

Niniejsza praca dyplomowa przedstawia porównanie różnych architektur do budowy systemów, które badają pokrycie genomu. Pokrycie genomu jest liczbą odczytów danego nukleotydu w procesie sekwencjonowania DNA. Jest to miara jakości sekwencjonowania, czyli procesu pozyskiwania kodu genetycznego. Wartość tą możemy zdefiniować dla każdego nukleotydu w genomie. Z uwagi na wielkość genomu, który ma ponad 3 miliardy nukleotydów na jednej nici, szczególnym wyzwaniem jest zapewnienie odpowiedniej wydajności aplikacji i zużycia pamięci dyskowej.

W pracy przedstawiono dwie różne architektury systemu. Jedna polega na wykorzystaniu technologii do dużych zbiorów danych takich jak Apache Spark oraz bibliotekę Adam. Drugie podejście wykorzystuje wstępne przetwarzanie danych i kompresję w dedykowanym formacie binarnym. Celem tej pracy jest uzyskanie odpowiedzi, która architektura lepiej sprawdza się w aplikacjach badających pokrycie genomu. By uzyskać odpowiedź na to pytanie zaimplementowano oba rozwiązania. Implementacja systemu bazuje na technologiach HTML5, Angular, .NET, Scala, Apache Spark. Wykonano również szereg badań wydajnościowych aplikacji w obu podejściach.

Niniejsza praca stanowi kontynuację i rozwinięcie innej pracy dyplomowej związanej z pokryciem genomu. Rozwój systemu jest konsultowany z badaczami pracującymi w Zakładzie Genetyki Medycznej Warszawskiego Uniwersytetu Medycznego.

Poza porównaniem dwóch różnych architektur, niniejsza praca ma walor dydaktyczny, gdyż wprowadza czytelnika w zagadnienia bioinformatyki. W związku z tym dość szczegółowo został opisany dorobek informatyczny tej nauki. Ponadto, rozważono przykładową analizę pokrycia genomu wykorzystując przy tym zaimplementowany system.

Słowa kluczowe: architektura systemów, bioinformatyka, sekwencjonowanie nowej generacji, genomika, pokrycie genomu, duże zbiory danych, kompresja

The comparison of systems architecture to examine genome coverage

ABSTRACT

The thesis describes comparison of systems architecture to examine genome coverage. The genome coverage is the amount of reading of a given nucleotide in the DNA sequence. It is a measure of the quality of genome sequencing. Sequencing is the process of determining the genetic code. Each nucleotide may have a different value of the genome coverage. Taking into consideration that the genome has more than 3 billion of nucleotides, the most challenging aspects of the application is performance and use of disk memory.

The thesis includes comparison of two different types of systems. The first one is based on dedicated tools for big data such as Apache Spark. Another concern pre-processing and compression to binary files. A purpose of this thesis is to find out which of architecture is more desirable. To achieve this both of architectures were implemented. Implementation based on .NET, Angular, Scala, Apache Spark, HTML5 technologies. To find out a final conclusion a number of performance tests were carried out.

This work is continuation of another thesis about genome coverage. The system development is consulted with specialists from the Medical University of Warsaw.

In addition to the comparison of architectures, the thesis also introduce reader to the bioinformatic issues. The informatic achievements have been described in detail. Furthermore, an exemplary genome coverage analysis was considered using implemented the system.

Keywords: system architecture, bioinformatics, next generation sequencing, genomics, genome coverage, big data, compression



Politechnika Warszawska

załącznik do zarządzenia nr 28/2016 r.
Rektora PW

„załącznik nr 3 do zarządzenia nr 24/2016 Rektora PW

.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płyce kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta”

Spis treści

1.	Wstęp	6
1.1.	Zakres i cel pracy	6
1.2.	Struktura pracy	7
2.	Zarys bioinformatyczny	8
2.1	Podstawowe pojęcia bioinformatyki	8
2.2	Wyrównanie sekwencji i pokrycie genomu	10
2.3	Projekty informatyczne związane z bioinformatyką	13
2.3.1	Formaty danych w bioinformatyce	13
2.3.2	Przegląd programów bioinformatycznych	15
2.3.3	Bioinformatyczne bazy danych	20
3	Omówienie architektur rozważanego systemu	24
3.1	Wymagania stawiane systemowi	24
3.2	Wykorzystane narzędzia informatyczne	26
3.3	Architektura oparta na plikach binarnych	27
3.4	Architektura oparta na platformie Spark	28
3.5	Hybrydowa architektura aplikacji	32
4	Implementacja systemu	33
4.1	Podział systemu na komponenty	33
4.2	Interfejs użytkownika	35
4.3	Warstwa serwerowa	37
4.4	Analiza pokrycia w technologii Spark	39
4.5	Testy systemu	40
5	Przeprowadzone badania	44
5.1	Wstępne rozważania na temat architektur	44
5.2	Analiza działania obu programów	46
5.3	Analiza pokrycia przy pomocy zaimplementowanego systemu	50
6	Wnioski	55
6.1	Zrealizowanie celu pracy	55
6.2	Możliwości dalszej rozbudowy	56
	Bibliografia	58
	Spis rysunków	61
	Spis tabel	62
	Spis załączników	63
	Dodatek A: Konfiguracja i uruchamianie aplikacji	64

1. Wstęp

Bioinformatyka jest młodą, interdyscyplinarną dziedziną wiedzy, która powstała w wyniku wykorzystania metod informatycznych w biologii. Jednym z obszarów wiedzy, którym zajmuje się ta nauka jest genomika. Badania nad genomami są natomiast skupione wokół sekwencjonowania. Z kolei sekwencjonowanie genomu jest procesem pozyskiwania informacji o kodzie genetycznym jakiegoś organizmu, a w szczególności człowieka. Na przestrzeni ostatnich lat rozwinęły się nowe metody pozyskiwania kodu genetycznego zwane sekwencjonowaniem nowej generacji (ang. *next generation sequencing*). Obecnie postęp we współczesnym sekwencjonowaniu genomu pozwala na pozyskiwanie informacji genetycznej w jednym laboratorium, w przeciągu dwóch tygodni przy koszcie kilku tysięcy złotych. Dla porównania pierwszy projekt związany z pozyskaniem genomu ludzkiego (ang. *Human Genome Project*) trwał od roku 1990 do 2001, zaangażował najlepsze laboratoria na świecie i pochłonął miliardy dolarów [1] [2].

Należy zauważyć, że postęp w bioinformatyce przyczynia się do pojawiania zupełnie nowych możliwości w medycynie, ale także otwiera nowe wyzwania. Nowoczesne techniki sekwencjonowania umożliwiają na przykład wykrywanie chorób genetycznych w pierwszym pokoleniu pojawienia się złośliwej mutacji. Rozwijają diagnostykę chorób nowotworowych. Co więcej odkryto, że wiele chorób, które nie były wcześniej uważane za mające źródło w nieprawidłowym kodzie genetycznym faktycznie należą do chorób genetycznych. Przykładem może być schizofrenia czy inne choroby psychiczne. Ponadto rozwija się zupełnie nowa gałąź medycyny, tak zwana medycyna spersonalizowana, która dostosowuje leki do konkretnego pacjenta [3]. Współcześnie pozyskanie kodu genetycznego nie jest już tak problematyczne, jednak w procesie tym powstaje bardzo wiele informacji. Coraz trudniejsze staje się ich przechowywanie i analiza. Rośnie więc zapotrzebowanie na narzędzia informatyczne, które będą w stanie poradzić sobie z ową wielką ilością danych.

Jednym z podstawowych pojęć, które jest związane z sekwencjonowaniem nowej generacji jest pokrycie genomu (ang. *genome coverage*), zwana także jako głębokość odczytu (ang. *coverage depth*). Wartość ta jest liczbą odczytów danego nukleotydu, czyli podstawowej jednostki strukturalnej kodu genetycznego w porównaniu do znanej sekwencji referencyjnej. Pokrycie genomu mówi przede wszystkim o pewności eksperymentu. Istnieje szereg publikacji określających jakiej głębokości odczytu powinniśmy stosować przy konkretnych eksperymentach. Przykładowymi badaniami (w których ważna jest ta wartość) są znajdowanie mutacji genowych czy wykrywanie polimorfizmu pojedynczego genu [4].

Należy wspomnieć, że niniejszy tekst stanowi rozszerzenie wcześniejszej pracy tego samego autora zatytułowanej „Tworzenie aplikacji do badania pokrycia genomu” [5].

1.1. Zakres i cel pracy

Przedmiotem tej pracy jest opracowanie nowej architektury do badania pokrycia genomu oraz jej implementacja. Praca powstała we współpracy z Zakładem Genetyki Medycznej

Warszawskiego Uniwersytetu Medycznego. Odbiorcami oprogramowania są badacze związani z genomiką.

Celem badań pracy jest porównanie dwóch architektur systemów do zastosowania w analizach głębokości pokrycia genomu. Niniejsze badania mają odpowiedzieć na pytanie, które podejście jest lepsze dla konkretnego zastosowania.

Pierwsze podejście zostało wypracowane we wcześniejszej pracy [5]. Podejście to opiera się na wstępnym przetworzeniu danych wejściowych do plików binarnych. Dzięki takiemu podejściu zostały uzyskane małe pliki binarne, które pozwalają w sposób szybki uzyskiwać informacje o pokryciu nawet dla wielu próbek.

Inny sposób realizacji architektury polega na wykorzystaniu rozwiązań informatycznych dedykowanych do radzenia sobie z dużymi zbiorami danych (ang. *big data*). Bardziej konkretnie chodzi tu o język Scala, technologię Spark oraz bibliotekę dedykowaną do rozwiązań genomicznych, a mianowicie bibliotekę Adam.

1.2. Struktura pracy

Niniejsza praca składa się z sześciu rozdziałów. Pierwszy stanowi krótki wstęp. Kolejne rozdziały obejmują opisane poniżej zagadnienia.

W drugim rozdziale przedstawiono podstawowe pojęcia bioinformatyczne istotne w celu dalszego zrozumienia pracy. Bardziej szczegółowo zostaną opisane pojęcia takie jak wyrównanie sekwencji czy pokrycie genomu. Ze względu na mnogość rozwiązań informatycznych zostaną również przedstawione obecne popularne programy, bazy danych i formaty plików dla zastosowań w genomice.

W trzecim rozdziale bardziej szczegółowo zostaną opisane wymagania, które są postawione przed systemem informatycznym związanym z badaniem pokrycia. Ponadto rozdział ten zawiera opis narzędzi informatycznych wykorzystanych w pracy. Ze względu na charakter pracy, czyli porównanie dwóch różnych architektur jest ich wiele. Na końcu zostaną opisane bardziej szczegółowo obie architektury.

Czwarty rozdział przedstawia opis implementacji zbudowanego systemu. Oddzielnie omówiono implementację warstwy użytkownika oraz warstwy serwerowej. Co więcej, opisano również testy wykonane w celu sprawdzenia poprawności aplikacji.

Przedostatni rozdział skupia się na przeprowadzonych badaniach aplikacji. W nim przedstawione zostanie porównanie dwóch architektur aplikacji oraz wyszczególnienie wad i zalet obydwu z podejść. Opisano również przykładową analizę pokrycia z użyciem zaimplementowanego systemu.

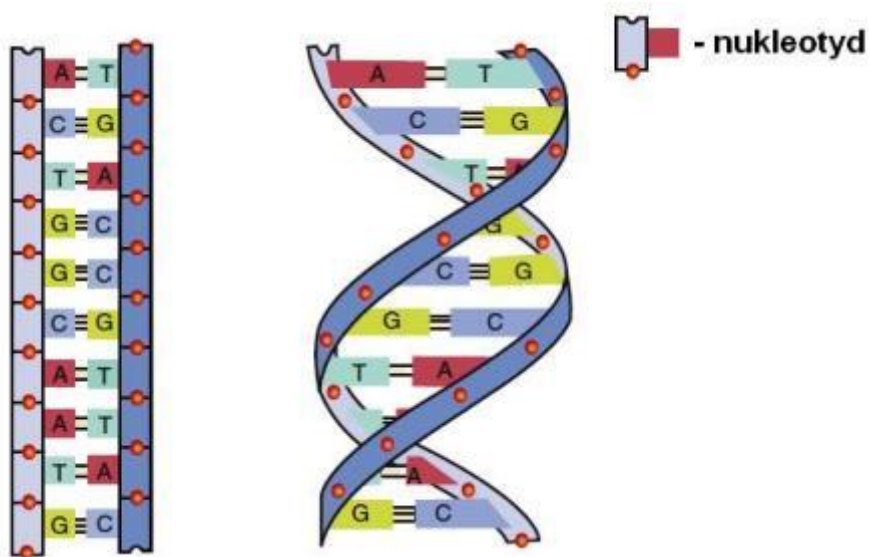
W ostatnim rozdziale zawarte jest podsumowanie przeprowadzonych badań. System ma wiele możliwości dalszego rozwoju, więc w ostatnim podrozdziale opisano kilka koncepcji, które mogą w przyszłości rozszerzyć pracę.

2. Zarys bioinformatyczny

Niniejszy rozdział wprowadza w pojęcia bioinformatyczne niezbędne do zrozumienia działania budowanego systemu. W większym stopniu zostaną opisane pojęcia wyrównania sekwencji i pokrycia genomu, które są szczególnie istotne w kontekście tworzonej pracy. Ostatni podrozdział opisuje dorobek informatyczny, czyli bazy danych, programy oraz formaty plików szczególnie popularne w bioinformatyce.

2.1 Podstawowe pojęcia bioinformatyki

Jak powszechnie wiadomo, dwoma podstawowymi kwasami nukleinowymi są DNA (kwas deoksyrybonukleinowy) i RNA (kwas rybonukleinowy). Podstawową jednostką strukturalną kodu genetycznego człowieka są nukleotydy. W DNA wyróżniamy cztery rodzaje nukleotydów adenina (A), guanina (G), cytozyna (C), tymina (T). Natomiast w RNA zamiast tyminy (T) występuje uracyl (U). Nukleotydy łączą się ze sobą na zasadach komplementarności. Reguła ta mówi, że guanina łączy się z cytozyną, a adenina łączy się z tyminą w DNA, a w RNA z uracylem. Z zasady tej wynika nazwa podstawowej jednostki długości kwasów nukleinowych, czyli liczby par zasad (ang. *base pair*). Na Rys. 1 zaprezentowana jest zasada komplementarności DNA.



Rys. 1 Nić DNA ilustrująca zasadę komplementarności [6]

Zbiorem nukleotydów mogą być przypisywane pewne funkcje, a wydzielone logicznie fragmenty często określane są mianem genów. Geny leżą w chromosomach. U człowieka mamy 22 pary autosomów i po jednej parze heterosomów. Oznaczone są odpowiednio chr1...chr22, chrX i chrY. 23 chromosomy składają się na genom. Można więc powiedzieć, że chromosomy są kontenerem na geny, a genom jest kontenerem na chromosomy.

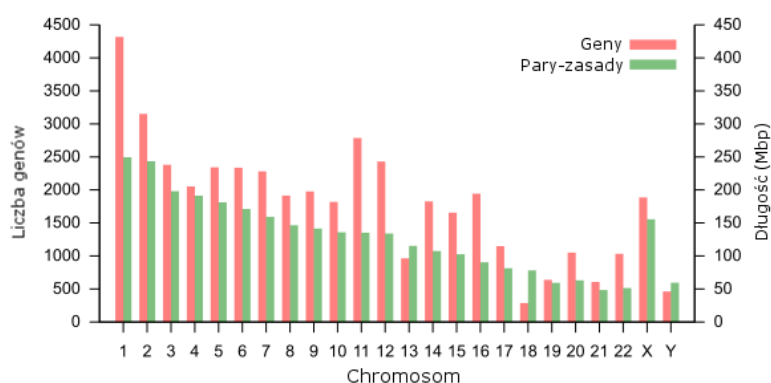
Sekwencje nukleotydów stanowią informację genetyczną człowieka, która zaś mówi o sposobie w jakim tworzone są białka. Białka są podstawowymi elementami, z których zbudowane są organizmy żywe. Pełnią one niezbędne funkcje życiowe takie jak funkcje budulcowe, magazynowe, transportujące. Proces tworzenia białek w organizmie człowieka

nazywany jest biosyntezą białek. Można tu wyróżnić dwa procesy transkrypcję i translację. Pierwszy z nich to przepisywanie DNA na RNA, drugi zaś jest procesem syntezy białek na matrycy mRNA.

Przyczynami chorób mających źródło w kodzie genetycznym są mutacje, czyli nagłe zmiany w materiale genetycznym. Mutacje mogą zachodzić spontanicznie lub być wymuszone przez tak zwane mutageny. Mutacje dzielimy na genowe, aberracje strukturalne chromosomów i genomowe. Do pierwszej kategorii należą zmiany pojedynczych nukleotydów jak substytucje (zamiana nukleotydu innym), insercje (dodanie dodatkowego nukleotydu), delecje (brak jakiegoś nukleotydu). Druga kategoria - aberracje strukturalne chromosomów mówi zaś o zmianach poszczególnych chromosomów jak utrata fragmentu chromosomu czy obrócenie fragmentu o 180 stopni. Ostatnia kategoria – zmiany genomowe to mutacje, które skutkują zmianą liczby chromosomów tak, że mamy ich nadmiar lub niedobór [7].

Sekwencje nukleotydów możemy podzielić na dwa rodzaje, czyli sekwencje kodujące białka – eksony i niekodujące – introny. Pierwsze z nich są o wiele bardziej istotne we wszelkiego rodzaju badaniach, gdyż zmiana w eksonie bezpośrednio oddziałuje na nasz zbiór cech zewnętrznych zwany fenotypem. Większość chorób genetycznych jest, więc upatrywana w eksonach. Natomiast introny są poddawane często procesowi składania genów (ang. *splicing*), polegającym na ich wycinaniu. Podczas tego procesu, zwanego również splataniem eksony są łączone ze sobą. Jednak nie muszą one łączyć się w ściśle określonej kolejności, dlatego też mamy wiele wariantów splatania (ang. *alternative splicing*) dla jednego genu [7].

Jednostka długości pary-zasady jest używana do określenia długości sekwencji, genów, chromosomów i genomów. Różne źródła podają długość genomu od 3 do 3,5 miliarda par zasad. W pracy Piotra Węglańskiego wartość ta określana jest rozmiarem 3,3 miliarda par zasad [7]. Podział ze względu na chromosomy oraz ilość poszczególnych genów w chromosomach zaprezentowana jest na Rys. 2.



Rys. 2 Długości chromosomów wraz z ilością genów [8]

W ostatnio potwierdzonym odkryciu okazało się, że nie tylko sekwencja nukleotydów stanowi źródło informacji genetycznej. W 2016 roku Helmut Schiessel ze swoją grupą badaczy odkrył, że informacja ta jest zawarta również za sprawą ułożenia nici w nukleosomach. Nukleosom jest jednostką strukturalną chromatyny składająca się z odcinka DNA o długości

około 200 par zasad. Chromatyna jest zbudowana z DNA, histonów i niehistonowych białek. Jest to włóknista substancja i stanowi główny składnik chromosomów [9].

2.2 Wyrównanie sekwencji i pokrycie genomu

Bardzo ważnym pojęciem w badaniach bioinformatycznych jest dopasowanie sekwencji (ang. *sequence alignment*). Jest to sposób porównywania dwóch lub więcej sekwencji kwasów nukleinowych bądź aminokwasów. Dopasowanie przedstawiane jest głównie jako macierz z rzędami, którymi są sekwencje, a kolumnami są dopasowane nukleotydy. Pojęcie te używane jest również w badaniach nad językami naturalnymi i w analizie danych finansowych, gdzie liczona jest odległość pomiędzy łańcuchami znaków. Krótkie sekwencje są często analizowane ręcznie, bez pomocy komputera. Natomiast bardziej interesujące są długie, złożone sekwencje. Wyróżniamy dopasowania lokalne, globalne oraz mieszane znane jako metody semiglobalne. Różnice pomiędzy dopasowaniem lokalnym i globalnym dobrze przedstawia Rys. 3. Techniki dopasowania globalnego są użyteczne, gdy chcemy znaleźć podobieństwo dla całej sekwencji. Natomiast dopasowanie lokalne charakteryzuje się podobieństwami w niektórych fragmentach sekwencji. Metody semiglobalne skupiają się na znalezieniu dopasowania początku lub końca sekwencji. Są one użyteczne w przypadkach, gdy początek jednej sekwencji zachodzi na koniec drugiej sekwencji.



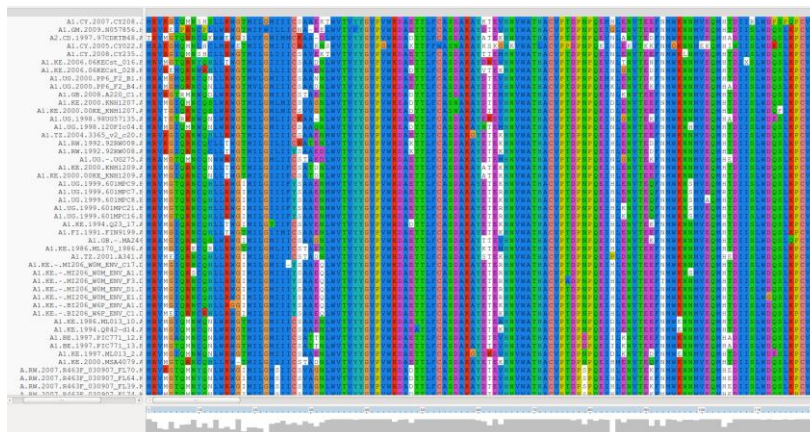
Rys. 3 Różnica pomiędzy dopasowaniem lokalnym i globalnym [10]

Istnieje wiele algorytmów, które znajdują podobieństwo pomiędzy łańcuchami znaków. Algorytmy realizujące te zadanie pochodzą z różnych kategorii na przykład programowania dynamicznego, metod heurystycznych czy probabilistycznych. Dwoma bardzo znanymi metodami programowania dynamicznego są algorytmy Needleman-Wunsh'a i Smith'a-Waterman'a. Pierwszy z nich ma zastosowanie do dopasowania globalnych, zaś drugi do lokalnych. W metodach tych najczęściej liczone są punkty za poprawne dopasowanie i kary za przerwy. Utrudnieniem dla tych algorytmów są insercje i delecje, a w celu zlikwidowania ich niekorzystnego wkładu stosuje się metody wyszukiwania ramki (ang. *framesearch*) [11].

Szczególnym przypadkiem jest znajdowanie dopasowania dla par sekwencji. Wiele algorytmów jest szybsza i bardziej dopasowana właśnie do dwóch łańcuchów. Używane są do zadań niewymagających dużej precyzji jak na przykład szukanie w bazie wskazanego łańcuchu. Wspólnym problemem dla wszystkich algorytmów są sekwencje, które zawierają powtarzające się łańcuchy nukleotydów. Metody, które służą do znajdowania par sekwencji to wspomniane wcześniej metody programowania dynamicznego, a także metoda słów (ang. *dot matrix*). Warto wspomnieć, że znajdowanie dopasowania sekwencji jest dużo bardziej istotne dla sekwencji kodujących niż niekodujących.

Rozszerzeniem metod znajdowania dwóch łańcuchów są metody znajdowania wielu łańcuchów oznaczane skrótem MSA (ang. *multiple sequence alignment*). MSA ma zastosowanie w znajdowaniu zależności ewolucyjnych na przykład przy tworzeniu drzew filogenetycznych. Drzewa takie przedstawiają zależności ewolucyjne między sekwencjami lub gatunkami organizmów żywych. Mamy dwa powody, dla których drzewa filogenetyczne są użyteczne. Pierwszy to fakt, że domeny funkcyjne znalezione w jednej sekwencji mogą być użyte w innej, niezbadanej wcześniej. Drugi to możliwość znalezienia regionów zachowanych (ang. *conserved regions*). Regiony zachowane są to miejsca w dwóch lub więcej sekwencjach, które są identyczne lub bardzo podobne [12].

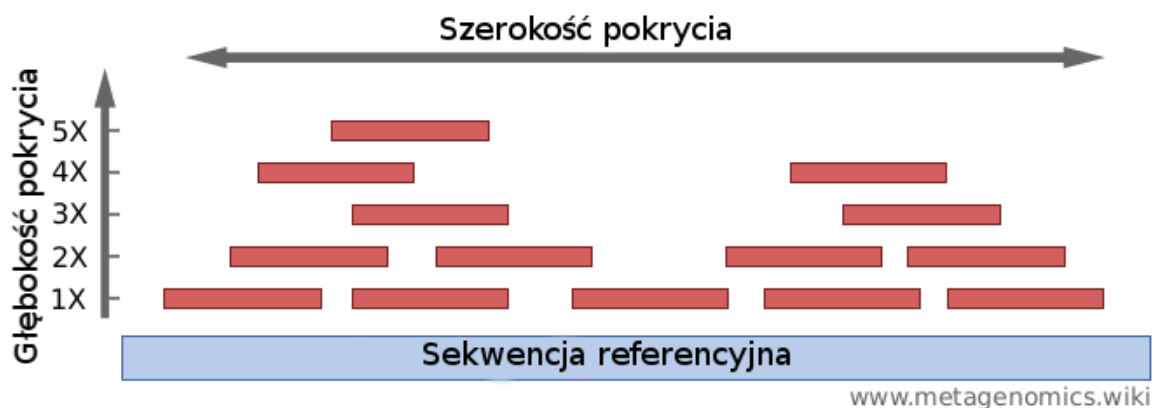
Znajdowanie dopasowania wielu łańcuchów wymaga dużo bardziej wyrafinowanych algorytmów niż metody znajdowania par podobnych. Większość z nich korzysta z metod heurystycznych zamiast szukać rozwiązania optymalnego globalnie. Często tworzone są na podstawie dwójkowych. Ponadto bardzo często w dopasowaniu sekwencji występują błędy. Badania wykonane przy pomocy programu BALiBase pokazały, że w programach wykorzystywanych do dopasowania genomu ponad 24% wszystkich par jest niepoprawnie wyrównanych. W celu analizy tych błędów stosuje się programy wizualizujące błędy przy dopasowaniu sekwencji, które później badacze wykorzystują przy analizie sekwencji. Przykładami takiego oprogramowania może być JalView czy UGENE. Istnieją również sposoby, które automatyzują proces wykluczania błędów dopasowania. Mimo wszystko automatyczne metody również nie są idealne. Niektóre z programów stosują też metody pokazywania pewności otrzymanego wyniku jak na przykład program FSA. Przykładowa analiza metody wielu łańcuch przy pomocy programu ClustalX znajduje się na Rys. 4.



Rys. 4 Wizualizacja MSA pochodząca z programu ClustalX [13]

Pokrycie genomu (ang. *genome coverage*) jest interesującą wartością dla badaczy zajmujących się sekwencjowaniem genomu. W literaturze naukowej można spotkać też je pod nazwą głębokości genomu (ang. *depth of coverage*). Wartość ta wykorzystywana jest przede wszystkim do oceny pewności eksperymentu. Wysoka wartość pokrycia jest wartością pożądaną. Wiele artykułów naukowych i książek opisuje badania wraz z zakładanym pokryciem. Jedne z takich zestawień możemy odnaleźć na stronie producenta maszyn do sekwencjonowania Illumina [4].

W celu wyjaśnienia szczegółów obliczania pokrycia weźmy pod uwagę Rys. 5.



Rys. 5 Ilustracja pojęć pokrycia i szerokości genomu [15]

Sekwencjonowanie w wyniku nie daje jednej długiej sekwencji, a wiele krótkich sekwencji, które są potem rekonstruowane na sekwencji referencyjnej. Krótkie sekwencje zobrazowane są jako bordowe prostokąty. Głębokość pokrycia mówi nam, ile razy dany nukleotyd z sekwencji referencyjnej był odczytywany w genomie. Istnieje także inna wartość szerokość pokrycia (ang. *coverage breadth*). Mówi ona jaki procent genomu (lub wybranej sekwencji) jest pokryta przez odczyty [15].

Innym przykładem, który może dać pewne zrozumienie obu wartości jest wynik działania programu bedtools. Wynik zilustrowano na Rys. 6. Pierwsze dwa bloki przedstawiają pliki z danymi (odpowiednio A.bed i B.bed). Są to odpowiedniki sekwencji referencyjnej (A.bed) i sekwencji, która została zsekwencjonowana (B.bed). Następne dwa bloki przedstawiają wynik działania programu bedtools. W sekwencji A.bed mieliśmy podany chromosom pierwszy od 0 do 100 (chr1 0 100). W drugiej zaś (B.bed) 3 kolejne sekwencje o długości 10 zawierają się w tej z pliku A.bed. W wynikowym pliku otrzymamy więc dla sekwencji chr1:0-100 szerokość pokrycia 0.3. Drugi blok wynikowy przedstawia podobną sytuację, ale do programu została dodatkowo podana opcja „-s”. Opcja ta pozwala odróżniać wyniki pokrycia ze względu na nić DNA, ponieważ jak wiemy kod genetyczny człowieka jest dwuniciowy. Poszczególne nici rozróżniane są za pomocą znaków plus (+) i minus (-).

```

$ cat A.bed
chr1 0 100 b1 1 +
chr1 100 200 b2 1 -
chr2 0 100 b3 1 +

$ cat B.bed
chr1 10 20 a1 1 -
chr1 20 30 a2 1 -
chr1 30 40 a3 1 -
chr1 100 200 a4 1 +

$ bedtools coverage -a A.bed -b B.bed
chr1 0 100 b1 1 + 3 30 100 0.3000000
chr1 100 200 b2 1 - 1 100 100 1.0000000
chr2 0 100 b3 1 + 0 0 100 0.0000000

$ bedtools coverage -a A.bed -b B.bed -s
chr1 0 100 b1 1 + 0 0 100 0.0000000
chr1 100 200 b2 1 - 0 0 100 0.0000000
chr2 0 100 b3 1 + 0 0 100 0.0000000

```

Rys. 6 Wynik działania programu bedtools [16]

Cechą pokrycia jest to, że jest ona wartością bardzo często taką samą na kolejnych odcinkach. Problemami w analizie pokrycia jest ilość danych. Długość genomu jest rzędu 3,3 miliarda par zasad, co może mówić o wielkości danych z jaką mamy do czynienia.

2.3 Projekty informatyczne związane z bioinformatyką

Dzisiejsze badania bioinformatyczne nie były by możliwe bez współczesnej informatyki. Dane zawarte w DNA czy informacje o białkach są zazwyczaj bardzo dużych rozmiarów. Stąd wynika duże zapotrzebowanie na stosowanie wyspecjalizowanych baz danych i oprogramowania. Dodatkowo sposób przechowywania danych również jest bardzo różny. Może różnić się on w zależności od konkretnych potrzeb. Ze względu na wielkość danych w bioinformatyce mamy duże zróżnicowanie formatów wejściowych. W kolejnych podrozdziałach zostaną omówione przykładowe formaty, bazy danych i oprogramowanie.

2.3.1 Formaty danych w bioinformatyce

Wśród specjalistów można spotkać się ze stwierdzeniem, że każdy nowy program bioinformatyczny to nowy format danych. Aplikacja, która została stworzona podczas niniejszej pracy nie jest tu wyjątkiem, gdyż zastosowano niestandardowe przechowywanie danych w postaci plików binarnych. Wyjaśnieniem tego zjawiska może być wielkość danych w bioinformatyce. Większość programów do analizy danych pochodzących z sekwencjonowania nie akceptuje znaków spoza tablicy ASCII [17]. Najbardziej popularne formaty opisane są poniżej. Jednak jest ich znacznie więcej jak na przykład CRAM, GFF, GTF,

HAL, MAF, WIG, VCF, PSL. Opis różnych formatów danych jest zgromadzony na stronie Uniwersytetu Kalifornia Santa Cruz <http://genome.ucsc.edu/FAQ/FAQformat.html>.

Przykładowe, najpopularniejsze formaty danych, które są stosowane w bioinformatyce:

- **BED (Browser Extensible Data)** - format jest używany jako wejście do programów optymalizujących dane. Jest to plik tekstowy z trzema wymaganymi polami i dziewięcioma opcjonalnymi. Najbardziej istotne pola to oznaczenie chromosomu, start chromosomu oraz jego koniec. Istnieją różne warianty tego formatu danych. Format jest wynikiem działania programu *bedtools*. Dwa najpopularniejsze z nich to *bigBed* i *BedDetailFormat*. *BigBed* jest używany, gdy zbiór danych jest bardzo duży (co najmniej ponad 50 megabajtów danych) wtedy jest on rekomendowanym formatem. Tworzony jest on przez program *bedToBigBed*. W przeciwieństwie do formatu BED używa plików binarnych. Jeżeli chodzi o *BedDetailFormat* format ten używa dodatkowych kolumn. Rys. 7 prezentuje przykładową sekwencję w tym formacie.

```
browser position chr7:127471196-127495720
browser hide all
track name="ItemRGBDemo" description="Item RGB demonstration" visibility=2 itemRgb="On"
chr7 127471196 127472363 Pos1 0 + 127471196 127472363 255,0,0
chr7 127472363 127473530 Pos2 0 + 127472363 127473530 255,0,0
chr7 127473530 127474697 Pos3 0 + 127473530 127474697 255,0,0
chr7 127474697 127475864 Pos4 0 + 127474697 127475864 255,0,0
chr7 127475864 127477031 Neg1 0 - 127475864 127477031 0,0,255
chr7 127477031 127478198 Neg2 0 - 127477031 127478198 0,0,255
chr7 127478198 127479365 Neg3 0 - 127478198 127479365 0,0,255
chr7 127479365 127480532 Pos5 0 + 127479365 127480532 255,0,0
chr7 127480532 127481699 Neg4 0 - 127480532 127481699 0,0,255
```

Rys. 7 Przykład formatu BED [18]

- **GenBank** – GenBank format składa się z dwóch sekcji: nagłówka z adnotacjami i sekwencji. Każda linia sekwencji zaczyna się liczbą występujących w niej nukleotydów [19].
- **FASTA** – popularny format stosowany w programach bioinformatycznych. Został stworzony wraz z oprogramowaniem o tej samej nazwie. Obecnie używany w programach takich jak BLAST (wyszukiwarka sekwencji) czy w programach do wyrównania sekwencji jak Clustal czy T-Coffee. W tym formacie każda sekwencja startuje od znaku „>”. Pierwsze słowo w sekwencji jest jej identyfikatorem, a następnie zawarty jest jej opis.
- **SAM (Sequence Alignment/Map)** – popularny format tekstowy. SAM jest używany do wyrównania sekwencji. Format składa się z opcjonalnego nagłówka i sekcji zawierającej dane o wyrównaniu sekwencji. Przykład tego pliku zaprezentowany jest na Rys. 8.

```

Coor      12345678901234  5678901234567890123456789012345
ref       AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT

+r001/1   TTAGATAAAGGATA*CTG
+r002     aaaAGATAA*GGATA
+r003     gcctaAGCTAA
+r004           ATAGCT.....TCAGC
-r003           ttagctTAGGC
-r001/2           CAGCGGCAT

```

Rys. 8 Przykład formatu SAM [11]

- **BAM** – wersja binarna formatu SAM. Format ten używa kompresji BGZF w celu wydajnego losowego dostępu do pliku. BGZF to specjalna wersja popularnej metody kompresji GZIP, a co ciekawe pliki BGZF są większe niż normalne pliki GZIP. Stosowanie BGZF jest jednak sensowne, gdyż używa on wirtualnych przesunięć w pliku by zwiększyć szybkość losowego dostępu. Bardziej szczegółowy opis można znaleźć w specyfikacji formatu SAM i BAM [11].
- **EMBL (European Molecular Biology Laboratory)** - w formacie EMBL nagłówek zaczyna się z identyfikatorem ID. Sekwencja zaczyna się ze słowem „SQ” i kończy się przez oznaczenie dwoma ukośnikami.
- **PDB (Protein Data Bank)** – format i jednocześnie wyspecjalizowana baza do przechowywania protein i kwasów nukleinowych. Dane w tym formacie są dostępne dla biologów z całego świata i nadzorowane przez organizację wwPDB [20]. Szczegółowa dokumentacja tego formatu dostępna jest w serwisie wwpdb.org pod adresem <http://www.wwpdb.org/documentation/file-format>.

2.3.2 Przegląd programów bioinformatycznych

Lista programów bioinformatycznych jest bardzo długa. Pewną specyfiką programów bioinformatycznych jest to, że często mają kod otwarto źródłowy (ang. *open source*). Bioinformatyka nie jest wyjątkiem, jeżeli chodzi o ogólnie panujący trend przenoszenia coraz większej liczby rozwiązań do wersji przeglądarkowej. Przykładem może być ExaC czy UCSC Genome Browser. Programy te różnią się zastosowaniem, obsługiwanymi formatami, licencjami czy systemami operacyjnymi. Szczególna mnogość oprogramowania znajduje zastosowanie w dopasowaniu sekwencji. Można bowiem wyróżnić tu oprogramowanie służące do różnych zastosowań

- Przeszukiwania bazy, a nie wykonywania faktycznego dopasowania. Na przykład: *BLAST*, *FASTA*.
- Dopasowania sekwencji dla par. Na przykład: *BioPerl*, *mAlign*.
- Dopasowania wielu sekwencji. Na przykład: *FSA*, *Clustal*, *T-Coffee*.
- Analizy genów. Na przykład: *MGA*, *AVID*.
- Znajdowaniu motywów. Na przykład *PMS*, *MERCI*.

- Wizualizacji dopasowania sekwencji. Na przykład: *Integrated Genome Browser*, *ClustalX*, *Jalview 2*.
- Testowania sprawności algorytmów. Na przykład: *SABmark*, *SMART*.
- Krótkiego odczytywania sekwencji. Na przykład: *BWA*, *GEM*.

Szczegółową listę oprogramowania do dopasowania genów, jak również oprogramowania otwarcie źródłowego do celów bioinformatycznych, można znaleźć na stronach angielskiej wikipedii [21] [22].

Innymi kategoriami, w których możemy znaleźć mnogość oprogramowania może być:

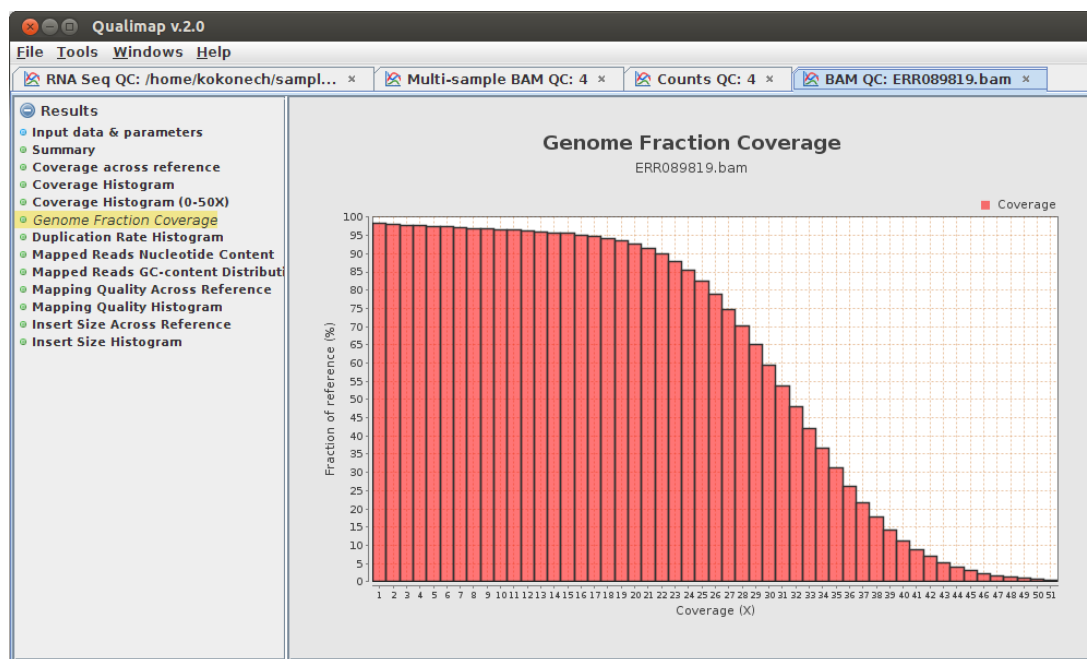
- Predykcja genów. Na przykład: *GENIUS*, *GeneMark*.
- Istotne w kontekście niniejszej pracy oprogramowanie do badania pokrycia genomu. Na przykład: *Bedtools*, *GATK*, *FreeBayes*.
- Oprogramowanie do wizualizacji bioinformatycznej. Na przykład: *DICOM*, *Ambra Health*, *VistA*.
- Modelowanie biomolekularne. Na przykład: *AMPHORA*, *MEGAN*, *UniFrac*.

Opis kilku najbardziej popularnych programów znajduje się poniżej. Istnieje jednak wiele innych ważnych programów nieopisanych w poniższym zestawieniu. Takich jak *BioPerl*, *BFAST*, *UGENE*, *Bioinformatics On Windows*.

Freebayes – to program używany w celu znajdowania polimorfizmu pojedynczego nukleotydu (ang. *single nucleotide polymorphisms*), insercji i delecji (ang. *insertions and deletions*), polimorfizmu wielu nukleotydów (ang. *multi-nucleotide polymorphisms*). Formaty używane w programie to BAM, FASTA, VCF, BED. Jest to program otwarcie źródłowy dostępny w serwisie github pod adresem: <https://github.com/ekg/freebayes>.

GATK (Genome analysis toolkit) – to pakiet oprogramowania używany i zaprojektowany przez *Broad Institute*. Jest bez żadnych opłat dla użytku niekomercyjnego i bardzo drogi w przeciwnym wypadku. Podobnie jak FreeBayes, GATK jest używany w celu znajdowania SNP, insercji i delecji. Ponadto może być używany do obliczania pokrycia genomu, obliczania wariacji kopi DNA (ang. *copy number variation*) i strukturalnej wariacji (ang. *structural variation*). GATK działa wraz z popularnymi formatami takimi jak SAM, BAM, CRAM i VCF. W celu obliczania pokrycia genomu w plikach BAM używa wzorca mapowania i redukcji (ang. *map reduce*) [23] [24].

QualiMap – program o otwartym kodzie źródłowym, który może być używany w każdym systemie operacyjnym. QualiMap współpracuje z formatami SAM, BAM, GFF. Wspiera wszystkie popularne rodzaje sekwencjonowania, czyli sekwencjonowania całych genów czy tylko eksonów, a także sekwencjonowanie RNA i ChIP [25]. Zrzut ekranu z tego programu zaprezentowany jest na Rys. 9.



Rys. 9 Zrzut obrazu z programu QualiMap [25]

.NET Bio – stworzony przez firmę Microsoft program do zastosowań bioinformatycznych. Początkowo znany pod nazwą Microsoft Biology Foundation. Napisany w języku C# pod platformą .NET. Możliwości tego programu to odczytywanie dopasowania z formatów FASTA i GenBank, znajdowania lokalnych i globalnych dopasowań za pomocą wskazanych algorytmów, dostęp do serwisów takich jak NCBI BLAST i wyszukiwanie w tych bazach interesujących fragmentów, zastosowanie wybranych algorytmów do składania sekwencji. Kod źródłowy i program są dostępne pod adresem <http://bio.codeplex.com/>. Środowisko .NET nie jest jedyne dla bioinformatyki, które w nazwie ma technologię, w której powstało. Istnieją również programy o nazwach BioPython, BioPerl, BioRuby czy BioJava.

FSA (ang. *fast statistical alignment*) – program do dopasowania wielu sekwencji. Wykorzystywany zarówno dla protein, RNA jak i DNA. Formatami obsługiwanymi przez program są FASTA i format sztokholmski. Używany w sekwencjonowaniu nowych genomów robaków czy czynników transkrypcyjnych.

Samtools - również projekt otwarcie źródłowy. Dostarcza narzędzi do badania wyrównania pokrycia. Więcej informacji można znaleźć na stronie narzędzia <http://samtools.sourceforge.net/>.

Fasta – służy do znajdowania dopasowania sekwencji w DNA i proteinach. Podczas tworzenia tego pakietu oprogramowania powstał format o tej samej nazwie, który obecnie jest wszechobecny w zastosowaniach bioinformatycznych. Nazwa pochodzi od złożenia wyrazów FAST-All, gdzie słowo „all”, czyli z angielskiego – wszystko, odnosi się do wyrównania na przykład protein, nukleotydów. Można spotkać też nazwy poszczególnych rozszerzeń FAST-P, gdzie P oznacza proteiny czy FAST-N, gdzie N oznacza nukleotydy [26]. W pakiecie wprowadzono implementację algorytmu Smith’a-Waterman’a, który służy do znajdowania

podobnych regionów w łańcuchach tekstowych. Program dostępny jest pod adresem: <http://fasta.bioch.virginia.edu>.

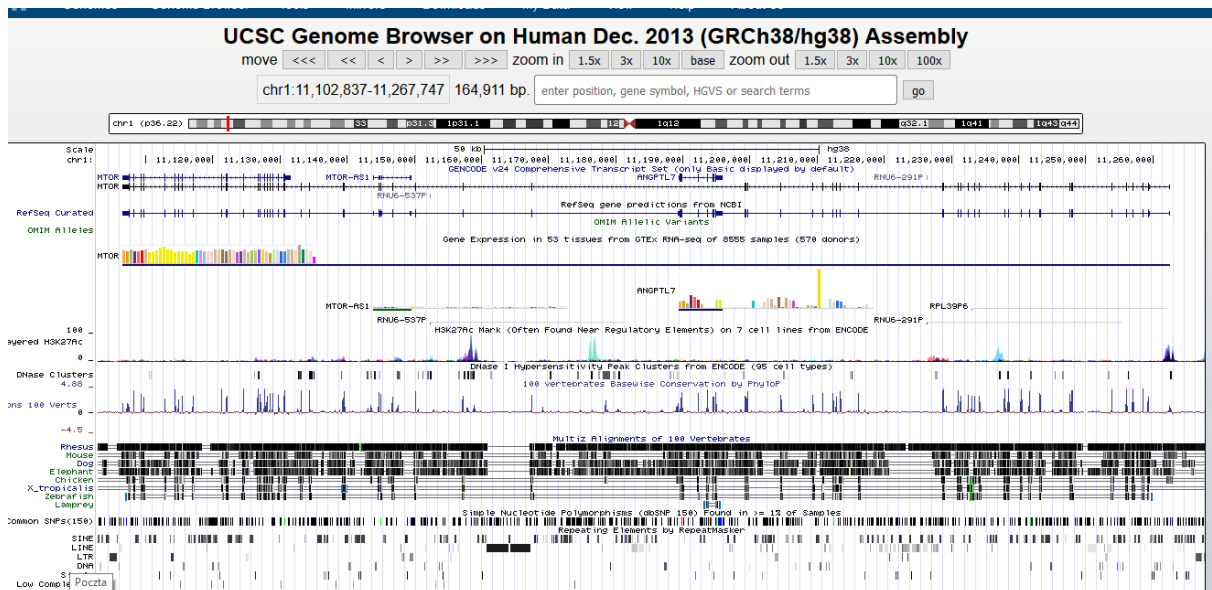
Bedtools – program będący na bezpłatnej licencji GNU. Decyzja o powstaniu programu wynikała z braku oprogramowania do analizy dużych plików genomowych, a jego odpowiedniki internetowe były niezbyt wygodne w użyciu. Niektóre z nich wymagały lokalnej instalacji przeglądarki jak UCSC Genome Browser. Obsługiwane przez program formaty to BED, GFF, VCF, BAM. Program potrafi liczyć pokrycie genomu na podstawie pliku BAM, konwertować pomiędzy różnymi formatami plików. Również potrafi znaleźć przecięcia i nakładania się na siebie nukleotydów [16].

BLAST (*Basic Local Alignment Search Tool*) – program i algorytm o tej samej nazwie. Używany do podstawowego zadania bioinformatycznego, czyli znajdowania biologicznego podobieństwa pomiędzy sekwencjami. Wejście stanowią formaty FASTA lub Genbank natomiast na wyjściu jest dostępnych wiele różnych formatów HTML, zwykły tekst czy XML. Dostępny w wersjach dla nukleotydów i protein. Przed wprowadzeniem programów FASTA i BLAST używany był algorytm Smith’a-Waterman’a. Algorytm ten jest bardzo czasochłonny dla analiz genomowych jednak w przeciwieństwie do programu BLAST potrafi znaleźć optymalne rozwiązanie [27].

Clustal – rodzina programów, których zastosowaniem jest odnajdywanie dopasowania wielu sekwencji. Program korzysta z licencji GPL. Istnieje format danych o tej samej nazwie. Występuje wiele różnych rodzajów programu takich jak: ClustalV, ClustalW, ClustalX i będący obecnie standardem ClustalΩ (Omega). W programie zaimplementowane są metody heurystyczne i wykorzystywane jest drzewo prowadzące. W internecie można odnaleźć przeglądarkowe wersje programu jak na przykład na stronie <http://www.genome.jp/tools-bin/clustalw>.

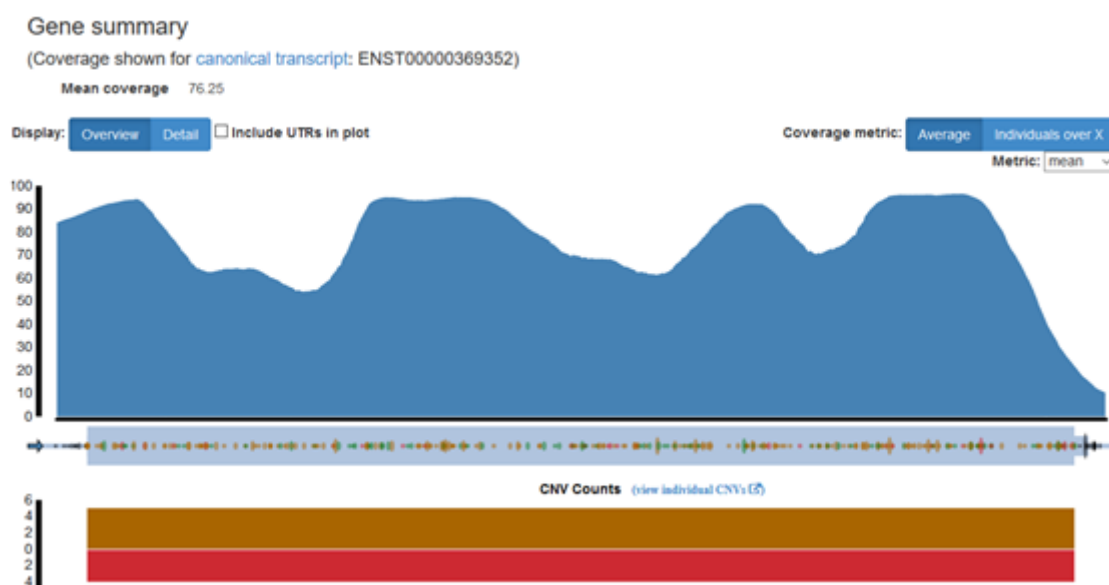
T-Coffee (ang. *Tree-based Consistency Objective Function For alignment Evaluation*) - podobnie jak Clustal służy do odnajdywania dopasowania wielu sekwencji. Wykorzystuje przy tym podejście progresywne. Również jest programem otwarto źródłowym dostępnym na wielu systemach operacyjnych. Formaty obsługiwane przez program to FASTA, PIR, PDB, CLUSTAL, MSF. Domyślnie korzysta ze zmodyfikowanego formatu Clustal. Istnieje wiele wariacji tego narzędzia jak na przykład M-Coffee (potrafiący łączyć wyniki z wielu różnych programów do dopasowania sekwencji), R-Coffee (wykorzystywany do sekwencji RNA) czy Espresso. Ponadto program T-Coffee używa wskaźnika TCS (ang. *Transitive Consistency Score*) jako miary jakości dopasowania sekwencji [28].

UCSC Genome Browser – oprogramowanie do wizualizacji genomów. Rys. 10 pokazuje zrzut ekranu z przeglądarki. Widać tam mnogość zastosowanych adnotacji. Adnotacje są wykorzystywane do oznaczania na przykład polimorfizmów pojedynczego nukleotydu, ekspresji genów i danych, predykcji genów. Przy tworzeniu przeglądarki skupiono się na optymalizacji pod względem szybkości przez co jest ona używana przez wielu badaczy. Ponadto badacze mogą wgrywać swoje pliki danych i wyświetlać je jako specjalne adnotacje.



Rys. 10 Zrzut ekranu z przeglądarki genomów Uniwersytetu Santa Cruz [29]

ExAC (ang. *Exome Aggregation Consortium*) – przechowuje wyniki analizy dla próbek, które poddane były sekwencjonowaniu. Istotną miarą jest tu właśnie pokrycie genomu. Pokrycie jest przedstawione głównie jako mediana i średnia z danego obszaru. Wbudowana wyszukiwarka pozwala szukać za pomocą nazw genów lub transkrypt. Jest to dzieło badaczy z całego świata w tym również polskich naukowców. Baza danych zawiera 60 tysięcy próbek. Kod źródłowy programu można odnaleźć w internecie. Na Rys. 11 pokazany jest zrzut ekranu ze wspomnianej przeglądarki dla genu GJA10.



Rys. 11 Zrzut z programu ExAC dla genu GJA10 [30]

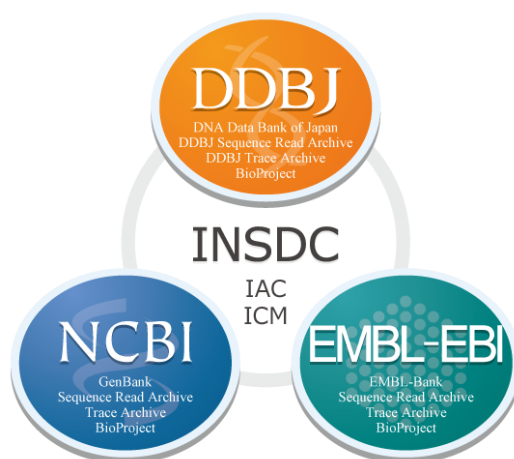
Innym rodzajem oprogramowania, które zyskuje na znaczeniu nie tylko ogólnie w informatyce, ale również w bioinformatyce są usługi sieciowe. Rosnący trend wynika z tego, że dzięki nim możemy udostępniać różnego rodzaju usługi lub dostęp do baz danych w postaci jakiej chcemy pokazać innym. W bioinformatyce istnieje wiele publicznie dostępnych baz, a ponadto często są one dostępne przez usługi sieciowe. Problemem z usługami sieciowymi było ich odnajdywanie. Został on rozwiązany za pomocą tak zwanego rejestru *EMBRANCE*. Stanowi on katalog bioinformatycznych usług sieciowych i pozwala poinformować o zmianie działania czy dostępności za pomocą poczty elektronicznej lub portalu twitter.com [31].

2.3.3 Bioinformatyczne bazy danych

Z uwagi na wielkość danych i ich specyfikę w bioinformatyce mamy nie tylko wiele różnych programów, ale także baz danych. Na stronach uniwersytetu Oxford, a konkretnie pod adresem <https://academic.oup.com/nar>, publikowane są co roku listy nowych biologicznych baz danych. W roku 2016 zostały zaktualizowane opisy 95 baz i dodane 62 nowe bazy. Bazy te najczęściej związane są z chorobami genetycznymi, a zwłaszcza rakiem, ale także często spotyka się bazy na temat narkotyków i ich skutków. Dostarczycielami danych są najczęściej duże instytuty bioinformatyczne, a głównymi z nich są:

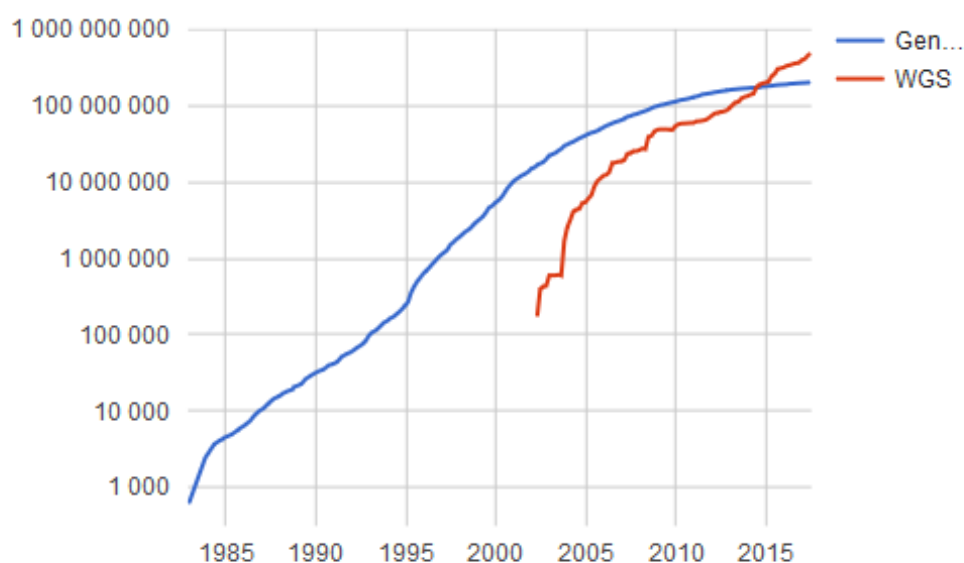
- Instytut informatyki Stanów Zjednoczonych (ang. *U.S. National Center for Biotechnology Information - NCBI*)
- Europejski instytut bioinformatyki (ang. *European Bioinformatics Institute - EMBL-EBI*)
- Szwajcarski instytut bioinformatyki (ang. *Swiss Institute for Bioinformatics - SIB*) [32].

Istnieją różnego rodzaju typy biologicznych baz danych. Możemy wyróżnić bazy kwasów nukleinowych, bazy aminokwasów czy protein lub metabazy, czyli zawierające informację do generowania innych baz danych. Bazy kwasów nukleinowych mogą być używane do przechowywania danych o DNA, RNA, genomach, ekspresji genowej czy fenotypach. Poniżej przedstawiono zestawienie kilku najbardziej popularnych baz danych.



Rys. 12 Zespół trzech organizacji i ich baz połączony w jedną INSDC [33]

- **INSDC** (ang. *International Nucleotide Sequence Database Collaboration*) – jest to w zasadzie zbiór trzech baz danych pochodzących z trzech instytutów: DNA Data Bank of Japan z bazą o tej samej nazwie, NCBI z bazą GenBank i EMBL-EBI z bazą EMBL. Zespół baz powstał przy okazji współpracy czołówki światowych instytutów bioinformatycznych. Zawierają dane na temat nukleotydów we wszystkich organizmach. Każda z trzech baz wymienia między sobą dane. Ponadto są one zsynchronizowane z bazą SRA (ang. *Sequence Read Archive*). Ilustracja INSDC zaprezentowana jest na Rys. 12.
- **SRA** (ang. *Sequence Read Archive*) – zawiera niesformatowane dane pochodzące z sekwencjonowania genomów nowej generacji. Zazwyczaj są to krótkie sekwencje o długości mniejszej niż 1000 par zasad długości. Baza należy do instytutu NCBI i jest synchronizowana z pozostałymi instytutami.
- **GenBank** – publicznie dostępna baza należąca do NCBI, zawierająca sekwencje DNA. Baza wydawana jest co dwa miesiące i dostępna przez protokół ftp. Ponadto na stronach NCBI możemy znaleźć opisy rekordów w tej bazie. Genbank jest budowany na podstawie danych udostępnianych przez niezależne instytuty. Dane w GenBanku podwajały się w przybliżeniu co 3 lata, choć jak widać na Rys. 13 w ostatnich latach wzrost nie był aż tak szybki [34].



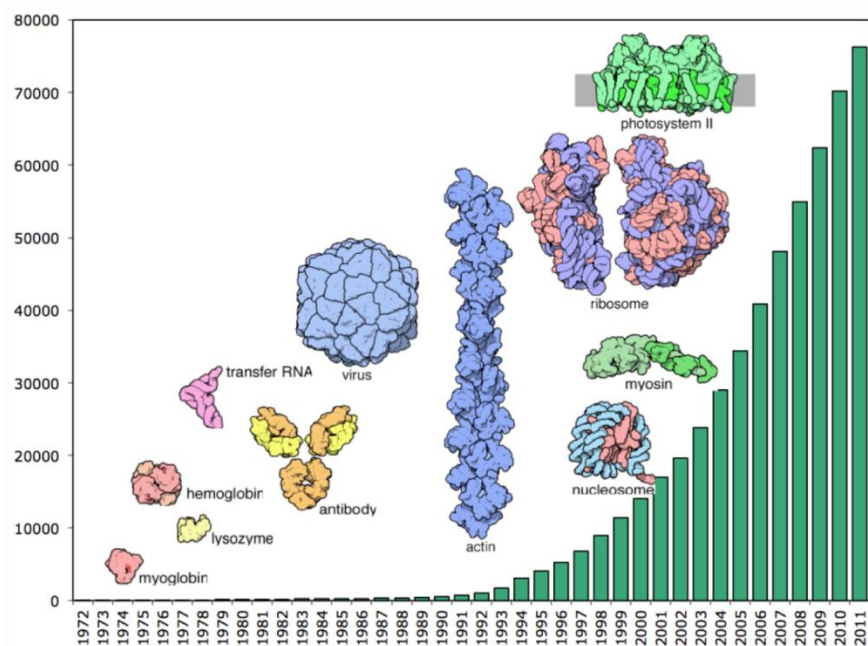
Rys. 13 Statystyki pokazujące wzrost ilości par zasad w bazach GenBank i WGS [34]

- **DDBJ** (ang. *DNA Data Bank of Japan*) – baza należąca do Narodowego Instytutu Genetyki (ang. *National Institute of Genetics*) zlokalizowanego w miejscowości Shizuoka w Japonii.
- **ENA** (ang. *European Nucleotide Archive*) – publiczna baza DNA i RNA udostępniana przez instytut EMBL-EBI (ang. *European Molecular Biology Laboratory European Bioinformatics Institute*). Należy do INSDC. Organizacja ma swoje siedziby w Niemczech, Włoszech, Wielkiej Brytanii i Francji. Baza ENA dostępna jest bezpośrednio lub za pośrednictwem usługi sieciowej typu REST.

Rekordy w bazie ENA zawierają informacje wejściowe do sekwencjonowania (konfiguracja maszyny, próbkę, konfigurację eksperymentu), wyjściowe wyniki sekwencjonowania oraz zinterpretowane informacje o mapowaniu czy adnotacjach funkcyjnych.

- **RefSeq** - otwarta baza zbudowana podobnie jak GenBank przez instytut NCBI. Różnica pomiędzy tymi dwoma bazami polega na tym, że GenBank zapisuje w bazie, dane o sekwencjonowaniu każdego organizmu, natomiast RefSeq pamięta tylko jeden rekord dla danego organizmu. RefSeq próbuje zbudować modele różnych organizmów.
- **Gene Disease Database** – zbiór baz danych zawierających informacje o chorobach genetycznych. Istnieją cztery rodzaje tego typu baz danych:
 - Bazy zbudowane na podstawie wiedzy eksperckiej (ang. *curated databases*) – ściśle uporządkowane, zatwierdzone przez jednego lub więcej ekspertów.
 - Bazy zbudowane na podstawie przewidywań (ang. *predictive databases*).
 - Bazy zbudowane na podstawie literatury (ang. *literature databases*) – zbudowane na podstawie artykułów, książek, rozpraw naukowych. Przykładem takiej bazy może być GAD (ang. *Genetic Association Database*). Baza ta gromadzi głównie dane na temat powszechnych chorób genetycznych. Zawiera zsumowane dane z innych magazynów danych na temat genów i genomów.
 - Bazy polegające na powiązaniu gen – choroba (ang. *integrative databases*) [35].
- **Protein Data Bank** – baza danych skierowana specjalnie do przechowywania molekuł i kryształów tak zwana krystalograficzna baza danych. Wykorzystywany jest do tego specjalny format danych o tej samej nazwie. Rys. 14 prezentuje liczby dostępnych struktur w formacie PDB w poszczególnych latach do 2011 roku. Jak widzimy w ostatnich latach zanotowanych na wykresie liczba struktur znacznie wzrastała. W latach osiemdziesiątych było to jedynie 7 struktur. Początkowo bazą opiekowała się wąska grupa badaczy, dziś jest ona znacznie większa, a kontrolą nad nią sprawuje organizacja *Worldwide Protein Data Bank*, wwPDB [20]. Baza dostępna jest publicznie, udostępniana przez takie organizacje jak RCSB (ang. *Research Collaboratory for Structural Bioinformatics*).
- **Meta bazy danych** – przykładem takich baz mogą być *GeneCards* z Instytutu Weizmann, *Genome Aggregation Database* z Broad Institute czy *mGEN* zawierający w sobie cztery największe bazy danych w postaci przyjaznego użytkownikowi programu. Meta bazy służą do budowania innych baz na ich podstawie.

Innym ciekawym przedsięwzięciem, które dotyczyło utworzenia bioinformatycznej bazy danych był projekt 1000 genomów. Jak nazwa wskazuje, zakładał on pozyskanie co najmniej 1000 genomów ludzkich. Dane z projektu są dostępne dla wszystkich w internecie [36].



Rys. 14 Wykres przedstawiający liczbę udostępnionych struktur w formacie PDB [37]

W najbliższych latach spodziewany jest jeszcze większy wzrost ilości narzędzi bioinformatycznych, zarówno programów, formatów jak i baz danych. Ponadto ilości zgromadzone w bazach będą z roku na rok rosły. Coraz większym wyzwaniem jest zapanowanie nad owym rozrostem danych.

3 Omówienie architektur rozważanego systemu

W pierwszym punkcie niniejszego rozdziału opisano jakie wymagania oraz problemy są związane z tworzoną systemem. Ze względu na charakter pracy użyto wielu narzędzi informatycznych i technologii. Drugi podrozdział opisuje je. Następne podrozdziały opisują dwie architektury, które są porównywane w niniejszej pracy. Pierwsza z nich wykorzystuje kompresję danych do plików binarnych. Druga zaś korzysta z narzędzi informatycznych do przetwarzania dużych zbiorów danych (ang. *big data*). Obie architektury zostały połączone w przeglądarkowym interfejsie, co zostało ujęte w ostatnim podrozdziale.

3.1 Wymagania stawiane systemowi

Głównym celem budowanego systemu jest możliwość pozyskiwania wartości o pokryciu genomu. Z uwagi jednak na specyfikę tej wartości nie jest sprawą prostą rozwiązanie kilku kwestii wiążących się z pokryciem. Przede wszystkim jest to wartość, którą ma każdy nukleotyd w genomie. W ludzkim DNA mamy 3,5 miliarda nukleotydów, więc potencjalnie dla jednego genomu mamy tyle też wartości pokrycia. Istnieje wiele programów, które potrafią zbierać wyniki o pokryciu genomu. Nie spełniają one jednak wszystkich wymagań, które stawiają przed nimi badacze. Wiele narzędzi potrafi odczytać pokrycie z jednej próbki, gdzie jedna próbka reprezentuje jeden genom. Badacze chcą mieć możliwość sprawdzania podczas jednego badania setek próbek dla setek różnych genów. Wymaganie to jest problematyczne w realizacji właśnie ze względu na ilość wartości pokrycia.

W procesie sekwencjonowania powstają krótkie odcinki, które są mapowane na genom referencyjny. W wyniku czego wartości pokrycia na sąsiednich nukleotydach są bardzo często takie same lub bardzo zbliżone do siebie. Co więcej, badaczom nie zależy na dokładnej wartości, a jedynie na tym jaki mamy procent wartości, których pokrycie jest większe niż zadane. Przykładowo wystarczająca jest informacja, że pokrycie jest większe od 20, a nie jest istotne czy jego konkretna wartość wynosi 25 czy 27. Doprecyzowując, najważniejsza dla badaczy jest informacja według tabeli 1. W pierwszej kolumnie tabeli zaprezentowana jest nazwa przedziału, która składa się z liter GE i wartości pokrycia. Litery GE oznaczają, że wartości pokrycia są większe równe (ang. *greater equal*) od występującej po niej wartości. Dla aplikacji wystarczające jest, więc pamiętanie zakresów podanych w tabeli 1.

Tabela 1 Zestawienie istotnych przedziałów dla wartości pokrycia

Nazwa	Opis wartości
GE5	Wartości większe niż 5
GE10	Wartości większe niż 10
GE20	Wartości większe niż 20
GE50	Wartości większe niż 50
GE100	Wartości większe niż 100
GE500	Wartości większe niż 500

Biorąc pod uwagę powyższe wartości pokrycia można dojść do wniosku, że dobrą metodą jest zastosowanie kompresji danych i dedykowany format. Drugim pomysłem jest wykorzystanie ostatnio popularnych narzędzi do wielkich zbiorów danych. Wykorzystując odpowiednią bibliotekę, mogą one również znaleźć zastosowanie w liczeniu pokrycia.

Jeżeli chodzi o konkretne wymagania, system powinien umożliwiać wprowadzanie próbek i wartości, o które chcemy pytać. Jedna próbka jest reprezentowana jako jeden plik, na przykład w formacie BAM. Jeżeli chodzi o rodzaje obsługiwanych zapytań, wyróżniamy dwa typy. Jednym z nich jest podanie chromosomu oraz przedziału dla którego chcemy znaleźć wartość pokrycia. W rozwiązaniach bioinformatycznych najczęściej spotykany jest on w formacie jak na Rys. 15.



Rys. 15 Przykładowe przedstawienie formatu, który umożliwia odpytywanie o wartości pokrycia

Rysunek ten pokazuje, że w genomie zostanie odpytany chromosom X w przedziale od 300 do 900. Wartość chromosomu może przyjmować następujące wartości chr1...chr22 dla autosomów, chrX, chrY dla heterosomów oraz chrM dla DNA mitochondrialnego. Natomiast w przedziałach ważne jest by pierwsza wartość była mniejsza od drugiej oraz żeby długość nie była dłuższa niż długość konkretnego chromosomu. Jednostką dla podawanych przedziałów rozpatrywane są pary zasady.

Powyżej opisany sposób jest jednym ze sposobów reprezentacji, który jest wymagany przez badaczy. Drugi sposób polega na podawaniu nazwy genu. Program musi za pomocą bazy danych odnaleźć odpowiedni wpis i odpowiadające mu przedziały. Istotne jest, że przedziałami tymi mają być tylko i wyłącznie eksony danego genu. Geny mogą mieć więcej niż jeden ekson. Przykładowo gen VAC14 dla sekwencji referencyjnej HG19 ma aż czternaście eksonów na chromosomie szesnastym. Eksony tego genu znajdują się w przedziale od 70721820 do 70834813. Pytając, więc o gen VAC14 powinniśmy zapytać o czternaście różnych eksonów.

Jak już wspomniano kluczowym wymaganiem jest tu czas przetwarzania analizy oraz zajętość samej bazy programu. Należy pamiętać, że próbki są dużych rozmiarów. Składowanie ich w nieprzetworzonych formatach jest już samo w sobie problematyczne. Zatem ważne jest, żeby program albo miał możliwość odpytywania bezpośredniego z pliku próbki albo po przetworzeniu wyniki zajmowały mało miejsca na dysku. Co więcej, nieakceptowalne jest czekanie na wyniki analiz kilku godzin. Opracowany system musi być jak najszybszy. Konkretnie, chcielibyśmy by system potrafił przetwarzać kilkaset zapytań w czasie nie dłuższym niż pięć minut. Jeżeli chodzi o same obliczenia istotnym usprawnieniem byłaby możliwość dzielenia ich na więcej niż jeden procesor lub jeszcze lepiej na więcej niż jeden komputer.

Poza opisanymi wyżej cechami pożądana jest dostępność z więcej niż jednego komputera. Różni badacze chcą mieć dostęp do wyników pokrycia, dlatego też naturalnym wydaje się realizacja systemu w wersji aplikacji internetowej.

3.2 Wykorzystane narzędzia informatyczne

Aplikacja, która nie jest artefaktem niniejszej pracy, a jedynie stanowi wzór dla drugiego systemu została napisana w języku C#, którego działanie opiera się o platformę .NET Framework w wersji 4.5. Jeżeli chodzi o środowisko programistyczne zostało wykorzystane Microsoft Visual Studio 2015 Community Edition. Ciekawym faktem jest, że system z powodzeniem może być wykorzystywany również w architekturze z serwerem na systemie Linux. Budowanie aplikacji w systemie uniksowym opierało się o platformę Mono, a środowiskiem programistycznym było MonoDevelop. Testy aplikacji były przeprowadzane w systemie Ubuntu i Windows.

Działanie aplikacji na systemie Linux pokazuje ogólny trend firmy Microsoft, a także wielu innych firm do tworzenia swoich rozwiązań dostępnych na wielu systemach operacyjnych. Firma Microsoft ostatnimi czasy poczyniła duże kroki w tworzeniu otwartego oprogramowania, czego dowodem jest otwarcie źródeł platformy .NET oraz rozwijanie projektów takich jak .NET Core czy .NET Standard.

Celem owych projektów jest dostęp do platformy .NET na wielu systemach operacyjnych. Graficzny interfejs użytkownika powstał w technologii WPF (Windows Presentation Forms). Dodatkowymi bibliotekami wspierającymi tworzenie oprogramowania były Nlog 4.0 WPF Xceed Toolkit, dodatek pozwalający stworzyć plik instalacyjny (Setup Project). Do testów jednostkowych i integracyjnych została wykorzystana biblioteka Nunit wspierana dodatkowo przez Moq 4.5 i Ploeh.SemanticComparison 3.4. Baza danych zawierająca nazwy genów wraz z ich lokalizacją powstała w technologii SQLite w wersji trzeciej.

Druga omawiana aplikacja, która powstała w czasie tworzenia niniejszej pracy została oparta na innym stosie technologicznym. Najważniejszą różnicą pod względem technologicznym jest to, że aplikacja nie jest aplikacją okienkową, a internetową. Wydzielone zostały dwie zasadnicze warstwy. Warstwa interfejsu użytkownika (ang. *frontend*), część serwerowa (ang. *backend*).

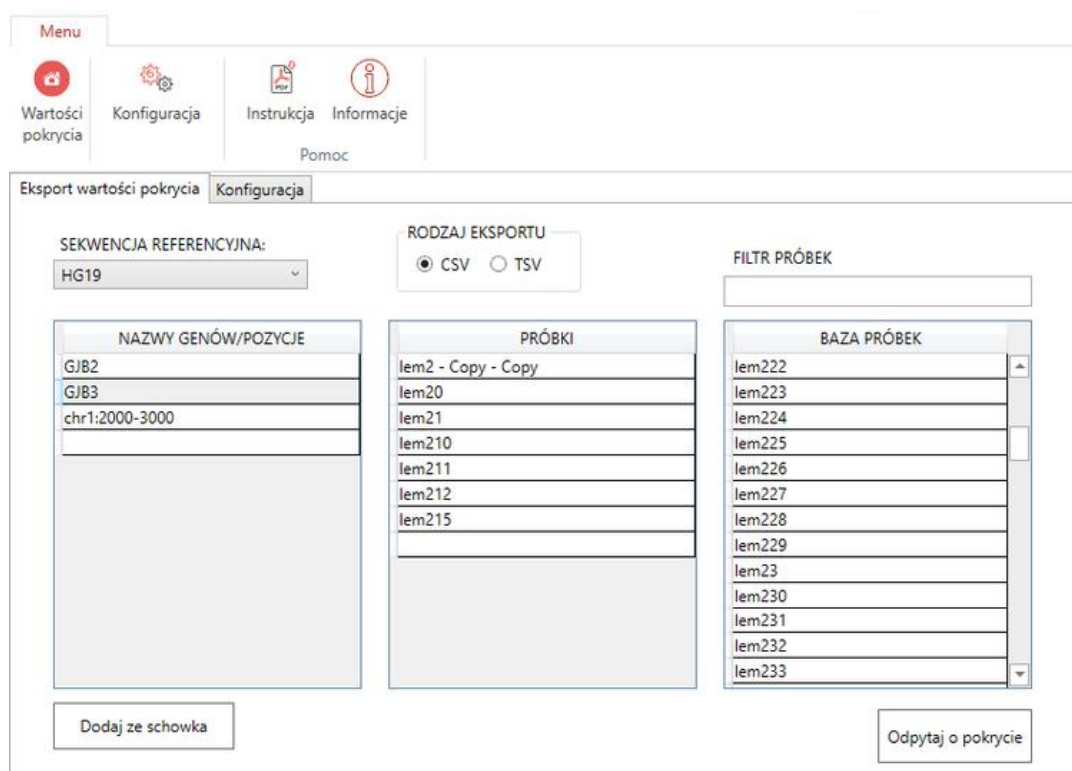
Aplikacja zrealizowana została przy pomocy nowoczesnych technologii internetowych takich jak HTML5, arkusze stylów CSS, biblioteka Bootstrap 4, JavaScript. Ponadto wykorzystuje język TypeScript, który jest rozwinięciem języka JavaScript. Język ten wprowadza ścisłą kontrolę typów zmiennych. Ułatwia to utrzymanie dużych aplikacji internetowych. Dodatkową technologią wykorzystywaną w warstwie użytkownika jest Angular w wersji 5.2. Wygląd aplikacji został wzbogacony dzięki szablonoowi firmy ColorLib, który dostępny jest w serwisie GitHub: <https://github.com/puikinsh/sufee-admin-dashboard>. Szablon wykorzystuje ponadto rozszerzenie arkuszy stylów SCSS. Dzięki temu rozszerzeniu w arkuszach stylów możemy użyć na przykład zmiennych, funkcji. Środowiskiem programistycznym, w którym powstawał interfejs użytkownika jest nowość firmy Microsoft,

a mianowicie Visual Studio Code, który pozwala programować na wielu systemach operacyjnych, nie tylko systemach Windows.

W warstwie serwerowej zbudowany jest interfejs programistyczny API (ang. *application programming interface*) za pomocą platformy .NET Core. Interfejs powstał przy pomocy środowiska Visual Studio 2017. Dodatkowo w warstwie serwerowej został użyty jeszcze jeden język, a mianowicie Scala. Jeżeli chodzi o środowisko programistyczne wspierające programowanie w Scali wykorzystano produkt firmy JetBrains IntelliJ Idea. Biblioteką wspierającą obliczenia związane z pokryciem jest biblioteka ADAM powstała w projekcie bigdatagenomics.

3.3 Architektura oparta na plikach binarnych

Jak wspomniano wcześniej początkowo aplikacja została przygotowana w wersji okienkowej. Aplikacja z poprzedniej wersji zaprezentowana jest na Rys. 16.



Rys. 16 Aplikacja w pierwszej, okienkowej wersji [5]

Aplikacja przyjmowała na wejściu duże pliki tekstowe, które są wynikami działania programu bedtools. Pliki te są podobne do tych zaprezentowanych na Rys. 6. Program poddawał kompresji tekstowe pliki wykorzystując właściwości wartości pokrycia. Aplikacja nie pamiętała dokładnie wartości pokrycia, a jedynie pewne przedziały. Na przykład zapamiętywana jest tylko informacja na przykład czy pokrycie jest większe od 5.

Wynikiem przetwarzania są pliki binarne. Format pliku binarnego składa się z nagłówka i danych. Nagłówek podzielony jest na 25 liczb całkowitych, odpowiadających odpowiednio dwudziestu dwóm chromosomom, chromosomowi X, Y i chromosomowi mitochondrialnemu.

Wartości te reprezentują ilości rekordów danych przechowywanych dla poszczególnych chromosomów. Rekordy danych są reprezentowane przez trójkę wartości, a są nimi zakres przed, zakres po i wartość pokrycia w opisanym poprzednimi wartościami zakresie. Takie podejście znacząco redukuje zużycie pamięci oraz pozwala w sposób szybki znajdować wartości pokrycia w pliku binarnym. Schematyczna reprezentacja tego formatu zaprezentowana jest na Rys. 17.

Założmy, że przykładowa próbka ma 5 rekordów mówiących o pokryciu z czego dwa są położone na chromosomie pierwszym i trzy na piątym. Dodatkowo chcemy znaleźć pokrycie na chromosomie piątym. Dzięki nagłówkowi możemy obliczyć, ile rekordów pokrycia musimy pominąć by uzyskać dostęp do zadanej wartości. Wystarczy zsumować wszystkie wartości poprzedzające nagłówek z zadaniem chromosomem. W przytoczonym przykładzie będzie to liczba rekordów danych na chromosomie pierwszym, czyli pominięte zostaną dwa rekordy danych.

Długość chr1	. . .	Długość chr22	Długość <u>chrX</u>	Długość <u>chrY</u>
Długość <u>chrM</u>	Rekord danych		Rekord danych	

Rys. 17 Schemat formatu pliku binarnego

Pierwotna architektura aplikacji dzieli system na dwie części: serwerową i interfejsu użytkownika. Po stronie serwerowej znajdowała się aplikacja, która kompresuje dane wejściowe do plików binarnych i umieszcza je na dysku sieciowym. Po stronie interfejsu użytkownika zlokalizowana jest aplikacja okienkowa, która pozwala na odpytywanie zgromadzonych na dysku sieciowym plików binarnych. Ponadto program korzysta z bazy danych w SQLite, która gromadzi nazwy genów wraz z ich pozycjami eksonowymi.

Architektura aplikacji została zmieniona na interfejs w przeglądarce oraz wykonana w najnowszych technologiach pozwalających budować nowoczesne interfejsy użytkownika.

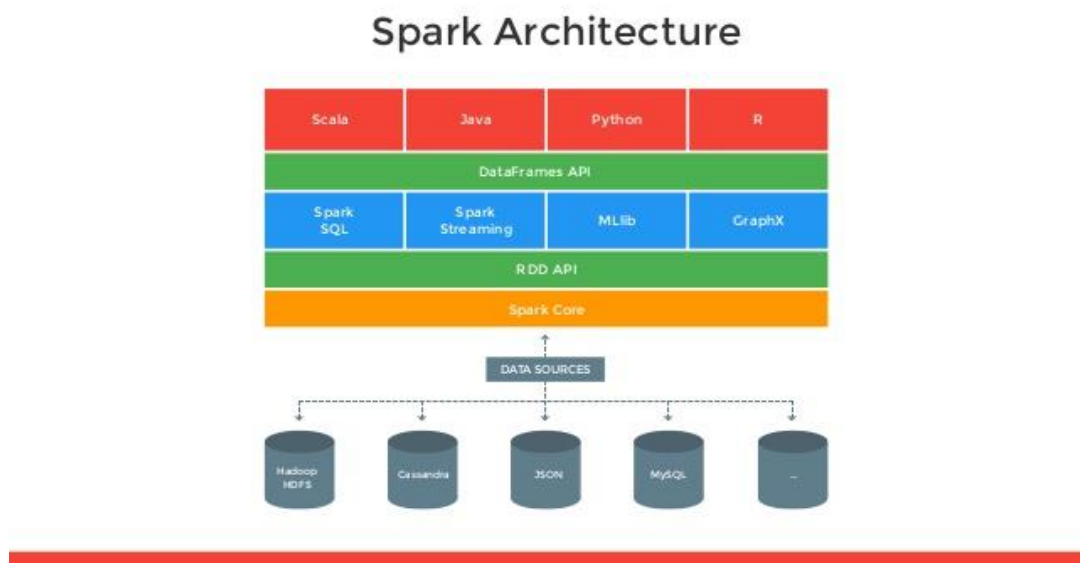
3.4 Architektura oparta na platformie Spark

Apache Spark jest technologią wspierającą budowanie rozproszonych aplikacji. Technologia może być z powodzeniem używana za pomocą popularnych języków Java, Scala, Python czy R. Spark wprowadza kilka dodatkowych modułów, które zwiększają jego możliwości takich jak:

- **Spark SQL** – wprowadza rozszerzenie, które pomaga operować na strukturze ramki danych (ang. *DataFrame*). Ponadto dostarcza interfejs, który pozwala poprzez sterowniki JDBC czy ODBC operować bazą. Co więcej, korzysta z implementacji Hive, która zapewnia abstrakcję nad konkretnym silnikiem bazodanowym.

- **Mllib** – dostarcza zbiór algorytmów uczenia maszynowego. Ponadto, zapewnia wsparcie dla funkcji matematycznych i statystycznych przydatnych w analizie danych. Zapewnia również możliwości zapisywania i ładowania ponownie modeli czy algorytmów.
- **GraphX** – zawiera zbiór algorytmów, które wspomagają równoległe przetwarzania w grafowej strukturze danych.
- **Spark Streaming** – zapewnia wsparcie dla równoległego przetwarzania strumieni danych. W Sparku istnieją strumienie danych, które pochodzą ze znanego portalu społecznościowego Twitter [38].

Znane w informatyce prawo Moora, które mówi o wykładniczej miniaturyzacji elektronicznych komponentów, a co za tym idzie wzrostu wydajności sprzętu przestaje obowiązywać [39]. Dlatego też obserwowane jest coraz większe zainteresowanie systemami rozproszonymi, które dzięki swoim możliwościom podtrzymują rozwój mocy obliczeniowej. Spark jest technologią, która właśnie wspiera obliczenia wykonywane w wielu węzłach. Ponadto ważną cechą systemów rozproszonych jest odporność na uszkodzenia (ang. *fault tolerance*). Polega ona na tym, że system w przypadku, gdy jakiś z węzłów przestaje działać nadal jest w stanie pracować. Rys. 18 przedstawia opisaną architekturę Spark. Widzimy na nim, że źródłem danych mogą być różnego rodzaju bazy jak na przykład MySQL, Cassandra, pliki Json.



Rys. 18 Architektura technologii Spark [40]

Spark jest często porównywany do technologii *Hadoop Map Reduce*. Apache Spark korzysta wewnętrznie z systemu Hadoop. Między innymi z rozproszonego systemu plików HDFS (ang. *Hadoop Distributed File System*). Obie technologie mają dostępne w internecie kody źródłowe oraz są na licencji Apache 2.0. Niemniej jednak Hadoop pozwala na przetwarzanie tylko z plików wsadowych, natomiast Spark, poza tym wspiera również strumieniowanie, przetwarzanie grafów i wiele innych operacji. Poza kilkoma kwestiami na

przykład związanymi z bezpieczeństwem Spark wypada lepiej niż jego konkurent jako technologia przetwarzania danych w klastrach [41].

W rozwiązaniach stosujących technologię Spark istotnym pojęciem jest struktura danych RDD (ang. *Resilient Distributed Dataset*). To właśnie ona zapewnia rozproszenie systemu i odporność na uszkodzenia. Ponadto struktura ta może być przetwarzana w pamięci operacyjnej komputera. Istnieją dwa sposoby tworzenia owej struktury. Pierwszy sposób polega na zrównolegleniu istniejących kolekcji w programie. Drugi natomiast wykorzystuje zewnętrzne źródła takie jak HDFS, współdzielony system plików czy Apache Hbase, który jest technologią przechowywania wielkich tabel, które sięgają miliardów rekordów i milionów kolumn. Struktura wspiera dwa typy operacji. Transformacje, które tworzą nowy zbiór danych na podstawie istniejącego oraz akcje, które tworzą zbiór na podstawie zewnętrznego źródła. Warto dodać, że wszystkie operacje są leniwe, czyli nie wykonują się od razu. Technika taka pozwala wykonać żadaną operację dopiero kiedy faktycznie potrzebujemy jej wyniku. Spark ponadto wprowadza też inne abstrakcyjne struktury danych, tak zwane zmienne współdzielone. Tworzą one kopie zmiennych dla każdego węzła sieci. W celu zapewnienia współdzielenia owych zmiennych Spark dostarcza dwa ich rodzaje, czyli:

- Zmienne rozgłaszane (ang. *broadcast variables*) – potrafią zapamiętać wartość w pamięci dla wszystkich węzłów.
- Akumulatory (ang. *accumulators*) – które są wartościami, do których można tylko dodawać wartości takie jak sumy czy liczniki [38].

Warto dodać, że oprócz wspomnianych i opisanych projektów fundacji Apache (czyli Hadoop, Spark, HBase). Istnieje również wiele technologii Apache związanych z przetwarzaniem dużym zbiorów danych. Często wykorzystywane są one wewnętrznie. Poniżej zaprezentowana jest lista kilku innych tego typu technologii:

- **Apache Avro** – zestaw narzędzi, które służą do serializacji obiektów i zapewniają interfejs sieciowy dostępu do danych. Wewnętrznie używają one formatu JSON. Narzędzia mają wsparcie zarówno dla Hadoop jak i Spark.
- **Apache Parquet** – dostarcza format dostępu do danych zorientowanych na kolumny. Jest to zupełnie inne podejście niż znane z baz danych, gdzie orientacja najczęściej jest w obrębie wiersza.
- **Apache Solr** – wspiera wyszukiwanie ciągów znaków w tekstowych bazach danych.

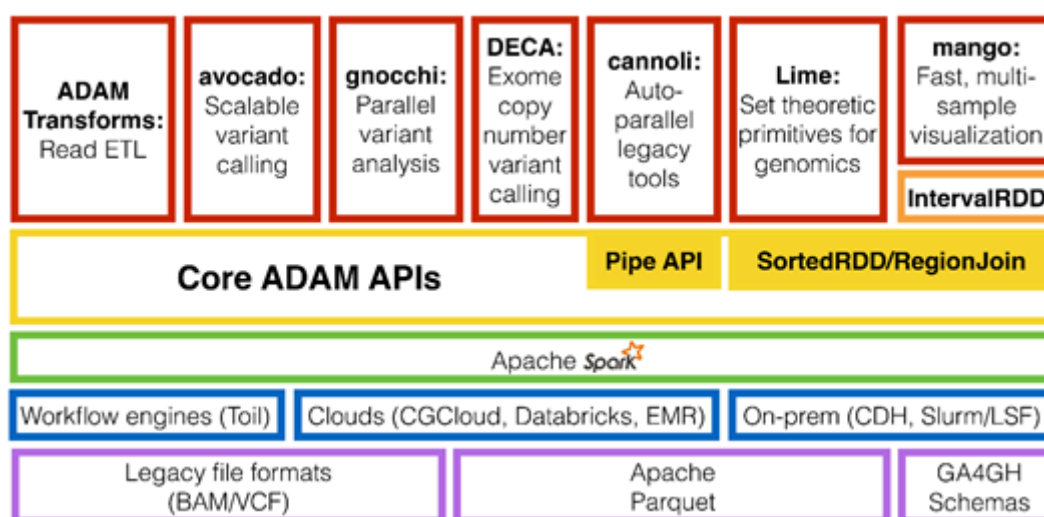
Pełną listę projektów znajdziemy na stronie fundacji Apache pod adresem: <https://projects.apache.org/projects.html?category#big-data>.

W genomice mamy do czynienia z dużą ilością danych i obliczeń, dlatego też naturalnym byłoby wykorzystanie technologii Spark, dla tego typu projektów. Z pomocą przychodzi tu biblioteka Adam i projekt bigdatagenomics, czyli duże zbiory danych dla genomiki. Adam również ma otwarty kod źródłowy. Poza projektem Apache Spark, Adam korzysta również z Apache Avro i Apache Parquet. Jak pamiętamy z rozdziału drugiego w bioinformatyce istnieje wiele różnych formatów. Adam zapewnia abstrakcje dla kilku

popularnych formatów takich jak SAM, BAM, CRAM, ADAM, BED, GFF3, GTF czy VCF. Projekt zapewnia wsparcie dla kilku popularnych języków, to jest Scala, Java, SQL, Python czy R. Jednak niektóre funkcje są dostępne tylko w języku Scala. Architektura biblioteki zaprezentowana jest na Rys. 19.

Poza samą biblioteką Adam istnieje kilka rozszerzeń do niego. Należą do nich:

- **Avocado** – związane ze znajdowaniem wariantów.
- **Cannoli** – pozwala na korzystanie z Adama w aplikacjach takich jak FreeBayes czy BWA.
- **DECA** – również związane ze znajdowaniem wariantów.
- **Gnocchi** – pomaga w przeprowadzaniu testów GWAS i eQTL na dużych genotypach.
- **Mango** – biblioteka wspomagająca wizualizację genomów.



Rys. 19 Schematyczne przedstawienie architektury ADAM [42]

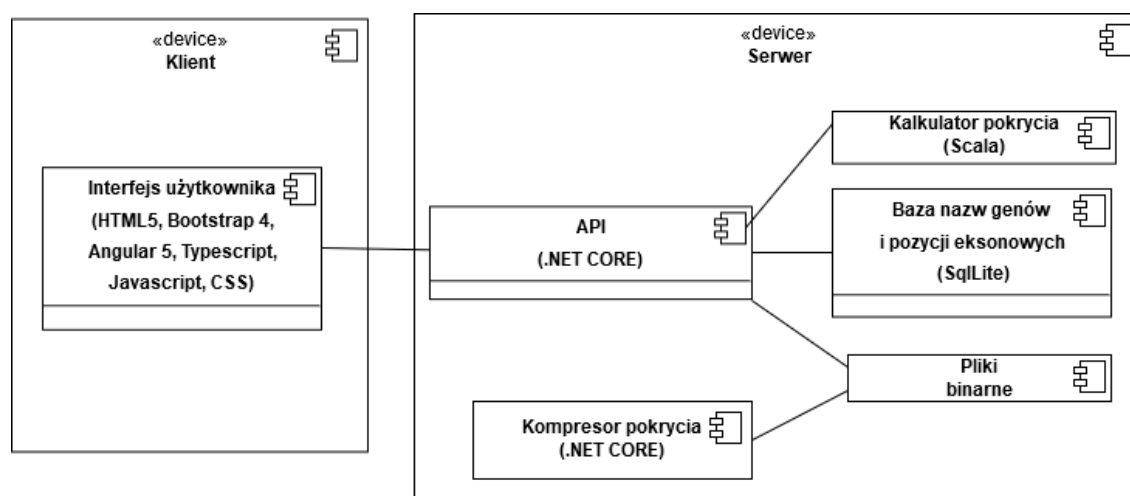
Adam udostępnia strukturę danych GenomicRDD, po której dziedziczą takie typy jak CoverageRDD. W kontekście budowanej pracy najważniejszą funkcją Adama jest wsparcie dla liczenia pokrycia genomu. W Adamie wyróżniamy trzy rodzaje narzędzi: akcji, konwersji i drukowania. Do pierwszych z nich należą różnego rodzaju transformacje do cech, genotypów, wariantów czy dopasowania (odpowiednio są to transformFeatures, transformGenotypes, transformVariants, transformAlignments). Interesujące wydaje się być narzędzie do czytania pokrycia (reads2Coverage). Jeżeli chodzi o narzędzia konwersji wspierana jest zmiana formatu z Fasta do Adam oraz w kierunku odwrotnym.

Ponadto, Adam wspiera dwa rodzaje operacji złączeń na regionach genomów, a mianowicie *Broadcast Region Join* oraz *Shuffle Region Join*. Wyniki obu złączeń są takie same, a różnią się jedynie sposobem działania. Pierwszy z nich najpierw dzieli na partycję prawostronnego zbioru zanim każdy z węzłów przeprowadzi złączenie. Ten rodzaj złączenia powinien być używany w sytuacji, gdy prawostronny zbiór jest duży i oba zbiory mają wysoką licznosc (ang. *cardinality*). Drugi rodzaj kopiuje cały lewostronny zbiór do każdego z węzłów

i powinien być stosowany, gdy prawostronny zbiór jest mały. Wyniki tego rodzaju złączenia są nieposortowane [42, 43].

3.5 Hybrydowa architektura aplikacji

Podejścia do architektury zaprezentowane w poprzednich dwóch rozdziałach złączono w jeden system. Zrealizowana aplikacja służy do zweryfikowania ich oraz oceny, które jest lepsze. Bardzo możliwe jednak, że dla niektórych przypadków jedno z nich będzie się lepiej sprawdzać. Zasadniczym założeniem budowanej aplikacji jest zmiana trybu jej działania za pomocą parametru konfiguracyjnego. System powstał jako aplikacja internetowa.



Rys. 20 Diagram rozmieszczenia komponentów obecnej architektury aplikacji

Rys. 20 przedstawia rozmieszczenie komponentów w obecnej architekturze aplikacji. Dwoma zasadniczymi elementami jest klient z interfejsem w postaci aplikacji internetowej i warstwa serwerowa. Klient komunikuje się za pomocą usługi sieciowej napisanej w oparciu o platformę .NET Core. Interfejs ten integruje również pozostałe elementy osadzone na serwerze. Pozwala on na odczyt danych z plików binarnych i bazy danych. Ponadto potrafi wystartować zadanie kalkulatora pokrycia napisanego w Scali i wykorzystującego technologię Spark oraz bibliotekę Adam. Kalkulator pokrycia na wejściu przyjmuje ciąg tekstowy, który jest w notacji JSON (ang. *JavaScript Object Notation*). Parametr ten zawiera próbki, o które chcemy zapytać oraz listę transkryptów. W warstwie serwerowej znajdują się również kompresor pokrycia również napisany przy pomocy platformy .NET Core. Kompresor ten przyjmuje plik tekstowy na wejściu, a jako rezultat jego działania powstaje plik binarny. Z tego właśnie pliku odczytywane są dane z usługi sieciowej.

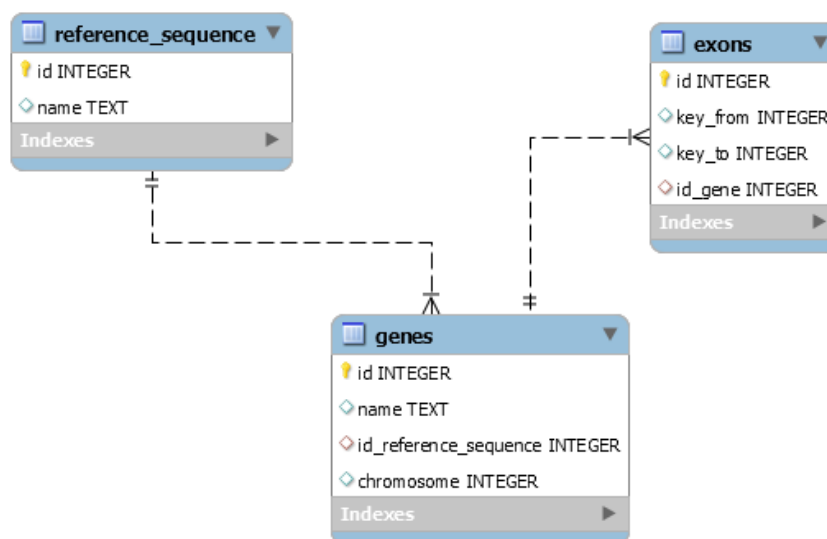
4 Implementacja systemu

W rozważanym teraz punkcie opisane zostały szczegóły techniczne tworzonego systemu. Pierwszy podrozdział wymienia wszystkie składowe projektu z uwzględnieniem nowych jak i użytych ponownie elementów. Następnie bardziej szczegółowo zostały opisane nowe elementy, czyli interfejs użytkownika, warstwa serwerowa i aplikacja, która liczy pokrycie. Ostatni podrozdział przeznaczony jest testom przeprowadzonym na systemie.

4.1 Podział systemu na komponenty

Na tworzoną aplikację składa się kilka różnych komponentów. Niektóre z nich zostaną omówione w dalszych punktach tego rozdziału. Część komponentów została przeniesiona z poprzedniego systemu. Do przeniesionych komponentów należą:

- **Baza nazw genów i eksonów** – przygotowana przy pomocy Sqlite. Baza zawiera nazwy genów oraz pozycje eksonowe tych genów. Dzięki niej program potrafi znaleźć eksony po wpisaniu jedynie nazwy genu. Dane pochodzą ze stron uniwersytetu California Santa Cruz i można je pobrać pod adresem <http://genome.ucsc.edu/cgi-bin/hgTables>. Dane te zostały zaimportowane dla dwóch sekwencji referencyjnych HG19 i HG38. Są one najczęściej używane w praktyce. Schemat bazy przedstawiony jest na Rys. 21.



Rys. 21 Model bazy danych

- **Program do importowania bazy** – ze stron uniwersytetu Santa Cruz można pobrać pliki, które zostały zaimportowane do bazy przy pomocy specjalnie przygotowanego programu. Importer ponadto dodaje 10 par zasad na początek i koniec każdego eksonu. Jest to zalecenie specjalistów, gdyż wtedy mamy większą pewność wyniku pokrycia genomu. Program został przygotowany przy użyciu platformy .NET. Jego nazwa techniczna w projekcie brzmi *ImportGene*.

- **Optymalizator pokrycia** – również przygotowany w technologii .NET z nazwą techniczną *BinaryCoverageOptimizer*. Służy do przemiany dużych plików tekstowych na pliki binarne. Przyjmuje on dwa parametry wejściowe, czyli sekwencje referencyjną i ścieżkę do pliku tekstowego. Nazwa pliku tekstowego jest identyczna jak nazwa próbki, którą później posługujemy się w programie.

Przykładowe wywołanie optymalizatora: *BinaryCoverageOptimizer.exe lem2.txt HG19*. Plik *lem2.txt* jest plikiem tekstowym, w którym zawarte są trzy kolumny oddzielone tabulatorami: chromosom, pozycja nukleotydu w chromosomie, wartość pokrycia. Natomiast drugie wywołanie stanowi sekwencje referencyjną. Aplikacja działa obecnie na jednej sekwencji referencyjnej HG19. Natomiast w bazie nazw genów zaimportowana jest również sekwencja HG38, dlatego też w łatwy sposób można dodać obsługę HG19 do systemu.

Po wykonaniu optymalizatora dane zapisane są jak na Rys. 17. Zadaniem optymalizatora jest tworzenie przedziałów z wartości pokrycia, które są zdefiniowane na podstawie tabeli 1 z podrozdziału 3.1. Jeżeli optymalizator na kolejnych nukleotydach napotka wartość znajdującą się w tym samym zakresie łączy nukleotydy w jeden przedział. Optymalizator zapamiętuje wszystkie zakresy dla każdego z chromosomów w pamięci operacyjnej, a na końcu zapisuje dane według formatu z Rys. 17.

W optymalizatorze pokrycia przeprowadzono niewielką zmianę. Zmienił się nieznacznie format wejściowy obsługiwany przez program. Poprzedni format pliku składał się z chromosomu, początku bloku, przesunięcia w bloku i wartości pokrycia. Zmiana polega na tym, że w poprzednim formacie by wyznaczyć pozycję nukleotydu należało zsumować wartości początku bloku i przesunięcia w bloku. Natomiast obecnie mamy ją jawnie podaną w nowej strukturze pliku. Nowy format jest zwracany przez program *bedtools*, który jest używany w celu przetwarzania danych z plików BAM.

Pozostałe komponenty zostały przygotowane całkowicie od nowa. Szerzej zostaną opisane w kolejnych punktach pracy. Lista nowych komponentów przedstawia się następująco:

- **Interfejs użytkownika** – wykonany w wersji przeglądarkowej. Został przygotowany przy pomocy nowoczesnych technologii takich jak na przykład Angular, TypeScript.
- **Warstwa serwerowa** – odpowiada za komunikację pomiędzy interfejsem użytkownika, a pozostałymi komponentami. Pozwala uruchamiać zadania przeliczenia pokrycia, które zostało zaimplementowane przy pomocy biblioteki Adam, i odczytać wartości pokrycia z przygotowanych plików binarnych. Warstwa ta komunikuje się również z bazą danych. Program został przygotowany w technologii .Net Core.
- **Program do liczenia pokrycia** – przygotowany w technologii Scala. Wykorzystuje on bibliotekę ADAM w celu liczenia pokrycia.

4.2 Interfejs użytkownika

Interfejs użytkownika zaprezentowany jest na Rys. 22. Działa on na wielu nowoczesnych przeglądarkach stron internetowych. Interfejs użytkownika powstał przy pomocy wielu technologii. Najważniejsze z nich to Angular 5, TypeScript, JavaScript, Bootstrap, Html 5, css, D3.js, Npm, JQuery. Użytym środowiskiem programistycznym jest Visual Studio Code. Dodatkowymi komponentami użytymi do implementacji interfejsu użytkownika jest File-Saver (służący do pobierania plików), ng2-completer (służy do podpowiadania danych do wprowadzania), ng4-loading-spinner (wyświetlający animację, która sygnalizuje użytkownikowi oczekiwanie podczas dłuższych operacji).

Menu programu, tabele, przyciski oraz inne elementy graficzne powstały przy pomocy biblioteki Bootstrap. Biblioteka ta jest darmowa o otwartym kodzie źródłowym i pozwala w sposób prosty tworzyć przyjazne użytkownikowi interfejsy w aplikacjach internetowych. W głównym menu dostępne są trzy opcje. Są nimi:

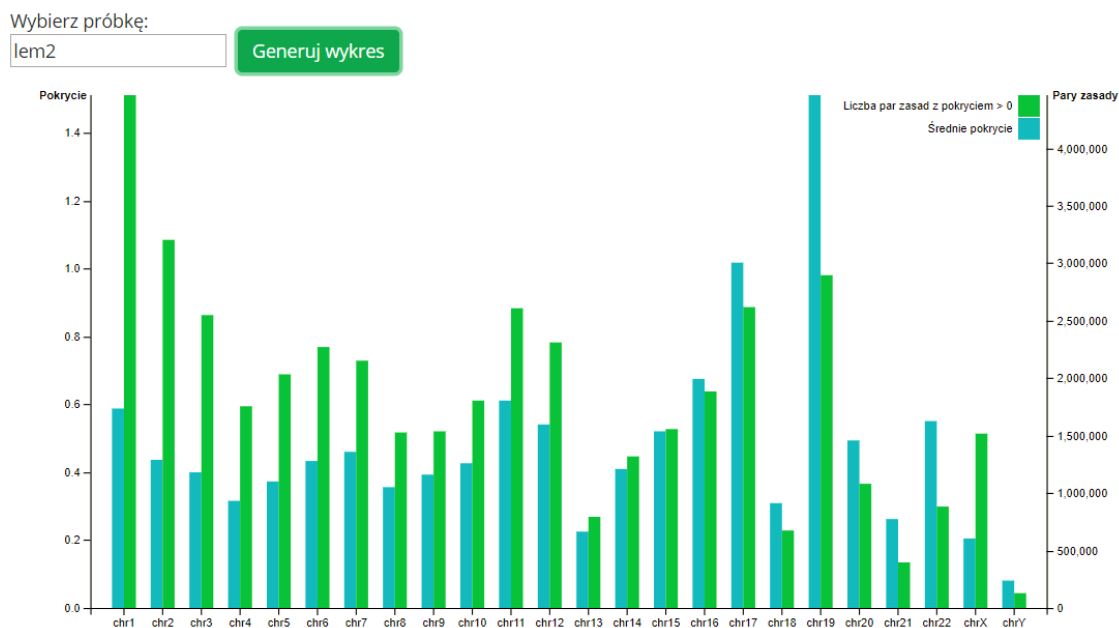
- **Analiza pokrycia** – główna strona programu, w której można przeprowadzać analizy pokrycia.
- **Pokrycie chromosomów** – na tej podstronie można wygenerować wykres, który przedstawia pokrycie na poszczególnych chromosomach oraz ilość par zasad z niezerową wartością pokrycia.
- **Informacje o programie** – zawiera informacje o wersji i autorze.

Główna strona składa się z dwóch tabel, a mianowicie listy próbek i drugiej tabeli, która zawiera transkrypty wraz z nazwami genowymi. Przy wprowadzaniu danych wykorzystane jest podpowiadanie możliwości do wyboru. Każda z dwóch tabel pozwala na dwa rodzaje operacji. Opcje widoczne są na przyciskach, czyli odpowiednio dodawanie i czyszczenie listy. Po uzupełnieniu listy badacz klika przycisk do przeprowadzania analizy. W wyniku tej akcji otrzymywany jest plik tekstowy zawierający wyniki badanego pokrycia.

Rys. 22 Interfejs użytkownika

Wykres na stronie *Pokrycie chromosomów* został przygotowany przy pomocy technologii D3.js. Na ekranie można wybrać interesującą nas próbkę. Po wybraniu

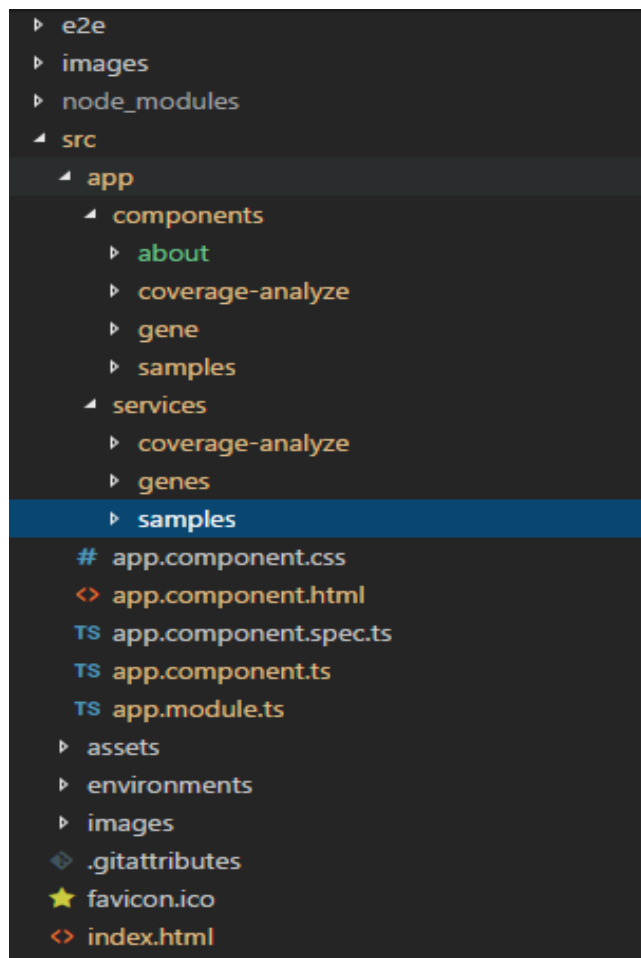
odpowiedniej próbki i kliknięciu przycisku *Generuj wykres* tworzony jest wykres słupkowy. Przykładowy wygląd tego wykresu przedstawiony jest na Rys. 23. Wykres ma trzy osie. Na osi poziomej znajdują się chromosomy. Dla każdego chromosomu mamy dwie wartości: średnie pokrycie oraz ilość par zasad z niezerowym pokryciem. Dzięki niemu badacze są w stanie w szybki sposób zweryfikować jakość sekwencjonowania oraz określić, które z chromosomów zostały najlepiej pokryte. Na przykładowym rysunku największe średnie pokrycie mamy dla chromosomu dziewiętnastego. Natomiast najwięcej niezerowych par zasad mamy na pierwszym chromosomie. Niezerowe wartości pokrycia istnieją na nim dla ponad 4 milionów par zasad.



Rys. 23 Przykładowy wykres pokrycia na poszczególnych chromosomach

W projekcie wykorzystano wzorce powszechnie stosowane w aplikacjach Angular, czyli zastosowano podział na komponenty, moduły i serwisy. Ich tworzenia dokonywano za pomocą *angular-cli*, czyli konsolowego interfejsu technologii Angular. Łączność z serwerem realizowana jest w serwisach. W serwisach stosowany jest wzorec obserwator, który polega na informowaniu obiektów o zmianie swojego stanu.

Podział projektu zaprezentowany jest na Rys. 24. Aplikacja składa się z jednego modułu *app.module*. Kod aplikacji został podzielony na komponent główny *app.component* oraz pięć innych komponentów. Komponent *about* odpowiada za stronę informacyjną o wersji. Kolejne trzy komponenty odpowiadają analizie pokrycia. W analizie pokrycia mamy jeden komponent rodzic *coverage-analyze*, w którym zawierają się dwa inne w postaci tabel w interfejsie użytkownika, czyli *gene* i *samples*. Ponadto jeden z komponentów zawiera w sobie wykres pokrycia w zależności od chromosomów. Wraz z komponentami zaimplementowano towarzyszące im serwisy. Odpowiadają one za zbieranie wyników i komunikację z serwerem. W katalogu *assets* zawarte są skrypty w języku JavaScript, arkusze stylów css oraz obrazki. Główną stroną projektu jest plik *index.html*.



Rys. 24 Struktura projektu interfejsu użytkownika

4.3 Warstwa serwerowa

Warstwa serwerowa zrealizowana jest w technologii .Net Core. Różni się ona od platformy .NET Framework tym, że pozwala na działanie na systemach innych niż Windows. Core umożliwia działanie aplikacji także w systemach Mac OS czy Linux. Warstwa serwerowa odpowiada za komunikację pomiędzy interfejsem użytkownika, a pozostałymi komponentami. Do pozostałych komponentów można zaliczyć bazę genów, program do obliczania pokrycia czy pliki binarne. Projekty składowe warstwy serwerowej są następujące:

- **Genomer** – projekt, który zawiera interfejs dostępu do warstwy serwerowej. W słownictwie technicznym nazywane jako programistyczny interfejs aplikacji, API (ang. *application programming interface*). Poza kontrolerami, które komunikują serwer z klientem zawiera również rejestracje wszystkich klas projektu.
- **Genomer.Services** – zawiera logikę, która jest wywoływana z projektu *Genomer*. W tym projekcie zawarta jest również komunikacja z programem obliczającym pokrycie zrealizowanym w technologii Scala.
- **Genomer.CoverageLib** – w projekcie zaimplementowana jest logika obsługi plików binarnych.

- **Genomer.Contract** – projekt jest zbiorem klas współdzielonych pomiędzy pozostałymi. Celem takiego rozmieszczenia klas jest zapobieganie zależnościom cyklicznym (ang. *circular dependency*).

W czasie implementacji intensywnie wykorzystywany jest wzorzec wstrzykiwania zależności (ang. *dependency injection*). Pozwala on na dostarczanie zależności do klas przy pomocy interfejsów. Interfejsy te podawane są w konstruktorach klasy, a całość realizowana jest z użyciem zasady odwrócenia kontroli (ang. *inversion of control*). Wszystkie zapytania do serwera realizowane są głównie w postaci metody post. Są one realizowane przy pomocy bezstanowego REST API (ang. *Representational State Transfer*). Projekt składa się z czterech kontrolerów:

- **Kontroler *GeneController*** - pozwala na pobranie listy genów. Lista genów wybierana jest z bazy. Ponadto kontroler ten zawiera metodę, która waliduje ciąg wejściowy dla tabeli transkrypt i nazw genowych. Jedną z możliwych wartości jest ciąg znaków w postaci *chromosom: region_od-region_do*, gdzie *chromosom* może przyjmować wartości chr1...chr22, chrX, chrY lub chrM, natomiast *region_od* i *region_do* są wartościami liczbowymi reprezentującymi dany region. Przykładowo taki ciąg może mieć postać chr1:100-2000. Drugą możliwą wartością jest nazwa genu, która istnieje w bazie. Kontroler zwraca informacje, jakiego typu jest szukany tekst. To znaczy czy jest on nazwą genu czy przedziałem oraz informację czy szukany ciąg jest poprawny.
- **Kontroler *SampleController*** – kontroler udostępnia jedną metodę, która zwraca listę wszystkich próbek. Lista generowana jest na podstawie plików w folderze określonym w konfiguracji aplikacji.
- **Kontroler *CoverageAnalyze*** – składa się z jednej metody *PerformAnalyze*. Metoda przyjmuje dwa parametry będące listami łańcuchów znaków. Pierwszym parametrem jest lista genów lub transkryptów, natomiast drugą jest lista próbek. Metoda może działać w dwóch trybach. Pierwszy z nich zwraca wyniki z plików binarnych. Natomiast drugi uruchamia zadanie w Scali i w owym programie liczone jest pokrycie genomu. Rodzaj metody wybrany jest na podstawie parametru konfiguracyjnego aplikacji. Metoda na początku tworzy plik, w którym zapisywane są wyniki analizy. Po przeprowadzeniu analizy plik w postaci strumienia bajtów przekazywany jest do przeglądarki i pobierany przez użytkownika.
- **Kontroler *Chromosomes*** – służy do pobierania danych dla wykresu i zawiera tylko jedną metodę *GetChromosomeAverage*. Metoda przyjmuje jako argument nazwę próbki, z której mają zostać odczytane dane. Kontroler został zaimplementowany tylko w architekturze wykorzystującej pliki binarne.

W skład warstwy serwerowej wchodzi również plik *appsettings.json*. Istnieją dwa rodzaje tego pliku w wersji roboczej i produkcyjnej. Plik zawiera w sobie ustawienia w formacie JSON. Ustawienia, które można podawać do aplikacji przedstawiają się następująco:

- Folder wejściowy, *InputFolder* – w nim zawarte są pliki binarne, które są używane w systemie. Pliki są rozmieszczone w katalogach dla odpowiadającej sekwencji referencyjnej. Mamy dwa rodzaje sekwencji referencyjnych HG19 i HG39. Dla każdej z nich w katalogu *InputFolder* mamy katalogi HG19 i HG38, które przechowują próbki dla konkretnej sekwencji referencyjnej.
- Folder wyjściowy, *OutputFolder* – w tym folderze zapisywane są wyniki analiz, które są następnie zwracane do interfejsu użytkownika.
- Metoda pracy analizy programu, *IsAdamRequest* – wartość tego parametru ustawiona na prawdę oznacza, że analiza będzie odbywała się przy pomocy programu napisanego w Scali. Natomiast wartość fałsz mówi, że dane o pokryciu genomu będą czytane z plików binarnych.
- Folder wejściowy przy działaniu w trybie architektury bazującej na Apache Spark, *ScalaInput* – w przypadku, gdy znacznik *IsAdamRequest* jest ustawiony na logiczną prawdę zwraca wynik z podanego folderu. Gdy wartość *IsAdamRequest* jest fałszywa folderem wejściowym jest określony w parametrze *InputFolder*.

4.4 Analiza pokrycia w technologii Spark

W niniejszym podrozdziale opisana jest aplikacja obliczająca pokrycie z wykorzystaniem technologii Spark i biblioteki Adam. Uruchamiana jest za pomocą komendy z wiersza poleceń: *sbt run*. Argumentem programu jest zbiór próbek i wartości, z których możemy liczyć pokrycie. Dodatkowo podawana jest nazwa pliku wynikowego. Formatem, w którym przekazywane są dane jest json. Przykładowe uruchomienie programu:

```
Sbt "run
{\\\\"Transcripts\\":[{\\\\"Chr\\":1,\\\\"From\\":100,\\\\"To\\":17000}],\\\\"Samples\\":[\\\\"wgEncodeUwRepliSeqBjS2AlnRep1.bam\\\"],\\\\"FileName\\":\\\\"CoverageAnalysis.txt\\\"}"
```

Powyższe wywołanie zawiera wiele znaków ukośnika wstecznego „\”. Jest on używany do prawidłowego przekazania znaków cudzysłów do programu. Te samo wywołanie prawidłowo sformatowane przedstawia się następująco:

```
{
  "Transcripts":[
    {
      "Chr":1,
      "From":100,
      "To":17000
    }
  ],
  "Samples":["wgEncodeUwRepliSeqBjS2AlnRep1.bam"],
```

```
"FileName": "CoverageAnalysis.txt"
```

```
}
```

Powyższy fragment mówi, że analiza zostanie przeprowadzona dla pierwszego chromosomu w zakresie od pozycji 100 do 17000. Jediną próbką poddaną analizie będzie plik *wgEncodeUwRepliSeqBjS2AlnRep1.bam*. Wyniki badania zostaną zapisane w pliku *CoverageAnalysis.txt*.

Na starcie aplikacja przetwarza podany ciąg wejściowy i zamienia go na obiekt. Do tej operacji wykorzystywana jest biblioteka *lift-json*. Następnie wyniki te przekazywane są do klasy, która zajmuje się przeprowadzaniem obliczeń. W klasie tej tworzone są obiekty kontekstu *SparkContext* i *AdamContext*. Po utworzeniu kontekstów na podstawie listy transkryptów tworzona jest tablica obiektów typu *ReferenceRegions*. Następnie dla każdej próbki wykonane jest obliczenie pokrycia przy pomocy funkcji biblioteki Adam *filterByOverlappingRegions*. Na samym końcu dla każdego ze znalezionych wyników odbywa się zapis do pliku. Kod, który zajmuje się owym obliczeniem zaprezentowany jest na Rys. 25.

```
val writer = new BufferedWriter(new FileWriter(coverageAnalyzeModel.FileName))

for (i <- 0 to referenceRegionCount - 1) {
  val transcript = coverageAnalyzeModel.Transcripts(i)
  referenceRegions(i) = new ReferenceRegion("chr"+transcript.Chr.toString(), transcript.From, transcript.To)
}

for (j <- 0 to coverageAnalyzeModel.Samples.length - 1) {
  println(coverageAnalyzeModel.Samples(j))
  val reads = ac.loadAlignments(coverageAnalyzeModel.Samples(j))
  val coverage = reads.toCoverage()
  val foundedCoverage = coverage.filterByOverlappingRegions(referenceRegions)
  val rddCoverage = foundedCoverage.rdd

  rddCoverage.collect().foreach { coverage =>
    val stringToWrite = coverage.contigName + "\t" + coverage.start + "\t" + coverage.end + "\t" + coverage.count + "\r\n"
    writer.write(stringToWrite)
  }
}
```

Rys. 25 Fragment kodu programu obliczającego pokrycie

4.5 Testy systemu

W niniejszym punkcie przeprowadzono testy funkcjonalne tworzonego systemu. Porównano czy wyniki w obu podejściach są takie same oraz czy nazwy genów poprawnie się mapują na eksony. Na końcu sprawdzono działanie w systemie Linux.

Weryfikacja poprawności działania polega tutaj na początkowym użyciu programu *bedtools*. Bardziej dokładny opis argumentów programu *bedtools* przedstawiony jest podczas badań wydajności architektur w punkcie 5.2. W celu zbadania poprawności użyto pliku BAM *wgEncodeUwRepliSeqK562S2AlnRep1.bam*. Program *bedtools* zwrócił plik tekstowy zawierający cały genom o pokryciu, a fragment został umieszczony w tabeli 2. Cały wynik dostępny jest w załącznikach w folderze *Wyniki* pod nazwą *wynik_dzialania_bedtools_1.txt*.

Tabela 2 Wynik działania programu bedtools

Chromosom	Start zakresu	Koniec zakresu	Wartość pokrycia
chr1	10155	10156	3
chr1	10156	10157	5
chr1	10157	10158	11
chr1	10158	10159	20
chr1	10159	10182	22
chr1	10182	10183	19
chr1	10183	10184	17
chr1	10184	10185	11
chr1	10185	10186	2
chr1	10237	10242	1
chr1	10242	10245	2
chr1	10245	10264	3
chr1	10264	10269	2
chr1	10269	10272	1
chr1	540963	540990	1
chr1	565427	565454	1
chr1	566021	566048	1

Następnie sprawdzono oba sposoby uzyskiwania wyników o pokryciu. W tym celu wybrano w interfejsie użytkownika chromosom pierwszy oraz określono przedział od 100 do 17000. Fragment wyniku działania dla programu w technologii Spark zaprezentowano w tabeli 2. Natomiast w załącznikach do pracy dostępne są pod nazwą *wynik_dzialania_scala_1.txt* w folderze *Wyniki*. Przyjrzyjmy się dokładniej obu tabelom. Na przykład w tabeli 3 w wierszu 10 pokrycie dla chromosomu pierwszego w przedziale od 10269 do 10272 jest równe dokładnie jeden. Natomiast w tabeli 3 widzimy pięć kolejnych linii odpowiadających temu wynikowi. Chodzi tu o linie od 32 do 36:

chr1 10237 10238 1.0

chr1 10238 10239 1.0

chr1 10239 10240 1.0

chr1 10240 10241 1.0

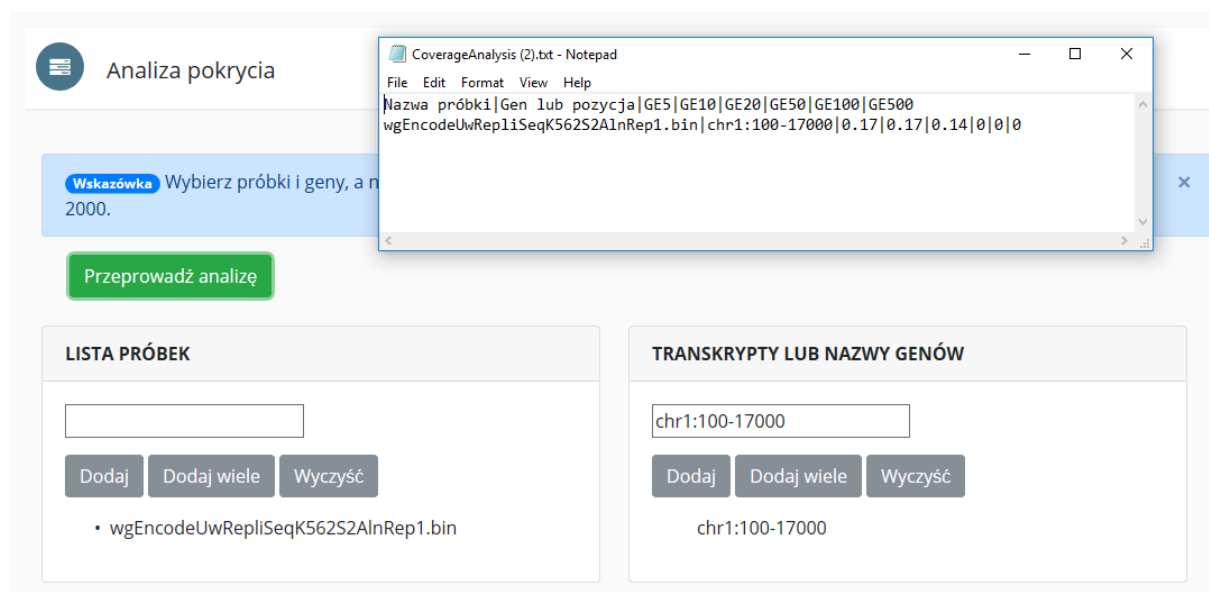
chr1 10241 10242 1.0

Wyniki w technologii Scala są zestawiane dla każdego nukleotydu, a nie dla przedziałów. Po dokładniejszym przyjrzeniu widzimy, że są one takie same, a więc program działa zgodnie z oczekiwaniami.

Tabela 3 Wynik działania aplikacji w programie technologii Spark dla przedziału chr1:100-17000

Chromosom	Start zakresu	Koniec zakresu	Wartość pokrycia
chr1	10180	10181	22
chr1	10181	10182	22
chr1	10182	10183	19
chr1	10183	10184	17
chr1	10184	10185	11
chr1	10185	10186	2
chr1	10237	10238	1
chr1	10238	10239	1
chr1	10239	10240	1
chr1	10240	10241	1
chr1	10241	10242	1
chr1	10242	10243	2
chr1	10243	10244	2
chr1	10244	10245	2
chr1	10245	10246	3
chr1	10246	10247	3

Kolejnym testem było sprawdzenie rezultatów podejścia związanego z plikami binarnymi. W celu realizacji na początku zamieniono plik źródłowy BAM na plik binarny. Następnie zmieniono tryb działania aplikacji by obsługiwał pliki binarne i wpisano przedział chr1:100-17000. Rezultat działania widoczny jest na Rys. 26.



Rys. 26 Wynik działania aplikacji w podejściu związanym z plikami binarnymi

Wyjaśnieniu wymaga plik wynikowy zaprezentowany na powyższym rysunku. Otóż rekord w pliku składa się z nazwy próbki, genu lub pozycji i wartości. Wartości to kolejno GE5,

GE10, GE20, GE50, GE100, GE500. Litery GE oznaczają, że pokrycie jest większe lub równe niż wartość występująca po nich (ang. *greater equal*). Wartości, które mają pokrycie mniejsze od 5 są pomijalne, gdyż nie mają one znaczenia dla badaczy. Wartości te są wyrażone w procentach, czyli gdy mamy wartość GE5 równą 40% oznacza to, że 40% zadanego przedziału miało pokrycie większe lub równe od 5.

Na podstawie Rys. 27 widzimy, że wartość GE20 jest równa 0.14%. Natomiast dzięki tabeli 3 wiemy, że pokrycie większe od 20 mamy w przedziale od 10158 do 10182. Długość całego wynosi 17000 – 100, czyli 16900, a długość przedziału, w którym wartości są większe od 20 wynosi 10158 – 10182, czyli 24. Po podzieleniu tych dwóch wartości przez siebie otrzymujemy, że mamy pokrycie większe bądź równe 20 na 0.14% długości całego przedziału.

Nazwa próbki	Gen lub pozycja	GE5	GE10	GE20	GE50	GE100	GE500
wgEncodeUwRep1SeqK562S2A1nRep1.bin	chr13:20763029-20763730	0	0	0	0	0	0
wgEncodeUwRep1SeqK562S2A1nRep1.bin	GJB2	0	0	0	0	0	0

Rys. 27 Wyniki dla genu GJB2

W celu sprawdzenia poprawności mapowania na geny zapytano o gen GJB2. Gen ten ma tylko jeden ekson na trzynastym chromosomie w przedziale od 20763029 do 20763730. Zapytano, więc o ten gen oraz o przedział chr13: 20763029-20763730. W przypadku poprawnego mapowania nazwy genowej na pozycje obie wartości powinny być identyczne. Wykonano, więc opisane zapytania. Rezultat jaki uzyskano zaprezentowany jest na Rys. 27. Oba rezultaty mają taką samą wartość, a mianowicie pokrycie równe jest zero. Test potwierdził, więc poprawność mapowania nazw genów na pozycje. W celu dalszego badania poprawności analogiczne testy jak te opisane powyżej zostały powtórzone dla innych przedziałów i genów. Inne testy również zakończyły się sukcesem.

Dodatkowym testem było przeprowadzenie działania aplikacji w systemie Linux. Docelowo bardzo często w zastosowaniach bioinformatycznych infrastruktura serwerowa działa na systemach typu Linux. Firma Microsoft zapewnia, że .NET Core działa z powodzeniem również właśnie w tych systemach. Pomimo to zweryfikowano działanie aplikacji w systemie Linux, a dokładniej Ubuntu 16.0.4. Za pomocą komendy *dotnet build* wykonano kompilację. Pozytywny rezultat kompilacji zaprezentowany jest na Rys. 28. Ponadto, cały system zachowuje się poprawnie właśnie w tym środowisku.

```
Genomer -> /home/marek/Pulpit/Genomer/genomer.backend/Genomer/bin/Debug/netcoreapp2.0/Genomer.dll
Genomer.Services.Tests -> /home/marek/Pulpit/Genomer/genomer.backend/Genomer.Services.Tests/bin/Debug/netcoreapp2.0/Genomer.Services.Tests.dll

Kompilacja powiodła się.
Ostrzeżenia: 0
Liczba błędów: 0

Czas, który upłynął: 00:00:25.87
marek@marek-ThinkPad-Edge-E531:~/Pulpit/Genomer/genomer.backend$
```

Rys. 28 Rezultat kompilacji w systemie Ubuntu

5 Przeprowadzone badania

Niniejsza część pracy zawiera badania nad architekturami aplikacji. Na początku uwaga została skupiona na ogólnym porównaniu architektur, bez korzystania z przygotowanej implementacji. Następnie przeprowadzono szereg testów wydajnościowych, gdyż to one głównie mogą powiedzieć, która architektura jest lepsza. Badano dwa podejścia do architektury:

- Zastosowanie dedykowanego formatu plików binarnych jako struktury przechowującej dane o pokryciu. Podejście opisane wcześniej w podrozdziale 3.3.
- Wykorzystanie technologii Spark i biblioteki Adam. Opis tego podejścia znajduje się w podrozdziale 3.4.

W ostatnim podrozdziale przeprowadzono czytelnika przez przykładową analizę pokrycia z wykorzystaniem zaimplementowanego systemu.

5.1 Wstępne rozważania na temat architektur

W tym punkcie zostanie dokonana ogólna analiza porównawcza podejść do obliczania pokrycia genomu. Dla przypomnienia pierwsze z nich wykorzystuje kompresję danych do specjalnego pliku binarnego. Następnie na skompresowanych plikach odczytywane są dane o pokryciu. Przetwarzanie odbywa się klasycznie na jednym węźle. Natomiast drugie podejście, wykorzystuje narzędzia do przetwarzania dużych zbiorów danych w tym technologii Spark i bibliotekę Adam.

Oczywistą zaletą technologii Spark jest skalowalność obliczeń na wiele węzłów. Są to technologie dedykowane do radzenia sobie z masowymi danymi, więc naturalnym dla nich zastosowaniem wydają się być genomy. Istotna jest również odporność na błędy, czyli jeżeli jeden węzeł przestanie z jakiegoś powodu działać, system jako całość nadal jest w stanie kontynuować swoją pracę. Ponadto przy pierwszym podejściu zastosowaliśmy kompresję stratną. Poniekąd spełnia ona założenia stawiane przed systemem. Jednak zastosowanie jej zawsze jest pewnym minusem, gdyż nie dostaniemy w stu procentach dokładnej informacji. Na pewnym etapie rozwoju systemu założenia, które poczyniliśmy mogą okazać się nieaktualne i chcielibyśmy mieć dostęp do wszystkich danych, nawet tych, które zostały utracone w kompresji stratnej. Podejście z wykorzystaniem technologii Spark jest wolne od tej wady. Bardzo dużym plusem drugiego podejścia jest odczyt bezpośrednio z danych źródłowych. W pierwszym podejściu należy wykonać kilka kroków by móc odczytywać wartości pokrycia z plików binarnych. Takie rozwiązanie ma jednak zasadniczą wadę potrzeby przechowywania i zarządzania przetworzonymi plikami.

Spark ponadto dobrze współdziała w rozproszonym systemie plików HDFS. W praktyce próbki bardzo często są właśnie z rozproszonych systemów. W pierwszym, klasycznym podejściu musimy sami zadbać o system plików. Adam potrafi współdziałać również z różnymi formatami bioinformatycznymi. Jak pokazano w rozdziale drugim bioinformatyka cechuje się bardzo dużą ilością formatów. Adam potrafi dobrze pracować

z SAM, BAM, CRAM, ADAM, BED, GFF3, GTF czy VCF co jest jego ogromnym plusem. Wielu badaczy pracowało już nad tym projektem, toteż został wykorzystany w różnych bioinformatycznych zastosowaniach. Podejście bazujące na plikach binarnych wymaga jednak rozwoju oprogramowania dla obsługiwanych formatów. Należałoby zaimplementować dodatkowe programy, które sprowadzą ze znanych i używanych formatów bioinformatycznych do plików binarnych. Kolejną przewagą zastosowania biblioteki Adam jest to, że zapewnia on wiele więcej funkcji niż tylko liczenie pokrycia. Zarówno Spark i Adam są bezpłatne i istnieje możliwość wglądu w ich kod źródłowy.

Z drugiej strony zastosowanie technologii Spark wydaje się mieć kilka wad. Po pierwsze stosowanie tych narzędzi ma sens tylko w dużej infrastrukturze informatycznej. Utrzymanie takiego rozwiązania wiąże się z wyższymi kosztami. Alternatywą jest stosowanie chmury, czyli przykładowo rozwiązań typu Microsoft Azure, Amazon Web Services czy Google Cloud Platform. Ponadto gromadzenie informacji o kodach genetycznych na serwerach wielkich korporacji może budzić kontrowersje.

Tabela 4 Zestawienie cech obu architektur

Cecha	Dedykowany format binarny	Technologia Apache Spark i biblioteka Adam
Odczyt z pliku	Odczyt po sprowadzeniu do dedykowanego formatu binarnego.	Bezpośredni odczyt, brak konieczności stosowania nowego formatu i wstępnego przetwarzania.
Działanie na wielu węzłach	Brak wbudowanego wsparcia. Algorytmy muszą być samodzielnie zaimplementowane. Możliwość łatwego skalowania wstępnego przetwarzania, gdyż jest ono wykonywane niezależnie dla każdej próbki.	Skalowalność obliczeń jest naturalna dla technologii Spark. Wsparcie dla rozproszonych systemów pliku. Ponadto, odporność na błędy na pojedynczych węzłach.
Zużycie pamięci dyskowej	Poza danymi źródłowymi sekwencjonowania, konieczność przechowywania plików binarnych.	Przechowywane muszą być tylko dane źródłowe, pochodzące z sekwencjonowania. Brak dodatkowego narzutu na zużycie pamięci.
Infrastruktura informatyczna	Brak wymagania na dodatkową infrastrukturę informatyczną. Możliwość działania nawet na komputerach osobistych.	Technologia dedykowana dla dużych infrastruktur. Wiąże się to z większymi kosztami utrzymania niż w pierwszym podejściu.
Rodzaj kompresji	Kompresja stratna. Zapisywana jest tylko informacja w jakim zakresie znalazło się pokrycie.	Odczyt bezpośrednio z pliku, więc nie stosowana jest żadna kompresja.

Dojrzałość projektu	Stosowanie dedykowanego formatu binarnego jest nowym pomysłem. Brak osób, które nad podejściem tym pracują.	Zarówno nad projektem Adam jak i Apache Spark pracuje wiele osób. Projekty rozwijane od lat.
Wsparcie dla formatów bioinformatycznych	Dla każdego formatu konieczność pisania dodatkowego programu, które utworzy z danych źródłowych pliki w dedykowanym formacie.	Adam zapewnia wsparcie dla wielu popularnych formatów bioinformatycznych.
Możliwość rozszerzenia systemu o inne bioinformatyczne funkcje.	Brak wbudowanego wsparcia dla innych funkcji niż liczenie pokrycia.	Adam wspiera nie tylko obliczanie pokrycia, ale też inne operacje jak na przykład złączenia zsekwencjonowanych odcinków, możliwość wyznaczania wyrównań.

Podsumowanie istotnych cech przedstawiono w tabeli 4. Biorąc pod uwagę powyższe zestawienie, podejście do architektury bazujące na technologii Spark w większości z nich ma przewagę. Jedynym minusem jest konieczność zastosowania dużej infrastruktury informatycznej. Natomiast wyraźnymi plusami jest bezpośredni odczyt z pliku, brak wstępnego przetwarzania, wsparcie dla wielu formatów bioinformatycznych, dojrzałość projektu czy zużycie pamięci dyskowej.

5.2 Analiza działania obu programów

W tym punkcie zostaną przeanalizowane pod względem wydajności oba podejścia. Testy wydajnościowe wykonywane są na tym samym komputerze osobistym na jednym klastrze. W badaniach zostaje pominięta część odpowiedzialna za samo uruchomienie programu z interfejsu użytkownika, gdyż jest ona identyczna dla obu architektur i nie jest ona interesująca. Procedura badań wygląda następująco:

- Uruchomienie programu, który przetwarza pokrycie za pomocą technologii Scala.
- Przetworzenie pliku BAM do pliku tekstowego za pomocą programu bedtools.
- Przetworzenie wyniku działania programu bedtools do pliku binarnego za pomocą specjalnie przygotowanego narzędzia.
- Odczytanie wyniku z przygotowanego pliku binarnego.

W każdym z punktów mierzony będzie czas wykonania i zapisywane będą uzyskane wyniki. Pierwszy punkt dotyczy tylko i wyłącznie podejścia opartego na technologii Scala. Trzy kolejne punkty dotyczą przetwarzania do plików binarnych.

Drugi punkt wymaga szerszego wyjaśnienia, gdyż użyte zostało w nim zewnętrzne oprogramowanie, czyli program bedtools. Wywołanie przetwarzania w bedtools wygląda następująco:

```
genomeCoverageBed -ibam myBamFile.bam -dz > output_coverage.txt
```

Szczegółowe wyjaśnienie powyższego wywołania:

- -ibam – jest parametrem określającym, że na wejściu zostanie podany plik bam.
- myBamFile.bam – jest przykładowym, wejściowym plikiem bam.
- -dz wskazuje, że pokrycie jest rozpisane dla każdego z nukleotydów. Litera „z” dodaje pomijanie wartości równych 0.
- output_coverage.txt – wynikowy plik tekstowy

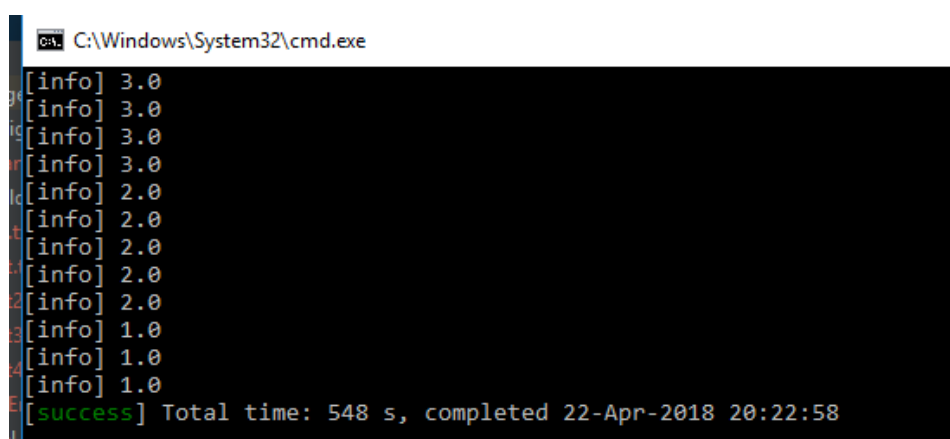
Ciekawym wydaje się być zastosowanie programu bedtools ze wskazaniem znacznika -bg zamiast -dz. Znacznik ten oznacza, że dane zostaną zwrócone w formacie bedgraph. Format ten oznacza, że mamy oddzielone tabulatorami wartości chromosomu początku bloku i końca, a na końcu wartość pokrycia. Przykładowo reprezentacja w tym formacie wygląda następująco:

```
chr1 1000 2000 60
```

Oznacza to, że w przedziale od 1000 do 2000 pokrycie jest równe 60.

W celu przeprowadzenia testów wydajnościowych poszukiwane były jak najbardziej różnorodne pliki BAM. Chodziło o to, żeby znaleźć pliki różnych rozmiarów, zarówno małe jak i duże. Takie rozmiary plików znaleziono na stronie: <http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeUwRepliSeq/>. Udostępnia ona pliki BAM z sekwencji referencyjnej HG19.

Badania zaprojektowano tak, by stopniowo zwiększać ilość zapytań do systemu. Początkowo uwzględniono tylko zapytanie dla jednej próbki i na jednym chromosomie. W pierwszym badaniu chcemy się dowiedzieć, jaka jest wartość pokrycia na chromosomie pierwszym w zakresie pokrycia od 100 do 17000.



```
C:\Windows\System32\cmd.exe
[info] 3.0
[info] 3.0
[info] 3.0
[info] 3.0
[info] 2.0
[info] 2.0
[info] 2.0
[info] 2.0
[info] 2.0
[info] 2.0
[info] 1.0
[info] 1.0
[info] 1.0
[success] Total time: 548 s, completed 22-Apr-2018 20:22:58
```

Rys. 29 Wyniki działania programu w Scali

Badanie wykonano na pliku BAM, który miał rozmiar 47 megabajtów. Plik ma nazwę wgEncodeUwRepliSeqK562S2AlnRep1.bam. Identyczne badanie wykonano dziesięciokrotnie, średni czas działania wynosił 540 sekund. Wyniki zaprezentowano na Rys. 29. Podczas

wykonania obserwujemy bardzo duże zużycie pamięci operacyjnej i procesora. Zużycie pamięci operacyjnej utrzymuje się przez większość czasu wykonania programu na poziomie 2-2,5 gigabajtów. Natomiast zużycie procesora zmienia się od kilku procent nawet do 70. Przykładowy zrzut ekranu z zużycia widzimy na Rys. 30.

	86%	62%	3%	0%
	CPU	Memory	Disk	Network
me				
Windows Command Processor ...	64.0%	2,167.3 MB	0 MB/s	0 Mbps

Rys. 30 Zużycie pamięci podczas działania programu

Spróbujmy, więc wprowadzić kilka innych przedziałów, w których będzie szukane pokrycie. Przykładowo gen VAC14 ma aż 19 eksonów. Średni czas, w którym wykonano badania w technologii Spark jest równy 413 sekund. Ciekawym jest fakt, że średni czas jest mniejszy niż dla badania jednego przedziału. Jednak, gdy przyjrzymy się bliżej większa liczba niezerowych wartości pokrycia została znaleziona dla jednego, większego przedziału. W przypadku pierwszego badania znaleziono pokrycie na 66 parach zasad. Natomiast w drugim badaniu znaleziono wartości pokrycia tylko na 27 parach zasad. Stąd otrzymujemy lepszy wynik w drugim przypadku. Wyniki z drugiego badania możemy obejrzeć w folderze Wyniki w pliku *wynik_dzialania_scala_2.txt*.

Kolejnym testem wykonanym w technologii Spark było sprawdzenie, jak zachowa się program przy większej liczbie próbek. W tym celu użyto tej samej próbki co wcześniej, ale była ona dwukrotnie odpytywana. Pytano o obszar na genie VAC14, który przy jednej próbce zanotował czas 413 sekund. Zgodnie z oczekiwaniami osiągnęliśmy dwa razy dłuższy czas wyszukiwania pokrycia. Średni czas wynosił 830 sekund, czyli prawie 14 minut.

W kolejnych badaniach zmieniono plik na nieco większy *wgEncodeUwRepliSeqBjS2AInRep1.bam*, który ma rozmiar 112 megabajtów. Podobnie jak poprzednio pytano o eksony genu VAC14. Okazało się jednak, że technologia Spark nie poradziła sobie z tym plikiem przy istniejących zasobach na komputerze osobistym. Podczas przetwarzania otrzymany został wyjątek *java.lang.OutOfMemoryError: Java heap space*. Przy mocniejszej infrastrukturze z dużym prawdopodobieństwem udałoby się przeprowadzić badania. Jednak świadczy to o słabości tego podejścia, gdyż pliki rzędu 100 megabajtów nie są jeszcze dużymi w świecie bioinformatyki. Spotykane są pliki BAM o rozmiarach kilka gigabajtów.

Te same badania zostały powtórzone dla drugiego podejścia związanego z plikami binarnymi. Początkowo sprowadzono dane do pliku binarnego za pomocą programu bedtools. Pierwszym rozpatrywanym plikiem był podobnie jak dla Sparka *wgEncodeUwRepliSeqBjS2AInRep1.bam*. Zanim jednak poddano go pytaniu o pokrycie wymagane było przeprowadzenie wstępnego przetwarzania. Wykonanie tego zadania sprowadza się do dwóch etapów:

- Sprowadzenie danych do pliku tekstowego za pomocą programu bedtools.
- Zmiana pliku tekstowego do dedykowanego formatu binarnego.

Przetwarzanie do formatu, w którym mamy plik tekstowy z wydzielonym pokryciem na każdy nukleotyd zajmuje średnio 1:45 min. Niestety po tej operacji wynikowy plik tekstowy miał 447 megabajtów, więc jest to całkiem sporo biorąc pod uwagę, że plik wejściowy stanowił tylko 49 megabajtów. Plik ten jednak po przetworzeniu możemy bez konsekwencji usunąć, gdyż jest on potrzebny tylko do uzyskania pliku binarnego, a potem nie potrzebujemy składować go w systemie. Następnie za pomocą optymalizatora pokrycia sprowadzono go do pliku binarnego. Etap ten trwał 14 sekund. Wytworzony plik zajmował tylko 29 kilobajtów.

Tak przetworzony plik mógł zostać już poddany odpytywaniu o pokrycie. Na początku spytano o przedział chr1:100-17000. Odpowiedź była natychmiastowa, a samo dostanie się do wyniku w pliku binarnym trwało kilka milisekund. Identyczne zachowanie obserwujemy również dla innych wartości, czyli na przykład dla genu VAC14. Podejście to również sprawdziło się przy badaniu na pliku *wgEncodeUwRepliSeqBjS2AlnRep1.bam* o większym rozmiarze. Po wstępnym przetworzeniu odpowiedzi były bardzo szybkie.

Tabela 5 Zestawienie wyników testów wydajności aplikacji

Opis testu	Dedykowany format binarny	Technologia Apache Spark
Zapytanie: chr1:100-17000 Próbka: wgEncodeUwRepliSeqK562S2AlnRep1.bam Rozmiar próbki: 46 MB Rozmiar pliku binarnego: 29 KB	1 etap przetwarzania: 105 s. 2 etap przetwarzania: 14 s. Czas odpowiedzi: 0.003 s.	Czas odpowiedzi: 540 s.
Zapytanie: VAC14 (19 eksonów) Próbka: wgEncodeUwRepliSeqK562S2AlnRep1.bam Rozmiar próbki: 46 MB Rozmiar pliku binarnego: 29 KB	1 etap przetwarzania: 105 s. 2 etap przetwarzania: 14 s. Czas odpowiedzi: 0.003 s.	Czas odpowiedzi: 410 s.
Zapytanie: VAC14 (19 eksonów) Próbka: 2x wgEncodeUwRepliSeqK562S2AlnRep1.bam Rozmiar próbki: 46 MB Rozmiar pliku binarnego: 29 KB	1 etap przetwarzania: 105 s. 2 etap przetwarzania: 14 s. Czas odpowiedzi: 0.003 s.	Czas odpowiedzi: 830 s.
Zapytanie: VAC14 (19 eksonów) Próbka: wgEncodeUwRepliSeqBjS2AlnRep1.bam Rozmiar próbki: 112 MB Rozmiar pliku binarnego: 25 KB	1 etap przetwarzania: 260 s. 2 etap przetwarzania: 62 s. Czas odpowiedzi: 0.004 s.	Brak poprawnego wykonania programu.
Zapytanie: ZCWPW1 (1 ekson) Próbka: wgEncodeUwRepliSeqSkshS3AlnRep1.bam Rozmiar próbki: 112 MB Rozmiar pliku binarnego: 3,5 MB	1 etap przetwarzania: 560 s. 2 etap przetwarzania: 240 s. Czas odpowiedzi: 0.004 s.	Brak poprawnego wykonania programu.

Zapytanie: GJB2 (1 ekson) Próbka: lem2 Rozmiar pliku binarnego: 10 MB	Dane w postaci tekstowej, brak pierwszego etapu. 2 etap przetwarzania 350 s. Czas odpowiedzi: 0.004 s.	Brak możliwości przeprowadzenia testu.
--	--	--

Zestawienie wyników testów wydajności przedstawione jest w tabeli 5. Widać w niej, że w przypadku większych plików Spark nie radzi sobie z przetwarzaniem wartości o pokryciu. Ponadto, ciekawą obserwacją jest to, że czas odpowiedzi dla różnych plików binarnych jest zawsze zbliżony do siebie i wynosi tylko kilka milisekund. Co więcej, czas przetwarzania do formatu binarnego rośnie wraz ze wzrostem plików wejściowych.

Nieco innym przypadkiem jest plik lem2, który został dostarczony przez badaczy z Warszawskiego Uniwersytetu Medycznego. Plik ten nie był w formacie BAM, a w postaci pliku tekstowego po pierwszym etapie przetwarzania, dlatego też nie wykonano na nim testów w technologii Spark.

Uwzględniając powyższe widać, że nawet z bardzo długim wstępnym przetwarzaniem implementacja związana z plikami binarnymi jest znacząco szybsza. Odpowiedź na zapytania po przetworzeniu jest natychmiastowa, a dane nie zajmują dużo miejsca. Zajętość pamięci wynosi niespełna 29 kilobajtów dla pliku, którego dane źródłowe zajmują 46 megabajtów. Oczywiście rozbudowana architektura sprzętowa mogłaby znacząco przyspieszyć badania oparte na Sparku. Jednak nadal musimy pamiętać o tym, że dane pomiędzy węzłami są przesyłane przy pomocy Apache AVRO z wprowadzaniem dodatkowego narzutu na rozmiar danych. Wysyłka pomiędzy węzłami realizowana jest w formacie JSON. Wynik powyżej 8 minut nie jest zadowalający nawet przy infrastrukturze złożonej z jednego klastra.

5.3 Analiza pokrycia przy pomocy zaimplementowanego systemu

W tym rozdziale przeprowadzona zostanie analiza przy pomocy końcowej implementacji systemu. Ten podrozdział ma na celu pokazanie, w jaki sposób korzystać z systemu oraz przeprowadzanie badań na kilku różnych próbkach.

W badaniach zostaną użyte pliki, które pochodzą z Warszawskiego Uniwersytetu Medycznego jako dane testowe. Poza tymi plikami zostały użyte również te, które można znaleźć na wspomnianej już stronie udostępniającej pliki BAM dla sekwencji referencyjnej HG19, czyli <http://hgdownload.cse.ucsc.edu>.

Pliki użyte w badaniach przedstawiają się następująco:

- lem2 – plik pochodzący jako przykład z Zakładu Genetyki Medycznej. Dostarczony w formie wyniku działania programu bedtools o rozmiarze około 3 gigabajtów. Po przetworzeniu do pliku binarnego zajmował 10 megabajtów. Próbka pochodzi z sekwencjonowania eksonowego, czyli takiego, w którym sekwencjonowaniu poddawane były tylko eksony.

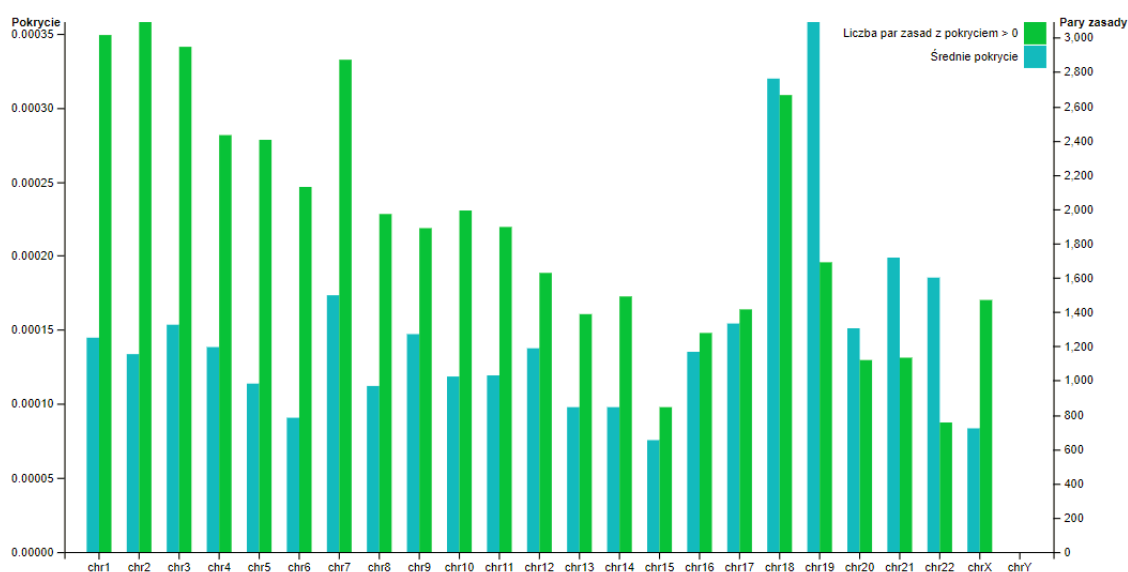
- wgEncodeUwRepliSeqK562S2AlnRep1.bin - plik, który po przetworzeniu miał rozmiar 29 kilobajtów. Przed przetworzeniem plik BAM o rozmiarze 46 megabajtów.
- wgEncodeUwRepliSeqSknshS3AlnRep1.bin – plik, który po przetworzeniu miał rozmiar 3,546 kilobajtów. Przed przetworzeniem plik BAM o rozmiarze 925 megabajtów.

Binarne pliki wynikowe zostały dodane w postaci załączników do pracy. Celem badań jest sprawdzenie jak dobrze zsekwencjonowano próbki. Ponadto chcemy sprawdzić, czy pokrycie na określonych genach jest większe niż zadane. Konkretnym celem jest sprawdzenie:

- Czy gen ZCWPW1 ma pokrycie powyżej 10 dla ponad 99% całej swojej długości.
- Czy mamy jakiekolwiek pokrycie w przedziale chr7:150000-200000.
- Czy gen VAC14 ma pokrycie powyżej 20 dla ponad 80% całej swojej długości.
- Czy gen AKR1B10 ma pokrycie powyżej 50 dla ponad 50% całej swojej długości

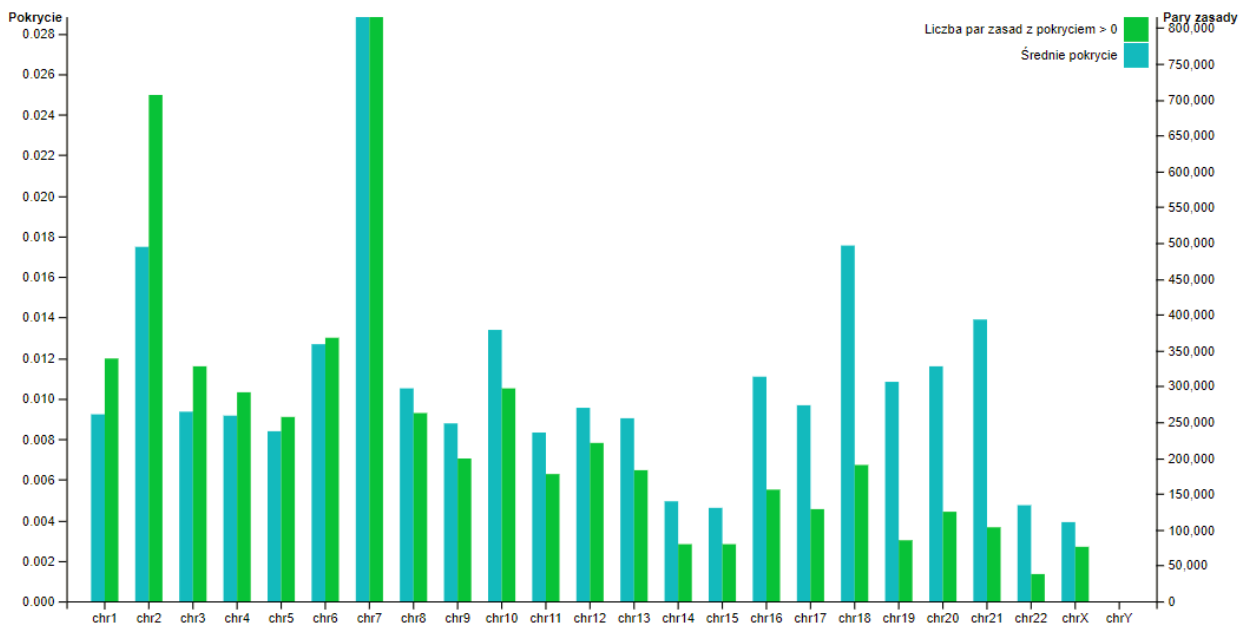
Tak obrany cel jest przedstawiony tylko dla prezentacji działania systemu. W rzeczywistości ocena określenie dana próbka jest dobrze pokryta bazuje na artykułach i publikacjach naukowych dla konkretnego badania. Jak wspomniano w rozdziale 2.2 tego typu publikacje możemy odnaleźć na stronie producenta maszyn do sekwencjonowania, firmy Illumina. Na stronie znajduje się informacja, że by poprawnie wnioskować o polimorfizmie pojedynczego nukleotydu, pokrycie musi być większe niż 20 [4].

Na początku zaczniemy analizę od najmniejszego pliku wgEncodeUwRepliSeqK562S2AlnRep1.bin, który zajmuje tylko 29 kilobajtów. Przed przetworzeniem do pliku binarnego będąc w postaci BAM zajmuje 46 megabajtów. Pierwszym krokiem badania będzie sprawdzenie jaki mamy ogólny rozkład pokrycia na poszczególnych chromosomach. Z pomocą przychodzi wykres, który możemy wygenerować w aplikacji w zakładce *Pokrycie Chromosomów*. Wykres prezentuje Rys. 31.



Rys. 31 Wykres dla najmniejszej próbki wgEncodeUwRepliSeqK562S2AlnRep1

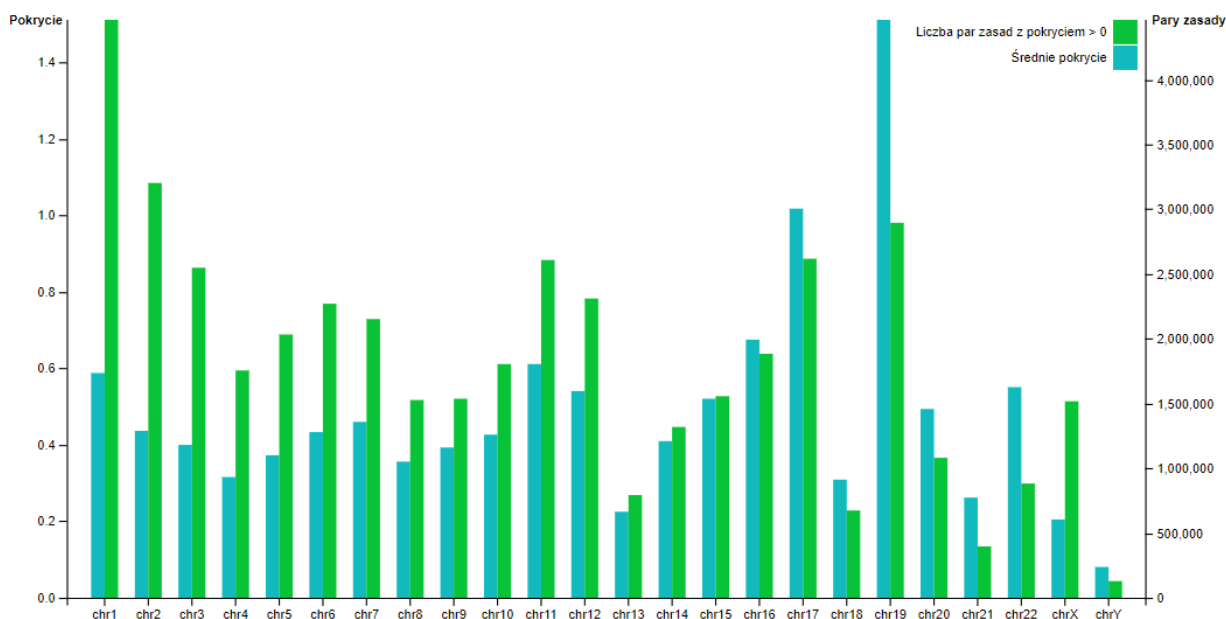
Na rysunku tym widać, że próbka jest słabo pokryta. Najwyższe pokrycie obserwujemy na chromosomie dziewiętnastym i wynosi ono nieco ponad 0.00034 par zasad. Sprawdźmy zatem jak wyglądają wykresy dla dwóch pozostałych próbek. Najwięcej nukleotydów z niezerowym pokryciem dla tej próbki otrzymano na chromosomie drugim. Jednak jest to niespełna 3000 par zasad. Długość chromosomu drugiego wynosi ponad 243 miliony par zasad. Zatem pokryty mamy jeden nukleotyd na kilkaset tysięcy.



Rys. 32 Wykres dla próbki wgEncodeUwRepliSeqSknshS3AlnRep1

Rys. 32 i Rys. 33 przedstawiają informację o kolejnych próbkach, odpowiednio wgEncodeUwRepliSeqK562S2AlnRep1.bin i lem2. Pierwsza z nich ma największe pokrycie na chromosomie siódmym i wynosi ono 0.028 par zasad. Natomiast w próbce lem2 najlepiej pokryty jest chromosom dziewiętnasty. Najwyższe średnie pokrycie wynosi 1.5 par zasad. Jako jedyna próbka lem2 ma też pokryty chromosom Y. Zważywszy na to, że sekwencjonowanie na tej próbce było eksonowe i wiedząc, że eksony zajmują tylko 10% naszego genomu otrzymujemy dużo lepszą jakość sekwencjonowania niż na dwóch pozostałych próbkach. Przy takim założeniu o długości eksonów średnie pokrycie na nich wynosi aż 15 par zasad.

Jeżeli chodzi natomiast o liczbę par zasad z niezerową wartością pokrycia na wykresie z Rys. 31 również obserwujemy na chromosomie siódmym. Wartość ta wynosi 800000 par zasad. Natomiast na Rys. 32 najwięcej niezerowych wartości pokrycia obserwujemy na chromosomie pierwszym z liczbą przekraczającą 4 miliony par zasad. Ciekawą obserwacją jest to, że zawsze najgorsze statystyki uzyskujemy na chromosomie Y. Na wykresach dwóch próbek wgEncodeUwRepliSeqK562S2AlnRep1 i wgEncodeUwRepliSeqSknshS3AlnRep1 widać, że nie mamy żadnego nukleotydu pokrytego w tym chromosomie. Można to uzasadnić tym, że chromosom występuje tylko u mężczyzn, a dwie próbki z zerową wartością pokrycia pochodzą z dużym prawdopodobieństwem od kobiet. Natomiast próbka lem2 prawdopodobnie pochodzi od mężczyzny. Chromosom Y jest tam najslabiej pokryty ze wszystkich, ale mimo wszystko możemy zaobserwować słupki pokazujące pary zasady z niezerowym pokryciem.



Rys. 33 Wykres dla największej próbki lem2

Po przeprowadzeniu wstępnej analizy za pomocą wykresu przechodzimy do bardziej szczegółowej interpretacji. W tym celu wybieramy najpierw zakładkę *Analiza pokrycia*. Następnie wybieramy interesujące nas próbki, a także nazwy genów i przedziały. Będą to geny i przedziały wskazane na początku rozdziału oraz próbki, dla których sporządzono wykresy. Klikając przycisk *Przeprowadź analizę* otrzymujemy wyniki jak w tabeli 6.

Tabela 6 Zestawienie wyników dla interesujących nas genów

Nazwa próbki	Gen lub pozycja	GE5	GE10	GE20	GE50	GE100	GE500
lem2	chr7:150000-200000	1.71	1.29	0.91	0	0	0
lem2	AKR1B10	99.22	99.22	99.14	52.89	0.78	0
lem2	ZCWPW1	99.34	99.34	92.33	21.81	0	0
lem2	VAC14	99.38	99.38	86.21	10.81	0.4	0
wgEncodeUwRepliSeq SknshS3AInRep1.bin	chr7:150000-200000	1.84	0.1	0	0	0	0
wgEncodeUwRepliSeq SknshS3AInRep1.bin	AKR1B10	0	0	0	0	0	0
wgEncodeUwRepliSeq SknshS3AInRep1.bin	ZCWPW1	0	0	0	0	0	0
wgEncodeUwRepliSeq SknshS3AInRep1.bin	VAC14	0	0	0	0	0	0
wgEncodeUwRepliSeq K562S2AInRep1.bin	chr7:150000-200000	0	0	0	0	0	0
wgEncodeUwRepliSeq K562S2AInRep1.bin	AKR1B10	0	0	0	0	0	0
wgEncodeUwRepliSeq K562S2AInRep1.bin	ZCWPW1	0	0	0	0	0	0
wgEncodeUwRepliSeq K562S2AInRep1.bin	VAC14	0	0	0	0	0	0

Tabela 6 potwierdza to co zostało zaobserwowane na podstawie analizy wykresów. Otóż najlepsze pokrycie obserwujemy dla próbki lem2. Dla tej próbki wszystkie cele, które podane zostały na początku rozdziału zostały więc spełnione. Ponadto, wiemy, że pokrycie na eksonach genu ZCWPW1 w 90% jest większe niż 20, więc przykładowo możemy z dużym prawdopodobieństwem przeprowadzać eksperymenty określające polimorfizm pojedynczego nukleotydu w tym obszarze. Oprócz przedziału chr7:150000-200000 na pozostałych próbkach obserwujemy zerowe pokrycie. Natomiast niezerową wartość dla tego przedziału można zauważyć na próbce wgEncodeUwRepliSeqSknshS3AInRep1.bin. W tym przypadku jest ono nawet większe niż dla próbki lem2. Na wykresie dla tej próbki widać było najlepsze pokrycie dla chromosomu siódmego.

Przeprowadzona analiza pokazuje w jaki sposób przy pomocy systemu analizować wartości pokrycia. Dzięki wykresom prezentującym rozkład na chromosomach, badacz szybko jest w stanie ocenić pokrycie dla całej próbki. W podobny sposób można dokonywać innych analiz w zakładach genetycznych.

6 Wnioski

Jak widać z powyższego materiału, przedstawiono szerokie spektrum rozważań, analiz i porównań architektur systemów do badania pokrycia genomu. Natomiast, ostatni rozdział koncentruje się na podsumowaniu całej pracy. Przedstawiono, w jaki sposób cel pracy został zrealizowany. Z uwagi na wiele różnych koncepcji, które pojawiły się podczas pisania niniejszej pracy, ostatni podrozdział zawiera propozycję udoskonaleń do programu i dalszych badań w tym zakresie.

6.1 Zrealizowanie celu pracy

Celem niniejszej pracy jest zbadanie dwóch różnych architektur systemów, które pozwalają obliczać pokrycie genomu. Badanie ma dać odpowiedź, która z dwóch architektur jest lepsza. Dla przypomnienia pierwsza z nich stosuje kompresję danych i pliki binarne. Druga architektura korzysta z technologii Apache Spark i projektu Adam. Po ogólnej analizie, bez faktycznej implementacji, wydawać by się mogło, że architektura oparta o technologię Spark i wspierana biblioteką Adam ma większy potencjał.

Aspekty, które świadczą o przewadze technologii Apache Spark i biblioteki Adam nad stosowaniem dedykowanego formatu binarnego:

- Odczyt bezpośrednio z pliku źródłowego. Brak wstępnego przetwarzania. Brak dodatkowego zużycia dysku.
- Możliwość łatwej pracy w wielu węzłach.
- Brak kompresji stratnej.
- Wiele badaczy pracuje nad rozwojem zarówno Apache Spark jak i biblioteki Adam.
- Adam wspiera popularne formaty bioinformatyczne.

Jedną z nielicznych wad, którą można by wskazać jest to, że Apache Spark potrzebuje większej infrastruktury informatycznej.

Po wstępnej analizie przeprowadzono testy wydajnościowe z wykorzystaniem zaimplementowanego systemu. Kompresja danych do plików binarnych dała bardzo dobre rezultaty, gdyż czas odpowiedzi na zapytania wynosił kilka milisekund, zarówno dla dużych jak i małych plików. W architekturze tej dochodzi jeszcze czas wstępnego przetwarzania, który wynosi kilka minut. Jednak należy pamiętać, że dokonuje się go jednokrotnie dla każdej z próbek. Natomiast przy stosowaniu technologii Apache Spark dla małych plików, czyli około 50 MB, otrzymano odpowiedź w kilka minut. Przy nieco większych plikach, które miały rozmiar około 100 MB, w technologii Apache Spark nie można było wykonać poprawnie programu. Badania odbywały się na jednym węźle, ale biorąc pod uwagę, że biblioteka Adam tak słabo radziła sobie przy stosunkowo małych plikach, nie można się spodziewać, że przy wielu węzłach będzie dużo lepiej. Biorąc pod uwagę, że w laboratoriach genetycznych niektóre pliki mają rozmiary rzędu kilka gigabajtów, wyklucza to całkowicie zastosowanie biblioteki Adam. Wnioskiem z tej pracy jest to, że dedykowany format binarny lepiej sprawdza się w obliczeniach pokrycia. Oczywiście, zarówno biblioteka Adam jak i technologia Apache

Spark ma wiele innych zastosowań, w których mogą się doskonale sprawdzać. Jednak w momencie pisania tej pracy w badaniu pokrycia nie sprawdzają się dobrze.

Praca ta poniekąd wyjaśnia też skąd bierze się tak duża liczba różnego rodzaju formatów w bioinformatyce. Tak więc praca ta ma walor dydaktyczny. Otóż, z uwagi na wielką ilość informacji często trzeba tworzyć odmienne formaty, które pozwalają w sposób efektywny uzyskiwać dostęp do danych. W niniejszej pracy również powstał nowy, dedykowany format binarny.

Innym ciekawym wnioskiem jest to, że bardzo ogólny trend panujący w informatyce nie zawsze jest w pełni słuszny. Chodzi tu o stosowanie gotowych bibliotek. Oczywiście nie ma to na celu kwestionowania ich wykorzystania, ponieważ wiąże się z tym mnóstwo zalet. Jedynie chodzi o pokazanie, że czasami gotowa implementacja dostarczona przez bibliotekę nie spełnia naszych oczekiwań. Być może wtedy jesteśmy w stanie zbudować własną, która lepiej realizuje nasze wymagania. Należy pamiętać, że mimo ich zastosowania musimy dokładnie wiedzieć, jak one działają. W przypadku biblioteki Adam sprawa jest o tyle ułatwiona, że kod biblioteki jest dostępny w internecie i każdy może go samodzielnie zweryfikować.

Zdaniem autora największymi osiągnięciami niniejszej pracy jest porównanie oraz implementacja dwóch różnych architektur systemu do badania pokrycia genomu. Zadanie to wymagało użycia wielu technologii informatycznych. Ponadto, zastosowano ciekawą wizualizację średniego pokrycia wraz z ilością niezerowych nukleotydów. Nowatorstwo tej pracy polega na stworzeniu i sprawdzeniu dedykowanego formatu binarnego, który okazał się lepszy niż architektura bazująca na technologii Apache Spark.

6.2 Możliwości dalszej rozbudowy

Istnieje szereg możliwości dalszego rozwoju opracowanej tu aplikacji zarówno pod kątem badawczym jak i czysto funkcjonalnym.

Ciekawym udoskonaleniem zdaniem autora wydaje się zastosowanie wspomianej w pracy bazy Apache Hbase. Według opisu Apache Hbase stworzony jest dla rozwiązań, gdzie w tabelach przechowujemy miliardy rekordów i miliony kolumn. Odnosząc go do zastosowania w liczeniu pokrycia niewątpliwie mielibyśmy miliardy rekordów nawet dla pojedynczych próbek. Kolumn wystarczyłoby kilka, czyli chromosom, start zakresu, koniec zakresu i ewentualnie kolumny identyfikujące próbkę. Z dokumentacji wynika, że Apache Hbase dobrze sprawdza się w systemach rozproszonych [44].

Innym wartym rozważenia przedsięwzięciem mogłoby być zbadanie jak biblioteka Adam zachowuje się w innych niż BAM formatach. Być może liczenie pokrycia w tym formacie jest wyjątkowo źle zaimplementowane, a pozostałe formaty dają dobre wyniki. Przykładowo moglibyśmy sprowadzić pliki BAM najpierw do formatu ADAM, a dopiero potem liczyć pokrycie. Oczywiście takie podejście przysparza wielu problemów znanych z rozwiązania z plikami binarnymi. Chodzi tu o zarządzanie plikami i ich przechowywanie. Warto tu dodać, że Adam jest projektem otwarto źródłowym, dlatego też sami moglibyśmy spróbować usprawnić liczenie pokrycia w Adamie.

Pod względem możliwości dalszej rozbudowy wiele można zrobić w podejściu związanym z plikami binarnymi. Przede wszystkim warto by było rozwiązać problem automatyzacji przetwarzania do tego formatu. Można zaimplementować funkcję, która ma do wglądu również pliki, które nie są obecnie w formacie binarnym, a popularnym dla bioinformatyki jak BAM, SAM czy VCF. Interfejs użytkownika musiałby wyszczególnić próbki, które nie są jeszcze sprowadzone do formatu binarnego oraz takie, które posiadają już specjalną reprezentację. Jeżeli użytkownik wybrałby próbkę, której nie ma w plikach binarnych, program powinien automatycznie przetworzyć ją i dopiero podać wyniki. Czas przetwarzania takiej próbki oczywiście byłby dłuższy, dlatego niezbędne jest rozróżnienie pomiędzy plikami przetworzonymi, a nie przetworzonymi w interfejsie użytkownika. Innym zadaniem jest przyspieszenie obliczeń plików binarnych poprzez realizację jej na wielu węzłach. Kompresja do plików binarnych mogłaby bez problemu zostać poddana zrównolegleniu na wiele serwerów, ponieważ pojedyncza próbka może być sprowadzana do pliku binarnego całkowicie niezależnie.

Można także usprawnić przetwarzanie plików binarnych. Otóż na ten moment system ma dwa etapy przetwarzania. Najpierw z pliku BAM do pliku tekstowego za pomocą programu bedtools, a następnie przy użyciu napisanego programu do dedykowanego formatu binarnego. Dużo bardziej efektywna byłaby redukcja tego przetwarzania do jednego etapu, czyli na podstawie pliku BAM od razu można tworzyć plik binarny.

Problemem jaki stoi przed narzędziem jest tu mnogość formatów, jakie stosowane są w bioinformatyce. Dotyczy on tylko wersji binarnej, gdyż Adam ma wsparcie dla wielu formatów. Tak jak wspomniano w trzecim rozdziale często nowy program wprowadza również nowy format pliku. Istnieje więc potrzeba obsługi chociaż najpopularniejszych formatów bioinformatycznych takich jak ADAM, FASTA, BAM, SAM, VCF. Do każdego takiego formatu należałoby napisać dedykowany program, który utworzy plik binarny z wartościami pokrycia.

Pod względem czysto informatycznego rozwoju systemu należałoby rozważyć możliwość większej ilości logów aplikacyjnych. Jest to istotne ze względów utrzymaniowych systemu. Co więcej ważną i popularną funkcją wydaje się możliwość logowania i zarządzania użytkownikami aplikacji. Jeszcze innym aspektem technicznym jest przeniesienie tekstów występujących w programie w jedno miejsce i zaimplementowanie obsługi wielu języków.

Tak więc jak widać, istnieje wiele koncepcji dalszego udoskonalenia oprogramowania wytworzonego w ramach niniejszej pracy.

Bibliografia

- [1] Wikipedia. Human Genome Project.
https://en.wikipedia.org/wiki/Human_Genome_Project, dostęp: 18.03.2018.
- [2] M. C. Proc. Strona instytutu ncbi. The Human Genome Project, 2011 04.
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3068906/>, dostęp: 18.03.2018.
- [3] R. Płoski. Film w serwisie youtube.com. Sekwencjonowanie następnej generacji: rewolucja w genetyce i onkogenetyce.
<https://www.youtube.com/watch?v=Y5Iyh5lFFxc>, dostęp: 05.05.2018.
- [4] Strona internetowa firmy Illumina. Sequencing Coverage.
<https://www.illumina.com/science/education/sequencing-coverage.html>, dostęp: 08.05.2018.
- [5] M. Moraczyński. Tworzenie aplikacji do badania pokrycia genomu. Praca dyplomowa, Politechnika Warszawska. Warszawa, 2017.
- [6] Serwis bnd.ibe.edu.pl. Nici DNA. <https://bnd.ibe.edu.pl/tool-page/524>, dostęp: 22.05.2018.
- [7] P. Węgleński, Genetyka molekularna, Warszawa: Wydawnictwo Naukowe PWN SA, 2015.
- [8] Wikipedia. Chromosome. <https://en.wikipedia.org/wiki/Chromosome>, dostęp: 09.05.2018.
- [9] Serwis physics.leidenuniv.nl. Second layer of information in DNA confirmed. Lion News.
<https://www.physics.leidenuniv.nl/index.php?id=11573&news=889&type=LION&ln=EN>, dostęp: 29.10.2017.
- [10] Serwis majordifferences.com. Difference between Global and Local Sequence Alignment. <http://www.majordifferences.com/2016/05/difference-between-global-and-local.html>, dostęp: 15.10.2017.
- [11] Strona projektu samtools. Sequence Alignment/Map Format Specification.
<https://samtools.github.io/hts-specs/SAMv1.pdf>, dostęp: 29.07.2017.
- [12] Wikipedia. Multiple sequence alignment.
https://en.wikipedia.org/wiki/Multiple_sequence_alignment, dostęp: 15.10.2017.
- [13] Serwis bioinfopoint.com. Multiple sequence alignment with Bioperl and Muscle.
<http://bioinfopoint.com/index.php/code/3-multiple-sequence-alignment-with-bioperl-and-muscle>, dostęp: 15.10.2017.
- [14] Serwis dwavesys.com. D-Wave Initiates Open Quantum Software Environment.
<https://www.dwavesys.com/press-releases/d-wave-initiates-open-quantum-software-environment>, dostęp: 15.10.2017.
- [15] Serwis metagenomics.wiki. Coverage (read or sequencing depth). metagenomics.wiki, <http://www.metagenomics.wiki/pdf/definition/coverage-read-depth>, dostęp: 30.03.2018.

- [16] Dokumentacja projektu bedtools. Bedtools: a powerful toolset for genome arithmetic. <http://bedtools.readthedocs.io/en/latest/>, dostęp: 07.10.2017.
- [17] E. Banachowicz. Serwis www.staff.amu.edu.pl/~ewas/pracownia/biomono/old08/Bioinformatyka4.pdf, dostęp: 29.07.2017.
- [18] Strona uniwersytetu Santa Cruz. Data file formats. <http://genome.ucsc.edu/FAQ/FAQformat.html#format1>, dostęp: 29.07.2017.
- [19] Serwis [quma.cdb.riken.jp](http://quma.cdb.riken.jp/help/gbHelp.html). GenBank format. <http://quma.cdb.riken.jp/help/gbHelp.html>, dostęp: 29.07.2017.
- [20] Strona bazy pdb. Protein Data Bank. <https://www.wwpdb.org/>, dostęp: 15.10.2017.
- [21] Wikipedia. List of sequence alignment software. https://en.wikipedia.org/wiki/List_of_sequence_alignment_software, dostęp: 29.10.2017.
- [22] Wikipedia. Bioinformatics software. https://en.wikipedia.org/wiki/Category:Bioinformatics_software, dostęp: 29.10.2017.
- [23] Dokumentacja aplikacji GATK. GATK tool documentation. <https://software.broadinstitute.org/gatk/documentation/tooldocs/current/>, dostęp: 30.07.2017.
- [24] M. H. E. B. Aaron McKenna. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. Spring, nr 16, 2014.
- [25] Strona aplikacji QualiMap. QualiMap. <http://qualimap.bioinfo.cipf.es/>, dostęp: 30.07.2017.
- [26] D. J. L. W. R. Pearson. Improved tools for biological sequence comparison.. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC280013/>, dostęp: 07.10.2017.
- [27] Strona instytutu ncbi. Basic Local Alignment Search Tool. <https://blast.ncbi.nlm.nih.gov/Blast.cgi>, dostęp: 08.10.2017.
- [28] Strona aplikacji TCOFFE. <http://www.tcoffee.org/>, dostęp: 15.10.2017.
- [29] Strona przeglądarki genomów Uniwersytetu Santa Cruz. UCSC Genome Browser. genome.ucsc.edu, dostęp: 29.10.2017.
- [30] Aplikacja ExAC. ExAC Browser Beta. <http://exac.broadinstitute.org>, dostęp: 29.10.2017.
- [31] Strona instytutu ncbi. An active registry for bioinformatics web services. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2722997/>, dostęp: 29.10.2017.
- [32] D. J. R. X. M. F.-S. M. Y. Galperin. Strona serwisu academic.oup.com. The 2016 database issue of Nucleic Acids Research and an updated molecular biology database collection. <https://academic.oup.com/nar/article/44/D1/D1/2503124/The-2016-database-issue-of-Nucleic-Acids-Research>, dostęp: 15.10.2017.
- [33] Strona bazy Data Bank of Japan. INSDC – International Nucleotide Sequence Database Collaboration. <http://www.ddbj.nig.ac.jp/insdc/insdc-e.html>, dostęp: 15.10.2017.

- [34] Strona instytutu ncbi. Genbank. <https://www.ncbi.nlm.nih.gov/genbank>, dostęp: 15.10.2017.
- [35] Wikipedia. Gene Disease Database. https://en.wikipedia.org/wiki/Gene_Disease_Database, dostęp: 22.10.2017.
- [36] Serwis internationalgenome.org. IGSR and the 1000 Genomes Project. <http://www.internationalgenome.org/>, dostęp 20.04.2018.
- [37] Serwis rcsb.org. RCSB PDB. <https://www.rcsb.org/pdb/secondary.do?p=v2/secondary/search.jsp>.
- [38] Dokumentacja projektu Apache Spark. Spark Overview. <https://spark.apache.org/docs/latest/index.html>, dostęp: 22.04.2018.
- [39] T. Simonite. Serwis technologyreview.com. Moore's Law Is Dead. Now What?, 13.05.2016. <https://www.technologyreview.com/s/601441/moores-law-is-dead-now-what/>, dostęp: 21.04.2018.
- [40] J. P. Moreno. Serwis slideshare.net. Introduction to Apache Spark, 3.10.2015. <https://www.slideshare.net/juanpedromoreno/introduction-to-apache-spark-54677095>, dostęp: 23.04.2018.
- [41] G. Gupta. Serwis dzone.com. Hadoop MapReduce vs. Apache Spark, 04.12.2017. <https://dzone.com/articles/apache-hadoop-vs-apache-spark>, dostęp: 21.04.2018.
- [42] Dokumentacja projektu Adam. Adam overview. <http://adam.readthedocs.io/en/latest/>, dostęp: 21.04.2018.
- [43] Strona projektu adam w serwisie github.com, <https://github.com/bigdatagenomics/adam>, dostęp: 21.04.2018.
- [44] Dokumentacja projektu Apache HBase. <https://hbase.apache.org/>, dostęp: 22.05.2018.

Spis rysunków

Rys. 1 Nić DNA ilustrująca zasadę komplementarności [6].	8
Rys. 2 Długości chromosomów wraz z ilością genów [8].	9
Rys. 3 Różnica pomiędzy dopasowaniem lokalnym i globalnym [10].	10
Rys. 4 Wizualizacja MSA pochodząca z programu ClustalX [13].	12
Rys. 5 Ilustracja pojęć pokrycia i szerokości genomu [15].	13
Rys. 6 Wynik działania programu bedtools [16].	14
Rys. 7 Przykład formatu BED [18].	15
Rys. 9 Przykład formatu SAM [11].	16
Rys. 10 Zrzut obrazu z programu QualiMap.	18
Rys. 11 Zrzut ekranu z przeglądarki genomów Uniwersytetu Santa Cruz [29].	20
Rys. 12 Zrzut z programu ExAC dla genu GJA10 [30].	20
Rys. 13 Zespół trzech organizacji i ich baz połączony w jedną INSDC [33].	21
Rys. 14 Statystyki pokazujące wzrost ilości par zasad w bazach GenBank i WGS [34].	22
Rys. 15 Wykres przedstawiający liczbę udostępnionych struktur w formacie PDB [37].	24
Rys. 16 Przykładowe przedstawienie formatu, który umożliwia odpytywanie o wartości pokrycia.	26
Rys. 17 Aplikacja w pierwszej, okienkowej wersji [5].	28
Rys. 18 Schemat formatu pliku binarnego.	29
Rys. 19 Architektura technologii Spark [40].	30
Rys. 20 Schematyczne przedstawienie architektury ADAM [42].	32
Rys. 21 Diagram rozmieszczenia komponentów obecnej architektury aplikacji.	33
Rys. 22 Model bazy danych.	34
Rys. 23 Interfejs użytkownika.	36
Rys. 24 Przykładowy wykres pokrycia na poszczególnych chromosomach.	37
Rys. 25 Struktura projektu interfejsu użytkownika.	38
Rys. 26 Fragment kodu programu obliczającego pokrycie.	41
Rys. 27 Wynik działania aplikacji w podejściu związanym z plikami binarnymi.	43
Rys. 28 Wyniki dla genu GJB2.	44
Rys. 29 Rezultat kompilacji w systemie Ubuntu.	44
Rys. 30 Wyniki działania programu w Scali.	48
Rys. 31 Zużycie pamięci podczas działania programu.	49
Rys. 32 Wykres dla najmniejszej próbki wgEncodeUwRepliSeqK562S2AlnRep1.	52
Rys. 33 Wykres dla próbki wgEncodeUwRepliSeqSknsH3AlnRep1.	53
Rys. 34 Wykres dla największej próbki lem2.	54

Spis tabel

Tabela 1 Zestawienie istotnych przedziałów dla wartości pokrycia.....	24
Tabela 2 Wynik działania programu bedtools.	41
Tabela 3 Wynik działania aplikacji w programie technologii Spark	42
Tabela 4 Zestawienie cech obu architektur.....	45
Tabela 5 Zestawienie wyników testów wydajności aplikacji.	49
Tabela 6 Zestawienie wyników z interesujących nas genów.....	53

Spis załączników

1. Pliki wejściowe:
 - a. wgEncodeUwRepliSeqK562S2A1nRep1.bam.
2. Wyniki:
 - a. wynik_działania_bedtools_1.txt.
 - b. wynik_działania_scala_1.txt.
 - c. wynik_działania_scala_2.txt.
 - d. wynik_plik_binarny_1.bin.
3. Pliki binarne:
 - a. lem2
 - b. wgEncodeUwRepli-SeqBjS2A1nRep1.bin
 - c. wgEncodeUwRepliSeqImr90S1A1nRep1.bin
 - d. wgEncodeUwRepliSeqK562S2A1nRep1.bin
 - e. wgEncodeUwRepliSeqSknshS3A1nRep1.bin

Dodatek A: Konfiguracja i uruchamianie aplikacji

Wymagane oprogramowanie:

- node.js 9.5.0
- Java 8
- Visual Studio 2017
- Scala 2.11
- Sbt 1.11
- bedtools

Uruchomienie aplikacji internetowej:

- Pobranie zależności interfejsu użytkownika, w katalogu *Genomer.Frontend* wykonać komendę *npm install*.
- Budowanie warstwy serwerowej: otworzyć plik *Genomer.sln* w Visual Studio 2017 i kliknąć *Build Solution* lub z linii poleceń w katalogu *Genomer.Backend*: *dotnet build*
- Uruchomienie warstwy serwerowej: otworzyć plik *Genomer.sln* w Visual Studio 2017 lub *dotnet run*.
- Uruchomienie interfejsu użytkownika: *ng serve --proxy-config proxy.config.json*.

Wgranie przykładowych plików:

- Przekopiować zawartość folderu "*Genomer Przykładowe pliki*" na dysk C:\
- Alternatywnie zmienić konfiguracje w plikach: *appsettings.json* i *appsettings.Development.json*. Domyślnie konfiguracja ustawiona jest jak poniżej:
 - "InputFolder": "C:\\Genomer",
 - "OutputFolder": "C:\\Genomer\\Output",
 - "ScalaInput": "C:\\Genomer\\ScalaInput",
 - "IsAdamRequest": false

Budowanie poprzedniej wersji aplikacji:

- W katalogu *Genomer.PreviousVersion* otworzyć plik *DnaCoverage.sln*.
- Zbudować projekt w środowisku Visual Studio 2017.

Wstępne przetwarzanie za pomocą programu bedtools (1 etap):

- Po instalacji programu bedtools. Wykonać komendę:
genomeCoverageBed -ibam myBamFile.bam -dz > output_coverage.txt,
gdzie:
 - *myBamFile.bam* – jest wejściowym plikiem w formacie BAM
 - *output_coverage.txt* – jest wynikiem działania programu bedtools

Wstępne przetwarzanie za pomocą optymalizatora pokrycia *BinaryCoverageOptimizer* (2 etap):

- Skompilować poprzednią wersję aplikacji, tak jak opisano w *Budowanie poprzedniej wersji aplikacji* lub skompilować projekt *BinaryCoverageOptimizer.csproj* przy pomocy Visual Studio
- W pliku *App.config* tego projektu ustawić katalog wyjściowy, domyślnie jest to folder *C:\Genomer*. Ustawienie jest zaprezentowane w sekcji poniżej

```
<BinaryCoverageOptimizer.Properties.Settings>
  <setting name="OutputFolder" serializeAs="String">
    <value>C:\Genomer</value>
  </setting>
</BinaryCoverageOptimizer.Properties.Settings>
```

- Uruchomić przetwarzanie aplikacji za pomocą komendy:
BinaryCoverageOptimizer.exe output_coverage.txt HG19,
gdzie:
 - *output_coverage.txt* – jest plikiem wejściowym, który jest wynikiem działania programu bedtools
 - *HG19* – jest sekwencją referencyjną, obsługiwane wartości przez optymalizator: HG19 i HG38, obsługiwane wartości przez nową aplikację: HG19

Zmiana trybu działania aplikacji:

- W katalogu *Genomer.Backend/Genomer* w plikach *appsettings.Development.json* i *appsettings.json* zmienić parameter *IsAdamRequest*.
 - *IsAdamRequest* ustawione na *true* działa z wykorzystaniem technologii Apache Spark i biblioteki Adam
 - *IsAdamRequest* ustawione na *false* działa z wykorzystaniem plików binarnych
- Po zmianie parametru należy ponownie uruchomić aplikację.