AKADEMIA ŚWIĘTOKRZYSKA
IM. JANA KOCHANOWSKIEGO W KIELCACH

WYDZIAŁ MATEMATYCZNO-PRZYRODNICZY
Kierunek: Informatyka

ŁUKASZ KOZŁOWSKI

Praca licencjacka

ZASTOSOWANIE ALGORYTMU GENETYCZNEGO DO WYZNACZANIA SKŁADU AMINOKWASOWEGO BIAŁEK

Praca przyjęta pod względem
merytorycznym i formalnym
w formie papierowej i elektronicznej

.....

Promotor pracy:
dr Ignacy Pardyka

KIELCE 2007

Spis treści:

1. WSTĘP

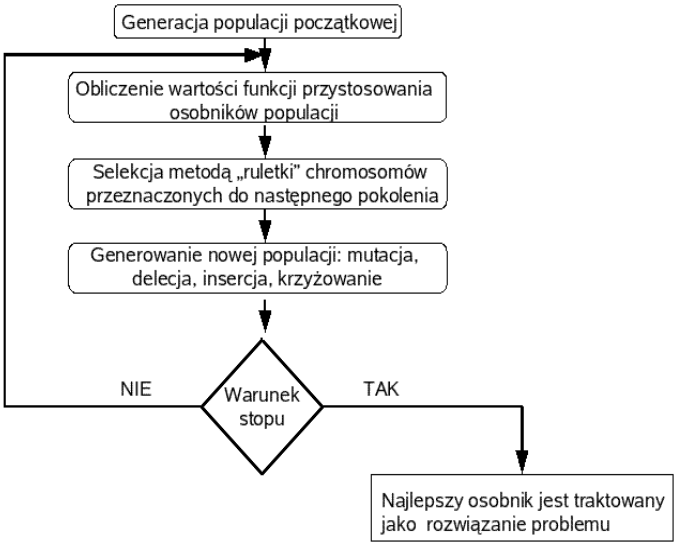
W chwili obecnej rozwój nauk biologicznych znajduje się w specyficznym momencie. Od kilkunastu lat cały wysiłek naukowców skierowany był na sekwencjonowanie materiału genetycznego różnych organizmów czego uwieńczeniem było odczytanie genomu człowieka w 2003 roku. Pomimo szybkiego rozwoju technik sekwencjonowania, w których pierwszoplanową rolę odegrały komputery, istnieje potrzeba zastosowania komputerów w coraz to nowych fazach tego procesu w celu jego przyspieszenia lub przerzucenia niektórych klasycznie „laboratoryjnych" etapów na barki komputera. Jest to szczególnie kusząca perspektywa zważywszy na olbrzymią moc obliczeniową dzisiejszych komputerów. W niniejszej pracy przedstawię analizę możliwości wykorzystania oprogramowania komputerowego w wyznaczaniu składu białek. Obecnie etap ten wykonywany jest za pomocą skomplikowanej i kosztownej metody spektrometrii mas. W pracy tej zaprezentowano inne podejście, które polega na wyznaczeniu składu aminokwasowego białek na podstawie parametrów, które można w prosty i tani sposób uzyskać nawet w słabo wyposażonym laboratorium. Parametrami tymi będą masa, punkt izoelektryczny i absorbancja. Dzięki znajomości składu aminokwasowego z dużym prawdopodobieństwem można wysunąć wnioski m.in. o funkcji i lokalizacji białek.

Należy zaznaczyć, że nie jest to jedyna tego typu praca. Wykorzystanie wymienionych parametrów do identyfikacji białek jest techniką szeroko stosowaną, jednak opiera się ona na przeszukiwaniu bazy danych białek (Protein Data Bank). W ten sposób działa między innymi serwis ProtParam dostępny na serwerze ExPASy Szwajcarskiego Instytutu Bioinformatyki [1]. Pomimo niewątpliwych zalet takiego podejścia, jakimi są prostota i szybkość, oczywistą wadą tej metody jest fragmentaryczność bazy PDB, przez co wynik ograniczony jest jedynie do białek o znanej aktualnie strukturze. Pod tym względem prezentowane w pracy podejście jest unikatowe, ponieważ pozwala ominąć tą wadę. Niestety nie jest to zadanie łatwe. Im dłuższa sekwencja tym więcej możliwych rozwiązań, które różnią się od siebie jedynie w nieznacznym stopniu. Już dla krótkich białek o długości 200 aminokwasów różnych sekwencji jest 10^{260}. Nawet jeśli ograniczymy się jedynie do składu aminokwasowego, tak jak w tym konkretnym przypadku, nadal pozostaje 10^{27} układów czyli o wiele za dużo, aby można wszystkie przetestować w celu znalezienia rozwiązania. W tym momencie pozostaje uciec się do metod numerycznych, które przeszukają przestrzeń potencjalnych rozwiązań w „inteligentny" sposób, skupiając się wokół obszarów w których znalezienie rozwiązania jest najbardziej prawdopodobne. Jedną z takich metod jest algorytmika genetyczna, która powstała w wyniku

naśladowania procesów ewolucji zachodzących w otaczającym nas świecie. Organizmy żywe, niejednokrotnie zadziwiające swoją złożonością, powstały w wyniku konsekwentnego stosowania prostej zasady polegającej na tym, że osobniki lepiej przystosowane do warunków środowiska mają większą szansę na pozostawienie potomstwa, w wyniku czego korzystne cechy przekazywane są do następnego pokolenia, a zanikają te gorsze. Jednocześnie informacja genetyczna może ulegać przypadkowym zmianom, zarówno w czasie życia osobnika jak również w trakcie wykonywania kopii DNA dla potomstwa (replikacja). Choć większość takich zmian jest niekorzystna, znikomy procent może okazać się korzystny - wnosząc nową jakość. Dodatkowym mechanizmem zwiększającym różnorodność populacji jest krzyżowanie się organizmów w czasie którego dochodzi do rekombinacji (crossing over) czyli wymiany homologicznych odcinków DNA, która umożliwia przetasowanie genów rodziców. Z matematycznego punktu widzenia ewolucja polega na optymalizacji olbrzymiej liczby parametrów zapisanych w genomie osobnika pod kątem wielu funkcji określających przystosowanie do otaczającego środowiska.

2. ALGORYTMY GENETYCZNE – PODSTAWOWE POJĘCIA I TERMINY

Najogólniej ujmując, algorytm genetyczny to szczególna forma probabilistycznego algorytmu przeszukiwania przestrzeni rozwiązań. W w każdej iteracji t tworzona jest populacja osobników $P(t) = \{x_1, ..., x_n\}$. Osobnik (zwany w algorytmach także chromosomem) reprezentuje potencjalne rozwiązanie problemu zakodowane w specyficznej strukturze danych (najczęściej jest to ciąg binarny). Ponadto rozwiązania te można ocenić pod kątem ich użyteczności, a następnie dokonać ich selekcji w wyniku czego powstaje następne pokolenie (iteracja t+1). Celem wprowadzenia różnorodności osobniki poddawane są działaniu specjalnych operatorów genetycznych, takich jak mutacje (transformacje jednoargumentowe) oraz krzyżowanie (operacje wieloargumentowe). Procedura selekcji i zmian powtarzana jest do momentu, gdy spełniony zostanie warunek stopu (Rys.1) [6].



Rysunek 1. Schemat blokowy prostego algorytmu genetycznego.

W celu prezentacji zasady działania algorytmów genetycznych rozważmy skrajnie prosty przykład polegający na znalezieniu największej pięciocyfrowej liczby dającej się zapisać w kodzie binarnym (będzie to oczywiście 11111 czyli 31 dziesiętne). W początkowym etapie tworzymy populację określonej wielkości składającą się z losowych pięcioelementowych łańcuchów binarnych (niech ich będzie sześć: 01010, 10001, 00111, 10100, 11000, 01111). Następnie oceniamy każde rozwiązanie (w tym przypadku wartość dostosowania eval(x_i) jest równoznaczna wielkość liczby dziesiętnej pomnożonej przez 2 w celu szybszej zbieżności) oraz liczymy całkowite dostosowanie populacji F, będące sumą wszystkich wartości dostosowania poszczególnych osobników. Mając te dwie wartości możemy obliczyć prawdopodobieństwo wyboru p_i każdego osobnika do następnego pokolenia oraz jego dystrybuantę q_i:

$$p_i = \frac{eval(vi)}{F} \tag{2.1}$$

$$q_i = \sum_{j=1}^{i} p_j \tag{2.2}$$

Po tych wstępnych obliczeniach (Tabela 1) możemy przystąpić do selekcji osobników, która opiera się na tzw. metodzie ruletki. Polega ona na n-krotnym wylosowaniu liczby r z zakresu [0,1], gdzie n oznacza wielkość populacji. Za każdym razem wybierany jest jeden osobnik do następnego pokolenia, dla którego zachodzi zależność $q_{i-1} < r \leq q_i$. Załóżmy, że zostały wylosowane liczby: 0,165; 0,795; 0,621; 0,432; 0,033 oraz 0,197. Wykorzystując powyższe

wzory otrzymujemy wyniki zebrane w tabeli 1. Do następnego pokolenia zostaną wybrane chromosomy 2, 5, 5, 4, 1, 2. Jak widać część osobników została wybrana kilkakrotnie, inne zaś zostały pominięte (umarły). Cechą charakterystyczną jest to, że osobniki słabsze stopniowo wymierają, a zastępują je silniejsze. Iteracyjne stosowanie takiej procedury powoduje stopniowy wzrost dostosowania całej populacji (zmiana z 186 na 224 w drugim pokoleniu), pomimo tego, że czasem może dojść do losowego wykluczenia silniejszego osobnika kosztem słabszego (osobnik 1 przetrwał mimo, iż jest słabszy od osobnika 6, który zginął). Postępowanie z powyższym schematem doprowadziłoby do zbiegnięcia się całej populacji do aktualnie najlepszego lub prawie najlepszego osobnika, dlatego następnym etapem jest stosowanie operatorów genetycznych: mutacji i krzyżowania. Pozwoli to zwiększyć różnorodność populacji przeszukując lokalną przestrzeń wokół potencjalnie dobrych rozwiązań.

Pierwszym typem operatorów genetycznych jest mutacja. Klasyczny operator mutacji polega na zmianie losowo określonego bitu na przeciwny ($0 \rightarrow 1$ lub $1 \rightarrow 0$) z wcześniej ustalonym prawdopodobieństwem. Innymi formami mutacji są insercja, delecja, duplikacja i inwersja. Wszystkie z wyjątkiem ostatniej nie dają się zastosować w klasycznym algorytmie genetycznym o stałej długości chromosomów, ponieważ powodują zmianę długości łańcuchów, co czasem, jak zostanie pokazane później, może być niezwykle korzystne. Jeśli chodzi o inwersję, to polega ona na odwróceniu kolejności pewnego fragmentu chromosomu [7].

Drugim typem operatorów jest rekombinacja. Jest to proces naśladujący krzyżowanie (crossing over) osobników rodzicielskich, pospolicie występujące w przyrodzie. Polega on na częściowej wymianie informacji między rodzicami. Powstałe potomstwo dziedziczy cechy od obu rodziców stanowiąc mieszaninę ich cech. Istnieje wiele rodzajów rekombinacji, często ukierunkowanych na rozpatrywany problem, wśród najpopularniejszych należy wymienić krzyżowanie jednopunktowe i dwupunktowe, krzyżowanie arytmetyczne, OX i PMX. Ponadto w procesie crossowania może brać udział więcej niż dwa osobniki. Ogólnie, procedura krzyżowania wygląda w następujący sposób. Najpierw wybieramy osobniki do krzyżowania (z określonym prawdopodobieństwem analogicznie jak w przypadku mutacji). Później je parujemy (jeśli w krzyżowaniu bierze udział dwa osobniki musimy zapewnić parzystą liczbę wylosowanych osobników), a następnie w przypadku krzyżowania jednopunktowego losujemy liczbę z zakresu [1, length], gdzie length oznacza długość chromosomu. Przykładowo wylosowano osobniki 01111 i **10100** z punktem krzyżowania między drugim a trzecim bitem. W wyniku krzyżowania powstaną dwa osobniki potomne o sekwencji: 01**100** i **10**111. Jak widać w procesie tym powstał osobnik posiadający wyższą wartość dostosowania niż którykolwiek z rodziców. Oczywiście wcale tak nie musiało być, równie dobrze mogło dojść do pogorszenia,

istotne jednak jest to, że efekt zarówno krzyżowania jak i mutacji zostanie zweryfikowany w odpowiedni sposób na etapie selekcji.

x_i binarnie	x_i dziesiętnie	eval(x_i)	p_i	q_i
01010	10	20	0,108	0,108
10001	17	34	0,183	0,291
00111	**7**	**14**	0,075	0,366
10100	20	40	0,215	0,581
11000	**24**	**48**	0,258	0,839
01111	15	30	0,161	1,000
		F = 186		

Tabela 1. Przykładowe obliczenia wykonywane przed selekcją (szczegóły i objaśnienia w tekście).

Powyższy przykład algorytmu genetycznego jest bardzo prosty, i nie ma praktycznego zastosowania. Został on przedstawiony jedynie dla wprowadzenia w tematykę. Szczegółowa realizacja algorytmu zależy od rozpatrywanego problemu. To on decyduje, czy użyjemy reprezentacji binarnej (w początkowym okresie stosowania algorytmów genetycznych tylko takiej używano) [5], zmiennoprzecinkowej [3] czy zupełnie innej opartej na specyficznym dla zadania alfabecie [4]. To samo dotyczy liczby i rodzaju zastosowanych operatorów genetycznych [9]. Wszystko to gwarantuje olbrzymią elastyczność algorytmów genetycznych, które stosowano w tak różnych dyscyplinach jak obróbka obrazów [2], tworzenie znaków wodnych [11], projektowanie anten wojskowych [8], skrzydeł samolotu [10], optymalizacja parametrów lasera [12], czy przewidywanie struktury drugorzędowej białek [13]. Właściwie trudniej znaleźć dyscyplinę, w której algorytmy genetyczne nie mogą mieć zastosowania niż na odwrót.

3. BIOLOGICZNE ASPEKTY WYZNACZANIA SKŁADU AMINOKWASOWEGO BIAŁEK

Białka to długie liniowe polimery zbudowane z około 20 różnych cegiełek nazywanych aminokwasami. Każda z nich posiada inną strukturę trójwymiarową, skład atomowy i wynikające stąd odmienne właściwości fizykochemiczne. Kolejność aminokwasów decyduje o strukturze białek, która wynika z oddziaływań zachodzących między poszczególnymi aminokwasami oraz między aminokwasami a środowiskiem zewnętrznym. Z kolei struktura

białek warunkuje ich możliwości. Większość wysiłku naukowców skupia się obecnie na poznaniu struktury trzeciorzędowej, ponieważ daje ona najpełniejszą informację o analizowanym białku. Jednak badania takie są niezwykle skomplikowane i drogie oraz wymagają wysoce specjalistycznego sprzętu. Niekiedy bardzo wstępne analizy są wystarczające i pozwalają wykluczyć pewne możliwości ukierunkowując badacza od samego początku na właściwy tor. Dla przykładu, proste określenie masy białka, czy jego hydrofilowości pozwala na jego separację w mieszaninie innych białek.

3.1. MASA CZĄSTECZKOWA BIAŁEK

Masa cząsteczkowa białek jest sumą mas poszczególnych aminokwasów wchodzących w skład białka powiększoną o masę 1 cząst. wody i wyrażana jest w Daltonach (Da) lub jego wielokrotnościach np. kDa (tysiąc Daltonów). Masy poszczególnych aminokwasów podano w tabeli 2.

Aminokwas	Skrót 3-liter.	Skrót 1-liter.	masa Da	częstość %	Aminokwas	Skrót 3-liter.	Skrót 1-liter.	masa Da	częstość %
Glicyna	Gly	G	57,0519	8,2	Leucyna	Leu	L	113,1594	9,1
Walina	Val	V	99,1326	7,2	Izoleucyna	Ile	I	113,1594	6,2
Alanina	Ala	A	71,0788	7,8	Seryna	Ser	S	87,0782	5,9
Cysteina	Cys	C	103,1388	2,4	Fenyloalanina	Phe	F	147,1766	4,4
Tyrozyna	Tyr	Y	163,1760	3,4	Glutamina	Gln	Q	128,1307	3,4
Histydyna	His	H	137,1411	2,5	Asparagina	Asn	N	114,1038	4,1
Lizyna	Lys	K	128,1741	5,6	Tryptofan	Trp	W	186,2132	1,4
Arginina	Arg	R	156,1875	5,1	Prolina	Pro	P	97,1167	4,3
Metionina	Met	M	131,1926	2,4	Treonina	Thr	T	101,1051	5,5
Kwas glutaminowy	Glu	E	129,1155	5,9	Kwas asparaginowy	Asp	D	115,0886	5,2

Tabela 2. Najczęściej występujące aminokwasy w białkach. Oprócz masy cząsteczkowej [20] podano także częstość występowania aminokwasów w naturze [21].

3.2. ABSORBANCJA

Następnym parametrem opisującym białko jest absorbancja, zwana także gęstością optyczną. Absorbancję wyznacza się doświadczalnie przepuszczając światło monochromatyczne

przez roztwór o określonym stężeniu. W zależności od ilości rozpuszczonej substancji zmienia się ilość pochłoniętego światła. W przypadku białek maksimum absorpcji przypada na fale o długości ok 285 nm i zależy głównie od trzech aminokwasów (tryptofan, cysteina i tyrozyna). Pozostałe aminokwasy praktycznie nie pochłaniają światła, więc nie mają wpływu na wynik końcowy. Parametr opisujący bezwzględną ilość pochłoniętego światła nazywany jest ekstynkcją E i wynosi odpowiednio $E_{Trp} = 5690$, $E_{Tyr} = 1280$, $E_{Cys} = 120$. W związku z tym można obliczyć absorbancję w sposób teoretyczny znając masę białka i ilość tych trzech aminokwasów:

$$E_{białka} = E_{Trp} \cdot ilośćTrp + E_{Tyr} \cdot ilośćTyr + E_{Cys} \cdot ilość\ Cys \tag{3.2.1}$$

$$absorbancja = \frac{E_{białka}}{masa_{białka}} \tag{3.2.2}$$

3.3. PUNKT IZOELEKTRYCZNY

Punkt izoelektryczny (pI) białka opisuje jego ładunek elektrostatyczny, czyli określa czy białko jest kwasowe, zasadowe lub obojętne, i w jakim stopniu. W sposób doświadczalny wyznacza się go umieszczają białko w polu elektromagnetycznym. W zależności od ładunku białko przemieszcza się w kierunku katody lub anody, aż do momentu osiągnięcia równowagi. Punkt izoelektryczny odpowiada więc wartości pH, w którym suma ładunków cząsteczki wynosi zero. W przypadku białek zależy on głównie od siedmiu naładowanych aminokwasów: kwasu glutaminowego (grupa δ-karboksylowa), kwasu asparaginowego (grupa ß- karboksylowa), cysteiny (grupa tiolowa), tyrozyny (grupa fenolowa), histydyny (pierścień imidazolowy), lizyny (grupa ε-amidowa) i argininy (grupa guanidynowa). Ponadto, wpływ mają także grupy NH_2 i COOH znajdujące się na końcach białka. Każda z grup funkcyjnych charakteryzuje się inną wartością stałej dysocjacji pK. Ładunek elektryczny białka odnosi się do pH roztworu (bufor), w którym się ono znajduje w związku z czym ma ono także wpływ na obliczanie pI. Uwzględniając te wszystkie parametry i wykorzystując równanie Hendersona-Hasselbacha możemy wywnioskować następujące wzory pozwalające obliczyć ładunek cząsteczki w określonym pH buforu:

- dla cząsteczek obdarzonych ładunkiem ujemnym:

$$\sum_{i=1}^{n} \frac{-1}{1+10^{pK_n - pH}} \tag{3.3.1}$$

gdzie pK_n oznacza wartość pK analizowanego aminokwasu zawierającego grupę negatywną .

– dla cząsteczek obdarzonych ładunkiem dodatnim:

$$\sum_{i=1}^{n} \frac{1}{1+10^{pH-pK_p}} \tag{3.2.2}$$

gdzie pK_p oznacza wartość pK analizowanego aminokwasu zawierającego grupę pozytywną.

Jak widać z równań jedyną zmienną jest pH buforu. Zmieniając jego wartość możemy ustalić punkt izoelektryczny danego białka. Obliczony punkt izoelektryczny jest jedynie wartością teoretyczną, i w pewnym stopniu będzie odbiegać od wartości ustalonej eksperymentalnie z kilku względów. Po pierwsze samo założenie, że punkt izoelektryczny zależy jedynie od wartości pK wspomnianych wyżej aminokwasów jest raczej uproszczeniem. Nie bierzemy pod uwagę modyfikacji białek, które mogą modyfikować ładunek białka (np. fosforylacja). Problematyczna jest także obecność cysteiny, ponieważ jeśli tworzy ona mostki siarczkowe to takich cystein nie należy brać pod uwagę przy obliczeniach. Pomimo tych wad można teoretycznie wyliczyć punkt izoelektryczny z dokładnością 0,5. Krytycznym momentem wyznaczania punktu izoelektrycznego jest użycie odpowiednich wartości pK. W tym przypadku mamy spory wybór. W tabeli 3 przedstawiono przykładowe wartości według różnych źródeł.

Źródło	**pK_{NH2}**	**pK_{COOH}**	**pK_{Cys}**	**pK_{Asp}**	**pK_{Glu}**	**pK_{His}**	**pK_{Lys}**	**pK_{Arg}**	**pK_{Tyr}**
EMBOSS	8,6	3,6	8,5	3,9	4,1	6,5	10,8	12,5	10,1
DTASelect	8,0	3,1	8,5	4,4	4,4	6,5	10,0	12,0	10,0
Solomon	9,6	2,4	8,3	3,9	4,3	6,0	10,5	12,5	10,1
Sillero	8,2	3,2	9,0	4,0	4,5	6,4	10,4	12,0	10,0
Rodwell	8,0	3,1	8,33	3,68	4,25	6,0	11,5	11,5	10,07
Patrickios	11,2	4,2	-	4,2	4,2	-	11,2	11,2	-
Wikipedia	8,2	3,65	8,18	3,9	4,07	6,04	10,54	12,48	10,46

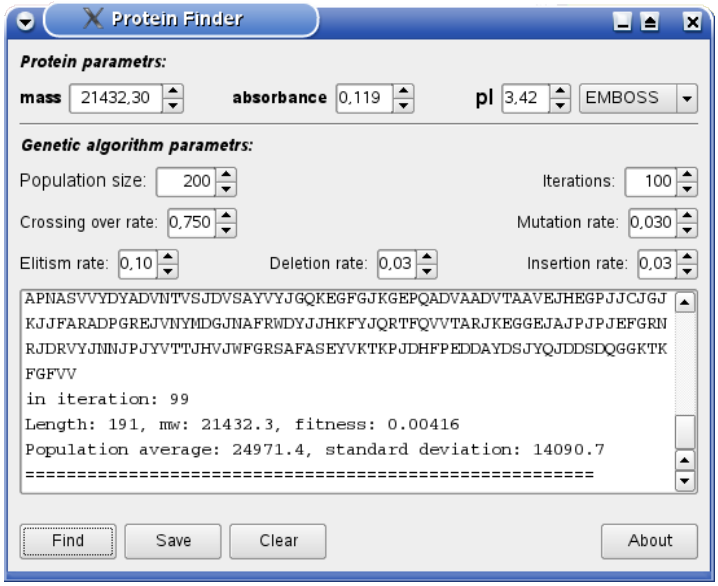
Tabela 3. Stałe dysocjacji aminokwasów.

4. PROGRAM WYZNACZAJĄCY SKŁAD AMINOKWASOWY BIAŁEK

Program ProteinFinder został napisany przez autora niniejszej pracy w języku C++ [14, 15, 22] z wykorzystaniem biblioteki QT4 firmy Trolltech [16]. Dzięki zastosowaniu biblioteki QT kod źródłowy jest w pełni przenośny i w nie zmienionej formie może być kompilowany na dowolnej platformie systemowej na której możliwe jest zainstalowanie biblioteki QT. Biblioteka ta jest dostępna nieodpłatnie dla praktycznie każdego istniejącego systemu operacyjnego włączając w to systemy z rodziny Windows i Unix.

4.1. INTERFEJS UŻYTKOWNIKA

Graficzny interfejs użytkownika (GUI) programu ProteinFinder pokazano na rysunku 2. Interfejs składa się z dwóch sekcji. W pierwszej od góry sekcji użytkownik ma możliwość ustawienia parametrów poszukiwanego białka, takich jak masa, absorbancja i punkt izoelektryczny z wartościami pK według wybranego modelu. W drugiej sekcji znajdują się ustawienia algorytmu genetycznego (wielkość populacji, ilość pokoleń oraz częstość działania poszczególnych operatorów).



Rysunek 2. Interfejs graficzny programu ProteinFinder.

4.2. BUDOWA ALGORYTMU GENETYCZNEGO

Ze względu na specyfikę problemu prosty algorytm genetyczny zaprezentowany rozdziale 2 zmodyfikowano. Szczegółowy opis zmian zostanie przedstawiony w bieżącym rozdziale podczas prezentacji kolejnych etapów algorytmu. Algorytm genetyczny swoją siłę zawdzięcza losowości działania. Przestrzeń potencjalnych rozwiązań przeszukiwana jest w sposób inteligentny pomimo, że zasada działania opiera się na losowym wyborze punktów przestrzeni. Jeśli wybrane punkty wykazują obiecujące właściwości, kontynuowane jest przeszukiwanie najbliższej okolicy. W związku z tym, w czasie wykonywania algorytmu istnieje

ciągła potrzeba dokonywania losowego wyboru kierunku dalszego działania. Z tej przyczyny, jednym z ważniejszych aspektów decydujących o jakości i szybkości algorytmu genetycznego jest sposób generowania liczb pseudolosowych. W przybliżeniu ten etap wypełnia 90% czasu pracy programu, dlatego zrezygnowano z generatora liczb oferowanego w ramach standardowej biblioteki i użyto generatora Mersenne Twister [17], który jest ponad dwukrotnie szybszy i wykazuje lepsze właściwości statystyczne (jak do tej pory przeszedł pomyślnie wszystkie testy) oraz cechuje się olbrzymim okresem powtarzalności (10^{6000}). Ponadto w czasie działania algorytmu niezbędne jest sortowanie generowanych chromosomów na podstawie funkcji oceny. W tym celu skorzystano z algorytmu QuickSort [18]. Szczegółową strukturę zmodyfikowanego algorytmu genetycznego przedstawiono na rysunku 3.

4.2.1. KODOWANIE

Specyfika rozwiązywanego problemu zadecydowała o wykorzystaniu innego kodowania niż binarne. W przypadku poszukiwania składu aminokwasowego możliwe było wykorzystanie bezpośredniej notacji odnoszącej się do kodu aminokwasowego. Osobnik przedstawiony jest w postaci ciągu liter symbolizujących poszczególne aminokwasy, z jednym zastrzeżeniem. Początkowy 20. literowy alfabet uproszczono do 19. liter, ponieważ dwa z aminokwasów (leucyna i izoleucyna) nie są rozróżnialne przez funkcję przystosowania (patrz rozdz. 4.2.3). Wynika to z identycznej masy tych aminokwasów. Oznaczono je literą J. Kolejnym założeniem jest zmienna długość chromosomu ograniczona jedynie w pewnym zakresie przez parametry wejściowe. Ponieważ szukamy składu aminokwasowego (a nie sekwencji) białek kolejność liter nie ma znaczenia.

```
procedure program ProteinFinder
begin
        t ← 0
        zainicjuj początkową populację P(t), populację elitarną ElitArray
        begin
                ustal dopuszczalną długość białka <min_length, max_length>
                wylosuj długość n chromosomu
                wypełnij chromosom aminokwasami
        end
        while (not warunek zakończenia) do
        begin
                t ← t + 1
                krzyżuj P(t)
                begin
                        utwórz listę rodziców
                        if lista nieparzysta
                                popraw listę
                        while (not lista pusta) do
                        begin
                                wybierz dwóch rodziców i skrzyżuj ich
```

```
                end
        end
        dokonaj insercji na P(t)
        dokonaj delecji na P(t)
        mutuj P(t)
        oceń P(t)
        begin
                oblicz mw
                oblicz pI
                oblicz absorbancję
                oblicz fitness
                posortuj P(t) według fitness algorytmem QuickSort
                zapisz ElitArraySize najlepszych osobników do ElitArray
                przeskaluj fitness z min na max
                wylosuj osobniki do nowej populacji P(t +1)
        end
    end
end
```

Rysunek 3. Struktura programu ProteinFinder - szczegółowy opis w tekście.

4.2.2. GENEROWNIE POPULACJI POCZĄTKOWEJ

Wielkość populacji jest stała i ustalona przez użytkownika. Pierwszym etapem algorytmu genetycznego jest stworzenie n-elementowej tablicy (PopArray) obiektów klasy Protein. Ponadto tworzone jest tablica tymczasowa (PopArrayTemp). Obie tablice mają rozmiar równy wielkości populacji. Jak zaznaczono wcześniej, chromosomy reprezentujące pojedyncze rozwiązanie, czyli szukane białko mają zmienną długość. Wynika to z tego, że w początkowym etapie możemy ustalić jedynie pewien zakres długości białka (przestrzeń przeszukiwań). Jedynym parametrem pośrednio wpływającym na długość białka jest masa cząsteczkowa (mw). Przykładowo, jeśli szukamy składu aminokwasowego białka o masie 20 kDa, nie wiemy ile dokładnie wynosi jego długość (w pewnym stopniu zależy to także od punktu izoelektrycznego i absorbancji), ale możemy oszacować, w jakim zakresie mamy szukać. Zakres ten mieści się między masą szukanego białka podzieloną przez odpowiednio najcięższy aminokwas i najlżejszy:

$$\text{max\_length} = \frac{mw - 18{,}01524}{57{,}0519} \tag{4.2.2.1}$$

$$\text{min\_length} = \frac{mw - 18{,}01524}{186{,}2132} \tag{4.2.2.2}$$

W przypadku białka o masie 20 kDa jedynie chromosomy o długości w zakresie [108, 350] liter mogą osiągnąć zamierzoną masę. Oznacza to, że białko zbudowane z 107 aminokwasów nie będzie posiadać zamierzonej masy, nawet jeśli składałoby się ono w całości z najcięższego aminokwasu tj. Tryptofanu. W związku z tym generowanie chromosomów o takiej długości pozbawione jest sensu.

W następnym etapie w sposób losowy wybierana jest długość chromosomu zawierająca się w wcześniej określonym przedziale. Później wywołana jest funkcja addAA() (Dodatek A), która odpowiada za wybór aminokwasów tworzonego chromosomu zgodnie z częstością występowania aminokwasów w naturze (Tabela 1). Pozwoli to zlokalizować wygenerowane rozwiązania w obiecujących okolicach przestrzeni. Przykładowo, nawet jeśli pominąć fakt, że nie występują białka zbudowane jedynie z jednego rodzaju aminokwasów to dodatkowo niektóre aminokwasy występują niezwykle rzadko, i dlatego powstanie białka zbudowanego w większości z tryptofanu jest mało prawdopodobne.

Szukanie składu aminokwasowego białek jest zadaniem prostszym od poszukiwania ich sekwencji, jednak potencjalna przestrzeń poszukiwań i tak jest bardzo duża, co uniemożliwia sprawdzenie wszystkich rozwiązań. Ogólnie przestrzeń ta ma rozmiar:

$$C = \sum_{k=min}^{max} \frac{(n+k-1)!}{k!(n-1)!} \qquad (4.2.2.1)$$

gdzie min i max oznacza minimalną i maksymalną długość poszukiwanego białka, zaś n wynosi 19 (liczba aminokwasów)

Przykładowo dla względnie małego białka o masie 20kDa zakres długości białka wynosi [115, 375] dając ostatecznie ~10^{33} możliwości.

4.2.3. FUNKCJA OCENY

Funkcja oceny (zwana funkcją celu lub przystosowania) jest podstawowym elementem algorytmu genetycznego. Na jej podstawie oceniana jest użyteczność każdego chromosomu w populacji. W przypadku algorytmu ProteinFinder funkcją celu jest odległość w kartezjańskim układzie współrzędnych między punktem reprezentującego poszukiwane białko a punktem odnoszącym się do bieżącego rozwiązania. Ponieważ algorytm bierze pod uwagę trzy parametry to każdy punkt przestrzeni będzie trójwymiarowym wektorem, a odległość między dwoma takimi punktami można obliczyć korzystając ze wzoru:

$$fitness = \sqrt{(mw_{target} - mw)^2 + (pI_{target} - pI)^2 + (absorb_{target} - absorb)^2} \qquad (4.2.3.1)$$

gdzie parametry z oznaczeniem „target” oznaczają wartości poszukiwane (docelowe białko), zaś pozostałe odnoszą się do bieżącego chromosomu

Dodatkowo dwa z parametrów (mw, absorbancja) zostały znormalizowane tak, aby ich wartości były proporcjonalne do wniesionej informacji. W przeciwnym przypadku masa cząsteczkowa białek, liczona w tysiącach Da, zdominowałaby pozostałe parametry, których wartości zmieniają

się w zakresie kilkunastu jednostek (patrz Dodatek A funkcja calculate_fitness()).

Należy podkreślić, że obliczenie masy cząsteczkowej i absorbancji jest zadaniem łatwym, które wymaga jedynie zliczenia poszczególnych aminokwasów (Dodatek A, funkcje calulate_mw() i calculate_absorbance()). Inaczej wygląda sytuacja w przypadku określania punktu izoelektrycznego. Po ustaleniu liczby aminokwasów obdarzonych ładunkiem niezbędne jest określenie punktu pH w którym ładunek elektrostatyczny białka i środowiska równoważą się. W praktyce oznacza to wielokrotne liczenie równania Hendersona-Hasselbacha (Rozdz. 3.3) przy różnych wartościach pH. W najprostszym wariancie procedura obliczania pI polega na stopniowym zwiększaniu parametru pH, aż do momentu, gdy będzie on mniejszy lub równy zeru. W celu skrócenia czasu obliczeń wykorzystano na tym etapie algorytm bisekcji (Dodatek A funkcja calculate_pI()) [24]. Pozwoliło to skrócić czas obliczania punktu izoelektrycznego 10-krotnie.

4.2.4. OPERATOR KRZYŻOWANIA

Działanie operatora krzyżowania (crossing over) polega na wymianie informacji między dwoma chromosomami. Istnieje wiele rodzajów operatorów krzyżowania. Algorytm genetyczny programu PrteinFinder korzysta z krzyżowania jednopunktowego i dwupunktowego (Rys. 4). Procedura krzyżowania polega na ustaleniu listy rodziców (wielkość listy zależy od ustawień użytkownika, i określona jest jako procent całej populacji). Ponieważ w krzyżowaniu biorą udział dwa osobniki lista ta musi być parzysta. Następnie w sposób losowy kojarzeni są rodzice, którzy tworzą potomstwo. Z dwóch osobników rodzicielskich powstają dwa zastępujące je osobniki potomne (Rys. 3). Krzyżowanie przeprowadzane jest w taki sposób, aby osobniki potomne były takiej samej długości jak rodzice. Gwarantuje to stabilność działania algorytmu, który przeszukuje przestrzeń zgodnie z nałożonymi ograniczeniami. Implementacja operatorów krzyżowania przedstawiona jest w dodatku B.

Rysunek 4. Działanie operatora krzyżowania. a) krzyżowanie jednopunktowe; b) krzyżowanie dwupunktowe.

4.2.5. OPERATOR MUTACJI

Operator mutacji polega na zmianie jednego genu chromosomu w inny. Działanie operatora zachodzi z określonym przez użytkownika prawdopodobieństwem. Przykładowo poziom mutacji na poziomie 0,1 oznacza, że statystycznie co dziesiąty aminokwas ulegnie mutacji. Teoretycznie, w momencie w którym dana pozycja zostanie przeznaczona do mutacji dochodzi do losowej wymiany na jeden z 18 pozostałych aminokwasów. Ze względu na specyfikę zadania można było zmodyfikować działanie mutacji. Otóż wybór aminokwasów nie jest w pełni losowy, i zależy on od prawdopodobieństw mutacji jednych aminokwasów w drugie. Wartości te zebrano w macierz BLOSUM62 [23] w wyniku porównania sekwencji białkowych o co najmniej 62% poziomie identyczności. Zastosowanie macierzy powinno oddawać dodatkowe właściwości białek, których nie uwzględniają parametry funkcji celu. Wartości z macierzy BLOSUM odbiegają od oczekiwanych, ponieważ istnieją pewne tendencje w procesie mutowania aminokwasów. Ogólnie, aminokwasy o podobnych właściwościach fizykochemicznych będą bardziej tolerowane, ponieważ takie zmiany nie zaburzają w znaczący sposób struktury białka. Przykładowo, mutacja zasadowej lizyny w aminokwas kwaśny jest dwukrotnie mniej prawdopodobna niż zamiana w inny zasadowy aminokwas, np. argininę. Podobnie mutacja W ↔ G jest rzadko spotykana, co tym razem wynika z różnicy w wielkości aminokwasów. Implementacja operatora mutacji i wartości prawdopodobieństwa mutacji opracowane na podstawie macierzy BLOSUM62 zamieszczono w dodatku C.

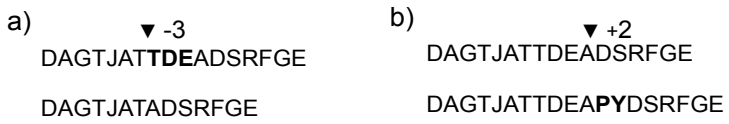
4.2.6. OPERATORY INSERCJI I DELECJI

Operatory insercji i delecji mają przeciwstawne działanie. Insercja polega na dodaniu określonej liczby aminokwasów, zaś delecja na ich odjęciu (Rys. 5). Głównym założeniem tych operatorów jest umożliwienie zmiany długości chromosomów. Inne operatory takiej właściwości nie mają. Zastosowanie zmiennej długości chromosomów wynika z potrzeby przeszukiwania całej przestrzeni rozwiązań, a nie jedynie jej fragmentów. Posłużmy się tu pewnym prostym przykładem. Załóżmy, że nasza populacja ma wielkość 100 osobników. Szukamy białka o masie 20 kDa. Jak wynika z wcześniejszych obliczeń szerokość zakresu długości chromosomów wynosi 260. Widać więc, że nawet jeśli założymy, że długości wylosowanych 100 osobników nie powtarzają się (jest to raczej mało prawdopodobne ze względu na losowość procesu) to i tak nie wyczerpią one wszystkich możliwości. Jeśli teraz okaże się, że rozwiązaniem jest białko o długości innej niż te spośród wylosowanych, to algorytm nigdy nie będzie w stanie „wskoczyć” na właściwą ścieżkę, bez względu na ilość iteracji i działanie operatorów krzyżowania i mutacji.

Po względnym ustabilizowaniu się algorytmu dopiero działanie insercji lub delecji może zmienić długość chromosomów, być może we właściwym kierunku.

Ze stosowaniem operatorów insercji i delecji związany jest problem wyjścia poza granice przestrzeni rozwiązań. Niekontrolowane stosowanie tych operatorów może doprowadzić do powstania osobników znajdujących się poza ograniczeniami, które nie są w stanie wyprodukować szukanego rozwiązania, a więc marnotrawią czas programu. Aby tego uniknąć po zakończeniu działania operatorów sprawdzany jest ich rezultat, i jeśli narusza on narzucone bariery jest on usuwany.

Innym problemem pośrednio związanym z ograniczeniami jest długość insercji/delecji. Ogólnie istnieje zasada, że im większa zmiana tym mniej prawdopodobna. Wynika to z tego, że duże zmiany mają małą szanse na wprowadzenie znaczących poprawek. Najczęściej bardziej korzystna jest mała zmiana, która jedynie nieznacznie zmodyfikuje wartość przystosowania. Jest to szczególnie istotne w końcowym etapie działania algorytmu, kiedy populacja zbiega się do określonego rozwiązania. W przypadku algorytmu ProteinFinder zdecydowano się na prostą aproksymację wyżej wspomnianej zasady. Polega ona na tym, że połowa zmian powoduje dodanie lub odjęcie jednego aminokwasu, jedna czwarta dwóch, jedna ósma trzech i tak dalej. Ponadto mała część delecji i insercji (1/32) może przebiegać z dowolną długością ograniczoną jedynie przez wymiar przestrzeni rozwiązań. Funkcje odpowiedzialne za opisane tu operatory umieszczono w dodatku D.



Rysunek 5. Działanie operatorów delecji (a) i insercji (b).

4.2.7. SELEKCJA

Decydującym etapem działania algorytmu genetycznego jest selekcja wygenerowanych wcześniej rozwiązań na podstawie przystosowania osobników. O ile wszelkie wcześniejsze działania, takie jak generowanie populacji i działanie na niej różnymi operatorami jest niezbędne dla prawidłowego funkcjonowania algorytmu genetycznego, to dopiero odpowiednia selekcja czyni ten rodzaj algorytmów różnym i przewyższającym zwykłe przeszukiwanie stochastyczne. Główna zasada selekcji polega na przetrwaniu osobników lepiej przystosowanych do warunków środowiska w stosunku do gorszych. W taki właśnie sposób ewoluują organizmy żywe, i to też

właśnie jest siłą sprawczą działania algorytmów genetycznych.

Selekcja odbywa się na zasadzie ruletki. Pierwszym krokiem przygotowującym do selekcji jest obliczenie funkcji przystosowania każdego z osobników. Następnie sumujemy otrzymane wartości otrzymując całkowite przystosowanie populacji. W tym momencie istnieje potrzeba przeskalowania funkcji oceny, ponieważ algorytm genetyczny w wydajny sposób pracuje na funkcji maksymalizującej, zaś w chwili obecnej zadanie polega na minimalizacji odległości między szukanym osobnikiem a bieżącym rozwiązaniem. Jest to wykonane w prosty sposób przez znalezienie maksymalnej wartości funkcji przystosowania w danym pokoleniu i następnie odjęcie od niej wartości analizowanego osobnika (Dodatek A, funkcja calculate_scaled_fitness()). Teraz możemy obliczyć prawdopodobieństwa wylosowania osobników do następnego pokolenia i ich dystrybuanty. Po tych wstępnych obliczeniach możemy zakręcić ruletką tyle razy ile wynosi wielkość populacji. Do następnego pokolenia wybrane zostają lepiej przystosowane osobniki w kilku kopiach kosztem słabszych, które zanikają. Następnie na częściowo jednorodną nowo utworzoną populację działają operatory. W ten sposób cykl się zamyka. Aby uniknąć losowej utraty najlepszych osobników zastosowano elitarny model selekcji, który polega na wyborze określonej liczby najsilniejszych osobników i zapisaniu ich do oddzielnej tablicy (ElitArray). W praktyce oznacza to, że ruletką kręcimy PopSize – ElitArray razy, a do następnego pokolenia niezależnie od selekcji trafia ElitArraySize najlepszych osobników. Dzięki temu wynik końcowy będzie przedstawiał najlepsze znalezione rozwiązania w czasie całego procesu w odróżnieniu do najlepszej wartości w końcowej populacji.

Warunkiem zatrzymania algorytmu jest znalezienie osobnika o zerowej wartości przystosowania lub powtórzenie głównej pętli programu przez określoną liczbę iteracji (pokoleń).

5. WYNIKI EKSPERYMENTÓW NUMERYCZNYCH

W tym rozdziale przedstawione zostały wyniki uzyskane przy wykorzystaniu sekwencji testowych (dodatek E) za pomocą programu ProteinFinder. Program uruchomiono na komputerze z procesorem Intel Celeron 2,6 GHz z 512 MB pamięci operacyjnej. W zależności od długości białka, czyli od podanej masy, algorytm wykazywał zmienną szybkość. W oczywisty sposób dla białek cięższych (100 kDa i więcej) wyraźnie widoczne było spowolnienie działania algorytmu. Dla mniejszych białek czas obliczeń był mniej więcej stabilny i wynosił

około 50 osobników na sekundę. Zależność ta wynika z tego, że w przypadku większych białek o czasie wykonania głównej pętli programu przestają decydować obliczenia związane z wyliczaniem parametrów funkcji celu i selekcją, a górę biorą procesy związane z działaniem operatorów, zwłaszcza mutacji. Sprawdzenie osobników o długości setek aminokwasów i ewentualna mutacja znacznie zwiększają złożoność obliczeniową algorytmu.

Przeprowadzono cały szereg testów na różnych białkach przy kilku odmiennych ustawieniach algorytmu genetycznego (tabela 4 i dodatek F). Zazwyczaj każdy z testów powtórzono kilkakrotnie. Wielkość populacji i liczba pokoleń zostały dobrane eksperymentalnie tak, aby otrzymać jak najlepsze rezultaty. Należy podkreślić, że wyraźnie widoczna jest tu pewna tendencja, która polega na tym, że na osiągnięte rezultaty pozytywnie wpływa zwiększenie liczebności populacji. Stosowanie małej populacji przy olbrzymiej liczbie pokoleń (patrz [6]) wydaje się nieuzasadnione. Otrzymane wyniki wskazują, że dla stacjonarnych problemów, w których zadanie nie zmienia się w czasie (tak jak w tym przypadku) algorytm bardzo szybko lokalizuje obiecujący punkt w przestrzeni i skupia się na przeszukiwaniu jego najbliższego otoczenia. W związku z tym lepsze wydaje się zastosowanie większej populacji przez co algorytm będzie mógł działać na większym obszarze odnajdując kilka dobrych punktów, których lokalne otoczenie zostanie lepiej przeszukane (przykładowo testy na białku flagellin, pomimo podobnych wyników wartości parametrów funkcji celu wykazują się mniejszym błędem w ocenie długości białka w przypadku zastosowania większej populacji). Z drugiej strony zwiększanie liczebności populacji nie może się odbywać się w nieskończoność, ponieważ zbytnio zwolni to wykonanie programu. Empirycznie ustalono, że liczebność populacji wynosząca kilkaset osobników (w przypadku długich białek kilku tysięcy) jest wystarczająca dla wydajnego działania algorytmu .

Ciekawe jest również, że wykorzystanie macierzy BLOSUM wnosi jedynie nieznaczną poprawę efektywności algorytmu (białko flagellin). Podobnie jest w przypadku zastosowania różnych operatorów krzyżowania. Doświadczalnie wykazano, że krzyżowanie dwupunktowe produkuje odrobinę lepsze rozwiązania, jednak różnica nie jest duża (białko ADM)

	właściwości białka					**ustawienia algorytmu genetycznego**						
nazwa białka	mw	pI	abs	długość	fitness	wielkość populacja	liczba pokoleń	częstość delecji	częstość insercji	częstość mutacji	rodzaj krzyżowania	BLOSUM
Flagellina	21432,33	3,42100	0,11900	201	0							
(4)	21432,25	3,42145	0,11945	200,5	0,00279252	2000	200	0,05	0,05	0,045	1	-
(1)	21433,40	3,42139	0,12131	203	0,01162350	200	2000	0,05	0,05	0,045	1	-
(1)	21432,60	3,42139	0,11944	195	0,00262229	200	2000	0,05	0,05	0,045	1	+
(4)	21432,65	3,42139	0,11944	198,5	0,00262253	2000	200	0,05	0,05	0,045	1	+
rbsA	96152,83	11,6640	0,23700	859	0							
(1)	93905,60	11,6489	0,23872	868	0,00299877	1000	100	0,05	0,05	0,045	1	-

	właściwości białka					ustawienia algorytmu genetycznego						
DMP	93902,90	3,45900	0,19450	934	0							
(3)	93904,30	3,45947	0,19481	872	0,00245350	2000	400	0,07	0,07	0,01	1	+
p104	101921	7,5770	0,79410	893	0							
(1)	101927	7,5766	0,79440	933	0,00162498	300	2000	0,03	0,03	0,01	1	+
(1)	101917	7,5766	0,79389	932	0,00119632	2000	300	0,03	0,03	0,01	1	+
(1)	101916	7,5766	0,79389	927	0,00119598	3000	200	0,03	0,03	0,01	1	+
(1)	101930	7,5766	0,79456	923	0,00252391	6000	100	0,03	0,03	0,01	1	+
Crambina	4736,46	6,0240	0,69250	46	0							
(4)	4736,47	6,0271	0,69250	42,75	0,0032147	1000	500	0,07	0,07	0,025	1	+
ADM	2445,86	11,6490	4,6528	20	0							
(1)	2445,84	11,6489	4,6528	23	0,00009903	500	500	0,03	0,03	0,01	1	+
(5)	2445,84	11,6489	4,6528	20,8	0,00018012	1000	200	0,05	0,05	0,05	1	+
(5)	2445,85	11,6489	4,6528	20,6	0,00012712	1000	200	0,03	0,03	0,03	2	+

Tabela 4. Sumaryczne wyniki działania programy ProteinFinder na wybranych białkach testowych. We wszystkich eksperymentach użyto częstości krzyżowania równej 0,75. W nawiasach poniżej nazwy białka podano liczbę analiz, a wartości pokazane są ich średnią.

Jak wynika z danych, w każdym z eksperymentów algorytm znalazł wysokiej jakości rozwiązanie różniące się od poszukiwanego jedynie w bardzo małym stopniu. Różnica ta w dużym stopniu wynika z błędów zaokrągleń (użytkownik podaje parametry białka z 2-3 przecinkową precyzją). Jednak należy zaznaczyć, że ze względu na to, że opieramy się na wartościach teoretycznych, które mogą się różnić od faktycznie zaobserwowanych o kilka procent, wszystkie znalezione rozwiązania są wystarczająco dobre.

Tak kształtują się wartości przystosowania znalezionych białek, ale czy mają one identyczny skład aminokwasowy jak poszukiwane białka? Niestety, w tym przypadku odpowiedź nie jest pozytywna, wystarczy spojrzeć na długości znalezionych białek. Prawie zawsze są one inne od poszukiwanych. Jedynie w przypadku ostatniego testu na białku ADM algorytm znalazł białka o odpowiedniej długości (na sukces ten niewątpliwie wpłynęła mała długość oraz wykorzystanie macierzy BLOSUM i krzyżowania dwupunktowego). Mimo tego, po bliższym sprawdzeniu, okazało się, że skład aminokwasowy otrzymanych peptydów był różny od ADM. Dopiero skrócenie poszukiwanych peptydów poniżej 10 aminokwasów pozwoliło na znalezienie prawidłowych rozwiązań. Wynik taki, to z jednej strony mało, z drugiej zaś dużo. Długość kilkunastu aminokwasów jest bardzo mała w porównaniu do analizowanych obiektów (rzędu setek aminokwasów), jednak wyniki algorytmu ProteinFinder są porównywalne do otrzymanych innymi metodami. Przykładowo w czasie spektrometrii mas białko jest rozkładane na peptydy o podobnej długości, i dopiero one są analizowane.

Należy podkreślić, że algorytm genetyczny znajduje wysokiej jakości rozwiązania zgodnie z parametrami funkcji celu (masa, pI i absorbancja). To, że nie są one identyczne z

szukanymi, nie wynika ze słabości stosowanej metody, lecz z małej ilości informacji ukrytej w zastosowanych parametrach. W przypadku długich białek istnieje duża ilość rozwiązań reprezentowanych za pomocą punktów przestrzeni położonych na wymaganym poziomie dokładności (dla zadania minimalizacji będą to doliny na trójwymiarowym krajobrazie przestrzeni). Algorytm genetyczny znajduje jeden z nich, a przy ich dużej liczbie istnieje małe prawdopodobieństwo, że będzie to akurat ten, o który chodziło. Niewątpliwie wykorzystanie większej liczby parametrów usprawniłoby działanie algorytmu poprzez obniżenie poziomu wrażliwości algorytmu i wyeliminowanie znacznej części fałszywych rozwiązań znajdujących się pierwotnie poniżej progu. Jednak trudno takie parametry znaleźć. Ustalenie innych właściwości białka jest już zadaniem skomplikowanym i kosztownym, przez co zastosowanie metody wykorzystującej algorytm genetyczny przestaje być lepsze od innych stosowanych obecnie technik.

PODSUMOWANIE

W pracy opisano zagadnienie wyszukiwania składu aminokwasowego w oparciu o właściwości fizyczne białek, które łatwo zmierzyć. Są nimi: masa, punkt izoelektryczny i absorbancja. Ostatecznie zadanie sprowadzono do wyznaczania ciągów znaków w 19. literowym alfabecie. Ciągi te reprezentują skład aminokwasowy białek. Procedura poszukiwania rozwiązania odbywa się przy wykorzystaniu odpowiednio zmodyfikowanego algorytmu genetycznego. Dla celów eksperymentalnych opracowano i zaimplementowano algorytm genetyczny w języku C++. Następnie przetestowano działanie programu ProteinFinder. W końcowym etapie dokonano analizy uzyskanych wyników.

Podsumowując należy stwierdzić, że algorytmy genetyczne są wydajnym sposobem postępowania w przypadku zadań, z którymi normalnie nie mogą sobie poradzić inne metody. Uzyskane wyniki świadczą, że algorytm genetyczny jest metodą konkurencyjną w stosunku do innych stosowanych metod wyznaczania składu aminokwasowego cechując się dużą szybkością działania i dobrą jakością uzyskanych wyników.

DODATEK A

Dodatek ten zawiera definicje ważniejszych funkcji wchodzących w skład klasy Protein znajdujące się w pliku Protein.cpp. Napisane są one w języku C++ z wykorzystaniem funkcji pochodzących z biblioteki QT4 (funkcje te pogrubiono). Opis funkcji zamieszczono w komentarzach.

```
//=============================================================================
void Protein::calculate_mw()                    //funkcja ta oblicza masę cząsteczkową oraz dodatkowo ekstynkcję
{
length = protein.size();

 //aminokwasy obdarzone ładunkiem
AspNumber = protein.count ("D");
GluNumber = protein.count ("E");
CysNumber = protein.count ("C");                //aminokwas wykorzystywany w obliczaniu absorbancji
TyrNumber = protein.count ("Y");                //aminokwas wykorzystywany w obliczaniu absorbancji
HisNumber = protein.count ("H");
LysNumber = protein.count ("K");
ArgNumber = protein.count ("R");

 //pozostałe aminokwasy
MetNumber = protein.count ("M");
PheNumber = protein.count ("F");
ValNumber = protein.count ("V");
AlaNumber = protein.count ("A");
GlyNumber = protein.count ("G");
GlnNumber = protein.count ("Q");
AsnNumber = protein.count ("N");
XleNumber = protein.count ("J");
TrpNumber = protein.count ("W");                //aminokwas wykorzystywany w obliczaniu absorbancji
SerNumber = protein.count ("S");
ThrNumber = protein.count ("T");
ProNumber = protein.count ("P");

//masa cząsteczkowa białka
mw = massH2O + massXle * XleNumber + massGly * GlyNumber + massAla * AlaNumber + massVal *
ValNumber + + massGlu * GluNumber + massSer * SerNumber + massLys * LysNumber + massThr * ThrNumber
+ massAsp * AspNumber + massArg * ArgNumber + massPhe * PheNumber + massPro * ProNumber + massAsn *
AsnNumber + massGln * GlnNumber + massTyr * TyrNumber + massHis * HisNumber + massCys * CysNumber
+ massMet * MetNumber  + massTrp * TrpNumber;

//ekstynkcja
proteinExtinction = TyrExtinction * TyrNumber + CysExtinction * CysNumber + TrpExtinction * TrpNumber;
}
//=============================================================================
void Protein::calculate_absorbance()            //funkcja obliczająca absorbancję
{
absorbance =  proteinExtinction/mw;
}
//=============================================================================
void Protein::calculate_pI()                    //funkcja obliczająca punkt izoelektryczny
{
double pKCterm, pKAsp, pKGlu, pKCys, pKTyr, pKHis, pKNterm, pKLys, pKArg;
```

```
//przypisanie zmiennym pK odpowiednich wartości zgodnie z przyjętym modelem.
			if (pKval == "EMBOSS")
				{#include "emboss.h"}
			else if (pKval == "DTASelect")
				{#include "dtaselect.h"}
			else if (pKval == "Solomon")
				{#include "solomon.h"}
			else if (pKval == "Sillero")
				{#include "sillero.h"}
			else if (pKval == "Rodwell")
				{#include "rodwell.h"}
			else if (pKval == "Patrickios")
				{#include "patrickios.h"}
			else if (pKval == "Wikipedia")
				{#include "wikipedia.h"}

   double NQ = 0.0;		//całkowity ładunek przy danym pH
   double QN1=0;		//ładunek grupy COOH
   double QN2=0;		//ładunek D
   double QN3=0;		//ładunek E
   double QN4=0;		//ładunek C
   double QN5=0;		//ładunek Y
   double QP1=0;		//ładunek H
   double QP2=0;		//ładunek NH2
   double QP3=0;		//ładunek K
   double QP4=0;		//ładunek R

   double pH = 6.5;			//początkowa wartość pI = 6.5 –  teoretycznie powinna wynosić 7
					//jednak białka średnio mają pI = 6.5, więc w ten sposób zwiększamy
   double pHprev = 0.0;		//szybkość zbieżności algorytmu do rozwiązania
   double pHnext = 14.0;		//możliwy zakres wartości pH [0, 14]

   double E = 0.01;		//epsilon –  dokładność algorytmu pI = pH ± E
   double temp = 0.0;

for(;;)
 {
   QN1=-1/(1+pow(10,(pKCterm-pH)));
   QN2=-AspNumber/(1+pow(10,(pKAsp-pH)));
   QN3=-GluNumber/(1+pow(10,(pKGlu-pH)));
   QN4=-CysNumber/(1+pow(10,(pKCys-pH)));
   QN5=-TyrNumber/(1+pow(10,(pKTyr-pH)));
   QP1=HisNumber/(1+pow(10,(pH-pKHis)));
   QP2=1/(1+pow(10,(pH-pKNterm)));
   QP3=LysNumber/(1+pow(10,(pH-pKLys)));
   QP4=ArgNumber/(1+pow(10,(pH-pKArg)));

   NQ=QN1+QN2+QN3+QN4+QN5+QP1+QP2+QP3+QP4;

//%%%%%%%%%%%%%%%%%%%%		BISEKCJA	%%%%%%%%%%%%%%%%%%%%%%
   if(NQ<0)		//jeśli bieżąca wartość pH jest mniejsza od 0 to nowa wartość pH musi być od niej mniejsza
   {
      temp = pH;
      pH = pH-((pH-pHprev)/2);
      pHnext = temp;
      //cout<<"pH: "<<pH<<", \tpHnext: "<<pHnext<<endl;
   }
   else		//w przeciwnym wypadku należy podnieść wartość pH
   {
```

```
        temp = pH;
        pH = pH + ((pHnext-pH)/2);
        pHprev = temp;
        //cout<<"pH: "<<pH<<",\tpHprev: "<<pHprev<<endl;
    }
 //warunek wyjścia z pętli for, znaleziono pI z zamierzoną dokładnością E
    if ((pH-pHprev<E)&&(pHnext-pH<E))
    {
            pI = pH;
            break;
    }
  }
}
//=============================================================================================
void Protein::calculate_fitness()               //funkcja obliczająca przystosowanie chromosomów
{
mw_scaling_factor = t_mw/10;                    //współczynnik skalujący mw
double mw_vector_length = (t_mw – mw)/mw_scaling_factor;
//dzięki temu parmametr mw osiągnie maksymalnie ~23 (przy obliczaniu mw brane jest pod uwagę 19 aminokawasów), w przeciwnym przypadku masa zdominowałaby absorbancję i punkt izoelektryczny

double pI_vector_length = t_pI - pI;
double temp = t_absorbance - absorbance;
double absorbance_vector_length = temp + temp + temp + temp + temp; //współczynnik skalujący absorbancji

//obliczamu długość między 3D wektorami o współrzędnych mw, pI i absorbancji
fitness = sqrt( (mw_vector_length * mw_vector_length) + (pI_vector_length * pI_vector_length) + (absorbance_vector_length * absorbance_vector_length) );
}
//=============================================================================================
void Protein::addAA (int number, int position)      //dodawanie number-razy losowych aminokwasów w pozycji
{                                                   //position zgodnie z częstością ich występowania w naturze
        MTRand mtrand1;
        double temp;
for (int i = 0; i < number; i++)
        {
        temp = mtrand1.rand();
        if (temp < 0.153149)
                protein.insert(position, "J");      //dodajemy do białka J w pozycji o indeksie position
        else if (temp < 0.236503)           protein.insert(position, "G");
        else if (temp < 0.314231)           protein.insert(position, "A");
        else if (temp < 0.385904)           protein.insert(position, "V");
        else if (temp < 0.444792)           protein.insert(position, "E");
        else if (temp < 0.503393)           protein.insert(position, "S");
        else if (temp < 0.558898)           protein.insert(position, "K");
        else if (temp < 0.613551)           protein.insert(position, "T");
        else if (temp < 0.665925)           protein.insert(position, "D");
        else if (temp < 0.717346)           protein.insert(position, "R");
        else if (temp < 0.761098)           protein.insert(position, "F");
        else if (temp < 0.803939)           protein.insert(position, "P");
        else if (temp < 0.844997)           protein.insert(position, "N");
        else if (temp < 0.879187)           protein.insert(position, "Q");
        else if (temp < 0.913251)           protein.insert(position, "Y");
        else if (temp < 0.937920)           protein.insert(position, "H");
        else if (temp < 0.962225)           protein.insert(position, "C");
        else if (temp < 0.986223)           protein.insert(position, "M");
        else                        {protein.insert(position, "W");}
        }
}
//=============================================================================================
```

DODATEK B

Niniejszy dodatek zawiera implementację operatora krzyżowania oraz przedstawia sposób jego działania. Oznaczenia jak w dodatku A.

```
//=============================================================================
void Protein::crossover(Protein &parent)                //krzyżowanie jednopunktowe
{
QString protein2 = parent.protein;
int length2 = parent.length;

MTRand randomFrom;
int smaller = length;                                   //sprawdza który z rodziców ma mniejszą długość
if (length > length2)
        smaller = length2;

int cross_point = randomFrom.randInt(smaller-1);  //losowo wybieramy punkt krzyżowania w zakresie [0, smaller]

QString protein1left = protein.left(cross_point);      //dzielimy osobniki w punkcie cross_point
QString protein2left = protein2.left(cross_point);
QString protein1right = protein.mid(cross_point);
QString protein2right = protein2.mid(cross_point);

        protein = protein1left + protein2right;         //wymieniamy części chromosomów między sobą
        protein2 = protein2left + protein1right;
        parent.protein = protein2;
        parent.length = protein2.size();
        length = protein.size();
}
//=============================================================================
void Protein::crossover2(Protein &parent)               //krzyżowanie dwupunktowe
{
QString protein2 = parent.protein;
int length2 = parent.length;

MTRand random;
int smaller = length;
if (length > length2)
        smaller = length2;

int cross_point1 = random.randInt(smaller - 1);                 //pierwszy punkt
int cross_point2 = random.randInt(smaller - 1);                 //drugi punkt

int smaller_crosspoint = cross_point1;
int bigger_crosspoint = cross_point2;
if (cross_point1 > cross_point2)
{
smaller_crosspoint = cross_point2;
bigger_crosspoint = cross_point1;
}

//dzielimy chromosom zgodnie z dwoma punktami krzyżowania
QString a = protein.mid(smaller_crosspoint, bigger_crosspoint - smaller_crosspoint+1);
QString b = protein2.mid(smaller_crosspoint, bigger_crosspoint - smaller_crosspoint+1);

protein.replace(smaller_crosspoint, bigger_crosspoint - smaller_crosspoint+1, b);
protein2.replace(smaller_crosspoint, bigger_crosspoint - smaller_crosspoint+1, a);
```

```
        parent.protein = protein2;
        parent.length = protein2.size();
        length = protein.size();
}
//=============================================================================

//Działanie operatora krzyżowania
//Tworzymy listę rodziców ParentsList o wielkości zgodnej z współczynnikiem krzyżowania CrossRate

for (int i = 0; i < PopSize; i++)               //wybieramy osobniki z bieżącej populacji do ParentList
        {
                MTRand random;
                temp = random();
                if (temp < CrossRate)
                        ParentsList.append(i);
        }

//Sprawdzamy czy lista jest parzysta (w razie potrzeby korygujemy)

if (ParentsList.size() % 2 == 1)
        {
                MTRand random;
                int wybor = random.randInt(1);
                if (wybor == 0)
                        ParentsList.removeLast();
                else { ParentsList.append(random.randInt(PopSize-1)); }

        }
//krzyżujemy osobniki, aż do ich wyczerpania
for(int i=0; ParentsList.isEmpty()==false ; i++)
        {
                MTRand random;
                momIndex = random.randInt(ParentsList.size()-1);    //losujemy indeks pierwszego z rodziców
                mom = ParentsList.takeAt(momIndex);       //pobranie osobnika z listy zgodnie z indeksem
                dadIndex = random.randInt(ParentsList.size()-1);      //losujemy indeks drugiego z rodziców
                dad = ParentsList.takeAt(dadIndex);           //pobranie osobnika z listy zgodnie z indeksem
                PopArray[mom].crossover2(PopArray[dad]);    //wywołanie funkcji krzyżowania dwupunktowego
        }
//=============================================================================
```

DODATEK C

Niniejszy dodatek zawiera implementację operatora mutacji. Oznaczenia jak w dodatku A. Ponadto podano wartości prawdopodobieństwa mutacji pomiędzy poszczególnymi aminokwasami zgodnie z macierzą BLOSUM62.

```
//=============================================================================
void Protein::mutate(double mutationRate)
{
MTRand random;
double temp;
for (int i = 0; i < protein.size(); i++)
        {
                temp = random.rand();
                if ( mutationRate > temp)              //sprawdza czy aminokwas ulegnie mutacji
                {
                        QChar aa = protein.at(i);
                        aa = mutation(aa);
                        protein.replace( i, 1, aa);
                }
        }
}
//=============================================================================
QChar mutation (QChar aa)
{
MTRand rands;
double mut = rands();                  //generuje pseudolosową liczbę w zakresie <0, 1>

if (aa == QChar('G'))
        {       if (mut < 0.5101215) aa = Qchar('G');          // zmiana G → G (największe prawdopodobieństwo)
                else if (mut < 0.5883941)  aa = Qchar('A');   // zmiana G → A
                else if (mut < 0.6396761)  aa = Qchar('S');   // zmiana  G → S
                else if (mut < 0.6869096)  aa = Qchar('J');   // itd.
                else if (mut < 0.7260459)  aa = Qchar('N');
                else if (mut < 0.7597841)  aa = Qchar('K');
                else if (mut < 0.7935223)  aa = Qchar('D');
                else if (mut < 0.8232119)  aa = Qchar('T');
                else if (mut < 0.8488529)  aa = Qchar('E');
                else if (mut < 0.8731444)  aa = Qchar('V');
                else if (mut < 0.8960864)  aa = Qchar('R');
                else if (mut < 0.9149798)  aa = Qchar('Q');
                else if (mut < 0.9338731)  aa = Qchar('P');
                else if (mut < 0.9500675)  aa = Qchar('F');
                else if (mut < 0.9635628)  aa = Qchar('H');
                else if (mut < 0.9743590)  aa = Qchar('C');
                else if (mut < 0.9851552)  aa = Qchar('Y');
                else if (mut < 0.9946019)  aa = Qchar('M');
                else if (mut < 1.0000000)  aa = Qchar('W');
        }
//analogicznie sprawdzany jest każdy aminokwas (skrócone ze względu na objętość)
//wartości prawdopodobieństwa przejścia jednego aminokwasu w inny zapisano w tabeli poniżej
return aa;
}
//=============================================================================
```

	A	C	D	E	F	G	H	J	K	M
A	0,29014845	0,0650407	0,0410448	0,0552486	0,0338266	0,0782726	0,0419847	0,0455909	0,0569948	0,0522088
C	0,02159244	0,4837398	0,0074627	0,0073665	0,0105708	0,0107962	0,0076336	0,0161968	0,0086356	0,0160643
D	0,02968961	0,0162602	0,3973881	0,0902394	0,0169133	0,0337382	0,0381679	0,0161968	0,0414508	0,0200803
E	0,04048583	0,0162602	0,0914179	0,2965009	0,0190275	0,0256410	0,0534351	0,0191962	0,0708117	0,0281124
F	0,02159244	0,0203252	0,0149254	0,0165746	0,3868922	0,0161943	0,0305344	0,0503899	0,0155440	0,0481928
G	0,07827260	0,0325203	0,0466418	0,0349908	0,0253700	0,5101215	0,0381679	0,0209958	0,0431779	0,0281124
H	0,01484480	0,0081301	0,0186567	0,0257827	0,0169133	0,0134953	0,3549618	0,0095981	0,0207254	0,0160643
J	0,10256410	0,1097561	0,0503731	0,0589319	0,1775899	0,0472335	0,0610687	0,4697061	0,0708117	0,0361446
K	0,04453441	0,0203252	0,0447761	0,0755064	0,0190275	0,0337382	0,0458015	0,0245951	0,2780656	0,2971888
M	0,01754386	0,0162602	0,0093284	0,0128913	0,0253700	0,0094467	0,0152672	0,0443911	0,0155440	0,1606426
N	0,02564103	0,0162602	0,0690299	0,0405157	0,0169133	0,0391363	0,0534351	0,0143971	0,0414508	0,0200803
P	0,02968961	0,0162602	0,0223881	0,0257827	0,0105708	0,0188934	0,0190840	0,0143971	0,0276339	0,0160643
Q	0,02564103	0,0121951	0,0298507	0,0644567	0,0105708	0,0188934	0,0381679	0,0149970	0,0535406	0,0281124
R	0,03103914	0,0162602	0,0298507	0,0497238	0,0190275	0,0229420	0,0458015	0,0215957	0,1070812	0,0321285
S	0,08502024	0,0406504	0,0522388	0,0552486	0,0253700	0,0512821	0,0419847	0,0245951	0,0535406	0,0361446
T	0,04993252	0,0365854	0,0354478	0,0368324	0,0253700	0,0296896	0,0267176	0,0359928	0,0397237	0,0401606
V	0,06882591	0,0569106	0,0242537	0,0313076	0,0549683	0,0242915	0,0229008	0,1289742	0,0328152	0,0923695
W	0,00539811	0,0040650	0,0037313	0,0055249	0,0169133	0,0053981	0,0076336	0,0065987	0,0051813	0,0080321
Y	0,01754386	0,0121951	0,0111940	0,0165746	0,0887949	0,0107962	0,0572519	0,0215957	0,0172712	0,0240964

	N	P	Q	R	S	T	V	W	Y
A	0,0426966	0,0568475	0,0558824	0,0445736	0,1099476	0,0729783	0,0699588	0,0303030	0,0404984
C	0,0089888	0,0103359	0,0088235	0,0077519	0,0174520	0,0177515	0,0192044	0,0075758	0,0093458
D	0,0831461	0,0310078	0,0470588	0,0310078	0,0488656	0,0374753	0,0178326	0,0151515	0,0186916
E	0,0494382	0,0361757	0,1029412	0,0523256	0,0523560	0,0394477	0,0233196	0,0227273	0,0280374
F	0,0179775	0,0129199	0,0147059	0,0174419	0,0209424	0,0236686	0,0356653	0,0606061	0,1308411
G	0,0651685	0,0361757	0,0411765	0,0329457	0,0663176	0,0433925	0,0246914	0,0303030	0,0249221
H	0,0314607	0,0129199	0,0294118	0,0232558	0,0191972	0,0138067	0,0082305	0,0151515	0,0467290
J	0,0539326	0,0620155	0,0735294	0,0697674	0,0715532	0,1183432	0,2949246	0,0833333	0,1121495
K	0,0539326	0,0413437	0,0911765	0,1201550	0,0541012	0,0453649	0,0260631	0,0227273	0,0311526
M	0,0112360	0,0103359	0,0205882	0,0155039	0,0157068	0,0197239	0,0315501	0,0151515	0,0186916
N	0,3168539	0,0232558	0,0441176	0,0387597	0,0541012	0,0433925	0,0164609	0,0151515	0,0218069
P	0,0202247	0,4935401	0,0235294	0,0193798	0,0296684	0,0276134	0,0164609	0,0075758	0,0155763
Q	0,0337079	0,0206718	0,2147059	0,0484496	0,0331588	0,0276134	0,0164609	0,0151515	0,0218069
R	0,0449438	0,0258398	0,0735294	0,3449612	0,0401396	0,0355030	0,0219479	0,0227273	0,0280374
S	0,0696629	0,0439276	0,0558824	0,0445736	0,2198953	0,0927022	0,0329218	0,0227273	0,0311526
T	0,0494382	0,0361757	0,0411765	0,0348837	0,0820244	0,2465483	0,0493827	0,0227273	0,0280374
V	0,0269663	0,0310078	0,0352941	0,0310078	0,0418848	0,0710059	0,2688615	0,0303030	0,0467290
W	0,0044944	0,0025840	0,0058824	0,0058140	0,0052356	0,0059172	0,0054870	0,4924242	0,0280374
Y	0,0157303	0,0129199	0,0205882	0,0174419	0,0174520	0,0177515	0,0205761	0,0681818	0,3177570

Tabela 5. Prawdopodobieństwa mutacji aminokwasów wyliczone z macierzy BLOSUM62 [23].

DODATEK D

Poniżej przedstawiono implementację operatorów insercji i delecji. Oznaczenia jak w dodatku A.

```
//=============================================================================
void Protein::insertion()
{
//w większości przypadków insercja/delecja dodaje/odejmuje 1 aminokwas, dlatego takie zmiany powinny być
// najczęstsze, ogólnie im dłuższa insercja lub delecja tym mniej jest ona prawdopodobna

MTRand random;
int current_length = protein.size();
int range = max_length-current_length;
int position = random.randInt(current_length-1);       //miejsce rozpoczęcia insercji
int part = range/5;

double choise = random.rand();
int number = 0;

if (choise < 0.5 || part == 0)          //połowa insercji dodaje 1 aminokwas
        addAA(1, position);             //opis funkcji addAA() w dodatku A
else if (choise<0.75)                   //1/4 insercji dodaje 2-3 aminokwasy itd.
        {
        number = random.randInt(part) + 2;
        addAA(number, position);
        }
else if (choise<0.875)
        {
        number = part + random.randInt(part)+ 2;
        addAA(number, position);
        }
else if (choise<0.9375)
        {
        number = part + part + random.randInt(part) + 2;
        addAA(number, position);
        }
else if (choise<0.96875)
        {
        number = part + part + part + random.randInt(part)+ 2;
                if (number < range) {addAA(number, position);}      //sprawdzamy czy nie wyszliśmy poza zakres
                else                {addAA(1, position);}           //ewentualna korekta
        }
else    {
        number = part + part + part + part + random.randInt(part)+ 2;
                if (number < range) {addAA(number, position);}
                else                {addAA(number-range, position);}
        }
}
//=============================================================================
void Protein::deletion()
{
MTRand random;
int current_length = protein.size();
int range = current_length - min_length;
int position = random.randInt(current_length-1);       //miejsce rozpoczęcia delecji
int part = range/5;

double choise = random.rand();
```

```
int number = 0;                                  //zmienna opisująca liczbę aminokwasów do usunięcia

if (choise < 0.5 || part == 0)
        protein.remove(position, 1);
else if (choise<0.75)
        {
        number = random.randInt(part) + 2;
        protein.remove(position, number);
        }
else if (choise<0.875)
        {
        number = part + random.randInt(part)+ 2;
        protein.remove(position, number);
        }
else if (choise<0.9375)
        {
        number = part + part + random.randInt(part) + 2;
        protein.remove(position, number);
        }
else if (choise<0.96875)
        {
        number = part + part + part + random.randInt(part)+ 2;
                if (number < range) {protein.remove(position, number);}          //korekta
                else                {protein.remove(position, 1);}
        }
else    {
        number = part + part + part + part + random.randInt(part)+ 2;
                if (number < range) {protein.remove(position, number);}
                else                {protein.remove(position, number-range);}
        }
}
//=============================================================================================
```

DODATEK E

Program ProteinFinder testowano na kilkunastu sekwencjach o różnej długości i punkcie izoelektrycznym pochodzących od różnych organizmów w celu zbadania czy wyniki zależą od właściwości analizowanych obiektów. Przy białkach zapisanych w formacie FASTA podano skróconą nazwę, numer dostępu GI w bazie GenBank, długość sekwencji w jednoliterowym kodzie oraz w razie potrzeby pozycję wykorzystanego fragmentu. Ponadto podano ich masę, absorbancję i punkt izoelektryczny obliczony według wartości pK pochodzących z EMBOSS.

>Maximin-H5; GI: 68052206; pozycje 124-143; długość: 20; mw: 2022,41; abs: 0; pI: 3,333
ILGPVLGLVSDTLDDVLGIL

>HSY; GI: 78100177; pozycje 111-130; długość: 20; mw: 2302,49; abs: 0,555; pI: 7,547
GRHDYVASPPPPKPQDEQRQ

>ADM; GI: 110825703; pozycje 22-41; mw: 2445,86; długość: 20; abs: 4,653; pI: 11,649
ARIDVAAEFRKKWNKWALSR

>AP9; GI: 74837068 ; pozycje 25- 74; długość: 50; mw: 5733,06; abs: 0,695; pI: 2,621
YPASYDDDFDALDDLDGLDLDDLLDSEPADLVLLDMWANMLDSQDFEDFE

>Crambin; GI: 6226577; długość: 46; mw: 4736.46; abs:0,693; pI: 6,024
TTCCPSIVARSNFNVCRLPGTPEALCATYTGCIIIPGATCPGDYAN

>DH; GI: 118665; długość: 46; mw: 5361.24; abs: 0; pI:12,499
TGAQSLSIVAPLDVLRQRLMNELNRRRRMRELQGSRIQQNRQLLTSI

>Flagellin; GI: 12230095; długość: 201; mw: 21432,33; abs: 0,119; pI: 3,421
MFEQNDDRDRGQVGIGTLIVFIAMVLVAAIAAGVLINTAGMLQTQAEATGEESTDQVSDRLDIVSVSGDVDDPDDPT
QINNISMVTATAPGSDPVDLNQTTAQFIGEGGEEMFNLSHEGVFINSIQGVTDEPDNNVLTESSDRAEVVFELDGAP
GSYDIGYEALDESERLTVILTTDAGASTEQEIRVPSTFIEDEESVRL

>DMP; GI: 17865460; długość: 934; mw: 93902,94; abs: 0,194; pI: 3,459
MKMKIIIYICIWATAWAIPVPQLVPLERDIVENSVAVPLLTHPGTAAQNELSINSTTSNSNDSPDGSEIGEQVLSED
GYKRDGNGSESIHVGGKDFPTQPILVNEQGNTAEEHNDIETYGHDGVHARGENSTANGIRSQVGIVENAEEAESSVH
GQAGQNTKSGGASDVSQNGDATLVQENEPPEASIKNSTNHEAGIHGSGVATHETTPQREGLGSENQGTEVTPSIGED
AGLDDTDGSPSGNGVEEDEDTGSGDGEGAEAGDGRESHDGTKGQGGQSHGGNTDHRGQSSVSTEDDDSKEQEGFPNG
HNGDNSSEENGVEEGDSTQATQDKEKLSPKDTRDAEGGIISQSEACPSGKSQDQGIETEGPNKGNKSIITKESGKLS
GSKDSNGHQGVELDKRNSPKQGESDKPQGTAEKSAAHSNLGHSRIGSSSNSDGHDSYEFDDESMQGDDPKSSDESNG
SDESDTNSESANESGSRGDASYTSDESSDDDNDSDSHAGEDDSSDDSSGDGDSDSNGDGDSESEDKDESDSSDHDNS
SDSESKSDSSDSSDDSSDSSDSSDSSDSSDSSDSSDSSDSSDSSDSSDSNSSSDSSDSSGSSDSSDSSDTCDSSDSSDSS
DSSDSSDSSDSSDSSDSSDSSDSSDSSSSSDSSDSSSCSDSSDSSDSSDSSDSSDSSDSSSSDSSSSSNSSDSSDSS
DSSSSSDSSDSSDSSDSSDSSGSSDSSDSSASSDSSSSSDSSDSSSSSDSSDSSDSSDSSDSSESSDSSNSSDSSDS
SDSSDSSDSSDSSDSSDSSDSSNSSDSSDSSDSSDSSDSSNSSDSSDSSDSSDSSDSSDSSDSSDSSDSSDSSDSSD
SSDSSDSSDSSDSSDSSDSSDSSDSSDSSDSSNSSDSSDSDSKDSSSDSSDGDSKSGNGNSDSNSDSNSDSDSDSEG
SDSNHSTSDD

>p104; GI: 74950752; długość: 893; mw: 101920,59; abs: 0,794; pI: 7,576
MKFLVLLFNILCLFPILGADELVMSPIPTTDVQPKVTFDINSEVSSGPLYLNPVEMAGVKYLQLQRQPGVQVHKVVE
GDIVIWENEEMPLYTCAIVTQNEVPYMAYVELLEDPDLIFFLKEGDQWAPIPEDQYLARLQQLRQQIHTESFFSLNL
SFQHENYKYEMVSSFQHSIKMVVFTPKNGHICKMVYDKNIRIFKALYNEYVTSVIGFFRGLKLLLLNIFVIDDRGMI
GNKYFQLLDDKYAPISVQGYVATIPKLKDFAEPYHPIILDISDIDYVNFYLGDATYHDPGFKIVPKTPQCITKVVDG
NEVIYESSNPSVECVYKVTYYDKKNESMLRLDLNHSPPSYTSYYAKREGVWVTSTYIDLEEKIEELQDHRSTELDVM
FMSDKDLNVVPLTNGNLEYFMVTPKPHRDIIIVFDGSEVLWYYEGLENHLVCTWIYVTEGAPRLVHLRVKDRIPQNT
DIYMVKFGEYWVRISKTQYTQEIKKLIKKSKKKLPSIEEEDSDKHGGPPKGPEPPTGPGHSSSESKEHEDSKESKEP

KEHGSPKETKEGEVTKKPGPAKEHKPSKIPVYTKRPEFPKKSKSPKRPESPKSPKRPVSPQRPVSPKSPKRPESLDI
PKSPKRPESPKSPKRPVSPQRPVSPRRPESPKSPKSPKSPKSPKVPFDPKFKEKLYDSYLDKAAKTKETVTLPPVLP
TDESFTHTPIGEPTAEQPDDIEPIEESVFIKETGILTEEVKTEDIHSETGEPEEPKRPDSPTKHSPKPTGTHPSMPK
KRRRSDGLALSTTDLESEAGRILRDPTGKIVTMKRSKSFDDLTTVREKEHMGAEIRKIVVDDDGTEADDEDTHPSKE
KHLSTVRRRRPRPKKSSKSSKPRKPDSAFVPSIIFIFLVSLIVGIL

>rbsA; GI: 118573270 ; długość: 859; mw: 96152,83; abs: 0,237; pI: 11,664
MRASLENGDDHDAHRLVDAGFRPPGRPRAARRRAFARARRGERRARGTAEDRHDVPGAEQPVLRDDAKGARRGGRVD
RRAGDRHRRASRREQAGERRRGHAAEEDRHPAREPDRFDGHPVGRRVGEEGGRRRRRGGRERERPGRRVRRLEEFRR
GRDVVRLPREGDRRRRRSRDPRRHPGRADSRARARLPRGAREIPEREDRRRAERQAGARERARRHREHDPGAPVAQG
RLQRQRRRLDGRAVRDRGVGPRHQAHERRRRAGGDRGDAEAELEVHRDVRAVPARPDSPRDRHRAREEVGRQCAEGD
SGRREADRQGQREDVQLVSGPRDEADDMDEASGAARAPDEASEEAMDTILALTGITKRFPGVVALRGIDLRVARGEI
HALLGENGAGKSTLMKILCGIHPPDEGVIALDGEPRRFANHHDAIAAGVGIVFQEFSLIPELNAVDNLFLGREWRGR
LGLRERARMRRAAADIFARLDVAIDLSAPVRELSVAQQQFVEIGKALSLDARLLILDEPTATLTPAEAAHLFGVMRE
LKRRGVAMIFISHHLDEIFEVCDRITVLRDGQYVGTTEVARTDVGALVEMMVGRRIEQSFPPKPRLARDAAPVLEVD
ALQVRENGPVNRFALREGEILGFAGLVGSGRTSSALALIGAKPARVRRMRVRGRPVCLADPAAALAAGIGLLPESRK
TQGLIPAFSIRHNIAINNLGKHRRLRWFVDAAAETRTTLELMQRLGVKAPTPHTRVDTLSGGNQQKVVIARWLNHHT
RILIFDEPTRGIDIGAKAEIYQLMRELSARGYSIVLISSELPEIVGLCDRVAVFRQGRIEAMLEGEAIEPNTVMTYA
TSDVRGANHEHA

DODATEK F

Przedstawiono tu bezpośrednie wyniki uzyskane za pomocą programu ProteinFinder z wykorzystaniem białek testowych (dodatek E). W pierwszym przykładzie pokazano zachowanie się algorytmu w co 20. iteracji, aby zaprezentować zbieżność do znalezionego rozwiązania. W pozostałych przypadkach ograniczono się jednie do podania najlepszego znalezionego białka. Ponadto podano zastosowane parametry algorytmu genetycznego (ilość iteracji, wielkość populacji, częstość mutacji i innych operatorów oraz średnią wartość przystosowania osobników bieżącej populacji wraz z jej odchyleniem standardowym). Jeśli nie zaznaczono tego wyraźnie w opisie parametry te są takie same dla wszystkich analiz danego przykładu. We wszystkich przypadkach użyto wartości pK według EMBOSS. Dodatkowo zaznaczono czy algorytm korzystał z macierzy BLOSUM lub krzyżowania dwupunktowego (oznaczenie cross2). Kompletne wyniki można znaleźć w folderze „ProteinFinder/wyniki".

Przykład 1. Białko Flagellin o długości 201 aminokwasów

```
Target protein:
mw: 21432.3, pI: 3.42, absorbance: 0.119
For isoelectric point calculation pK values according EMBOSS were used
-----------------------------------------------
Genetic algorithm parametrs:
population size:2000, number of iterations: 200
crossing over rate:0.75, elit rate: 0.1
mutation rate: 0.045, insertion rate: 0.05, deletion rate: 0.05
===============================================
```

Analiza 1 (całość)

```
Best protein :
NPDEDSEJAGDGDYJSFFMGVMJGMGNEVAAJAAPFMDJAGJEDEVGVJFSASSNTPVPRMGQGDVEGJGPFTEDVQJJDJGJFKFTFJQEEEGAEP
HGQVEECJJNQVVMHEAVQFJJNDMFPJTVVAGMJASDCTVVVSPJEEFGSJPVNTGJJTGDYQAJQNGEJJEQSJGVDEGASTEGVAETHQJKPKT
MCSCDVPJ
in iteration: 19
Length: 202, mw: 21463.1, pI: 3.45947, absorbance: 0.141638, fitness: 0.120735
Population average: 1531.52, standard deviation: 886.79
=========================================================
Best protein :
TNTJSNPVERJJDTTJJSAAPCSGFVRCTVETGCHAATYGMSQHJTEDASVEGVDEFCJJCCNGQFAJTSADJNJNSQJJRAAVGJDSFGEADVJJP
JJFJADNDJDJADNGDSTDTADPDGJDETTNGTNCJEJMTTVVVSPJEPVDSNPVNEGJAJJGAQQAJQNGEJJSQPJGGDEGFSSTEGVAETMQJC
FKJMCSCDVDJ
in iteration: 39
Length: 205, mw: 21456.9, pI: 3.3833, absorbance: 0.115581, fitness: 0.0420716
Population average: 1218.21, standard deviation: 690.399
=========================================================
Best protein :
QJPAGVQJNNVNFDGSEVJSDFAPMGNEVAAJAAPFMDJAGYPDEVGVEFVASSNVPVPRMGQEDVEGVPPPPEDVQTJDEAGJAAAKJNYRDESJE
PQDJEJDGFMSAGPEJJDEEDGSDQVVQMNFJQETJGPTAAVPGDJDTEJKSJJVVNGJPESQFQTDGNNDJRMHKEGVMQEQGVAATPDSSJTNDV
VDEJEHJ
in iteration: 59
Length: 201, mw: 21421.5, pI: 3.42139, absorbance: 0.119506, fitness: 0.00580999
Population average: 1179.11, standard deviation: 664.027
=========================================================
Best protein :
QJPAGVQJNNVNFDGSEVJSDFAPMGNEVAAJAAPFMDJAGYPDEVGVEFVASSNVPVPRMGQEDVEGVPPPPEDVQTJDEAGJAAAKJNYRDESJE
PQDJEJDGFMSAGPEJJDEEDGSDQVVQMNFJQETJGPTAAVPGDJDTEJKSJJVVNGJPESQFQTDGNNDJRMHKEGVMQEQGVAATPDSSJTNDV
VDEJEHJ
in iteration: 79
Length: 201, mw: 21421.5, pI: 3.42139, absorbance: 0.119506, fitness: 0.00580999
Population average: 1257.5, standard deviation: 723.59
```

```
==========================================================
Best protein :
NGTVDSEAAQDGDYJSJFMTPMEETDAGGSNAEQDFADNAGPVAENGVJFSASSNTPVGRJGEVRJSFADENVVQSQVJTAHEEAGTJJSJTEEJEE
TFPSAVVNGSSJPGQJDJAYFGKFEEEVDFEJAMPTPVJPJGGVSDVSNENGJDDJDVNSSFRNPJESSJFMGANGJJVEVNGETVJAVJSMEJRET
TSPVAEESR
in iteration: 99
Length: 203, mw: 21426.4, pI: 3.42139, absorbance: 0.119479, fitness: 0.00392229
Population average: 1197.75, standard deviation: 678.363
==========================================================
Best protein :
NGTVDSEAAQDGDYJSJFMTPMEETDAGGSNAEQDFADNAGPVAENGVJFSASSNTPVGRJGEVRJSFADENVVQSQVJTAHEEAGTJJSJTEEJEE
TFPSAVVNGSSJPGQJDJAYFGKFEEEVDFEJAMPTPVJPJGGVSDVSNENGJDDJDVNSSFRNPJESSJFMGANGJJVEVNGETVJAVJSMEJRET
TSPVAEESR
in iteration: 119
Length: 203, mw: 21426.4, pI: 3.42139, absorbance: 0.119479, fitness: 0.00392229
Population average: 1234.05, standard deviation: 700.865
==========================================================
Best protein :
AQDEJSESAJDMDFJGFJVGGDJGMJNEVJQERNJFMYJDGEEDETGJJESASSNPJVMRJGQGJVEGJGPGTAAVQPJPAGEEEGAJJSATEEJEE
TTPSAVVSGSJESGEJDJVYAGKFEEDVDFEJAMPVPVJPJJGVSDVSNENGTDEMVKNSSFRNPJESSFFMGANGJJSKPQGQNVJAJJSVEJREJ
AAVGDEJJV
in iteration: 139
Length: 203, mw: 21437, pI: 3.42139, absorbance: 0.11942, fitness: 0.00331486
Population average: 1252.62, standard deviation: 705.82
==========================================================
Best protein :
JPDDMMDVNJTJNEDVVFVGJEAGTKNEVJQEATPFMYJAGEESEVGJJEEASSNPDVMVMGVGTVEGJGPGTQARTPJPAGEEEGAJJSJTEEJEE
THPSAVVNGSJEPDQJDJAYGGKFEEDTVDFDJAMPVPVJJJJJVNDVSNENGJDDDVKNGSFRNPJESSJTMGANGJASEPQGJAFJAJJASEVRJ
JTSPVAVSJA
in iteration: 159
Length: 204, mw: 21429, pI: 3.42139, absorbance: 0.119464, fitness: 0.00312477
Population average: 1216.03, standard deviation: 697.819
==========================================================
Best protein :
JPDDMMDVNJTJNEDVVFVGJEAGTKNEVJQEATPFMYJAGEESEVGJJEEASSNPDVMVMGVGTVEGJGPGTQARTPJPAGEEEGAJJSJTEEJEE
THPSAVVNGSJEPDQJDJAYGGKFEEDTVDFDJAMPVPVJJJJJVNDVSNENGJDDDVKNGSFRNPJESSJTMGANGJASEPQGJAFJAJJASEVRJ
JTSPVAVSJA
in iteration: 179
Length: 204, mw: 21429, pI: 3.42139, absorbance: 0.119464, fitness: 0.00312477
Population average: 1218.85, standard deviation: 687.862
==========================================================
Best protein :
JPDDMMDVNJTJNEDVVFVGJEAGTKNEVJQEATPFMYJAGEESEVGJJEEASSNPDVMVMGVGTVEGJGPGTQARTPJPAGEEEGAJJSJTEEJEE
THPSAVVNGSJEPDQJDJAYGGKFEEDTVDFDJAMPVPVJJJJJVNDVSNENGJDDDVKNGSFRNPJESSJTMGANGJASEPQGJAFJAJJASEVRJ
JTSPVAVSJA
in iteration: 199
Length: 204, mw: 21429, pI: 3.42139, absorbance: 0.119464, fitness: 0.00312477
Population average: 1270, standard deviation: 713.396
==========================================================
```

Analiza 2

```
JVVAVJDEVKJYEDTEPJJAJJDJAJKFVJDMGGFJDEGAGARHDANQJJSJTTGNAAJTJNDHGFJQQPHVSJQFEDGSGNPGJDGEAMPAPJEAT
AYJPEEFTAMNQDDPJNJJJEJJEEAJNPJANPFGNTDSSNVPFJSPTFDQJHEEVJTJEJGQASDDGSTJGGEEEDGQNJSNNJVFTJJAGVDGTQ
FDQJVVS
Length: 201, mw: 21434.7, pI: 3.42139, absorbance: 0.119432, fitness: 0.00279836
Population average: 881.788, standard deviation: 499.968
```

Analiza 3

```
TDVTNTDDNVTAVVJSADJNDEGPMVPPEAJYETGTDNDDGTSGDAPKTGFEMNJDJFJAJPJEGDQEMKQGEKJFQEGJMJJJPYAGESDPGEASD
EQRTPSGTDFJKJTFJPEVPDJPFARAEENRPFJJPESDAVQTSEJDNEVDDADJVDSGAETJDVJSJSQGJTVVAJVETDKVJATJJJVFMSJKEE
FGDAM
Length: 199, mw: 21432.7, pI: 3.42139, absorbance: 0.119444, fitness: 0.00262161
Population average: 835.365, standard deviation: 473.169
```

Analiza 4

```
QVFDMGANEJDJPEJKSEVRTFFAAJGGJJJJNGEDGJNRNEJDDNPQPVDVGMADVHYAJDDAJQMFTQGJADTNGTJDENJMJGSAJTMJMNPER
EJDGAGJMEJSQGVEJAEEEHAJJGDVJPMEEJJTJEDPNJVAQJGKQPDAEEJVJMGFGJPDJVFSNNEAAEJETGFSVVEEVSEDYNGTTRVAEA
FFAJ
Length: 198, mw: 21432.9, pI: 3.42139, absorbance: 0.119442, fitness: 0.00262534
Population average: 912.161, standard deviation: 523.308
```

Analiza 5 (ustawienia jak wyżej z wyjątkiem wielkości populacji – 200 i liczby iteracji - 2000)

```
NFVTAKEJFGJDAJGJDJTRVEDADGKSPDADSVGEGJTECATTRNNJJJGJASTSJFJDFEECAKCJVCCJPVAJEEGJDSNVAPSDPJJJJTRFG
APJJDDPJEAVQTYJTJFAPTATDNVPVSNTJJJTPJDEQDJJAEVPGPNCDFPMVCQJCPTVJRNQAJDCEJAGGDTGMAJJEPGDMTJJEJNCVP
GGMDCDJEV
Length: 203, mw: 21433.4, pI: 3.42139, absorbance: 0.121306, fitness: 0.0116235
Population average: 62.7904, standard deviation: 37.0711
```

Analiza 6 (ustawienia jak wyżej z wyjątkiem wielkości populacji – 200 i liczby iteracji – 2000, ponadto mutacja z macierzą BLOSUM)

JVVNQJNSVSEJJNFDJHAEJFDJQMJEPKEQDTJDRTGQSRPJJANASGEEJFSDPDJDHJDAGTJATTDEADSRFGEJTJJNASNJVAEAEVRFT
NASSPTAPTFJTVYNJJQDRPDNJNDTSJEGEGEJGPEPJEJAMNJVSNDEFDDPJADPPJJSFJEGEADFJJTJYDJVFPPEDFAVJFJGJEQETS
M

Length: 195, mw: 21432.6, pI: 3.42139, absorbance: 0.119444, fitness: 0.00262229
Population average: 62.7784, standard deviation: 35.2977

Analiza 7 (mutacja z macierzą BLOSUM)

HADRDJDDAVVQJDFFSGTPGSARNJSGTJJNAPJGDVVGAVDFJDEAJEGAVTQQTJGTEDJTNDTDMATJJJPDYTJNVGJVJVSFJGJSEFFDQ
SQAJFFFNDPJEGSPGDDPFJMTTYDJGREFVGGTFGPGKMMDAGTEJTQEVGTENJTSDEVJFJJMESSJJEPDJGKHEDMDPGJJJEFADGDJHF
EGPTVA

Length: 200, mw: 21432.7, pI: 3.42139, absorbance: 0.119443, fitness: 0.00262313
Population average: 618.812, standard deviation: 351.413

Analiza 8 (mutacja z macierzą BLOSUM)

Best protein :
VTTEGSFFJQGHNMDAGDGDVJNNJEJDGDKHAJSEGGEJJDGEAJJNFFFRASEAGFDJQJJREATSJGSNFJJYVFJVJJMDDVJDMDVVAAJEA
QPSDGSQVDFNENAEVDEFEHQTQMJEAJDJSKTQJDJGJRJVMENEADPVEPDEGVJSQFJVTJENJAJEDVGNTQJADSVTDTDRDEYGJDADVJ
JJ

Length: 196, mw: 21432.6, pI: 3.42139, absorbance: 0.119444, fitness: 0.00262214
Population average: 683.316, standard deviation: 387.368

Analiza 9 (mutacja z macierzą BLOSUM)

GFYKJGFJDVJGNSJGTJYNMAAQDKAGQSSEJGSDESSQDDQDQSTQJJEDTNJJJQJJJEGVAAGVRNEGFTPVJEAJDANDSHSNNQJFJJAJQ
STGTTPJFJNVEDVVJVDGFJDJPGPGPGJDETSSQJGFDTVJNSJATJJDFAHGAKJDEQSDJDPFFVVPEAMNJDJSVSPEDDJRGVNVSFSVGD
AEGGGGPVVV

in iteration: 199
Length: 204, mw: 21432.6, pI: 3.42139, absorbance: 0.119444, fitness: 0.00262214
Population average: 674.331, standard deviation: 380.074

Analiza 10 (mutacja z macierzą BLOSUM)

APTQDVFNFQEJSJGAEJNFJEFJQPDSAMGVDNEYJSQMVFGJJVJEVETAJDEJAPFVQDMTGAVDKSAEGQFDGSEEYVPDDFSFQJENVQVJV
DARAKFVNVEFQSGAEQAVGSJEFFVEEMVSVNGDJJAPAGEFFSDSFEVDJDEKDFJNTVTVEJTSERVDJMGFANADFFMGDREJJJTNGPKRGD
Length: 194, mw: 21432.7, pI: 3.42139, absorbance: 0.119444, fitness: 0.0026227
Population average: 706.471, standard deviation: 402.208

Przykład 2. Białko rbsA (100 pokoleń, 1000 osobników).

DMGCRSEYQDYVDJDTRESFNKRKSNFJPGATJNDHNATPSSAJHHJGQQVRKFRRPKPKSFFKJVJHVGDRJKAVSJVRQPAJEVAPFGJSSGVJR
KGKJGVRTYNQHJASNEJDJKSPMARJJFJCRAREFQEJJTGHRKGJFRARADJGJJDGSAJSJSSADAJTJJRGSKDCQFVTHQJERMGVVPVANJ
RPFKTSRGJTJQFGPEEJJGKEEVRVPMKAAQJKNENNAJMVGPJSERMCAVJTVAFMVGKRNTVSJPJVTGTKRMSGSWRKFDJRSGRVNRCYAAV
GVFTCRTERFHCNRAEVGFJAKHSYMDRRTNAKRJCNGRANGAGNSGRAFNJNNAJVTSRJFEJVHFRRQESRKJPSGEPTFJEGTNPQHJRTKSRJ
GDPDGPSGRSGTAHSJSMTGJFJNMHKFARKKJRVAMQRESRTPQDMMGVJGGJJRJRVEVVYTGTAJGTQQSAHKKRRDEARSRNGJQAEJRKDVR
TAKNKRAGGJEQVVANRSTCNJJSJTJTCVJPFEJSVRGGVPKREYTVVJAAMJSQARCGJMDJDGPPRKTJJSGRNVVCAADSRKAKKJAQTAAAP
TCMNRTHTJJNRJNAMNFGAGARAVJSJSJJVTSQJTVMSAQVQRVAQETPJJFAARCJVKRQRQAQJEJARJNNMMGQJQAHYNJRRSJSKRRKGV
NPNHRVEJYKAAGRVGSMJJEREJYMNRTFGEVGKREAATGJGJKTKGJSHAGRTTVSSAPJFKMANJVANDSTCHTGPDSKRSNJRQJVRREHJPQ
FRFJMYTRMJRKGJJSRFKGSVGGJTRKAJVEMSMGJAJJFQGVVQGRJGKJSPNJDNGCSJVTTARAFJSPJJGMKPSAFYRVSSFDFSJR

in iteration: 99
Length: 868, mw: 96219.9, pI: 11.6489, absorbance: 0.238724, fitness: 0.0156705

Przykład 3. Białko DMP (te same ustawienia dla wszystkich trzech powtórzeń). W tym jak we wszystkich przykładach podstawionych poniżej wykorzystano mutację uwzględniającą macierz BLOSUM.

Analiza 1

mw: 93902.9, pI: 3.459, absorbance: 0.1945
population size:2000, number of iterations: 400
crossing over rate:0.75, elit rate: 0.03
mutation rate: 0.01, insertion rate: 0.07, deletion rate: 0.07
JJKGJAQAJAQTKSFFDJPGJASDDJTVGSVJTSPVDGSVPNFNVEETSJJDKVJJVEDJKDVVGADJQEEENADYGDRFEDJGJASSGDGSEJVET
VCADSSJVEJJJDNEVEQPEEDFEFTDVVVTMFJEFKVJEJYJTDPSGEETPCKJJJYSVJFMJSMYVEASSFDTFVFVGDSRSDADNTNEJNEVJG
CRDTJJGGNETVGEPEFRJFSDDGANGDPETMDJDGAEJFGAPAJSAGQERDDTEGFGJDCMGAJVTJEQJGTEPNGDKEVEEDNCGGVAKVSAFDH
EMGEKASGSAMJTGMSJFJYSJJTFPGVFKNMQESEVJJQSQEFJVJPJSJQTSATAFJPPTGSSDSCJJVSQDMPQDGCCAGJFPPGTSVJYJFJP
DQPRPNGSATJGAEDJYPQEJJTKJJTEJCPDYDFAJVJASDAMSJJJGCVFKSJJSTFQTDAVEFMPTJNGJEAVKTAATVDTPGJTMJSJESSTN
DVNYAPQAESNDGVJVETJTDDFSGJPJJSDDADNGMKEDVJEENFDVJNVEEMRTDDJYEKJVJJEAQAJSTGCDRTDAJMQMJVFPPTDDFSDAJ
FDAGGEDSGYKQDMJEJSGJJFFVPTJVNNPFSMNSJDFPPEEGFSAATJJASDEDSSFKEDJJEJHNVJFFVPTDKNNRFSVVGQGQFGFTSJSGA
JJEJSAJSFDJJNRRENASTDMJYPDEVKPEGQGVGTGEDSMVQEMNMDDJJHPDPJAPEGQEQJJNDEDAJDAEJRNCEEPCEGJEJANCEJJGEG
JKAJAPADFDFQJDEMVVFDDGGAVVEJQAJFGJJJPGNKJEEJGJMRJNSSTYGSEPAEEPHEJJAFDQJTFSAQEJQSTJJVJDVFVGJTT

Length: 869, mw: 93905.6, pI: 3.45947, absorbance: 0.195089, fitness: 0.00299877
Population average: 854.068, standard deviation: 504.335

Analiza 2

EJENDTJDGFJJCDKTJJPQJJEPJJGDGGVASAKGFGJJSEVEENNGKTENDJAJFJVJFPSJJSVNEAJQYEJDAJAEAASFNPNYDSJNJJJDE
QPEESEVTJGMDJJDEPADDQRTEJEVGJFSADGVRNDSAEEHENJTJGGFAPGSFFEFPQDVQGSSJEDGGVCGFTJJAHJEDGGJGAJPJJAAMT
DHDNDAGVTESJGRDSCQJJQNDAJQDJCFQYFESJJDDTJEETTAEQAVFDDGDTJKQQAYJDDEDJJGJAGDGJDJPTTEVATJJDAMJJNGTYC

TEQQFJJFEAJCEJJJNTGQGJSDGVDTEDAJQTDRDKEJNEVJEVJVJTJNJATQVEGJQAGJCJMJFEAGJVGDJSNQEVJJGGGFPFPEMSCNF
DDEVJKPVJFJEEVDGFNTPVECJDJJJNDJCJJJENGVACMAENETGAEDGTGSJNGANJYEVNAPRASVPPECYNEANGJDSJNPAVEVDJDEAS
VAKAJAKRJFTGQMDMRJGAGPHRYAYTPSJJSEJFRFKEASMEFRJVQGEVENESKJEJEGVJNRAAFEPCDEVSDEHGVPJJJCFDJJJESGGSV
NGEDQQGJPJJSJARNNTGETSQGVNGDPPEVSDASANVTVRMJMVNAGDJEFTFSJDFCEEEDJDDJJTPAESVVVJCMVVAVHDRJVDSNNESNG
AEVJJMTFGNFQGSDNAEJYDTMKCJYAJFTGGJRDDDQDJYGVJAAJJTVJADAEGMVGCFNNSGQJVJGDPVTPFETCGAGVMAMEEVJEDFCJA
JDVJJRVEMDCGFJSAJGPVCEGDAGFPCAJVVAFGDEFKRGEFPVAESDDFQFGJFECDTFJSRJADEJFPRNTVKAEJJENGAFSVASTQJFFP
Length: 872, mw: 93901.7, pI: 3.45947, absorbance: 0.194246, fitness: 0.00136297
Population average: 821.177, standard deviation: 478.079

Analiza 3
EMJJQDSFQDSAGJVFTGFJJJAGDJEANNSTQJJJVPEGJJNJQETASETESFQTDGPDQJNAGQDJATEFJSJDGDGVYJCDJSRJVCTSQVSJD
TFJVQENVEPRJADCGTVJTTGVPFJDAEDJDFGSDTSPAVAMESPYKPTAEJGVNJVMJJJAGEAJEDEJHPSTEVQQPMATSJGJJJAAVMEJFS
EFSCAAEJYVJJGESFQMEVNAATKVEEJEFJJJSFYGRDGJKGEKGQNJJGTSVDEESCEVJDDPVPVTQJFSVAREPVNDJVADQGGFAGAJTEN
MJNSNSQVPJAVDAFAQTGETJTGSQJDRNNVEESVNYYEAFTJSEAAJJEEGAFJJDGCEHSJGMAJVJJAGFJSDDHJAQPVEEQFRGQTTJYFJ
SEDVTTEVFTJJDNVGVQJESEVDTAJEJJGCSNVNVEEGGJNAANEJNVNVEPDPEASSASGAAMSNSEAGHJAAJGEFDFYTGDEQJGVPSFDMD
KPFFJNEGGNJQNEJJEHTJJDATTJMFJTHSNDPEAFVECYJGDDJEYTAFCHVTVPJMGPMNQJAJJVTFDVMVNJJJFVSJVVNDJJTAPGKAH
TNPATTVJJEFQGJQPEGPPEJNFACTRNAGEGSDHPMSEDEDDTVVENGAHJPDEDDCADAFVTTNMJNQDFDFHMGGJSGJJNCTJTQVSSPEJN
TGNAJJTGAQJFANNJAVJVEGDGFAJMVJGYEJDSSEHQAJEEVHNKJFTFATDGGEJGDEADJSJEJSPJRNKTAYVFVJQJJAEVFVVQPFVEQ
EJNPQCTGEAJQTHQNTSAAYNTEFMVDDJSSTJMATSTVTJJTTGMVTSTTVDJEJGGJGPJJDJMSJQDGESGEPJNQRQJRGSCJJVEDVJJVG
TSA
Length: 876, mw: 93905.6, pI: 3.45947, absorbance: 0.195089, fitness: 0.00299877
Population average: 757.456, standard deviation: 436.666

Przykład 4. Białko p104. Suma przeanalizowanych osobników we wszystkich analizach jest taka sama i wynosi 600 tys. Zmieniono jedynie proporcje wielkości populacji i pokoleń.

mw: 101921, pI: 7.577, absorbance: 0.7941

Analiza 1
population size:300, number of iterations: 2000
crossing over rate:0.75, elit rate: 0.1
mutation rate: 0.01, insertion rate: 0.03, deletion rate: 0.03
KJJJHTTVVHCQSVVVAGFDVFENGAJAHAJAJDJAMJJAFGVJJDEKFSGDVNFSAGVJJSQVAAQQVAAHVQMRSJHJGDEGJTVJHAVVJMEAP
GGJCTTHAJEJSJKJNQRMSFDJKEFVRESGAJVDJAAASPJMVHRVJCPSJFGRGHFADRGPRQVKSEDQVADJDWFNCJCCJVSJPJJJNCVJGS
HQJJNKHDJJHFJTJGRNRYJJQKFWETMKTDSSDJNJJAVHHMVDGSJWARGPVPEAYVJCSCTHGTCJYKFTQVPGJAYQKPJEMGJASQTTHJG
VVWNRCNRNGJDKQJHKQKJDADGPEFASGKQSSAADMGJJFEVASYJJSSHNDSKJJGSGPSRJGAJJGVTTPVKMDSWJHGTCJVJEKVGTKNTG
JRPJJDJVSETVEEGGAMKCQPESYAPNWEJACAAJGQGJADRJQTDYJREPNTCSDVHHGKJJHRAHJGGKJRAEDPKGGJJJVANEPJGKMQGDR
GGTNKRANJTJADGKHESPPJHVKJVJDKYVTEDRSCJMVFNKRDHAHNJPQAKFVRDQTYJMSGTDKKEGTEGEAKESPPJDKRHSRTDYVDTFFP
DGGJGYMGCTASTDRDNJRVPSJDKEJJEAPAWPKJFGGKVQMQKMMFPASQFDRKSJFDVWJERGVVWJKMGGFAYANFJJGHJKGTQCVHJJJJG
EKCSJKTPNRKAJFGJGNGSTPAVAJVAGRSCSMEJSPAVEGDTJFQSRJNHERKGJJQJSJQAAKJPVTVKVJVSMSGVRNVGKANVMEAVDETFR
FDPFQCJSJRTHGVVCJAHJVKYJSYVSESSMVJTJGSJYDQGJRFNYADTATPFYTTSKYKJMYDRTTDAAHGVJFFVFTGVQPSVJDRGTREEEA
JJSSGAFPAJCTVAYAJVDVANRPHTJYAVAGGATJTGSJPPVHEJSRKATACMFFJDSK
Length: 933, mw: 101927, pI: 7.57666, absorbance: 0.794396, fitness: 0.00162498
Population average: 73.7047, standard deviation: 41.2948

Analiza 2
population size:2000, number of iterations: 300
VPEJNSHAJEKEPCHSJMDKCDVTQWDVAQGGYCKJPDPKMVJJKANJQNVJJACCJDJCJGEDSRADRARTRVNGFJACGKJMGYJAVCEAVAKVJ
JJCEMKAGAQNPTSQRMJADSEQSVHSNYGGKGDSMYYJHETQPDJSNQNTHATAAYFKVMTGPSEVJTAKRGNVVGJAKJGADYSAJVPJQNYDSH
DTGVAKANPGJKQTGJVDCRSASATJCYTDFJYPKSTJJSSAYDEVJASSPFAAQTJJYJFKJGJGPEGNDYJGDEGDPJEKDNSKPJJVGDKAVJQ
GVVJRSTQNGNKJNCKFVWEHVJNKPAKJRJJCGAQJGEEVRTJJHGASECJNJAPTMSCRFRWAKRQHVYFVGVNVTGGNSNJFEJQFSSGGEDDP
AJCGEKYEGQGJJMJSGQHKVHKYDNRRAJKJKRKYJPGRERTRPDJACERGEVASNWEJKHTSFJGRCSVVGGAFJEYSKAGVSETGQATCDSDCK
NTPGJAJNEFARPYSVAKQVRCASGFGAKGQFDVTJQNJWRTTQCPVNCJQVRKECDTVVKKNSASSTTGEMRYNNJNATQEEJJKGKTRJVNAGQR
AAGJFANGNGTECTARJKQAKSEJJEJDKKDGTKRGQPGJGVDJJNAJDJSJPREKRACVMMPSEGSAAFJEJYERSPKGEQKJEGJJJJMQEAQEN
DNGTWTFAGFEPSCSAJANTRFVGQATSPYJVRVHRTJYEDSAJKNVVGAFHVNHFJFTPJEVPVGPTKFKAKTTAFJACJQJDQVQFJJFJFJAST
HWADVFDJJJJAKCJGGJJFNGJHGPJNJPJVGVCNTYDJNEACEYJESGVDVYEJSSVCMTEVNRVHRRSTJTTJTTGJYJJJVNTJHNJJEJANQ
TSYTAJSGCJJNHVFTYKAJQDJRNFEJFESAFNGRGJGKGGHAGFJGRVPJJPKDNMV
Length: 932, mw: 101917, pI: 7.57666, absorbance: 0.793885, fitness: 0.00119632
Population average: 558.068, standard deviation: 317.217

Analiza 3
population size:3000, number of iterations: 200
EJRYVKJRGRSRACECJJJSJVTJRVHGQSAKNWGASTAGMJTGNRRHQYPHFPYGSADYVRGJQSQEJVRSEJGJDGVAQFETSETJCPDSAJVJT
VJKGYYJVPJEDVRPAENEMKMNYPKDEWANGGTGGNNERKAVQAKANAKYJPGKAGDETJGQGVECDCCJSAYKQPGYPGGVPKYJNSYQTTRVKQ
EAEKDQEAKSJKEFVAFASDTCCPJPSEDKNSJHKTVMQTGJAJGVVFSNJCPKKTNDJJTYTGJVJKKGAPJSQSFTDGRRQFVKSJVDVEVEHEE
SEJPDKTPAKDRQTAFDJDAQRQCJJKNEYVHNHKDQSSJGJRQGGSJPCFACQDSGKADTCSQJJKSARTGGVKMTCTMHSVDWFQQHFRSJPNSF
AJAHYJTJJJPDVSDCPRAVEFHQJVGJAJCAHPGAAYQDVJWCAEAVWPPHHADQAJHYCTJNGADDVTSRCPCJKNSHPVJQDVKPHKAGJAEEN
SEYNJAVKSYSFGJYRCEJAGATPDRDDJJKENGRGAVNVJGWJJJSDVGVVJHAATGQNCFSDEKAESKAHJJNESYVVJPGATJKNDCJAHNSTC
JECKSKDDRECCTAKJNRSCGTRJGHJFGRTKAVMEFAAAJNYYGJEJDPJPHJETCJDAPSTAPTSJSTHVWYTVRKTEDJHKMENYRJSMAREQE
SPJTCGDGJCHNNJQAECSNARNVMJQJYPJRKJJRVSQVHAFRRTAVJSSSFGGDGTVGFJVJHPDNSJVHGSMJQJVERJVAFYVGPKEAGFNJV
VGASHDGPVVKFJKJEESSJRFDKRRHPEPDPSAKAGPPNKFSJVKJKDVAAJGQHNRVVHRRHMEEKJVGKVJJFETKGSNTRAJTTJMFAJARQA
VAVAKHPVJDMAJVFMPGGYATAJSGJSJAMDMQTHCFKMHGEGTESRJTAVYT
Length: 927, mw: 101916, pI: 7.57666, absorbance: 0.793888, fitness: 0.00119598
Population average: 1172.79, standard deviation: 675.39

Analiza 4
population size:6000, number of iterations: 100
JVSFDGJDARNGADEJJAVVPYTKVJAASCPADKASWJTJVFKDQAAFJJQKRRVKJDYGTGYGVDTADADJKCVQAKAJEFVVESVEAYMHKTJSV
FTJAGAVAKSRFRDPVKGFQJKHGTCKQSYPCAEEJJHSSYDTAFJVVJSDJRYSAGHJTEKAVVJVRRQAAEVJVJTRGGVGJQNNTJEAJJEEGA
VVJAESGJESMEHPTKJKJCSAJJJARHPVJRATSWGEDVANJYVJDGVRJFTNRAJHQGRKAHAKDJNNJYVNNQYJJAVCJSYPGGQKVCGNQCV
AAEVKRKJJTNAVGSFRJSFHPFJPFJEVGSNSJCGAGRQGRJJVEFJPTJJVDEJNYKEPACJGKGYKHHJJRAETRKHJKRJQECJJDGKAFSTV
AEPYMJDTGKVMGJTGPHJGGNVVMDHJAHJJFJPRMTASNRKJCMEEVSEMWPCMJFSHJKJQGJJVNETNVJGTTVWKMJMEGNKQGGJGJCGMV
DFJDGRJJANVVYDGARAAFSVSCNEDANJANGVENPYATHVRKJJAJNJTEHDDAHJVVKJAPVCATKJEJASNJAQSGJMJKFTVTTFHJKJWPT
SNTJKTPHHJJJAJAFTJRVDHQJWNRCEGAQFSJEEJKKKRJASJPRYKHCKGSJJVQJAVSJAKQJTEQSYVRJTVJJAYGJYFTJJAVEDGTNJ
JJVCGQPEKGJTFHNYKMDSJTGJJSTKPJAWMJRPKVKEQAHHJAJYNRQGJMFGJJSGPEDEJSPGTDNJMMKVQMJQKYJMEJYNSNAJKAJAV
AJRFHCDMDNNNCSRTGEJEEAJDDPNSRCGJAVVEGSDDVVRGJPAAJHGJJRNKJMVGAYEYPNJVCJHHNFHJDQDDVPEFJHTEHAJMDDJMV
AGFTYJNYJSKSQGPTVJTEGKDGDKVJYYAJJPGDREQEVKAKSFSAGJ
Length: 923, mw: 101930, pI: 7.57666, absorbance: 0.794563, fitness: 0.00252391
Population average: 2861.73, standard deviation: 1632.74

Przykład 5 Crambina

mw: 4736.46, pI: 6.024, absorbance: 0.6925

population size:1000, number of iterations: 500
crossing over rate:0.75, elit rate: 0.05
mutation rate: 0.025, insertion rate: 0.07, deletion rate: 0.07

Analiza 1
EFYFJGHGENCCFHDPJCFPACCHTFFHQJJGAYNJNJCPJ
Length: 41, mw: 4736.47, pI: 6.02393, absorbance: 0.692499, fitness: 7.69593e-05
Population average: 744.801, standard deviation: 429.981

Analiza 2
NCSCCJHSDHSCNQFYJJTJMPPJHJCDAAGQYJHJCJJGEAS
Length: 43, mw: 4736.48, pI: 6.02393, absorbance: 0.692497, fitness: 8.65238e-05
Population average: 800.603, standard deviation: 454.229

Analiza 3
FJNACCSSGJPNGDCTCCGRSJEFGJJJASKVYNATTCPJYJSQT
Length: 45, mw: 4736.46, pI: 6.02393, absorbance: 0.692501, fitness: 7.45891e-05
Population average: 938.244, standard deviation: 536.875

Analiza 4
DHCCJCDEHYJKNCJGJJJRSJSHJFGJDYJGNHNSCJECGJ
Length: 42, mw: 4736.47, pI: 6.03662, absorbance: 0.692499, fitness: 0.0126211
Population average: 731.749, standard deviation: 419.086

Przykład 6 ADM

mw: 2445.86, pI: 11.649, absorbance: 4.6528

Analiza 1
population size:500, number of iterations: 500
crossing over rate:0.75, elit rate: 0.05
mutation rate: 0.01, insertion rate: 0.03, deletion rate: 0.03

SKWMMVQQPPARGTGAAASVVSW
Length: 23, mw: 2445.84, pI: 11.6489, absorbance: 4.6528, fitness: 9.90337e-05
Population average: 257.884, standard deviation: 154.247

Analiza 2
population size:1000, number of iterations: 200
crossing over rate:0.75, elit rate: 0.02
mutation rate: 0.05, insertion rate: 0.05, deletion rate: 0.05
NAGVATKPTSJFRENWAJWKR
Length: 21, mw: 2445.81, pI: 11.6489, absorbance: 4.65286, fitness: 0.000348388
Population average: 2123.38, standard deviation: 1233.11

Analiza 3
DSGJWJNJRKSJWRKAFNAJG
Length: 21, mw: 2445.85, pI: 11.6489, absorbance: 4.65277, fitness: 0.000147889
Population average: 2331.16, standard deviation: 1353.54

Analiza 4
HSWQARPKMRPSFFKJNWE
Length: 19, mw: 2445.83, pI: 11.6489, absorbance: 4.65282, fitness: 0.000156972
Population average: 2081.51, standard deviation: 1241.32

Analiza 5
ASMVNRTJSWSNKPSJMHSJW
Length: 21, mw: 2445.84, pI: 11.6489, absorbance: 4.6528, fitness: 9.94698e-05
Population average: 2011.34, standard deviation: 1183.94

Analiza 6
WPQNGAAQQNRJWPVGGJVJJK
Length: 22, mw: 2445.85, pI: 11.6489, absorbance: 4.65277, fitness: 0.000147889
Population average: 1982.21, standard deviation: 1145.13

Analiza 7 (z wykorzystaniem operatora cross2)
population size:1000, number of iterations: 200
crossing over rate:0.75, elit rate: 0.02
mutation rate: 0.03, insertion rate: 0.03, deletion rate: 0.03

FJJQWPJAAGGFQRENKKRW
Length: 20, mw: 2445.86, pI: 11.6489, absorbance: 4.65277, fitness: 0.00017032
Population average: 1658.32, standard deviation: 962.717

Analiza 8 (z wykorzystaniem operatora cross2)
mutation rate: 0.05, insertion rate: 0.05, deletion rate: 0.05

DJHARGJTKAJJWRQKGAGAPW
Length: 22, mw: 2445.86, pI: 11.6489, absorbance: 4.65277, fitness: 0.00017435
Population average: 2547.95, standard deviation: 1411.43

Analiza 9 (z wykorzystaniem operatora cross2)
JANNHRAKNJJWJTFFWSJT
Length: 20, mw: 2445.85, pI: 11.6489, absorbance: 4.65278, fitness: 0.000146943
Population average: 2746.35, standard deviation: 1594.7

Analiza 10 (z wykorzystaniem operatora cross2)
FJJQWPJAAGGFQRENKKRW
Length: 20, mw: 2445.86, pI: 11.6489, absorbance: 4.65277, fitness: 0.00017032

Analiza 11 (z wykorzystaniem operatora cross2)
WAWKKGGMNGJEJGNHRRHVV
Length: 21, mw: 2445.83, pI: 11.6489, absorbance: 4.65281, fitness: 0.000144028
Population average: 1376.4, standard deviation: 823.131

LITERATURA

[1] Appel R. D., Bairoch A., Hochstrasser D. F. (1994) A new generation of information retrieval tools for biologists: The example of the ExPASy WWW server. Trends Biochem. Sci. 19, 258-260.

[2] Brumby S. P., Harvey N. R., Perkins S., Porter R. B., Szymanski J. J., Theiler J., Bloch J. J. (2000) A genetic algorithm for combining new and existing image processing tools for multispectral imagery. W: Algorithms for Multispectral, Hyperspectral, and Ultraspectral Imagery VI. Proceedings of SPIE 4049, 480-490.

[3] Goldberg D. E. (1990) Real-coded genetic algorithms, virtual alphabets and blocking. IlliGAL Report No. 90001.

[4] Bacardit J., Stout M., Hirst J. D., Sastry K., Llora X., Natalio Krasnogor N. (2007) Automated Alphabet Reduction Method with Evolutionary Algorithms for Protein Structure Prediction. IlliGAL Report No. 2007015.

[5] Holland J. H. (1975) Adaptation in natural and artifical systems. University of Michigan Press. Ann Arbor.

[6] Michalewicz Z. (1999) Algorytmy genetyczne + struktury danych = programy ewolucyjne. Wydanictwo Naukowo-Techniczne, Waszawa.

[7] Goldberg D. E. (2003) Algorytmy genetyczne i ich zastosowania. Wydanictwo Naukowo-Techniczne, Waszawa.

[8] Santarelli S., Yu T. L., Goldberg D. E., Altshuler E., O'Donnell T., Southall H., Mailloux R. (2005) Military Antenna Design Using Simple and Competent Genetic Algorithms. IlliGAL Report No. 2005013.

[9] Lima C. F., Sastry K., Goldberg D. E., Lobo F. G. (2005) Combining Competent Crossover and Mutation Operators: A Probabilistic Model Building Approach. IlliGAL Report No. 2005002.

[10] Oyama, A., Obayashi, S., Nakahashi, K. (1999) Wing design using real-coded adaptive range genetic algorithm. Systems, Man, and Cybernetics, IEEE SMC apos, 99 Conference Proceedings IEEE International Conference 4, 475-480.

[11] Kumsawat P., Attkitmongcol K., Srikaew A, Sujitjorn S. (2004) Wavelet-Based Image Watermarking Using the Genetic Algorithm. W: Knowledge-Based Intelligent Information and

Engineering Systems, 643-649, Springer Berlin/Heidelberg.

[12] Carroll D. L. (1996) Chemical laser modeling with genetic algorithms. AIAA J. 34(2), 338-346.

[13] Schultze-Kremer S. (2000) Genetic algorithms and protein folding. W: Methods in Molecular Biology 143 - Protein Structure Prediction: methods and protocols, 175-222.

[14] Gręboszl J. (1999) Symfonia C++. Programowanie w języku C++ orientowane obiektowo. Oficyna Kallimach, wydanie czwarte uzupełnione, Kraków.

[15] Liberty J., Pancewicz M. (2002) C++ dla każdego. Wydawnictwo Helion, Warszawa.

[16] Dokumentacja online biblioteki QT4 http://doc.trolltech.com/

[17] Matsumoto M. i Nishimura T. (1998) Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator", ACM Transactions on Modeling and Computer Simulation 8(1), 3-30.

[18] Hoare C. A. R. (1962) Quicksort. Computer J. 5(1), 10-15.

[20] http://www.expasy.ch/tools/findmod/findmod_masses.html

[21] Nelson D. L., Cox M. M. (2000) Lehninger Principles of Biochemistry. Worth Publishers.

[22] Kuczmarski K. (2004) Kurs C++. Od zera do gier kodera. Megaturtotial. Avocado Software.

[23] Henikoff S., Henikoff J. G. (1992) Amino acid substitution matrices from protein blocks. PNAS 89(22), 10915-9.

[24] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P. (2007) Numerical Recipes 3rd Edition: The Art of Scientific Computing. Cambridge University Press, New York.