УНИВЕРЗИТЕТ ,,Гоце Делчев“ – ШТИП

ФАКУЛТЕТ ЗА ИНФОРМАТИКА

Катедра за компјутерски технологии и интелигентни системи

ШТИП

м-р Доне Стојанов

АЛГОРИТМИ ЗА ПОРАМНУВАЊЕ И ПРЕБАРУВАЊЕ НА ДНК СЕКВЕНЦИ

- ДОКТОРСКИ ТРУД -

Штип, 2015

УНИВЕРЗИТЕТ ,,Гоце Делчев“ – ШТИП
ФАКУЛТЕТ ЗА ИНФОРМАТИКА
Катедра за компјутерски технологии и интелигентни системи
ШТИП

м-р Доне Стојанов

АЛГОРИТМИ ЗА ПОРАМНУВАЊЕ И ПРЕБАРУВАЊЕ НА ДНК СЕКВЕНЦИ

- ДОКТОРСКИ ТРУД -

Штип, 2015

Интерен ментор: проф. д-р Цвета Мартиновска Банде
Факултет за информатика
Универзитет ,,Гоце Делчев” – Штип

Екстерен ментор: проф. д-р Ана Мадевска Богданова
Факултет за информатички науки и компјутерско инженерство
Универзитет ,,Св. Кирил и Методиј” – Скопје

Научно поле:
Информатика
Научна област:
Биоинформатика

Трудови од научната област, објавени во меѓународни списанија со фактор на влијание:

- Stojanov, D., Koceski, S., Mileva, A., Koceska, N., & Bande, C. M. (2014). Towards computational improvement of DNA database indexing and short DNA query searching. *Biotechnology & Biotechnological Equipment*, *28*(5), 958-967. (IF = 0,379 за 2013 год)

- Stojanov, D., Koceski, S., & Mileva, A. (2013). FLAG: Fast Local Alignment Generating Methodology. *Romanian Biotechnological Letters*, *18*(1), 7881-7888. (IF = 0,351 за 2013 год)

Трудови од научната област, објавени во меѓународни списанија без фактор на влијание:

- Stojanov, D., & Martinovska, C. (2014). Improved alignment of homologous DNA sequences. Annals of West University of Timişoara, ser. Biology, 16(2), 97-106.

- Stojanov, D., Mileva, A., & Koceski, S. (2012). A new, space-efficient local pairwise alignment methodology. Advanced Studies in Biology, 4(2), 85-93.

КРАТОК ИЗВАДОК

Предмет на анализа на докторскиот труд се два новопредложени алгоритми за локално порамнување на ДНК секвенци и еден новопредложен алгоритам за ефикасно индексирање и пребарување на масивна ДНК база на секвенци. Со примена на првопредложениот алгоритам за парово порамнување на ДНК секвенци се намалува временскиот и меморискиот трошок за добивање на решение во споредба со алгоритмот на Smith и Waterman, додека со примена на второпредложениот алгоритам се зголемува точноста на порамнување, односно предложениот алгоритам генерира решение во кое се вклучени и базните совпаѓања кои алгоритмот на Smith и Waterman и хевристичните алгоритми (MUMmer, Avid, Glass, Lalign) ги отфрлаат поради намалувањето на резултатот на порамнување кај Smith-Waterman или поради нивна класификација како помалку значајни совпаѓања од страна на хевристичните алгоритми. За да се овозможи ефикасно индексирање на ДНК база на секвенци, предложена е нова формула, со чија примена процесот на индексирање на содржината од ДНК базата на секвенци повеќекратно се забрзува во споредба со современите хеш-базирани пристапи за иста намена, како SSAHA и алгоритмот на Reneker и Shyu. Во однос на претходните алгоритми се намалува и мемориската зафатнина на индексираната податочна структура, како и времето на пребарување на ДНК базата на секвенци по клучен ДНК прашалник. Со примена на предложениот алгоритам за пребарување се овозможува детекција на секое совпаѓање на ДНК прашалник во ДНК база на секвенци, независно од неговата почетна положба во индексираните секвенци, што не претставува случај кај SSAHA и алгоритмот на Reneker и Shyu.

Клучни зборови: *Порамнување, Индексирање - Пребарување, ДНК секвенци, ДНК база на податоци, оптимизација*

ABSTRACT

The PhD thesis analyses two new algorithms for local alignment of DNA sequences and one new algorithm for effective indexing - searching of a massive DNA database. By applying the first proposed algorithm for pairwise alignment of DNA sequences, the time and memory complexity in comparison to Smith-Waterman algorithm are improved, while by applying the second proposed algorithm the alignment accuracy can be increased, due to the fact that all hits that are rejected by the algorithm of Smith and Waterman and the heuristic algorithms (MUMmer, Avid, Glass, Lalign) are included in the solution generated by the proposed algorithm. The hits that are rejected by Smith-Waterman are hits that would decrease the alignment score, if they are included in the solution or hits witch are classified as less significant hits by the heuristic algorithms. A new formula for effective DNA database indexing is also proposed. By applying the proposed formula the process of DNA database indexing is multiply increased in comparison to the contemporary hash-based solutions for the same purpose, such as: SSAHA and the algorithm of Reneker and Shyu. In comparison to the previous algorithms, the memory storage of the indexed data structure, as well as the time span for searching the DNA database against DNA query, are decreased. By applying the proposed DNA database searching algorithm, each query hit in the database can be detected, regardless it’s starting position in the indexed sequences, what is not case with SSAHA and the algorithm of Reneker and Shyu.

Key words: *Alignment, Indexing - Searching, DNA sequences, DNA database, optimization*

СОДРЖИНА

ВОВЕД

Времето на извршување и меморијата се фактори кои во реални услови ја ограничуваат компјутерската примена на алгоритмите за порамнување и пребарување на генетски секвенци. Квадратната временска комплексност ја ограничува примената на алгоритмите базирани на динамичко програмирање (*Needleman-Wunsch*, *Wagner-Fischer*, *Gotoh*, *Smith-Waterman*, *Sellers*, *Waterman-Eggert*, *Ficket* и др.) само на кратки секвенци. Она што е карактеристично за алгоритмите базирани на динамичко програмирање е тоа што истите секогаш генерираат оптимално решение од аспект на максимизација на линеарна или нелинеарна математичка функција позната како резултат на порамнување. Со примена на овој пристап се отфрлаат совпаѓањата со чие вклучување би се намалила конечната вредност на резултатот на порамнување. Таквите совпаѓања најчесто се кратки совпаѓања кои се значајно оддалечени во однос на ДНК фрагментите со висока густина на совпаѓање. Истите не се вклучени во рамки на порамнувањето од причина која има повеќе математичка оправданост одошто биолошка, а тоа е намалување на конечната вредност на резултатот на порамнување. Од биолошки аспект тоа е неповолно, бидејќи така се губи можноста за идентификација на далечна хомологија (далечна структурна, функционална и еволуциска поврзаност) помеѓу анализираните генетски секвенци. Ратата на отфрлање на помалку значајните совпаѓања е далеку поголема кај хевристичните алгоритми (*FLASH, YASS, LAlign, PatternHunter* и др.) на што се должи повеќекратното забрзување на процесот на порамнување, што во основа ги прави овие алгоритми применливи како за прокариотски, така и за еукариотски секвенци (комплетни еукариотски хромозоми и геноми). Во принцип хевристичните алгоритми жртвуваат дел од точноста на решението за да се добие решение за пократок временски интервал. Намалувањето на просторот каде се бара решение, односно отфрлувањето на помалку значајните совпаѓања ги прави овие алгоритми неприменливи за детална и сеопфатна анализа на хомологија помеѓу две генетски секвенци.

Во првиот дел од истражувањето во рамки на овој труд се разгледуваат два нови алгоритми со кои се подобрува временската и мемориската комплексност и се зголемува точноста на порамнување во однос на алгоритмите базирани на динамичко програмирање.

При порамнување на ДНК секвенци со висок процент на базна идентичност, предложениот алгоритам за брзо и мемориски ефикасно порамнување на ДНК секвенци, повеќекратно го намалува бројот на извршени споредби во споредба со алгоритмите базирани на динамичко програмирање. Намалувањето на бројот на извршени споредби се должи на промената на редоследот на анализа на подпроблемските простори каде се бара решение. Алгоритмот работи на принцип на поместување на пократка секвенца долж подолга секвенца, така што во првата фаза од извршувањето решението се бара во рамки на преклопувачки прозорци со должина еднаква на должината на пократката секвенца m, по што во зависност од вредноста на локалниот максимум за резултат на порамнување се ограничуваат левите и десни поместувања на пократката секвенца долж подолгата со кои се образуваат преклопувачки прозорци со должина помала од должината на пократката секвенца. Воведен е пристап за мемориска репрезентација на секое совпаѓање

со податочна тројка која ги бележи почетните положби на совпаѓање во рамки на секвенците и должината на совпаѓање. Предложениот алгоритам не е само временски, туку е и мемориски поефикасен во споредба со просторно линеарните алгоритми за парово порамнување на ДНК секвенци како: *Hirschberg*, *Myers u Miller*, *Huang et al*., и др.

Вториот предложен алгоритам е алгоритам за празнинско порамнување на две ДНК секвенци, со чија примена се зголемува точноста на порамнување. Предложениот алгоритам во првата фаза од извршување ги утврдува сите конзистентни совпаѓања помеѓу две секвенци, од кои се добива порамнување во втората фаза од извршувањето. Множеството на сите конзистентни совпаѓања се утврдува со минимален број на базни споредби со примена на пристап за споредба на еднаквост на зборови до прво пронајдено несовпаѓање. За секвенци со висок процент на базна идентичност, вкупниот број на споредби кои се извршуваат за да се утврди множество на сите конзистентни совпаѓања е помал од $n \times m$, каде n и m се должини на секвенците кои се порамнуваат. По утврдување на множество на конзистентни совпаѓања, порамнувањето се добива со празнинско порамнување на фрагментите кои ги одделуваат паровите последователни совпаѓања. Во зависност од фреквенциите на појавување на субституциските нуклеотидни парови XY, алгоритмот во секој чекор избира да додаде празнина помеѓу субституциски пар со минимална фреквенција на појавување. Како и претходниот алгоритам, така и овој алгоритам користи пристап за мемориско бележење на почетните положби на совпаѓањата и должините на истите во рамки на податочен вектор, со што се гарантира минимална мемориска побарувачка во фаза на извршување.

Предноста на овој алгоритам во споредба со хевристичните алгоритми и алгоритмите базирани на динамичко програмирање е тоа што овој алгоритам не ги отфрла помалку значајните совпаѓања, како и оддалечените и кратки совпаѓања. Ваквиот пристап го прави овој алгоритам применлив за детална анализа на блиска и далечна хомологија помеѓу ДНК секвенци, не делејќи ги конзистентните совпаѓања на помалку значајни и повеќе значајни совпаѓања, односно на секое совпаѓање му се задава подеднаква важност. Со примена на пристапот за додавање на празнина (празнини) помеѓу субституциски парови со најмала фреквенција, предложениот алгоритам води сметка за тенденцијата за конзервираност на субституциски парови по секвенца врз основа на тезата дека веројатноста да се избрише елемент во средина на субституциски пар XY со помала фрекфенција на појавување е поголема од веројатноста елемент да се избрише во средина на субституциски пар XY со повисока фреквенција на појавување.

На крај од истражувањето предложен е нов алгоритам за хеш-базирано индексирање и пребарување на ДНК база на податоци со чија примена: повеќекратно се забрзува процесот на индексирање на содржина од ДНК база на податоци, се оптимизира мемориската побарувачка за зачувување на индексираната ДНК содржина, се забрзува процесот на пребарување по клучен ДНК прашалник и пред сè се зголемува бројот на детектирани совпаѓања во базата на податоци во споредба со хеш-базираните современици на предложениот алгоритам како SSAHA и методот на Reneker и Shyu. Забрзувањето на процесот на индексирање е резултат на примена на нова хеш-базирана формула, со чија примена вредноста на пресликување за секој

нареден збор во должина од k нуклеотиди се пресметува од вредноста на пресликување за претходниот збор, со што во основа се минимизира бројот на извршени операции по пресликан ДНК збор. Предложената формула за брзо индексирање се темели на фактот дека секои два последователни и преклопувачки ДНК зборови во должина од k нуклеотиди имаат $k-1$ заеднички нуклеотиди, со што во споредба со постојните хеш-базирани пристапи за индексирање на ДНК база на податоци се добива забрзување за фактор k, каде k е број на нуклеотиди по пресликан ДНК збор. За разлика од SSAHA, каде во хеш-табела се чуваат индекси за секој можен збор во должина од k нуклеотиди, независно дали зборот за кој се чува индекс постои во базата на податоци или не, предложениот алгоритам ги индексира само зборовите кои постојат во базата на податоци, со што се заштедува на меморија без никакви импликации врз точноста на пребарување. Предложениот пристап е особено ефикасен во случаите кога се индексира мала ДНК база на податоци. Подобрувањето на перформансите на пребарување во споредба со SSAHA, каде се користи хеш-табела за меморирање на ДНК индексите и методот на Reneker и Shyu, каде се користи индексирана датотека, во прв план се должи на употребата на сортиран речник за меморирање на ДНК индексите. Употребата на сортиран речник овозможува идентификација сите совпаѓања на ДНК прашалник во база на податоци без да се пребаруваат сите записи. Ова е можно бидејќи записите во сортираниот речник се подредени по клуч (вредност на пресликување). Од биолошки аспект, главна предност на предложениот алгоритам во споредба со SSAHA и методот на Reneker и Shyu е во тоа што со примена на алгоритмот се детектираат совпаѓања на ДНК прашалник во базата на податоци кои претходните два пристапи во никој случај не можат да ги детектират, бидејќи со примена на суфикс-базирано пребарување можат да се утврдат само совпаѓања кои се наоѓаат на почетни положби $p > k - |q|$ во индексираните ДНК секвенци, каде k е број на нуклеотиди по пресликан ДНК збор и $|q|$ е должина на ДНК прашалник. Во основа станува збор за совпаѓања кои се локализирани на почетоците на индексираните секвенци за чија идентификација предложениот алгоритам, освен суфикс-базирано пребарување, имплементира и префикс-базирано пребарување. Во зависност од биолошко-функционалната анотираност на ДНК прашалникот кој се пребарува, со примена на предложениот алгоритам можат да се детектираат недетектирани: промотор секвенци, интрони, егзони и теломери во рамки на хромозоми, во случаите кога хромозомите се пребаруваат во инверзен редослед.

Истражувањето е организирано во осум поглавја. Во првото поглавје „*Поими од биоинформатиката*“ приложен е краток осврт на основните концепти од молекуларна биологија (хемиска структура на ДНК, класификација на ДНК, превод/транскрипција на ген).

Историска и компаративна анализа на постојните алгоритми за порамнување и пребарување на ДНК секвенци е приложена во второто поглавје „*Истражувања во областа*“, во кое е апстрахиран севкупниот научен напредок во областа, започнувајќи од алгоритмот на Needleman-Wunsch за глобално порамнување на две ДНК секвенци, сè до современите хеш-базирани пристапи за брзо и просторно ефикасно индексирање/пребарување на ДНК база на секвенци по ДНК прашалник, како: SSAHA и алгоритмот на Reneker и Shyu.

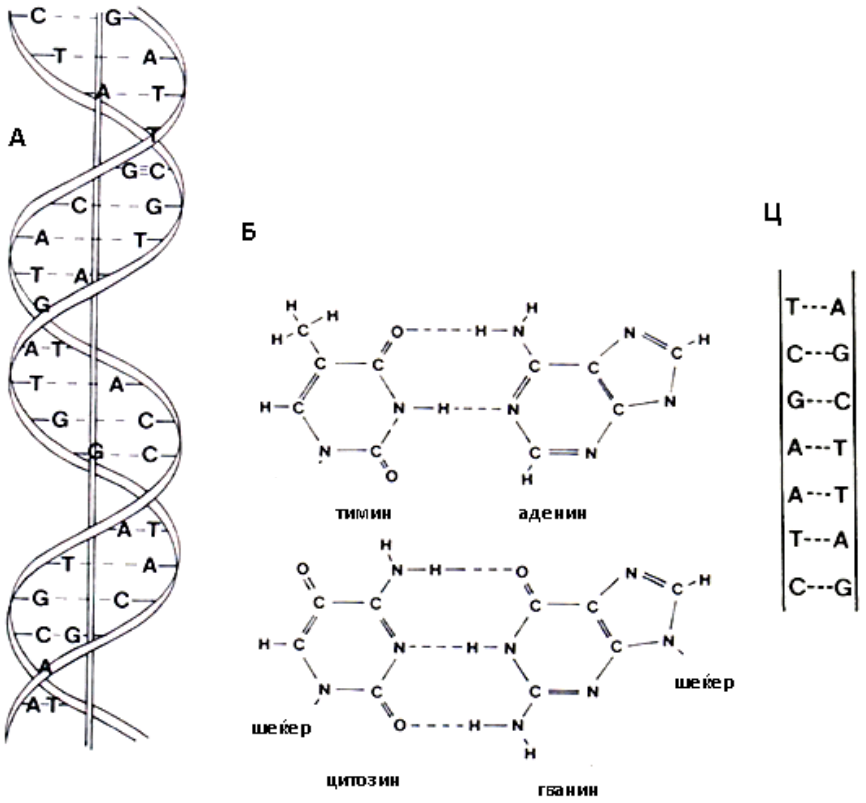
Предлог алгоритмот за брзо и мемориски ефикасно порамнување на ДНК секвенци, како и предложените алгоритми за празнинско порамнување врз основа на моделите за последователно додавање на празнини по совпаѓања и утврдување на положбите на базни мутации во функција од фреквенциите на базни супституции во неусогласените ДНК фрагменти, се опишани во поглавјето „*Опис на предложените алгоритми*“. Поглавјето изобилува со примери и дискусии на случаи кои ја објаснуваат концепциската поставеност на предложените алгоритми и предностите на предложените алгоритми во однос на постојните/сродни алгоритми. На крајот од ова поглавје елаборирана е алгоритмот за индексирање/пребарување на ДНК база на податоци по ДНК прашалник, со чија примена се забрзува процесот на индексирање на ДНК база на податоци за фактор k (k е должина на зборови кои се индексираат) и се подобрува точноста на детекција на совпаѓања во ДНК базата на податоци.

Во поглавјето „*Софтверска библиотека*“ даден е опис на имплементацијата на предложените алгоритми во C++/C#. Развиениот софтвер е применет на податоци за реални ДНК секвенци (Albumin ДНК од различни видови и целосни E. Coli хромозоми), кои апликацијата ги презема од Европската Нуклеотидна Архива. Добиените резултати укажуваат на тоа дека предложените алгоритми овозможуваат побрзо и мемориски поефикасно порамнување/пребарување на ДНК секвенци во споредба со алгоритмите базирани на динамичко програмирање и современите хеш-базирани пристапи како: SSAHA и алгоритмот на Reneker и Shyu, но и поточно порамнување/пребарување. Анализа на добиените резултати е направена во поглавјето „*Експериментални резултати*“.

Особеностите на предложените алгоритми и нивните предности во однос на постојните алгоритми се сумирани во табела која е приложена во поглавјето „*Дискусија*“, додека во поглавјето „Додаток“ е приложена комплетна листа на псевдо-кодови.

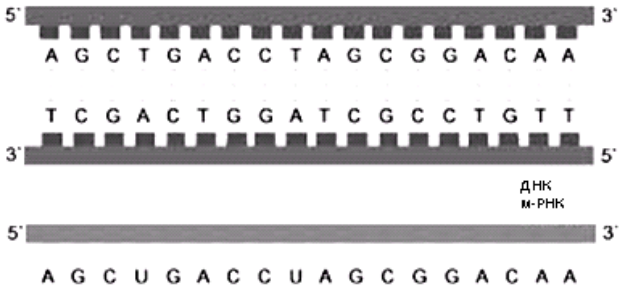
1. ПОИМИ ОД БИОИНФОРМАТИКАТА

Деоксирибонуклеинската киселина е двојно-нитна, издолжена и содржински комплементарна биолошка молекула, Сл. 1.1 (А). ДНК молекулата содржи наследен генетски материјал. Секоја нитка е низа од четири основни нуклеотиди: A (*аденин*), G (*гванин*), C (*цитозин*) и T (*тимин*), Сл. 1.1 (А, Ц). Фосфодиестер врските овозможуваат сериско поврзување на нуклеотидите по нитка. Важи правилото дека на секој нуклеотид од една нитка соодветствува нему комплементарен нуклеотид од спротивната нитка, Сл. 1.1 (А, Б, Ц). Аденин комплементарен нуклеотид е тиминот и обратно, додека цитозин комплементарен нуклеотид е гванинот и обратно, Сл. 1.1 (Б). Базните поврзувања се овозможени со двојни, односно тројни водородни врски, формирајќи стабилна двојно-нитна структура со дијаметар од 20 ангстроми (1A=10^{-10} m), која има природен потенцијал за обнова во случај на оштета. Секој нуклеотид е составен од: *деоксирибоза шеќер*, *фосфатна група* и *азотна база*. Структурата на азотната база е различна за секој нуклеотид, додека хемиската структурата на фосфатната група и шеќерот е иста кај сите четири нуклеотиди.



Слика 1.1. Структура на ДНК молекула
Figure 1.1. Structure of DNA molecule

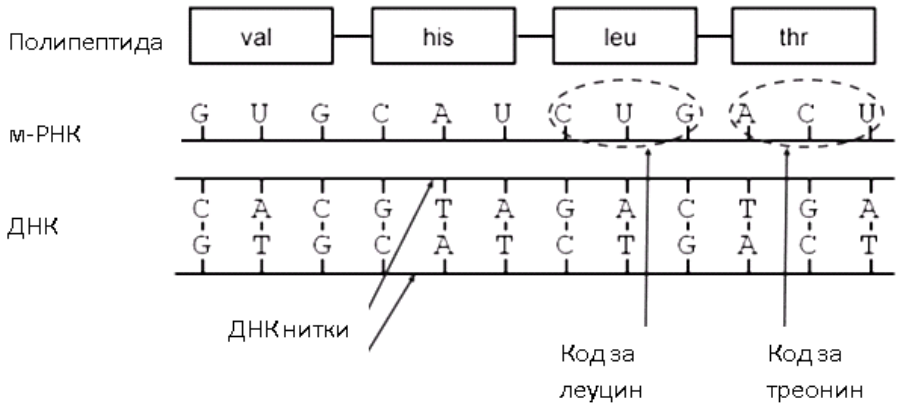
Информацијата кодирана во ДНК молекулата се дели на *кодирачка* и *некодирачка*. Најголем процент од некодирачката ДНК припаѓа на кратките тандемски повторувања, кои во основа се повеќекратни повторувања на краток ДНК фрагмент како, на пример, CA фрагментот. Честотата на повторување се зема како главен индикатор за утврдување на генетскиот профил. Кодирачката ДНК содржина ја детерминираат *гените – протеин кодирачки фрагменти*, чии почетоци и краеви ги определуваат *промотор* и *терминатор* секвенци. Имајќи ја предвид варијабилноста на промотор секвенците, почетокот на генот го определува *промотор консензус* секвенца, која се наоѓа на 10 или 35 базни положби на лево во однос на транскрипцискиот почеток. Со посредство на ензимот РНК полимераза, генот се транскрибира во *гласник рибонуклеинска киселина* (*м-РНК*), Сл. 1.2.



Слика 1.2. Транскрипција на ген
Figure 1.2. Transcription of gene

Постојат три видови на РНК: *рибозомска РНК* (*р-РНК*), *транспортна РНК* (*т-РНК*) и *гласник РНК* (*м-РНК*). Ризбозомската РНК е структурна компонента на рибозомскиот комплекс и истата учествува во синтезата на протеини. Транспортната РНК е кратка молекула со должина од 70 до 90 базни парови. Истата учествува во преносот на амино-киселините во фазата на синтеза на протеин. Информацијата кодирана во гласник рибонуклеинската киселина служи како шаблон за синтеза на конкретен протеин. Протеините се вклучени во повеќето биолошки процеси и истите извршуваат најразлични функции. Протеините можат да дејствуваат како: забрзувачи на биохемиските процеси, преносници на кислород и железо и одржувачи на структурата на клетката. Структурата на информациската РНК е комплементарна во однос на структурата на генот од ДНК нитката кој е предмет на траскрипција, каде важи правилото дека аденин комплементарен рибонуклеотид е Урацилот (U), Сл. 1.2. Кај РНК нуклеотидите, шеќерниот јаглероден атом 2' е поврзан со хидроксилна – OH група, наместо само со водород како кај деоксирибозата.

Низата од три последователни нуклеотиди се нарекува *кодон*. Секој кодон соодветствува на една од дваесетте протеински амино-киселини. Започнувајќи од *старт кодонот*, завршувајќи со *стоп кодонот*, секој кодон се преведува во конкретна амино-киселина, Сл. 1.3, според *универзалниот генетски код*, Сл. 1.4. Кодонот го препознава конкретна т-РНК молекула, која ја носи и додава соодветната амино-киселина на крајот од растечката протеинска полипептида. Битно е да се напомене дека *примарниот м-РНК транскрипт* содржи *интрони* и *егзони*. Интроните се некодирачки РНК потсеквенци, додека егзоните се кодирачки РНК потсеквенци. Со отстранување на интроните се добива *конечен РНК транскрипт*, кој е предмет на превод.



Слика 1.3. Превод на м-РНК
Figure 1.3. Translation of m-RNA

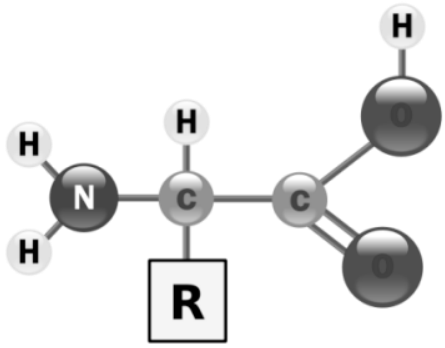
	U	C	A	G
U	UUU Phe UUC UUA Leu UUG	UCU UCC UCA Ser UCG	UAU Tyr UAC UAA Stop UAG	UGU Cys UGC UGA Stop UGG Trp
C	CUU CUC CUA Leu CUG	CCU CCC CCA Pro CCG	CAU His CAC CAA Gln CAG	CGU CGC CGA Arg CGG
A	AUU AUC Ile AUA AUG Met	ACU ACC ACA Thr ACG	AAU Asn AAC AAA Lys AAG	AGU Ser AGC AGA Arg AGG
G	GUU GUC GUA Val GUG	GCU GCC GCA Ala GCG	GAU Asp GAC GAA Glu GAG	GGU GGC GGA Gly GGG

Слика 1.4. Универзален генетски код
Figure 1.4. Universal genetic code

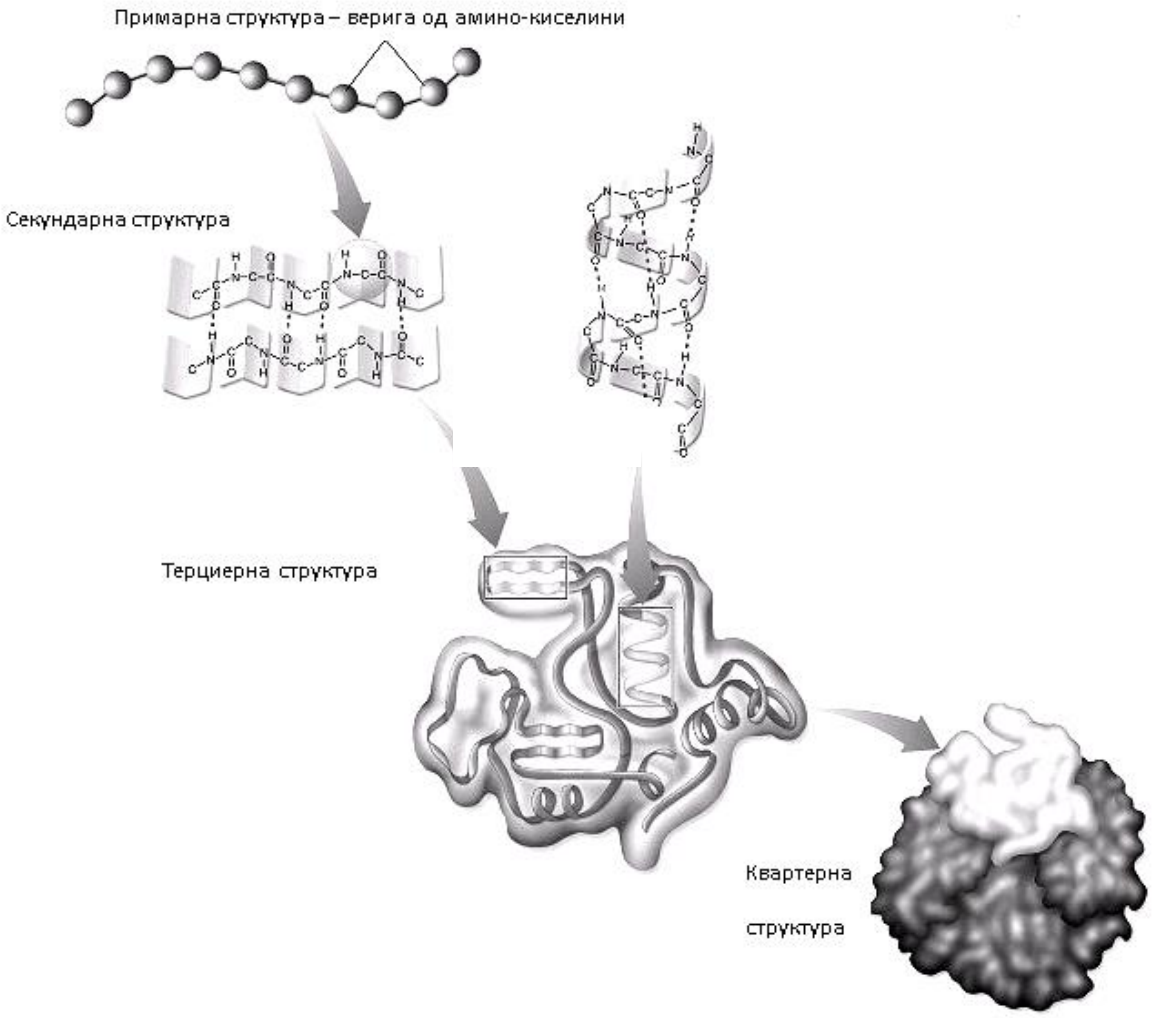
Досегашната фенотипска анализа укажува на промени во структурата на гените. Последицата од промената на информацијата кодирана во ген е синтеза на изменет протеин во фазата на превод. Синтезата на изменет протеин може да биде причина за различни функционални и структурни нарушувања. Драстични генетски промени се случуваат во услови на радијација и изложеност на надворешни хемиски фактори.

Базните промени во ниво на ген се познати како мутации. Можни се следниве мутации: *замена на еден нуклеотид со друг*, *бришење на нуклеотид* и *додавање на нуклеотид*. Во најлош случај, мутацијата: нетерминален триплет во стоп кодон ќе предизвика синтеза на скратен и најверојатно нефункционален протеин, кој е потенцијална причина за функционално нарушување.

Резултатот од преводот на информацијата содржана во м-РНК е протеин. Протеините се линеарни вериги кои се составени од амино-киселини. Во градбата на протеините се вклучени 20 различни амино-киселини, со различни биохемиски својства. Секоја амино-киселина е изградена од: централен јаглероден атом, кој е поврзан со амино група, карбоксилна група и остаток, чија структура е единствена за секоја амино-киселина, Сл. 1.5. Поврзувањето во верига е овозможено со формирање на пептидни врски помеѓу амино и карбоксилната група. Фрагменти од протеините се извиткуваат во регуларни геометриски облици, како што се *алфа хеликсите* и *бета рамнините*, Сл. 1.6. Просторното поврзување на секундарните извиткувања ја детерминира терциерната структура на протеините. Од друга страна пак, просторната структура на протеинот е во тесна релација со неговата функционалност. Протеини со слична просторна структура имаат и слична функционалност.



Слика 1.5. Структура на амино-киселина
Figure 1.5. Structure of amino-acid



Слика 1.6. Четири нивоа на организација на структурата кај протеините
Figure 1.6. Four levels of organization of proteins' structure

Порамнувањето на генетски секвенци (*ДНК*, *РНК* и *протеини*) и пребарувањето на генетска база на податоци се фундаментални пристапи за компјутерска анализа на генетски податоци. Компјутерската анализа на генетските податоци се врши за да се утврдат: *структурни*, *функционални* и *еволуциски* особености на анализираниот генетски материјал.

По дефиниција порамнувањето на генетски секвенци е процес на преуредување на содржината на две или повеќе низи од знаци над азбука Σ (Σ ={A,C,T,G} за ДНК, Σ ={A,C,G,U} за РНК, Σ ={A,R,N,D,C,D,G,E,H,I,L,K,M,F,P,S,T,W,Y,V} за протеини) со кои се утврдуваат фрагменти на сличност (хомологија), кои се последица на можна: структурна, функционална и еволуциска поврзаност. Од апликативен аспект, порамнувањето наоѓа примена за: филогенетска анализа, идентификација и квантификација на конзервирани фрагменти или функционални мотиви и профилирање на генетски заболувања. Добиените резултати се земаат како хипотеза за хомологија и истите ги определуваат взаемните сличности и разлики. Разликите помеѓу две или повеќе генетски секвенци се изразени во базни несогласувања и додавања (бришења) на елементи, додека сличностите ги определуваат фрагментите на совпаѓања.

Од друга страна, база на генетски секвенци се пребарува по анотиран генетски темплејт (генетски прашалник) за да се најдат парцијални или целосни совпаѓања на прашалникот во базата на секвенци. Во зависност од анотираноста на прашалникот по кој се врши пребарувањето, пронајдените совпаѓања се земаат како индикатори за: промотор секвенци, регулирачки елементи, почетоци и завршетоци на гени, интрони, егзони итн. Процесот наоѓа примена за функционална анотација на неанотирани генетски секвенци и селекција на слични секвенци од база на секвенци.

2. ИСТРАЖУВАЊА ВО ОБЛАСТА

Алгоритмите за порамнување на ДНК секвенци се делат во зависност од: *бројот на секвенци кои се порамнуваат*, *опсегот на порамнување*, *пристапот на порамнување* и *оптималноста на добиеното решение*. По дефиниција, порамнувањето е *паровo* кога се порамнуваат две секвенци. Ако се порамнуваат повеќе од две секвенци порамнувањето е *повеќекратно*. Во зависност од опсегот на порамнување, порамнувањата се делат на: *локални* и *глобални*. Ако се порамнуваат фрагменти од секвенци, порамнувањето е *локално*, во спротивност *глобално*. Независно од бројот на секвенци кои се порамнуваат и опсегот на порамнување, порамнувањата можат да бидат *празнински* или *безпразнински*. Моделот на празнинско порамнување вклучува додавања (бришења) на нуклеотиди, со што во целост се прикажува структурната и еволуциската зависност помеѓу порамнетите секвенци. За разлика од празнинското порамнување, беспразнинското порамнување ја исклучува можноста од постоење на настани од тип на додавања (бришења) на нуклеотиди и истото се извршува за да се утврди степенот на сличност помеѓу порамнетите секвенци. Степенот на сличност е фундамент на секоја филогенетска анализа, со која се врши групирање на најмалку оддалечените секвенци во заеднички кластери. Во зависност од оптималноста на добиеното решение, алгоритмите за порамнување можат да бидат: *детерминистички* или *хевристични*. Извршувањето на детерминистички алгоритам, за конкретна метрика на порамнување, секогаш резултира со оптимално решение од аспект на максимизација на резултат на порамнување или минимизација на растојание. Детерминистичките алгоритми во основа се примена на динамичко програмирање, каде во никој случај не смеат да се занемарат временските и меморискиите трошоци за добивање на решение. Неповолната временска и просторна комплексност ги прави алгоритмите базирани на динамичко програмирање применливи само на кратки секвенци (кратки прокариотски секвенци) или фрагменти од подолги секвенци (кратки фрагменти од еукариотска ДНК). Од друга страна, извршувањето на хевристични алгоритам не резултира секогаш со оптимално решение, но временскиот и меморискиот трошок за добивање на решение е минимален. Поволната временска и просторна комплексност ги прави овие алгоритми применливи на долги ДНК секвенци, како хромозоми и комплетни еукариотски геноми.

2.1. АЛГОРИТАМ НА NEEDLEMAN И WUNSCH

Алгоритмот на *Needleman и Wunsch* (Needleman et al., 1970) врши глобално порамнување на две ДНК секвенци. Порамнувајќи ја секоја база, се бара решение за кое важи дека *резултатот на порамнување* е максимален. Резултатот на порамнување се зема како мерка за сличност (хомологија) и истиот е функција од *награди* и *казни*. Награда се доделува за порамнување на еквивалентни нуклеотиди, додека казна се доделува за порамнување на различни нуклеотиди или порамнување на нуклеотид со празнина. Вообичаено, казната за порамнување на различни нуклеотиди се избира да биде помала од казната за порамнување на нуклеотид со празнина.

За да се порамнат секвенците: $\{a_i\}_{i=1}^{n}$ и $\{b_j\}_{j=1}^{m}$, се пресметува матрица $[s_{i,j}]_{(n+1)\times(m+1)}$, каде $s_{i,j}$ е резултат на оптимално порамнување за префикс секвенците: $a_1 \dots a_i$ и $b_1 \dots b_j$, кој се пресметува како: $s_{i,j} = \max\{s_{i,j-1} + g, s_{i-1,j} + g, s_{i-1,j-1} + s(a_i, b_j)\}$, каде: g е казна за порамнување на база со празнина, $s(a_i, b_j)$ е казна (награда) за порамнување на база a_i со база b_j ($s(a_i, b_j) < 0$ ако $a_i \neq b_j$ и $s(a_i, b_j) > 0$ ако $a_i = b_j$).

Ако $s_{i,j} = s_{i,j-1} + g$, тогаш b_j се порамнува со празнина која се додава на положба i во a. Ако $s_{i,j} = s_{i-1,j} + g$, тогаш a_i се порамнува со празнина која се додава на положба j во b. Нуклеотидите a_i и b_j взаемно се порамнуваат ако $s_{i,j} = s_{i-1,j-1} + s(a_i, b_j)$. Полињата $s_{i,0}, 0 \leq i \leq n$ и $s_{0,j}, 0 \leq j \leq m$ се иницијализираат на $i \times g$ и $j \times g$ соодветно. Следејќи ја патеката на покажувачи, порамнувањето се печати со враќање назад, од полето $s_{n,m}$ до полето $s_{0,0}$ (Сл. 2.1). Дијагонален покажувач долж патеката означува базно совпаѓање (несовпаѓање), додека вертикален (хоризонтален) покажувач означува додавање на празнина во секвенца на положба кон која покажувачот покажува. Резултатот на оптимално глобално порамнување е вредноста на полето $s_{n,m}$ од матрицата на динамичко програмирање.

Временската и меморискатa комплексност на алгоритмот на Needleman и Wunsch изнесува $O(n \times m)$ (алгоритмот извршува $n \times m$ споредби за да се пресметаат вредностите $s_{i,j}$, кои се чуваат во $n \times m$ податочна матрица).

		A	T	…	G
	0	g	2×g		m×g
A	g	$\max\{g+g, g+g, 0+s(A,A)\}$			
G	2×g	…….			
…	…		←	↖	
G	n×g				

Слика 2.1. Needleman-Wunsch матрица на динамичко програмирање
Figure 2.1. Needleman-Wunsch dynamic programming matrix

2.2. АЛГОРИТАМ НА SELLERS

Алгоритмот на Needleman-Wunsch ја максимизира *сличноста* помеѓу две секвенци, додека алгоритмот на *Sellers* (Sellers, 1974) го минимизира *растојанието* помеѓу две секвенци. Математичка дефиниција за растојание помеѓу две секвенци за првпат дава (Ulam, 1972). Според Ulam, растојание помеѓу две секвенци е минимален број на чекори со кои една секвенца се трансформира во друга, каде во секој чекор може да се: *избрише*, *додаде* или *замени* еден елемент. Алгоритмот на Sellers го пресметува растојанието помеѓу две секвенци во $O(m^2 n)$ чекори, каде m е должина на пократката секвенца. Ако при пресметка на растојанието се примени сугестијата од страна на Albert Nijenhuis и David Sankoff, според која растојанието $d_{i,j}$ помеѓу секој пар на потсеквенци: $a_1 \dots a_i$ и $b_1 \dots b_j$ се пресметува според (2.1), времето на извршување на алгоритмот се намалува од $O(m^2 n)$ на $O(nm)$.

$$d_{i,j} = min \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i,j} + \delta, \delta = 0 \text{ ако } a_i = b_j;\ \delta = 1 \text{ ако } a_i \neq b_j \end{cases} \quad (2.1)$$

		A	T	…	G
	0	1	2		m
A	1	$\min\{1+1, 1+1, 0+0\}$			
G	2	…..			
…	…				
G	n				

Слика 2.2. Пресметка на растојание според Sellers
Figure 2.2. Distance calculation according Sellers

2.3. АЛГОРИТАМ НА WAGNER И FISCHER

Wagner и Fischer (Wagner et al., 1974) задаваат различни цени за: додавање, бришење и замена на елемент. Ако со $\gamma(a_i \to b_j)$ се означи цената за замена на нуклеотид, со $\gamma(a_i \to \Lambda)$ се означи цената за бришење на нуклеотид a_i и со $\gamma(\Lambda \to b_j)$ се означи цената за додавање на нуклеотид b_j, тогаш според равенството (2.2), алгоритмот на Wagner и Fisher го пресметува растојанието помеѓу две секвенци со должини n и m во $O(nm)$ временски интервал.

$$d_{i,j} = min \begin{cases} d_{i-1,j} + \gamma(a_i \to \Lambda) \\ d_{i,j-1} + \gamma(\Lambda \to b_j) \\ d_{i,j} + \gamma(a_i \to b_j) \end{cases} \quad (2.2)$$

Проширување на метриката на Sellers предлагаат *Waterman, Smith и Beyer* (Waterman et al., 1976), кои освен единичните уредувања (додавање, бришење и замена на еден елемент во еден чекор) ги разгледуваат и случаите на повеќекратни додавања (бришења) на елементи.

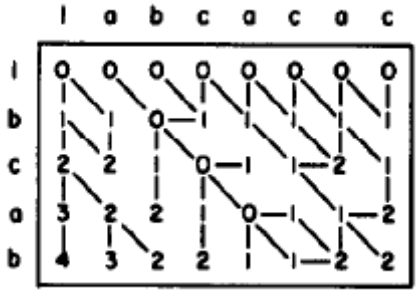
2.4. УСЛОВИ ЗА ЕКВИВАЛЕНТНОСТ НА АЛГОРИТМИТЕ НА NEEDLEMAN-WUNSCH И SELLERS

Иако се концепциски различни, односно алгоритмот на Needleman-Wunsch врши пресметка на сличност (хомологија), додека алгоритмот на Sellers врши пресметка на растојание, *Smith, Waterman и Fitch* (Smith et al., 1981) докажаа дека овие две процедури се еквивалентни. Условите за еквивалентност, независно од тоа дали ќе се примени алгоритмот на Needleman-Wunsch или алгоритмот на Sellers, се:

- $w'_k = \frac{k}{2} + w_k$, *каде* w'_k *е цена за последователно додавање на* k *празнини кај Sellers, додека* w_k *е цена за последователно додавање на* k *празнини кај Needleman-Wunsch (случај кога за порамнување на еквивалентни нуклеотиди се доделува награда 1);*
- *константност на збирот од резултатот на порамнување и растојанието пресметано според Sellers.*

2.5. АЛГОРИТАМ НА SELLERS ЗА ЛОКАЛНО ПОРАМНУВАЊЕ

Sellers (Sellers, 1980) предлага алгоритам за локално порамнување, со кој се врши пресметка на максимален степен на локална сличност за секој пар на интервали од две секвенци: $a = a_1 \dots a_n$ и $b = b_1 \dots b_m$. Максималниот степен на сличност е растојание, односно минимален број на единични уредувања (замени, додавања, бришења), со што потсекvенца од $a(b)$ се трансформира во потсекvенца од $b(a)$. Ако извршувањето на алгоритмот оди до пар на интервали кој ги вклучува секвенците a и b во целост, тогаш алгоритмот се извршува во $O(nm)$ време. Алгоритмот работи на начин, така што се пресметува матрица на растојанија $\left[d_{i,j}\right]_{(n+1)\times(m+1)}$, со почетни услови: $d_{0,j} = 0$ и $d_{i,0} = i$. Пресметувајќи го секое поле $d_{i,j}$ од матрицата $\left[d_{i,j}\right], i \geq 1, j \geq 1$ според (2.1), почетоците и завршетоците на најсличните фрагменти од секвенцата b со фрагмент $a_1 \dots a_i$, ги определуваат патеки кои вклучуваат елементи со најмала вредност од редица i (Сл 2.3). Истите ги определуваат и единични уредувања, со кои еден фрагмент се трансформира во друг.



Слика 2.3. Пресметка на максимален степен на локална сличност за пар на интервали според Sellers (1980)

Figure 2.3. Calculation of maximal similarity degree for pair of intervals according Sellers (1980)

2.6. АЛГОРИТАМ НА SMITH И WATERMAN

Smith и Waterman (Smith et al., 1981) предлагаат модификација на алгоритмот на Needleman и Wunsch со примена за локално порамнување на две секвенци. За да се порамнат секвенците $\{a_i\}_{i=1}^{n}$ и $\{b_j\}_{j=1}^{m}$, се пресметува матрица $[s_{i,j}]_{(n+1)\times(m+1)}$, каде полињата: $s_{i,0}, 0 \le i \le n$ и $s_{0,j}, 0 \le j \le m$ се поставени на 0. Вредностите на полињата $s_{i,j}, 1 \le i \le n, 1 \le j \le m$ се пресметуваат според (2.3). Ако $s_{i,j} = s_{i-1,j-1} + s(a_i, b_j)$, тогаш се порамнуваат нуклеотидите a_i и b_j. Ако $s_{i,j} = s_{i-1,j} + s(a_i, b_j)$, тогаш нуклеотидот a_i се порамнува со празнина која се додава на положба j во секвенцата b, во спротивност нуклеотидот b_j се порамнува со празнина која се додава на положба i во секвенцата a. По аналогија на Needleman-Wunsch, се чува листа на покажувачи, од каде се печати решението, со враќање назад, од полето со максимална вредност сè до првото нулто поле долж патеката.

$$s_{i,j} = max \begin{cases} 0 \\ s_{i-1,j-1} + s(a_i, b_j) \\ s_{i-1,j} + g \\ s_{i,j-1+g} \end{cases} \quad (2.3)$$

Ако $s_{i,j} = s_{i-1,j-1} + s(a_i, b_j)$, полето $s_{i-1,j-1}$ е означено со дијагонален покажувач од полето $s_{i,j}$. Ако $s_{i,j} = s_{i-1,j} + s(a_i, b_j)$, полето $s_{i-1,j}$ е означено со вертикален покажувач од полето $s_{i,j}$, во спротивност полето $s_{i,j-1}$ е означено со хоризонтален покажувач од $s_{i,j}$. Дијагонален покажувач означува порамнување на базите a_i и b_j (совпаѓање или несовпаѓање), хоризонтален покажувач означува додавање на празнина на положба i во секвенца a, додека вертикален покажувач означува додавање на празнина на положба j во секвенца b.

Иако алгоритмот на Smith и Waterman ја максимизира хомологијата на ниво на фрагменти, додека алгоритмот на Needleman и Wunsch ја максимизира хомологијата на ниво на секвенци во целост, временската и меморискатa комплексност на нови два алгоритми е еднаква, односно истата изнесува $O(nm)$.

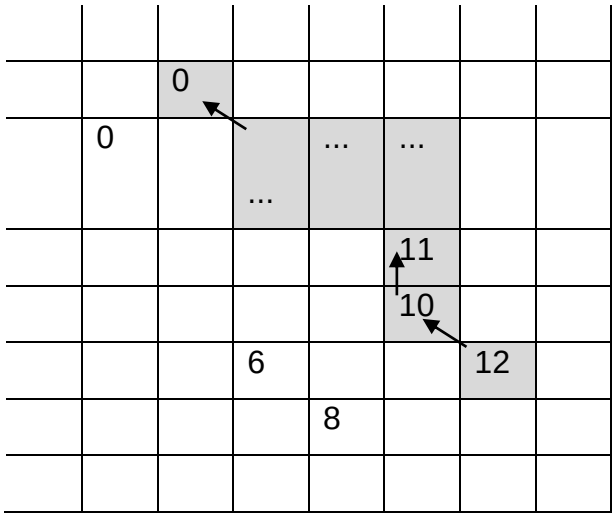
		A	T	…	T
	0	0	0		0
A	0	$\max\{0, 0+g, 0+g, 0+s(A,A)\}$			
G	0	…..			
…	…			$\max\{s_{i,j}\}$	
G	0				

Слика 2.4. Smith-Waterman матрица на динамичко програмирање
Figure 2.4. Smith-Waterman dynamic programming matrix

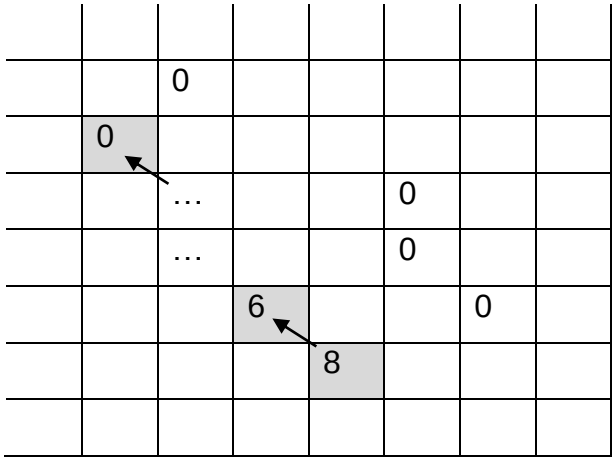
2.7. АЛГОРИТМИ НА WATERMAN–EGGERT И GOAD–KANEHISA

Алгоритмот на Smith и Waterman генерира едно оптимално локално решение, додека алгоритмите на *Waterman и Eggert* (Waterman et al., 1987), *Goad и Kanehisa* (Goad et al., 1982) и *Sellers* (Sellers, 1984) генерираат *листа на значајни локални порамнувања*.

Со примена на алгоритмот на *Waterman и Eggert* може да се утврдат *k*-најдобри локални порамнувања. *K*-најдобри локални порамнувања се *k* различни локални решенија со највисок резултат на порамнување, кои меѓусебно не се сечат, односно не вклучуваат заеднички нуклеотиди. Истите се утврдуваат етапно, во фаза на враќање, откако Smith-Waterman матрицата на динамичко програмирање ќе се пресмета во целост. Првин се утврдува локалното порамнување со највисок резултат на порамнување, веднаш потоа се утврдува локалното порамнување со втор по големина резултат на порамнување кое не вклучува елементи од претходното порамнување итн. Постапката се извршува на начин, така што откако ќе се определи локалното порамнување со највисок резултат на порамнување, полињата кои го определуваат истото во рамки на Smith-Waterman матрица на динамичко програмирање се поставуваат на 0, па се бара патеката на покажувачи од поле со нова максимална вредност, која го определува второто најдобро локално порамнување итн.



Прво најдобро локално порамнување



Второ најдобро локално порамнување

... ...

Слика 2.5. Waterman-Eggert алгоритам за субоптимално порамнување
Figure 2.5. Waterman-Eggert algorithm for suboptimal alignment

Алгоритмот на *Goad u Kanehisa* генерира листа на локални порамнувања, подредени по статистичка значајност. Доделувајќи различна тежина за секој вид на несогласување (бришења на 1,2,3,... нуклеотиди, замена на 1,2,3,... нуклеотиди), алгоритмот утврдува парови на фрагменти за кои густината на тежинските несогласувања, односно тежината на несогласување по елемент е помала од кориснички зададена прагова вредност.

Врз основа на концептот предложен од *Goad u Kanehisa* за пресметка на тежина на несогласување по елемент (пресметка на густина на совпаѓање), *Sellers* предлага алгоритам со кој се скенираат сите можни парови на сегменти од две секвенци во $O(nm)$ време, при што предвид се земаат само сегментите со задоволителна сличност. Формулата по која се пресметува сличноста помеѓу различни парови на сегменти е $rl - d$, каде l е средна должина на сегментите, d е минимален број на единечни уредувања со кои еден од сегментите се трансформира во друг и r е релативна пропорционална константа, со која е определен взаемниот удел на l и $-d$ во пресметката на сличноста.

2.8. АЛГОРИТМИ НА FITCH–SMITH И GOTOH

Според *Fitch u Smith* (Smith et al., 1983), за да се добие точно решение, треба да се земат предвид и повеќекратните додавања (бришења) на нуклеотиди. Според заклучоците кои ги предлагаат:

- *тежината за додавање на празнина мора да зависи од должината на празнината;*
- *во никој случај не смее да се допушти избор на нулта тежина за додавање на празнина, бидејќи во тој случај би се генерирале оптимални решенија со различна структура (различен број на празнини);*
- *метриката на Needleman u Wunsch е супериорна во однос на метриката на Sellers, поради бестежинскиот удел на терминалните празнини;*
- *може да се допушти избор на различни вредности за тежини за совпаѓање (несовпаѓање) од 1(0).*

Во овој случај казната за додавање на празнина ја определува тежинска функција $\gamma(g)$, која зависи од должината на празнината. При пресметка на вредноста на поле $s_{i,j}$ од матрица на динамичко програмирање, независно од тоа дали се работи за локално или глобално порамнување, освен взаемната споредба на нуклеотидите a_i и b_j и случаите на додавање по една празнина во $a(b)$: $(a_i, \_), (\_, b_j)$ се разгледуваат и случаите на повеќекратни додавања (бришења) на нуклеотиди: $(\ldots a_{i-1}a_i, \ldots \_\_\ )$, $(\ldots \_\_, b_{j-1}b_j)$. Во вакви услови, за да се пресмета вредноста на поле $s_{i,j}$ се земаат предвид $i + j + 1$ претходници, наместо три, со што времето на извршување се зголемува од $O(n^2)$ на $O(n^3)$. Кога се бара глобално порамнување, матрицата на динамичко програмирање се пресметува според (2.4). Кога се бара локално порамнување, секое поле $s_{i,j}$ од матрицата на динамичко програмирање се пресметува според (2.5).

$$s_{i,j} = max \begin{cases} s_{i-1,j-1} + s(a_i, b_j) \\ s_{k,j} + \gamma(i-k), \ \text{за } k = 0, \ldots, i-1 \\ s_{i,k} + \gamma(j-k), \ \text{за } k = 0, \ldots, j-1 \end{cases} \quad (2.4)$$

$$s_{i,j} = max \begin{cases} s_{i-1,j-1} + s(a_i, b_j) \\ s_{k,j} + \gamma(i-k), \ \text{за } k = 0, \ldots, i-1 \\ s_{i,k} + \gamma(j-k), \ \text{за } k = 0, \ldots, j-1 \\ 0 \end{cases} \quad (2.5)$$

Со линеаризација на тежинската функција $\gamma(g)$, $\gamma(g) = go + k \times ge$, каде: go е казна која се задава за отворање на празнина (додавање на прва празнина по нуклеотид), ge е казна за издолжување на празнина (додавање на една или повеќе празнини по веќе отворена празнина) и k е број на празнински издолжувања, *Gotoh* (Gotoh, 1982), (Gotoh, 1987) успева да ја намали временската комплексност на извршување од $O(n^3)$ на $O(n^2)$.

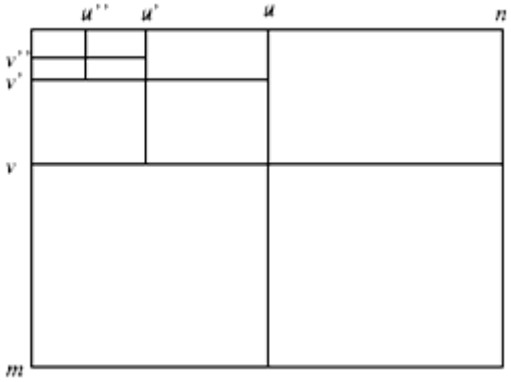
2.9. МЕМОРИСКИ ЛИНЕАРНИ АЛГОРИТМИ (HIRSCHBERG, MAYER–MILLER, HUANG и HUANG–MILLER)

Hirschberg (Hirschberg, 1975) прв ја разгледува идејата за линеаризација на мемориската комплексност. Врз основа на таа идеја, *Myers u Miller* (Myers et al., 1988) развиваат мемориски линеарна верзија на алгоритмот на Gotoh. Истите автори (Miller et al., 1988), разгледуваат ефикасни пристапи за пресметка на растојание помеѓу две секвенци, каде цената за додавање (бришење) на блок елементи е конкавна функција од должината на блокот. Huang et al., 1990 предлагаат алгоритам за локално порамнување, чија мемориска комплексност е пропорционална на збирот од должините на ДНК секвенците кои се порамнуваат. Алгоритмот е базиран на динамичко програмирање и истиот освен за локално порамнување може да се примени и за наоѓање на повторувачка ДНК содржина во подолга ДНК секвенца. *Huang u Miller* (Huang et al., 1991) предлагаат временски ефикасна и просторно линеарна верзија на алгоритмот на Waterman и Eggert за утврдување на *k*-најдобри локални порамнувања. Мемориската комплексност на алгоритмот изнесува $O(m + n + K)$, каде m и n се должини на ДНК секвенците кои се порамнуваат и K е должина на порамнување.

Наивниот пристап за намалување на мемориската комплексност се базира на пресметка на вредност на поле од редица i, во зависност од вредностите на соседните полиња по хоризонтала, вертикала и дијагонала од истата и претходната редица (редица $i-1$). Извршувајќи ги пресметките редица по редица, од лево па на десно, вредноста на оптимално порамнување може да се пресмета, меморирајќи две по две редици, што во основа ја линеаризира мемориската комплексност. Во тој случај, патеката на оптимално порамнување се губи, бидејќи на крајот од обработката, во меморија се чуваат само вредностите на полињата од последните две редици од матрица на динамичко програмирање.

За реконструкција на патеката на оптимално порамнување, може да се искористи стратегијата раздели па владеј, со која проблемот се дели на повеќе помали и независни потпроблеми. Решението се добива со соединување на потпроблемските решенија.

Идејата е да се најде пресек (v,u), каде со спојување на оптималните порамнувања за потсеквенците: $a_1 \dots a_u$; $b_1 \dots b_v$ и $a_{u+1} \dots a_n$; $b_{v+1} \dots b_m$ се добива оптимално порамнување за секвенците a и b. Забележете дека пресекот (v,u) ја дели матрицата на динамичко програмирање на четири квадранти, игнорирајќи го североисточниот и југозападниот квадрант, бидејќи патеката на оптимално порамнување не поминува низ нив, (Сл. 2.6). Алгоритамот рекурзивно се повикува за северозападните и југоисточните квадранти, сè до единично порамнување на база со база или база со празнина, кога со соединување на потпроблемските решенија се добива оптимално порамнување за секвенците a и b.

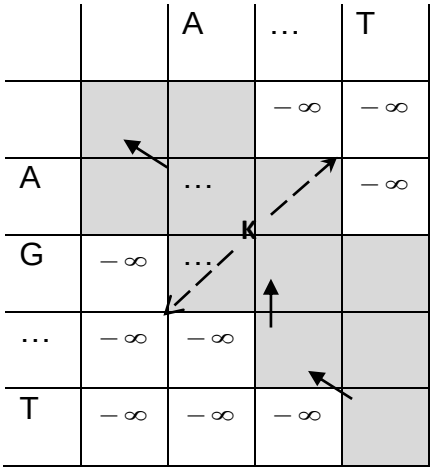


Слика 2.6. Делење на матрица на динамичко програмирање (Hirschberg)
Figure 2.6. Division of dynamic programming matrix (Hirschberg)

2.10. ПОРАМНУВАЊА ВО ДИЈАГОНАЛЕН ОПСЕГ (FICKET, UKKONEN, CHAO и SPOUGE)

Претходните алгоритми ја линеаризираат мемориската комплексност, но не и временската комплексност. Првите пристапи за подобрување на временската комплексност се базираат на концептот за порамнување во дијагонално симетричен и ограничен опсег околу главната дијагонала од матрица на динамичко програмирање. Овие алгоритми, наместо долж целата матрица, решението го бараат долж скратен и симетричен опсег околу главната дијагонала од матрица на динамичко програмирање. Ваквиот пристап не ја загрозува оптималноста на решението при порамнување на секвенци со висок степен на сличност, бидејќи во тие случаи оптималното порамнување го определува патека на покажувачи која конвергира околу главната дијагонала. Отфрлувајќи ги, односно не пресметувајќи ги полињата $s_{i,j}$ кои се значајно оддалечени од главната дијагонала ($|i-j| > \frac{k}{2}$), равенство (2.6), (Сл. 2.7), алгоритмот во фаза на извршување ги пресметува само полињата $s_{i,j}$ за кои важи $|i-j| \leq \frac{k}{2}$, каде $k, k \leq m$ е широчина на дијагонално симетричниот опсег, со што времето на извршување се намалува од $O(nm)$ на $O(kn)$.

$$s_{i,j} = max \begin{cases} s_{i-1,j-1} + s(a_i, b_j) \\ s_{i-1,j} + g, \text{ако } |i-1-j| \leq \frac{k}{2} \\ s_{i,j-1} + g, \text{ ако } |i-j+1| \leq \frac{k}{2} \end{cases} \quad (2.6)$$



Слика 2.7. Порамнување во дијагонално симетричен опсег
Figure 2.7. Alignment within specific diagonal band

Проблемот на порамнување во ограничен опсег први го разгледуваат *Sankoff u Kruskal* (Sankoff et al., 1983). Решенија на проблемот се предложени од *Ficket* (Ficket, 1984), *Ukkonen* (Ukkonen, 1985) и (*Chao et al*., 1992). *Spouge*, 1991 ја разгледува примената на пристапот за порамнување во ограничен опсег за да се пресмета растојание помеѓу две секвенци.

Алгоритмот на *Ficket* врши парцијална пресметка на матрица на растојанија со примена на динамичко програмирање. Ако $d_{i,j}$ е растојание помеѓу потсеквенците: $a_1 \dots a_i$ и $b_1 \dots b_j$ и D е растојание помеѓу секвенците: $a_1 \dots a_n$ и $b_1 \dots b_m$, според Ficket неопходно е да се пресметаат само растојанијата $d_{i,j}$ за кои важи $d_{i,j} \leq D$. Растојанијата $d_{i,j}$ се пресметуваат според (2.7) и истите се земат како вредности на полиња i,j од матрица на растојанија. Бидејќи се пресметува растојание помеѓу потсеквенци, растојанието $d_{i,j}$ се зголемува единствено при додавање на празнина g или порамнување на различни нуклеотиди x.

$$d_{i,j} = min \begin{cases} d_{i-1,j} + g \\ d_{i,j-1} + g \\ d_{i-1,j-1} + x, a_i \neq b_j \end{cases} \quad (2.7)$$

Пополнувањето на матрицата се врши редица по редица. На почеток, се пресметуваат првите l_1 полиња од првата редица на матрица на растојанија, $d_{1,1} \dots d_{1,l1}$, така што l_1 е минимумот за кој важи $d_{1,l1} \geq D$. Постапката продолжува со пресметка на полињата $d_{2,1} \dots d_{2,l2}$ од втората редица, така што l_2 е минимумот за кој важи $d_{2,l2} \geq D$ и $l_2 > l_1$. Истата постапка се применува за пресметка на полињата од секоја наредна редица.

Пресметувајќи ги исклучиво растојанијата $d_{i,j}$ за кои $|i-j| < \frac{D}{g} + 1$, решението се бара во рамки на дијагонално-симетричен опсег со должина $2 \times \left(\frac{D}{g} + 1\right)$, каде D е растојание помеѓу ДНК секвенците a и b и g е казна за додавање на празнина, во овој случај $g > 0$.

Исклучувањето на пресметките за $d_{i,j}$ за кои $|i-j| \geq \frac{D}{g} + 1$, го ограничува бројот на пресметани полиња на $2 \times \min(m,n) \times \left(\frac{D}{g} + 1\right)$. При порамнување на ДНК секвенци со висок процент на базна идентичност, должината на дијагонално-симетричниот опсег $2 \times \left(\frac{D}{g} + 1\right)$ е значајно помала од должината на подолгата ДНК секвенца n. Намалувањето на должината на опсегот каде се бара решение резултира со помал број на извршени пресметки, на што се должи подобрувањето на временската комплексност.

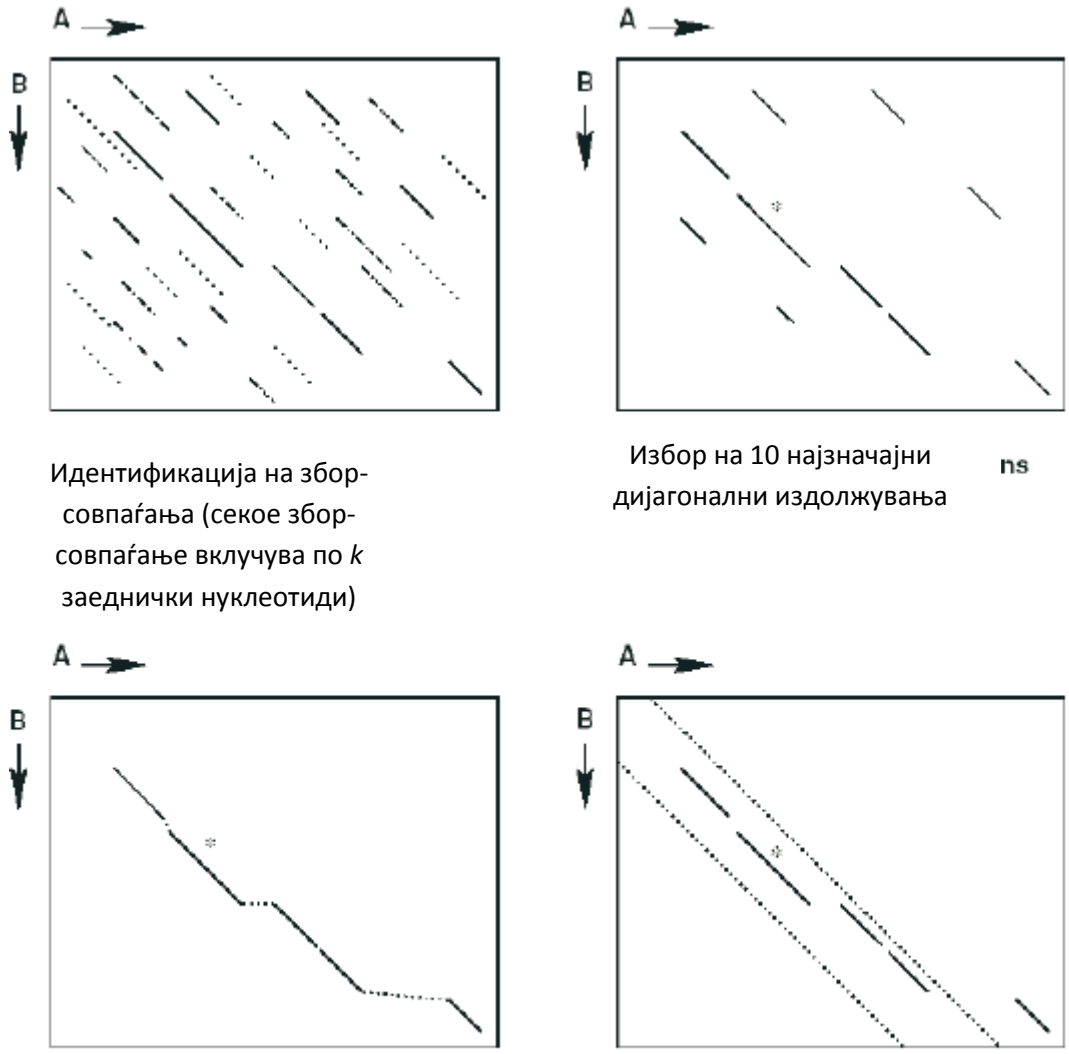
Со примена на алгоритмот на *Ukkonen* се генерира решение во $O(s \times \min(m,n))$ време и простор, каде s е растојание помеѓу секвенците кои се порамнуваат. Ако казната за додавање (бришење) и замена на нуклеотид е иста, мемориската комплексност на алгоритмот изнесува $O(s \times \min(m,n,s))$.

Алгоритмот на *Chao* врши оптимално порамнување во дијагонално-симетричен опсег со должина W во $O(mW)$ време и $O(m)$ простор, каде m е должина на пократката од двете ДНК секвенци кои се порамнуваат. Овој алгоритам првин ги наоѓа почетоците и краевите на најдобрите локални порамнувања, по што со порамнување на ДНК фрагментите кои истите ги одделуваат, се генерира конкретно решение.

2.11. БАЗИЧНИ ХЕВРИСТИЧКИ АЛГОРИТМИ (FASTA И BLAST)

Неповолната временска комплексност, која е пропорционална на производот од должините на ДНК секвенците кои се порамнуваат $O(nm)$, во реални услови ја ограничува примената на алгоритмите базирани на динамичко програмирање. Поради долгото време на извршување, претходните пристапи не можат да се применат за порамнување на долги секвенци (хромозоми или целосни геноми) или пребарување на база на податоци по референтна прашалник секвенца. За да се издвојат слични секвенци од база на податоци во однос на прашалник секвенца, секоја секвенца од базата на податоци се порамнува со прашалникот, каде за слични секвенци се избираат секвенците со највисоки резултати на порамнување. Извршувањето на овој процес со примена на динамичко програмирање бара време кое е пропорционално на производот од должината на прашалникот и големината на базата на податоци (вкупниот број на нуклеотиди од сите секвенци во базата). За да може да се изврши овој процес во реални услови, истиот неопходно е повеќекратно да се забрза. Хевристичните пристапи како: *FASTP* (Lipman et al., 1985), *FASTA* (Pearson et al., 1988) и *BLAST* (Altschul et al., 1997) го забрзуваат овој процес од 50 до 100-пати во споредба со алгоритмите базирани на динамичко програмирање, со што за кратко време се добива решение, кое не секогаш е оптимално. Она што е заедничко за овие пристапи, е тоа што истите генерираат решение со изнаоѓање на значајни совпаѓања, кои во подоцнежната фаза од извршувањето се соединуваат (издолжуваат).

FASTP е прв применлив пристап за наоѓање на значајни совпаѓања помеѓу прашалник секвенца и база на податоци. FASTP е претходник на FASTA и за разлика од FASTA, која може да се примени за ДНК и протеински секвенци, FASTP е применлив само за протеини. Во првата фаза од извршувањето, FASTA конструира пребарувачка хеш-табела, која ги содржи сите зборови со должина *k* од прашалник секвенца. Се бараат совпаѓања на овие зборови во базата на секвенци. Секое совпаѓање се бележи со податочен пар од вид: (i, j), каде i е почетна положба на збор во рамки на прашалникот и j е почетна положба на истиот збор во рамки на секвенца од базата на податоци. Последователните збор-совпаѓања долж иста дијагонала од матрица на динамичко програмирање се соединуваат со што се образуваат издолжени совпаѓања, познати како дијагонални издолжувања. Секое дијагонално издолжување се вреднува, при што алгоритмот ги зема предвид само десетте најзначајни дијагонални издолжувања. Кога се работи со протеински секвенци, за вреднување на дијагоналните издолжувања се користи PAM 250 матрица на замени. Порамнувањето се добива со примена на Smith-Waterman алгоритмот во ограничен опсег, со празнинско порамнување на ДНК фрагменти кои одделуваат најповолна комбинација на дијагонални издолжувања. Перформансите на извршување на FASTA и FASTP зависат од параметарот *k* (должината на зборовите). Кога се работи со ДНК секвенци за *k* се избира 6, додека кога се работи со протеини за *k* се избира 2. Фазите на извршување на FASTA се прикажани на Сл. 2.8.

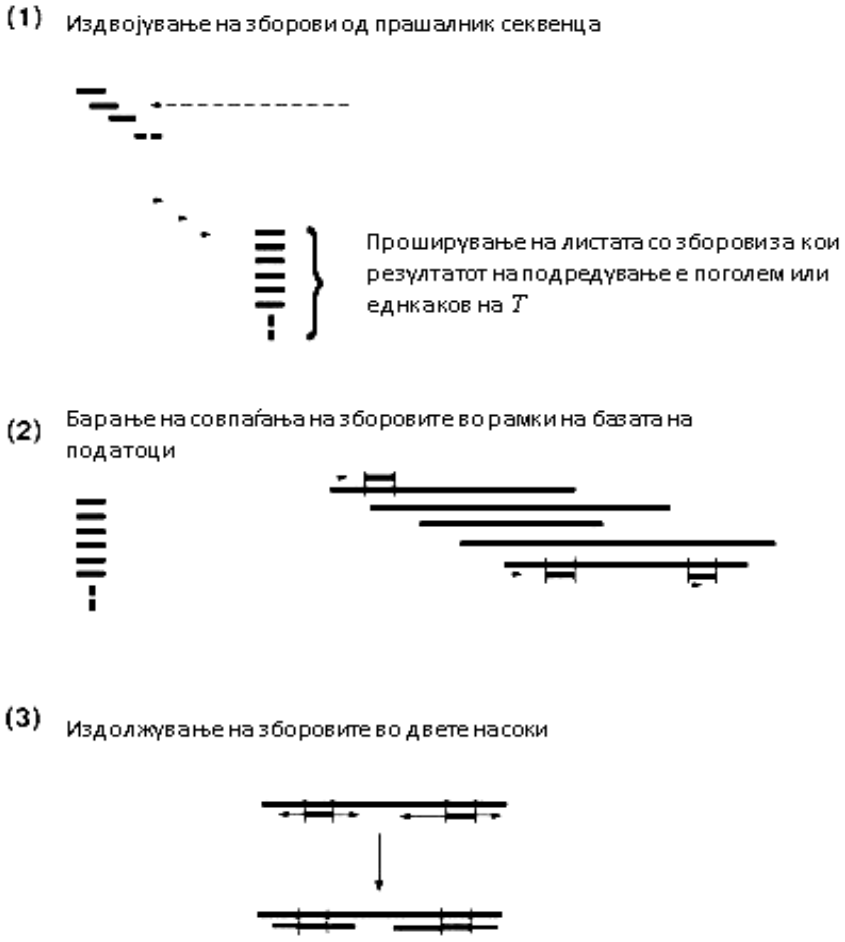


Слика 2.8. Фази на извршување на FASTA
Figure 2.8. FASTA running phases

BLAST (Basic Alignment Search Tool) е исто така хевристични пристап за пребарување на база на податоци по прашалник секвенца. Првата верзија на BLAST извршува безпразнински порамнувања. Промена во пристапот на порамнување, односно извршување на празнински наместо безпразнински порамнувања, предлагаат Altschul et.al (1997) (*PSI-BLAST*). Како FASTA, така и BLAST е концепт базиран на пребарување по парцијални (целосни) совпаѓања на зборови од референтна прашалник секвенца, кои се издолжуваат во подоцнежната фаза од извршувањето. Во споредба со FASTA, BLAST е побрз, но за разлика од FASTA кога се работи со протеински секвенци, BLAST пребарува по слични зборови од прашалник секвенца во база на податоци.

При работа со протеински секвенци, во првата фаза од извршувањето, BLAST ги издвојува сите зборови со должина w од прашалник секвенца. Во проширена листа на зборови се додава секој збор со должина w за кој резултатот на беспразнинско порамнување со претходно издвоен збор од прашалник

секвенца е поголем од минимален праг на сличност T. Резултатот на беспразнинско порамнување при порамнување на пар на зборови се пресметува според BLOSUM или PAM матрица на замени. Во понатамошниот тек од извршувањето, се бараат целосни совпаѓања на зборовите од проширената листа во базата на податоци, при што се бележи почетната положба на секое совпаѓање. Секое совпаѓање (семе) беспразнински се издолжува во две насоки (налево и надесно) се додека резултатот на порамнување расте, како последица на издолжувањето. BLAST ги зема предвид само значајните издолжувања, односно беспразнинските порамнувања за кои резултатот на порамнување е поголем од минимум дозволена вредност S. При извршување на BLAST, параметрите: w, T и S се влезни параметри. Структурата на решението зависи од изборот на конкретни вредности за овие параметри. Параметарот T има најголемо влијание врз брзината и чувствителноста на пребарување.



Слика 2.9. Фази на обработка кај БЛАСТП
Figure 2.9. BLASTP phases

При работа со ДНК секвенци, BLAST исто така генерира листа на зборови со должина w, кои се издвојуваат од прашалник секвенца. Должината на зборовите кои се издвојуваат при работа со нуклеотидни секвенци најчесто изнесува 12. Издвоените зборови се пребаруваат во базата на податоци, при што се користи двојно-бит кодирање по нуклеотид (од 00 до 11), со што се заштедува на меморија (простор).

Денес постојат различни имплементации на BLAST. *BLASTN* споредува нуклеотиден прашалник со ДНК база на податоци, *BLASTP* врши споредба на протеински прашалник со протеинска база на податоци, *TBLASTP* споредува протеин со ДНК база на податоци, додека *BLASTX* споредува нуклеотиден прашалник со протеинска база на податоци.

2.12. PATTERN HUNTER

Чувствителноста на пребарување кај BLAST зависи од должината k на зборовите кои се пребаруваат. При избор на голема вредност за k се губи можноста за идентификација на далечни хомологии (не постои можност за идентификација на совпаѓања со помалку од k карактери), додека за избор на мала вредност за k се забавува процесот на пребарување. *Pattern Hunter* (Ma et al., 2002) е пристап со кој се подобрува чувствителноста на пребарување. Pattern Hunter бара совпаѓања на зборови кои вклучуваат најмалку k заеднички нуклеотиди, кои освен последователно може да се појавуваат и непоследователно, со што во споредба BLAST, кој бара исклучиво зборови составени од последователни совпаѓачки карактери, се подобрува чувствителноста на пребарување, односно се зголемува бројот на пронајдени совпаѓања. Според Bin Ma et al., за регион со 70% сличност и за $k = 11$, чувствителноста на пребарување е максимална за модел: 111010010100110111, каде 1 означува совпаѓање на карактери на положба означена со 1, додека 0 означува несовпаѓање (совпаѓање) на карактери на положба означена со 0. Секој збор за кој моделот е задоволен (на положбите означени со 1 постојат совпаѓачки карактери, додека на положбите означени со 0 постојат несовпаѓачки или совпаѓачки карактери) се детектира. Излезот зависи од избраниот модел, односно за различни модели се добиваат различни резултати на излез. Во фаза на извршување Pattern Hunter за секоја положба од секвенца пресметува индекс врз основа на совпаѓање на моделот на дадената положба, со што се генерира листа на совпаѓања кои подоцна се издолжуваат во две насоки.

2.13. БЛАТ (BLAT)

Блат (Kent, 2002) е алтернативен пристап за детекција на хомологни секвенци. За разлика од Бласт, каде за секое совпаѓање се пресметува статистичка значајност, Блат бара перфектни или речиси перфектни совпаѓања. Ваквиот концепт ја намалува флексибилноста на пребарување (се намалува чувствителноста за детекција на совпаѓања), но од друга страна се подобруваат временските аспекти на извршување во споредба со Бласт. Од апликативен аспект, Бласт е применлив за споредба на еволуциски блиски и далечни секвенци, додека Блат е применлив само за споредба на еволуциски блиски секвенци.

Методолошки, Блат се извршува во две фази: фаза на пребарување и фаза на порамнување. Во фазата на пребарување, Блат бара: перфектни, речиси-перфектни и повеќекратни перфектни совпаѓања помеѓу прашалник секвенца и база на секвенци. Врз основа на листа на заеднички совпаѓања помеѓу прашалник секвенца и хомологни фрагменти од база на секвенци, Блат во втората фаза извршува празнински порамнувања.

За разлика од Бласт, кој ја индексира прашалник секвенцата, Блат ја индексира базата на секвенци, односно го индексира секој непреклопувачки збор кој вклучува k карактери (k-mer) од база на секвенци. При споредба на нуклеотидни секвенци за параметарот k се избира вредност помеѓу 8 и 16, додека при споредба на протеински секвенци вредноста на параметарот k варира помеѓу 3 и 7. Секој преклопувачки збор од прашалник секвенцата со должина k се споредува со секој непреклопувачки збор од базата на секвенци за да се утврдат хомологии како: перфектни, речиси-перфектни и повеќекратни перфектни совпаѓања. Според Kent 2002, перфектно совпаѓање е целосно совпаѓање на збор од прашалник секвенца со збор од базата на секвенци, речиси-перфектно совпаѓање вклучува едно базно несовпаѓање, додека повеќекратните перфектни совпаѓања вклучуваат N перфектни совпаѓања, кои меѓусебно се оддалечени не повеќе од W карактери во рамки на прашалник секвенцата и базата на секвенци.

Очекуваниот број на случајни перфектни совпаѓања изнесува: $\mathrm{F} = (\mathrm{q} - \mathrm{k} + 1) \times \left(\frac{\mathrm{D}}{\mathrm{k}}\right) + (\frac{1}{\mathrm{A}})^{\mathrm{k}}$, очекуваниот број на случајни речиси перфектни совпаѓања изнесува: $\mathrm{F} = (\mathrm{q} - \mathrm{k} + 1) \times \left(\frac{\mathrm{D}}{\mathrm{k}}\right) \times (\mathrm{k} \times \left(\frac{1}{\mathrm{A}}\right)^{\mathrm{k}-1} \times \left(1 - \frac{1}{A}\right) + \left(\frac{1}{\mathrm{A}}\right)^{\mathrm{k}})$, додека очекуваниот број на повеќекратни совпаѓања кои вклучуваат по N единечни перфектни совпаѓања се пресметува по формулата: $\mathrm{F_N} = \mathrm{F_1} \times \mathrm{S}$, каде: $\mathrm{F_1} = (\mathrm{q} - \mathrm{k} + 1) \times \left(\frac{\mathrm{D}}{\mathrm{k}}\right) \times (\frac{1}{\mathrm{A}})^{\mathrm{k}}$, $\mathrm{S} = 1 - (1 - \left(\frac{1}{\mathrm{A}}\right)^{\mathrm{k}})^{\mathrm{W/k}}$, q е должина на прашалник секвенца, D е големина на базата на секвенци, W е максимална дозволена оддлаченост на перфектни совпаѓања во рамки на повеќекратни совпаѓања и A е број на карактери од азбуката (за нуклеотидни секвенци $\mathrm{A} = 4$, за протеински секвенци $\mathrm{A} = 20$).

2.14. FLASH

FLASH (Califano et al., 1993) е пристап за идентификација на хомологни фрагменти помеѓу две секвенци. На почеток FLASH генерира две одделни листи на индекси, кои ги содржат почетните положби на сите преклопувачки зборови со должина k. Една листа на индекси од тип: $(i, w_i w_{i+1} \dots w_{i+k-1})$, каде i е почетна положба на збор $w_i w_{i+1} \dots w_{i+k-1}$ се генерира за целна секвенца од база на податоци, додека друга, одделна листа од ист тип се генерира за прашалник секвенца, чија должина најчесто е помала од должината на целната секвенца. Хомологните региони се утврдуваат со споредби на потсеквенци од целна секвенца со прашалник секвенца. Имено, за секој пар (*потсеквенца, прашалник секвенца*) се пресметуваат разликите помеѓу почетните положби на заедничките зборови, при што се утврдува број на еднакви разлики. Потсеквенцата од споредбениот пар (*потсеквенца, прашалник секвенца*) за која бројот на еднакви разлики е максимален е хомологен фрагмент од целна секвенца во однос на прашалник секвенца, односно тоа е фрагмент кој вклучува максимален број на совпаѓања.

На пример, нека ACTGATTG е целна секвенца, додека GAGTG е прашалник секвенца. За $k = 2$ се генерира листа на индекси: Lt={(1,AC),(2,CT),(3,TG),(4,GA),(5,AT),(6,TT),(7,TG)} за целната секвенца и листа на индекси за прашалник секвенцата: Lq={(1,GA),(2,AG),(3,GT),(4,TG)}. При

споредба на потсеквенцата GATTG од целната секвенца со прашалник секвенцата GAGTG, се утврдуваат две совпаѓања: ((4,GA)|Lt, (1,GA)|Lq), ((7,TG)|Lt, (4,TG)|Lq). Разликите помеѓу почетните положби на заедничките совпаѓања се еднакви, 4-1=7-4=3, од каде за број на еднакви разлики се добива 2. Бидејќи овој број е максимален во однос на која било друга споредбена потсеквенца, потсеквенцата GATTG е хомологен фрагмент од целната секвенца ACTGATTG во однос на прашалник секвенцата GAGTG.

2.15. YASS

YASS (YASS: enhancing the sensitivity of DNA similarity search, Noé et al., 2005) е алатка за пребарување на хомологија, која се базира на имплементација на ефикасен и сензитивен алгоритам за филтрирање. YASS бара групи на блиски совпаѓања. Максималната оддалеченост на совпаѓањата во рамки на групите на совпаѓања изнесува d. Растојанието d се пресметува во функција од фрекфенциите на замени и бришења (додавања) на елементи. Покрај филтрирањето по основ на меѓусебна оддалченост, YASS го ограничува и минималниот број на совпаѓања n од кои се формира група на блиски совпаѓања.

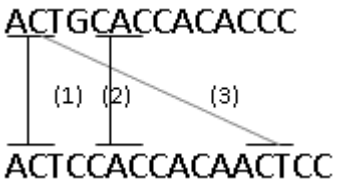
2.16. АЛГОРИТМИ ЗА ПОРАМНУВАЊЕ БАЗИРАНИ НА ПРЕБАРУВАЊЕ НА СТЕБЛО НА СУФИКСИ (MUMmer и AVID)

MUMmer (Delcher et al., 1995) е хевристичен пристап за брзо порамнување на долги секвенци, најчесто геноми. MUMmer се базира на три концепти: ***стебло на суфикси***, ***идентификација на најдолги заеднички потсеквенци*** и ***примена на Smith-Waterman порамнување***, со чија интеграција се овозможува брзо и ефикасно порамнување на долги ДНК секвенци. Стеблото на суфикси овозможува идентификација на најдолги и единствени заеднички потсеквенци помеѓу два геноми. Овие совпаѓања *Delcher* ги нарекува MUM-ери (*Maximal Unique Matches*) и истите во основа се најдолги можни совпаѓања помеѓу два геноми, кои не се дел од ниту едно друго подолго заедничко совпаѓање.

На пример, потсеквенцата ACT е MUM-ер за пример секвенците секвенците: a:ACTGCACCACACCC и b:ACTCCACCACAACTCC. Совпаѓањата: AC и CT не претставуваат MUM-ери, бидејќи истите се дел од подолга совпаѓачка секвенца ACT. CCA совпаѓањето исто така не претставува MUM-ер, бидејќи истото се повторува двапати во b.

Множеството на MUM-ери за два геноми со употреба на стебло на суфикси се наоѓа во $O(n+m)$ време и линеарен простор. Множеството се сортира, од каде се издвојува најдолго подмножество на MUM-ери кои се појавуваат во ист редослед во геномА и геномБ.

За претходните пример секвенци, од множеството на MUM-ери се издвојува подмножеството на MUM-ери со максимална должина, кое ги вклучува совпаѓањата (1) и (2), (Сл.2.10). Совпаѓањето (3) не е вклучено во рамки на подмножеството, бидејќи истото е во пресек со совпаѓањето (2), (Сл. 2.10).



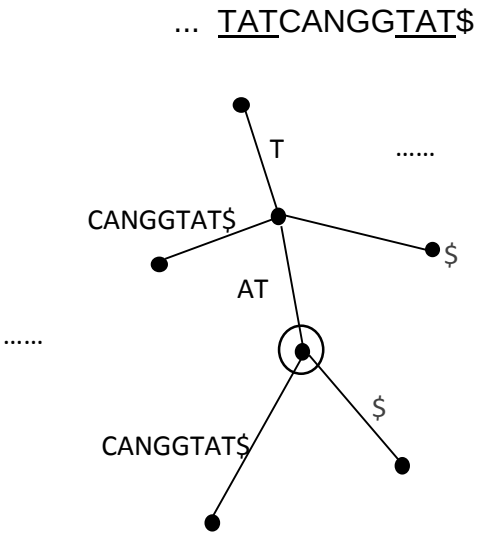
Слика 2.10. Единствени совпаѓања со максимална должина
Figure 2.10. Maximal Unique Matches

При затворање на празнините, пред да се испечати порамнувањето, се врши локална идентификација на: *долги додавања*, *повторувања*, *кратки мутирани фрагменти*, *тандемски повторувања* и *единични нуклеотидни полиморфизми*.

Од биолошки аспект, MUMmer нуди можност за идентификација на:

- единични нуклеотидни полиморфизми (*единични мутации опкружени со фрагменти на совпаѓање*);
- фрагменти со степен на дивергенција поголем од единичен нуклеотиден полиморфизам;
- фрагменти на додавања во еден од геномите, како последица на транспозиција или латерален трансфер од друг организам;
- Фрагменти на ДНК дуплирања на различни локации во еден од геномите;
- Кратки тандемски повторувања (*последователни повторувања на краток ДНК фрагмент*).

Освен MUMmer, стебло на суфикси користи и Програмата За Глобално Порамнување (*AVID*, Bray et al., 2003). Временски линеарен алгоритам за градба на стебло на суфикси за првпат предлага Gusfield, 1997. AVID, како и MUMmer, го употребува стеблото на суфикси за да се утврдат најдолги заеднички совпаѓања помеѓу две секвенци. AVID конструира стебло на суфикси за секвенца која се добива со соединување на секвенците кои се порамнуваат. Соединетите секвенци меѓусебно се одделени со делиметар „N“, (Сл. 2.11). Најдолгите повторувачки зборови во резултантната секвенца се најдолги заеднички совпаѓања помеѓу секвенците и истите се утврдуваат со пребарување на стеблото на суфикси, (Сл. 2.11). Од множеството на сите совпаѓања се бара подмножество на непреклопувачки и паралелни совпаѓања, при што на почеток се отфрлуваат совпаѓањата чија должина е помала од половина должина на совпаѓањето со максимална должина. Совпаѓањата ги избира изменета верзија на Smith-Waterman алгоритамот. За секое совпаѓање се пресметува различен резултат, кој зависи од должината на совпаѓање и од резултатот на порамнување на соседните региони, најчесто во должина до 10 базни положби.



Слика 2.11. Пребарување на стебло на суфикси за соединета секвенца
Figure 2.11. Joint sequence suffix tree search

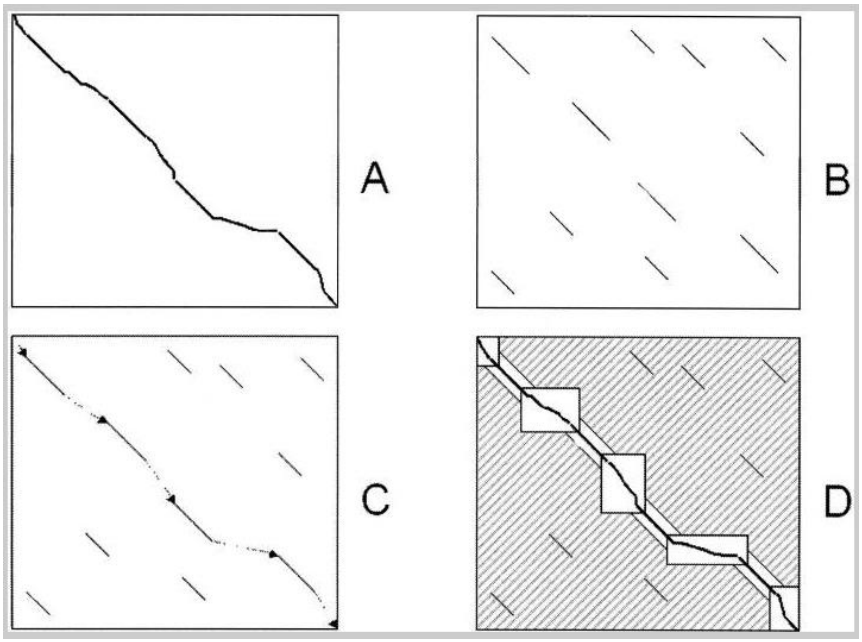
2.17. GLASS

GLASS (GLobal Alignment SyStem, Batzoglou et al., 2000) во споредба со MUMmer е побавен, но далеку почувствителен пристап. Извршувањето на GLASS започнува со идентификација на заеднички совпаѓања со должина k помеѓу две секвенци. Разгледувајќи го секое заедничко совпаѓање како одделен „*карактер*“, секвенците кои се порамнуваат се трансформираат во зборови од вакви карактери, во зависност од фреквенциите на појавување. Откако ќе се изврши порамнување со примена на динамичко програмирање, од порамнувањето се издвојуваат локални порамнувања со задоволителен резултат на порамнување ($> T$, T е прагова вредност за резултат на порамнување). Од новата листа на парови на совпаѓања се отстрануваат неконзистентните совпаѓања (совпаѓања чии положби се преклопуваат). Совпаѓањата со помала должина помеѓу порамнетите нуклеотиди се утврдуваат на рекурзивен начин, со примена на претходите чекори за помали вредности на $k, k \epsilon 20,15,12,9,8,7,6,5$, на што се должи зголемувањето на чувствителноста во споредба со MUMmer. На крај се порамнува непорамнетата ДНК содржина.

2.18. LALIGN

LALIGN (Brudno et al., 2003) е пристап за брзо и глобално порамнување на генетски секвенци, со чија примена може да се утврдат и подалечни хомологии. За разлика од MUMmer, AVID и GLASS, кои пребаруваат перфектни совпаѓања со максимална должина, што во основа ја ограничува нивната примена само на еволуциски блиски секвенци, LALIGN детектира локални порамнувања, кои покрај совпаѓања, вклучуваат и несовпаѓања. Локалните порамнувања се утврдуваат врз основа на повеќе кратки и нецелосни совпаѓања на зборови, со примена на алгоритмот *CHAOS* (Brudno et al., 2002). Од множеството на локални порамнувања со различни резултати на порамнување, LALIGN избира подмножество на последователни порамнувања со максимален резултат на порамнување. Избраното подмножество го определува просторот каде се бара решение, кое се добива со празнинско порамнување на фрагментите помеѓу

селектираните локални порамнувања. За извршување на празнинските порамнувања помеѓу последователните локални порамнувања, LALIGN го користи алгоритмот на Needleman и Wunsch.



Слика 2.12. Фази на извршување на LALIGN
Figure 2.12. LALIGN running phases

2.19. АЛГОРИТАМ ЗА ПРЕБАРУВАЊЕ БАЗИРАН НА СПОРЕДБА НА ДНК ФРАГМЕНТИ (DALIGN)

Концептот за парово порамнување, предложен од Моргенштерн (Morgenstern et al., 1996) се темели на споредување на сегменти, наместо базни парови. Сегментите кои се споредуваат имаат еднаква должина. Бидејќи сегментите образуваат дијагонали во рамки на регион со точки, овој пристап за порамнување е познат како дијагонално порамнување (DIALIGN) (Burkhard et al., 1998).

Потребно е да се најде множество на конзистентни дијагонали: $D_1 \dots D_k$ со максимален збир на тежини на дијагонали. Дијагоналите D_W и D_{W+1} се конзистентни ако и само ако последната база од D_W претходи на првата база од D_{W+1}. Веројатноста дијагонала D да содржи најмалку m совпаѓања може да се најде според равенката (2.8), каде $|D|$ е должина на дијагонала, p е веројатност за совпаѓање на две бази. Биолошки незначајните дијагонали се отфрлаат според равенка (2.9), каде T е кориснички дефиниран праг на отфрлање.

$$P(|D|, m) = \sum_{i=m}^{|D|} \frac{|D|}{i} p^i (1-p)^{(|D|-i)} \qquad (2.8)$$

$$w(D) = \begin{cases} -\ln\big(P(|D|, m)\big), \text{ако} -\ln\big(P(|D|, m)\big) > T \\ 0, \text{ инаку} \end{cases} \qquad (2.9)$$

Се утврдуваат дијагонали $D, w(D) > 0$ од позициски пар $(i-k, j-k)$ до (i, j), за $0 \leq k \leq \min(i-1, j-1)$. Ако со $\sigma(D)$ се означи максималниот збир на тежини на дијагонали до вклучување на дијагоналата $D, w(D) > 0$, тогаш $\sigma(D)$ го задоволува равенството (2.10), каде $s_{i,j} = \max\{s_{i-1,j}, s_{i,j-1}, \sigma(D_{i,j})\}$, $s_{i,j}$ се вредности од стандардна матрица на динамичко програмирање, додека $D_{i,j}$ е најдолгата дијагонала која завршува во точка (i, j).

$$\sigma(D) = s_{i-k-1,j-k-1} + w(D) \quad (2.10)$$

2.20. СУПЕР ПАРОВО ПОРАМНУВАЊЕ (SPA: Super Pairwise Alignment)

Супер Паровото Порамнување (SPA: Super Pairwise Alignment, Shen et al., 2002) е временски линеарен алгоритам за парово порамнување на хомологни секвенци, базиран врз примена на методи од веројатност и комбинаторика. Алгоритамот ги наоѓа положбите на кои се додаваат празнини, врз основа на процентот на сличност во рамки на прозорец со должина n. Прозорецот опфаќа по еден сегмент од секвенца, со еднаква должина. Споредувајќи две по две бази од прозорецот, може да се пресмета процентот на локална сличност, како количник помеѓу бројот на совпаѓања и должината на прозорецот. Празнини се додаваат секогаш кога со минимално поместување на дел од нуклеотидите, драстично се зголемува процентот на сличност, (Сл. 2.13). Процентот на сличност во рамки на прозорец со должина n се бележи со $r(x, y, k, n, t)$, каде: x и y се секвенци кои се порамнуваат, k е почетната положба на сегмент, n е должина на сегмент (должина на прозорец), t е број на единечни поместувања, кој може да биде позитивен, негативен или нула.

ACTGATTT….

CTGATT….

За n=4,

ACTGATTT….

CTGATT…., процент на сличност $\frac{1}{6} \times 100 = 16{,}7\%$

ACTGATTT….

CTGATT…., процент на сличност $\frac{6}{6} \times 100 = 100\%$

ACTGATTT….

_ CTGATT….,

Слика 2.13. Супер парово порамнување
Figure 2.13. Super Pairwise Alignment

За да се овозможи ефикасно пребарување на ДНК база на податоци, базата на податоци на почеток се индексира. Индексираната податочна структура бележи појавувања на ДНК фрагменти и ДНК зборови од база на податоци и истата се пребарува за да се најдат појавувања на недегенериран или делумно дегенериран ДНК прашалник. Денес постојат различни пристапи за индексирање и пребарување на генетска база на податоци, како од аспект на избор на тип на индексирана податочна структура, така и од аспект на избор на конкретна стратегија за пребарување на индексираната генетска содржина. Кога се работи со мали бази на податоци, како на пример прокариотска ДНК база на податоци, индексираната податочна структура се чува во главната меморија (RAM меморија). Кога се работи со масивни бази на податоци, какви што се еукариотските бази на податоци (на пример база на податоци за човечки геном), поради ограниченият капацитет на главната меморија, индексираната содржина се чува во датотека која се наоѓа на хард-диск на локална машина или оддалечен сервер. Табелите на пребарување, хеш-податочните структури, граф податочните структури, стеблата на суфикси и полињата на суфикси се најчест избор за тип на податочна структура за индексирање на генетска база на податоци.

2.21. MICA

MICA (Stokes et al., 2006) е пристап за брзо пребарување на масивна база на податоци на компјутери со ограничен капацитет на RAM меморија. За да се намали мемориската побарувачка, MICA ја дели ДНК секвенцата на фрагменти од по $2^{16}-1$ последователни нуклеотиди, бележејќи ја со $2B$ положбата на секој преклопувачки збор со должина k во фрагмент. Апсолутните положби на совпаѓањата се пресметуваат врз основа на релативните положби на индексираните зборови во фрагментите, пребарувајќи ја базата на податоци по делумно дегенериран или недегенериран прашалник, во зависност од должината на прашалникот. Кога се пребарува по прашалник чија должина е помала од k се врши пребарување по делумно дегенериран и издолжен прашалник, кој се добива со додавање на ‘N’ карактери до должина k (‘N’ може да биде било кој од четирите основни нуклеотиди: A,C,T,G), додека кога се пребарува по прашалник чија должина е поголема од k, прашалникот се разделува на конституенти со должина k, чии положби меѓусебно се споредуваат. Кога се пребарува по ДНК прашалник чија должина изнесува k, релативните положби на совпаѓањата по фрагмент се преведуваат во апсолутни положби на совпаѓања.

2.22. TANDEM REPEATS FINDER

Tandem Repeats Finder (Benson, 1999) е програма за анализа на ДНК секвенца со која можат да се утврдат тандемски повторувања, без да се специфицира на влез структурата на повторувањето и растојанието помеѓу повторувањата во тандемот. По дефиниција, тандемско повторување е повеќекратно повторување на краток ДНК фрагмент на растојание d. Пример за тандемско повторување е CTACTACTA, каде ДНК фрагментот CTA последователно се појавува трипати на растојание $d=0$. Алгоритмот работи на принцип на скенирање на ДНК секвенца со лизгачки прозорец со должина k, бележејќи ги почетните положби на појавување на секој збор од секвенцата во

должина од k нуклеотиди. Ако $\{p_1, p_2, \dots, p_{l-1}, p_l\}$ е листа на почетни положби на сите појавувања на збор , тогаш секое растојание $d = p_i - p_j$ е кандидат за оддалечност помеѓу повторувањата во рамки на тандем. Тандемите се утврдуваат како копии на збор w на еднакво растојание $d = .. = p_k - p_{k-1} = p_{k+1} - p_k = p_{k+2} - p_{k+1} = ..$.

2.23. TEIRESIAS

TEIRESIAS (Rigoutsos, 1998) е комбинаторички алгоритам за детекција на ДНК шаблони, кои се појавуваат во најмалку a секвенци, каде a е кориснички зададен параметар. Алгоритмот се извршува во две фази, и тоа: *фаза на скенирање* и *фаза на конволуција*. Во фазата на скенирање се детектираат елементарни ДНК фрагменти, кои парцијално задоволуваат зададен ДНК шаблон. Овие фрагменти во фазата на конволуција со примена на суфикс-префикс соединување се издолжуваат, со што се детектираат совпаѓања со максимална должина кои во целост го задоволуваат ДНК шаблонот. Така, на пример, за да се најдат појавувањата од тип A..CT.C, каде ‘.’ означува „било кој карактер“, може да се изврши пребарување по елементарни A..CT и CT.C ДНК темплејти, со чие суфикс-префикс соединување по CT во фазата на конволуција се детектираат сите појавувања кои во целост го задоволуваат зададениот ДНК шаблон. За идентификација на максималните шаблонски совпаѓања, алгоритмот не извршува парово порамнување, ниту го енумерира целокупниот простор каде се бара решение, што во принцип влијае позитивно на ефикасноста на извршување на алгоритмот.

2.24. SEQUENCE SEARCH TREE

SST: Sequence Search Tree (Giladi et al., 2002) е алгоритам за детекција на приближни совпаѓања од база на секвенци, кој се извршува во $O(logn)$ време, каде n е големина на базата на секвенци. Со издвојување на преклопувачки прозорци кои вклучуваат по W нуклеотиди од прашалник секвенца и база на податоци, каде за степенот на преклопување Δ важи $5 \leq \Delta \leq \frac{W}{2}$, SST може да детектира меѓусебно слични прозорци. Секој прозорец се пресликува во 4^k димензинален вектор, кој ги содржи фреквенциите на појавувања на преклопувачките зборови во должина од k нуклеотиди од прозорецот. Пресликувањето се врши според равенката (2.11), каде: $M(A) = 0, M(C) = 1, M(G) = 2, M(T) = 3$.

$$I(a_1 a_2 \dots a_{k-1} a_k) = \sum_{i=1}^{k} M(a_i) \times 4^{k-1}, \quad a_i \in \{A, C, T, G\} \quad (2.11)$$

На пример, за $k = 2$ и должина на прозорец $W = 6$, прозорецот ACGCTT се пресликува во вектор: $(0100001101000001)^T$, Табела 2.1. Вредностите во векторот се утврдуваат врз основа на бројот на појавувања на преклопувачките зборови: AC,CG,GC,CT и TT од прозорецот, кои вклучуваат по $k = 2$ нуклеотиди.

Во вакви услови, проблемот на пронаоѓање на слични прозорци помеѓу прашалник секвенца и секвенца од база на секвенци се сведува на проблем на пронаоѓање на најблиски соседи во 4^k димензионален векторски простор, за што

може да се примени некој од постојните алгоритми за податочно кластерирање, како што е k-Means кластерирањето (MacQueen, 1967).

Табела 2.1. Вектор на пресликување за прозорецот: ACGCTT
Table 2.1. Mapping vector for window: ACGCTT

$k-$збор	Фрек.
AA	0
AC	1
AG	0
AT	0
CA	0
CC	0
CG	1
CT	1
GA	0
GC	1
GG	0
GT	0
TA	0
TC	0
TG	0
TT	1

2.25. OASIS

OASIS (Meek et al., 2003) базата на податоци ја индексира во стебло на суфикси, со чие изминување во фазата на пребарување се пронаоѓаат совпаѓања на прашалник секвенца. Во фазата на пребарување, OASIS применува A* пребарување на стебло на суфикси базирано на динамичко програмирање. Добиените резултати се подредени во опаѓачки редослед по резултат на порамнување, што во принцип го прави овој пристап подобен за online имплементација. За разлика од OASIS, *Vmatch* (Abouelhoda, et al., 2004) употребува подобрено поле на суфикси. Во основа, подобрените полиња на суфикси се состојат од четири конституенти: *поле на суфикси*, *поле на најдолги заеднички префикси*, *поле на нитки* и *поле на суфикс линкови* и истите ги нудат истите можности за пребарување како и стеблата на суфикси. Стеблата на суфикси и подобрените полиња на суфикси се чест избор за тип на податочна структура за индексирање на база на податоци, бидејќи истите се конструираат во $O(n)$ време и зафаќаат $O(nlogn)$ простор. Имплементацијата на стебло на суфикси според Kurtz, (Kurtz, 1999) зафаќа 17.25 B по нуклеотид, додека

подобрените полиња на суфикси најчесто зафаќаат 4-8 B по нуклеотид. Наместо да се индексира секоја положба од база на податоци, Khan (Khan et al., 2009) предлага пристап за индексирање на секоја k-та положба, со што значајно се намалува мемориската побарувачка, што од една страна може да претставува ограничувачки фактор во случаите кога се индексираат долги ДНК секвенци, како што е човечкиот геном. Алатката essaMEM (Vyverman et al., 2013) го оптимизира претходниот метод.

2.26. ПРИСТАПИ БАЗИРАНИ НА КОМПРЕСИЈА НА ИНДЕКСИ (Ferragini-Manzini и трансформација на Burrows-Wheeler)

Постои група на алгоритми, кои со примена на техники за компресирање на индекси, успеваат просторната комплексност да ја намалат на $O(n)$ бити. Примери за такви техники се *Ferragini-Manzini* индексирањето (Ferragina et al., 2000), компресираните полиња на суфикси (Grossi et al., 2005) и индексирањето со примена на *Burrows-Wheeler* трансформација (Burrow et al., 1994). За ДНК секвенци, индексирањето со примена на Burrows-Wheeler трансформација е најефикасно, со просторна побарувачка која е помала од 0.3 B по нуклеотид. Примената на овој пристап за индексирање на човечки геном бара 1GB простор од RAM меморијата, што во основа го прави овој пристап применлив за извршување на персонален компјутер. Lippert et.al 2005, Lippert 2005 Burrows-Wheeler трансформација ја применуваат во комбинација со компресирано поле на суфикси.

2.27. SSAHA

SSAHA (Ning et al., 2001) и методот на *Reneker u Shyu* (Reneker et al., 2005) се хеш-базирани пристапи за индексирање и пребарување на ДНК база на податоци. Истите во фазата на индексирање пресликуваат ДНК зборови во должина од k нуклеотиди. Пресликувањата (клучевите) се целобројни вредности, кои се добиваат со примена на хеш-функција за енкрипција на текстуални податоци над азбука $\Sigma = \{A, C, T, G\}$. Совпаѓањата во базата на податоци при пребарување по ДНК прашалник се наоѓаат со пребарување на множеството на генерирани пресликувања (клучеви), кои се чуваат хеш-индексирана податочна структура (хеш-табела кај SSAHA и индексирана датотека кај методот на Reneker и Shyu).

SSAHA е хеш-базиран пристап за индексирање и пребарување на ДНК база на податоци. На почеток, базата на податоци $S = \{S_1, S_2, \dots, S_{n-1}, S_n\}$ се индексира во хеш-табела со 4^k клучеви. Секоја комбинација од k последователни нуклеотиди $w_i = a_{i,1}a_{i,2} \dots a_{i,k-1}a_{i,k}$ над азбука $\Sigma = \{A, C, T, G\}$, односно секој ДНК збор од $\underbrace{A \dots A}_{k}$ до $\underbrace{T \dots T}_{k}$ се енкриптира во клуч $f(w_i)$ според равенката (2.12), кодирајќи ја секоја база со целобројна вредност: 0, 1, 2 или 3. Клучевите $f(w_i)$ се распределени во опсег: $0 \leq f(w_i) \leq 4^k - 1$ и истите референцираат појавувања на ДНК зборови $w_i = a_{i,1}a_{i,2} \dots a_{i,k-1}a_{i,k}$ од базата на секвенци. Појавувањето на збор $w_i = a_{i,1}a_{i,2} \dots a_{i,k-1}a_{i,k}$ во ДНК секвенца S_x на почетна положба y е определено со податочен пар од вид: (x, y) кон кој покажува

клуч $f(w_i)$. Во зависност од бројот на појавувања на ДНК збор $w_i = a_{i,1}a_{i,2} \dots a_{i,k-1}a_{i,k}$, клуч $f(w_i)$ може да покажува кон $0,1,2, \dots$ податочни парови.

$f(A) = 0$

$f(C) = 1$

$f(G) = 2$

$f(T) = 3$

$f\left(w_i: a_{i,1}a_{i,2} \dots a_{i,k-1}a_{i,k}\right) = \sum_{j=1}^{k} f(a_{i,j}) \times 4^{j-1}, a_{i,j} \epsilon \{A, C, T, G\}$ (2.12)

Хеш-табелата за пример базата на податоци $S = \{S_1: ACTGCC, S_2: TCACCC, S_3: AATCCG\}$, за $k = 2$, е дадена во Табела 2.2. Од секоја секвенца се издвојуваат непреклопувачки зборови со должина $k = 2$: AC (1,1), TG (1,3), CC (1,5), TC (2,1), AC (2,3), CC (2,5), AA (3,1), TC (3,3) и CG (3,5). Издвоените непреклопувачки зборови ги определуваат податочни парови: (x, y), каде y е почетна положба на непреклопувачки збор во секвенца чиј индекс во рамки на базата на секвенци е x. Така, на пример, зборот AC се појавува во секвенците S_1 и S_2. Појавувањето на зборот AC во S_1 се бележи со податочен пар: (1,1), додека појавувањето на истиот збор во секвенцата S_2 се бележи со податочен пар: (2,3), бидејќи зборот AC во S_2 се наоѓа на почетна положба 3.

Табела 2.2. SSAHA хеш-табела за пример базата на секвенци S
Table 2.2. SSAHA hash-table for the sample DNA database S

Збор w	$f(w)$	(x, y)
AA	0	(3,1)
CA	1	
GA	2	
TA	3	
AC	4	(1,1), (2,3)
CC	5	(1,5), (2,5)
GC	6	
TC	7	(2,1), (3,3)
AG	8	
CG	9	(3,5)
GG	10	
TG	11	(1,3)
AT	12	
CT	13	
GT	14	
TT	15	

За да се најдат парцијални или целосни совпаѓања на ДНК прашалник во база на секвенци, од прашалник секвенцата q се издвојуваат: $|q|-k+1$ преклопувачки зборови q_i, $1 \le i \le |q|-k+1$, каде $|q|$ е должина на ДНК прашалник. Ако хешот на $q_i, f(q_i)$ покажува кон барем еден податочен пар (x,y) во хеш-табелата, тогаш секој податочен пар (x,y) референциран од $f(q_i)$ се проширува во податочна тројка од тип: $(x, y-t, y)$, каде со t е означена почетната положба на зборот q_i во рамки на ДНК прашалникот q. Со примена на овој концепт за сите преклопувачки зборови q_i, се добива листа на совпаѓања H составена од податочни тројки од тип: $(x, y-t, y)$.

Во Табела 2.3, последна колона, е дадена листата на совпаѓања H, ако на пример базата на секвенци S се пребарува по ДНК прашалник q:TGCC. Вкупно $|q|-k+1=4-2+1$ преклопувачки зборови со должина 2 се издвојуваат од ДНК прашалникот: TG(0), GC(1) и CC(2) (броевите во заградите ги означуваат почетните положби на зборовите во рамки на прашалникот). Единаесеттиот клуч е хеш на зборот TG(0), $f(TG)=f(T)\times 4^0+f(G)\times 4^1=3+2\times 4=11$. Истиот покажува кон податочен пар (1,3), од каде се пресметува податочна тројка (1, 3-0, 3)=(1, 3, 3) која се додава во листата на совпаѓања H. Хеш на зборот GC(1) е клуч $f(GC)=f(G)\times 4^0+f(C)\times 4^1=2+1\times 4=6$ кој не референцира податочен пар, додека хеш-вредноста на зборот CC(2), $f(CC)=f(C)\times 4^0+f(C)\times 4^1=1+1\times 4=5$ е клуч кој ги референцира паровите: (1,5) и (2,5), од каде се пресметуваат податочните тројки: (1,5-2,5)=(1,3,5), (2,5-2,5)=(2,3,5), кои исто така се додаваат во листата на совпаѓања H.

Табела 2.3. Листа на совпаѓања H
Table 2.3. List of hits H

Збор w	$f(w)$	(x,y)	Листа на совпаѓања H
AA	0	(3,1)	
CA	1		
GA	2		
TA	3		
AC	4	(1,1), (2,3)	
CC	5	(1,5), (2,5)	(1,3,5), (2,3,5)
GC	6		
TC	7	(2,1), (3,3)	
AG	8		
CG	9	(3,5)	
GG	10		
TG	11	(1,3)	(1,3,3)
AT	12		
CT	13		
GT	14		
TT	15		

Листата на совпаѓања H се сортира последователно по $x, y-t$ и y вредностите, по што се добива сортирана листа M: (1,3,3), (1,3,5), (2,3,5). Совпаѓањето на ДНК прашалникот е определено со последователно множество на тројки: $(x_1, y_1 - t_1, y_1) \dots (x_{\frac{|q|}{k}}, y_{\frac{|q|}{k}} - t_{\frac{|q|}{k}}, y_{\frac{|q|}{k}})$, за кои важи $x_1 = \cdots = x_{\frac{|q|}{k}}$, $y_1 - t_1 = \cdots = y_{\frac{|q|}{k}} - t_{\frac{|q|}{k}}$.

За конкретниот пример, совпаѓањето на ДНК прашалникот q:TGCC во рамки на секвенцата S_1 на положба 3, е определно со множеството на последователни податочни тројки: (1,3,3), (1,3,5).

2.28. АЛГОРИТАМ НА RENEKER И SHYU

Reneker и *Shyu* решаваат дел од недостатоците на SSAHA. Еден од недостатоците на SSAHA е неправењето на разлика помеѓу ДНК зборови со идентичен завршеток (идентичен суфикс), на кој претходи повторување од тип: $\mathrm{AA} \dots \mathrm{AA}$, со различна должина. Ваквиот недостаток се должи на нултото пресликување на нуклеотидот A (Аденин), што може да доведе до ситуација два различни ДНК зборови да имаат идентична хеш-вредност, како на пример w_1: AACT и w_2: ACT, $f(w_1: AACT) = f(w_2: ACT) = f(CT)$ и како такви да се земат за еквивалентни. Изборот на ненултата шема за базно пресликување, дефинирана на начин како кај Reneker и Shyu: $f(A) = 1, f(T) = 2, f(G) = 3, f(C) = 4$ го решава овој проблем, односно се оневозможува постоење на различни ДНК зборови со идентичен хеш.

Поради тоа што SSAHA индексира непреклопувачки зборови во должина од k нуклеотиди, во фаза на пребарување не постои можност за детекција на преклопувачки совпаѓања. За да се реши овој проблем, Reneker и Shyu ги индексираат сите зборови од базата на секвенци, независно од тоа дали станува збор за преклопувачки или непреклопувачки ДНК збор, со што се овозможува детекција на преклопувачки и непреклопувачки совпаѓања во фазата на пребарување по ДНК прашалник.

При пребарување на базата на секвенци по ДНК прашалник чија должина е помала од должината на индексирање k, SSAHA не детектира совпаѓања. Парцијално решение на овој проблем предлагаат Reneker и Shyu. Според нив, за да се најдат совпаѓања во базата на секвенци при пребарување по ДНК прашалник чија должина е помала од должината на индексирање (k), ДНК парашалникот треба да се издолжи до k нуклеотиди, со додавање на $k - |q|$ A (Аденини) и C (Цитозини) од левата страна од прашалникот. Во вакви услови, бидејќи $f(A) = 1 = \min\{f(A), f(T), f(G), f(C)\}$ и $f(C) = 4 = \max\{f(A), f(T), f(G), f(C)\}$, совпаѓањата се наоѓаат како суфикси во рамки на пресликани зборови чии хеш-вредности се наоѓаат во опсег помеѓу: $f(\underbrace{A \dots A}_{k-|q|} q)$и $f(\underbrace{C \dots C}_{k-|q|} q)$.

Поголем дел од совпаѓањата со должина помала од k можат да се најдат на овој начин, но не и совпаѓањата кои се наоѓаат на почетни положби: $p \leq k - |q|$, бидејќи Reneker и Shyu го бараат ДНК прашалникот само како суфикс во рамки на ДНК пресликувања, но не и како префикс.

Тоа може да се разгледа и дискутира на пример. За должина на индексирање $k = 2$, нека ДНК секвенцата s: ACTGAC се пребарува по ДНК прашалник q: A. Reneker и Shyu за секој пресликан збор чуваат податочен пар од тип <*location, PID*>, каде *location* е почетната положба на пресликан збор, изразена во однос на почетокот на секвенцата и *PID* е идентификатор на најблискиот ген до локацијата од каде е пресликан зборот. Ако должината на секвенцата изнесува n, тогаш во листа *L* се чуваат $n - k + 1$ податочни парови од тип <*location*, *PID*>. Листата L е сортирана по вредност на пресликување. За пример секвенцата s, множеството на пресликувања подредени по вредност на пресликување е: S={GA $(f(GA) = 7)$, CT $(f(CT) = 12)$, TG $(f(TG) = 14)$, AC$(f(AC) = 17)$, AC $(f(AC) = 17)$}. Под претпоставка дека најблискиот ген до првите три нуклеотиди од секвенцата е ген со ID 1234, додека најблискиот ген до последните три нуклеотиди е ген со ID 5678, на Слика 2.14 е прикажана структурата на листата L за пример секвенцата s: ACTGAC.

<0003, 5678>|<0001, 1234>|<0002, 1234>|<0000, 1234><0004, 5678>

0 8 16 24

Слика 2.14. Структура на листата L
Figure 2.14. Structure of the list L

Листата L е индексирана од поле A кое содржи 4^k записи. Форматот на секој запис е од тип: *key*|*offset*, *occurrences*, *bytes*, *numbersize*, каде *offset* е почетната положба на првиот пар <*location, PID*> за клуч од L, *occurrences* е број на појавувања на клуч во рамки на секвенцата, *bytes=occurrences*$\times 8$ и *numbersize* е број на бајти во L по <*location, PID*> пар.

За пример секвенцата, структурата на A е прикажана во Табела 2.4. Издолжувања на ДНК прашалникот q:A до должина k се добиваат со додавање на $k - |q|$ A(Аденини) и C(Цитозини) од левата страна на прашалникот q, $q_{low} = \underbrace{A \ldots A}_{k-|q|} q = Aq = AA$; $q_{high} = \underbrace{C \ldots C}_{k-|q|} q = Cq = CA$. Совпаѓањата се пронаоѓаат со исчитување на $r = \frac{q_{high}.offset - q_{low}.offset}{low.numbersize} + high.occurences \times 2 = \frac{8-0}{4} + 0 \times 2 = 2$ бајти од L, започнувајќи од $f(q_{low}) - f\left(\underbrace{A \ldots A}_{k}\right) = f(AA) - f(AA) = 5 - 5 = 0$. Резултатот од читањето е зборот GA: <0003, 5678>, кој го содржи ДНК прашалникот A како суфикс. Совпаѓањето на почетна положба 0 не може да се детектира со примена на методот на Reneker и Shyu, бидејќи во рамки на зборот AC, ДНК прашалникот A (Аденин) не се појавува како суфикс, туку како префикс.

Табела 2.4. Структура на поле A
Table 2.4. Structure of the array A

Збор	*клуч*	*offset, occurrences, bytes, numbersize*
AA	5	0,0,0,4
TA	6	0,0,0,4
GA	7	0,1,8,4
CA	8	8,0,0,4
AT	9	...
…		

3. ОПИС НА ПРЕДЛОЖЕНИТЕ АЛГОРИТМИ

3.1. АЛГОРИТАМ ЗА БРЗО И МЕМОРИСКИ ЕФИКАСНО ПОРАМНУВАЊЕ НА ДНК СЕКВЕНЦИ

ДЕФИНИЦИЈА НА ПРОБЛЕМ: Нека $a = a_1a_2 \dots a_{n-1}a_n$ и $b = b_1b_2 \dots b_{m-1}b_m$ се две ДНК секвенци, за кои важи: $a_i, b_j \in \{A, C, T, G\}, 1 \leq i \leq n, 1 \leq j \leq m$ и $n \geq m$. Потребно е да се најде оптимално локално порамнување на ДНК секвенците a и b: $A_o = R_1R_2 \dots R_{k-1}R_k$, каде: $R_i, 1 \leq i \leq k$ означува фрагмент на совпаѓање помеѓу a и b.

Дефиниција 1: Локално порамнување на ДНК секвенци $a = a_1a_2 \dots a_{n-1}a_n$ и $b = b_1b_2 \dots b_{m-1}b_m$ е порамнување на фрагмент од секвенцата a, $a_f = a_sa_{s+1} \dots a_{|a_f|-2}a_{|a_f|-1}$ со фрагмент од секвенцата b, $b_f = b_rb_{r+1} \dots b_{|b_f|-2}b_{|b_f|-1}$, каде секој нуклеотид a_i од a_f е порамнет со совпаѓачки или несовпаѓачки нуклеотид b_i од b_f.

Дефиниција 2: Порамнетите фрагменти a_f и b_f имат еднаква должина, $|a_f| = |b_f|$.

Нека $a: CTCGGAC$ и $b: GTAGGT$ се две пример секвенци. Локално порамнување на a и b може да се образува со порамнување на фрагментите: $a_f = TCGG$ и $b_f = TAGG$, (Сл 3.1). Должината на фрагментите кои се порамнуваат е еднаква, $|a_f| = |b_f| = 4$.

Слика 3.1. Локално порамнување на a и b
Figure 3.1. Local alignment of a and b

Дефиниција 3: Порамнувањето на a_f и b_f вклучува $k, k \geq 1$ последователни фрагменти на совпаѓање $R_\xi, 1 \leq \xi \leq k$, одделени со $k-1$ фрагменти на несовпаѓање.

Локалното порамнување за пример секвенците a и b, вклучува два фрагменти на совпаѓање: $R_1: T$ и $R_2: GG$, кои меѓусебно се одделени со C-A (цитозин-аденин) фрагмент на несовпаѓање.

Дефиниција 4: Фрагмент на совпаѓање R_ξ е низа од $|R_\xi|$ последователни базни совпаѓања во a и b.

Дефиниција 5: Должината на фрагмент на совпаѓање $|R_\xi|$ е број на нуклеотиди во R_ξ .

Дефиниција 6: Фрагмент на несовпаѓање е низа од последователни базни несовпаѓања.

Дефиниција 7: Два последователни региони на совпаѓање R_ξ и $R_{\xi+1}$, меѓусебно се одделни со фрагмент на несовпаѓање $diff(R_\xi, R_{\xi+1})$, кој вклучува најмалку едно базно несовпаѓање, односно $|diff(R_\xi, R_{\xi+1})| \geq 1$.

Должината на фрагментот на совпаѓање $R_1{:}T$ изнесува 1, додека должината на фрагментот на совпаѓање $R_2{:}GG$ изнесува 2, $|R_1| = 1, |R_2| = 2$. Фрагментите на совпаѓање T и GG се два последователни фрагменти на совпаѓање, меѓусебно одделени со C-A (цитозин-аденин) фрагмент на несовпаѓање, $|diff(R_1, R_2)| = 1$.

Дефиниција 8: Нека $\Sigma A = \{A_1, A_2, \dots, A_{|\Sigma A|-1}, A_{|\Sigma A|}\}$ е множество на ΣA различни локални порамнувања на a и b. Локалното порамнување A_o, $A_o \epsilon \Sigma A$ е оптимално, ако за секое порамнување $A_r, A_r \neq A_o$ важи: $f(A_o) \geq f(A_r)$, каде $f(A)$ е резултат на локално порамнување A.

Дефиниција 9: За локално порамнување $A = R_1 R_2 \dots R_{k-1} R_k$, резултатот на порамнување $f(A)$ е линеарна функција од награди и казни и истиот се пресметува согласно равенката (3.1), каде $\mu, \mu > 0$ е награда за порамнување на совпаѓачки нуклеотиди, додека $\delta, \delta < 0$ е казна за порамнување на несовпаѓачки нуклеотиди.

$$f(A) = \mu \sum_{i=1}^{k} |R_i| + \delta \sum_{j=1}^{k-1} |diff(R_j, R_{j+1})| \quad (3.1)$$

Со порамнување на фрагментите $a_f{:}GGA$ и $b_f{:}GTA$, (Сл. 3.2), се образува локално порамнување, различно од претходното. Ова порамнување е само едно од множеството на порамнувања ΣA. Во услови на постоење на множество на локални порамнувања ΣA, се поставува прашањето на избор на порамнување на ДНК фрагменти со највисок степен на сличност, односно се поставува прашањето на избор на оптимално порамнување.

Слика 3.2. Можно локално порамнување на a и b
Figure 3.2. Candidate for local alignment of a и b

Оптималното порамнување, според *Дефиниција 8*, е порамнување со максимален резултат на порамнување од множеството на порамнувања ΣA, пресметан како линеарна функција од награди и казни според равенката (3.1).

Резултатот на локалното порамнување $A(TCGG;TAGG)$, се пресметува според равенката (3.1), и истиот изнесува $f(\ A(TCGG;TAGG)) = 2 \times (|R_1| + |R_2|) - 1 \times |diff(R_1, R_2)| = 2 \times (1+2) - 1 \times 1 = 5$, воведувајќи метрика на доделување награда $\mu = 2$ и казна $\delta = -1$. Резултатот на локалното порамнување $A(GGA;GTA)$, изнесува $f\big(A(GGA;GTA)\big) = 2 \times (|R_1| + |R_2|) - 1 \times |diff(R_1, R_2)| = 2 \times (1+1) - 1 \times 1 = 3$.

Оптимално локално порамнување за пример секвенците $a{:}\,CTCGGAC$ и $b{:}\,GTAGGT$ е порамнување од множеството на локални порамнувања $\Sigma A = \{A(T;T), A(G;G), A(TCGG;TAGG), A(G;G), A(GGA;GTA), A(G;G)\}$, за кое е задоволен условот (3.2).

$A_o = A \epsilon \Sigma A | f(A) = \max\{f(A_i)\}, 1 \le i \le \Sigma A$ (3.2)

Резултатите на порамнувањата од множеството ΣA се: $f\big(A(G;G)\big) = f\big(A(T;T)\big) = 2$, $f\big(A(GGA;GTA)\big) = 3$ и $f\big(A(TCGG;TAGG)\big) = 5$, врз основа на што за оптимално локално порамнување се избира порамнувањето $A(TCGG;TAGG)$, со максимален резултат на порамнување 5.

Множеството на локални порамнувања ΣA, може да се образува со поместување на пократката секвенца b, долж подолгата a, согласно редоследот на поместување на Сл. 3.3. При секое поместување се образува фрагмент на преклопување (на Слика 3.3 фрагментите на преклопување се означени со правоаголници) со должина $l, 1 \le l \le m$. Максималниот број на поместувања на пократката секвенца b долж подолгата a, што соодветствува на максималниот број на фрагменти на преклопување со различна содржина, изнесува $n - m + 1$. Со споредба на паралелните нуклеотиди во рамки на фрагмент на преклопување, може да се утврдат $k, k \ge 0$ фрагменти на совпаѓање R_ξ. За поместување, каде во рамки на фрагмент на преклопување се утврдени $k, k \ge 1$ фрагменти на совпаѓање, може да се формираат $\frac{k(k+1)}{2}$ различни локални порамнувања, со соединување на $\gamma = 1,2,\dots,k-1,k$ последователни фрагменти на совпаѓање.

Дефиниција 10: Оптимално локално порамнување за поместување s е порамнување $A_{s,o}$ со максимален резултат на порамнување од $\frac{k(k+1)}{2}$ различни локални порамнувања, добиени со соединување на $\gamma = 1,2,\dots,k-1,k$ последователни фрагменти на совпаѓање.

Дефиниција 11: Оптимално локално порамнување за секвенците a и b е оптимално порамнување A_o за поместување s со максимален резултатот на порамнување од оптималните порамнувања за поместувања $A_{s,o}, 1 \le s \le n + m - 1$.

Ако со $A_{s,o}$ се означи оптималното локално порамнување за поместување s, додека со A_o се означи оптималното локално порамнување за секвенците a и b, тогаш за $A_{s,o}$ и A_o, важат равенките (3.3) и (3.4) соодветно, каде $A_{s,i}, 1 \le i \le \frac{k(k+1)}{2}$ означува локално порамнување за поместување s.

$$A_{s,o} = A \in \Sigma A_s | f(A) = \max\{f(A_{s,1}), f(A_{s,2}), \dots, f\left(A_{s,\frac{k(k+1)}{2}-1}\right), f\left(A_{s,\frac{k(k+1)}{2}}\right)\} \quad (3.3)$$

$$A_o = A \epsilon \Sigma A | f(A) = \max\{f(A_{1,o}), f(A_{2,o}), \dots, f(A_{n+m-2,o}), f(A_{n+m-1,o}) \quad (3.4)$$

$$\begin{array}{l} \\ b_1 b_2 \dots b_{m-1} \boxed{\begin{matrix} a_1 \\ b_m \end{matrix}} a_2 \dots a_{n-1} a_n \\ \\ b_1 b_2 \dots \boxed{\begin{matrix} a_1 & a_2 \\ b_{m-1} & b_m \end{matrix}} \dots a_{n-1} a_n \\ \dots\dots \\ a_1\ a_2 \dots \boxed{\begin{matrix} a_{n-1} & a_n \\ b_1 & b_2 \end{matrix}} \dots b_{m-1} b_m \\ a_1\ a_2 \dots a_{n-1} \boxed{\begin{matrix} a_n \\ b_1 \end{matrix}} b_2 \dots b_{m-1} b_m \end{array}$$

Слика 3.3. Редослед на поместување на пократката секвенца b долж подолгата a

Figure 3.3. Shifting the shorter sequence b over the longer sequence a

Во фаза на извршување во меморија се чуваат два вектори: $v_{s,o}$ и v_o. Векторот $v_{s,o}$ го бележи оптималното порамнување за поместување s, додека векторот v_o го бележи оптималното порамнување до поместување s. Податочната структура на векторите $v_{s,o}$ и v_o е од тип: $[(p_{a,1}, p_{b,1}, l_1), (p_{a,2}, p_{b,2}, l_2), \dots, (p_{a,k-1}, p_{b,k-1}, l_{k-1}), (p_{a,k}, p_{b,k}, l_k)]$ каде секоја податочна тројка $(p_{a,\xi}, p_{b,\xi}, l_\xi)$ соодветствува на фрагмент на совпаѓање $R_\xi, 1 \leq \xi \leq k$. Почетната положба на R_ξ во секвенцата a е $p_{a,\xi}$, почетната положба на истиот фрагмент во секвенцата b е $p_{b,\xi}$, додека l_ξ е должина на R_ξ .

Пред секое наредно поместување, се проверува дали резултатот на оптималното порамнување за поместување s, $f(A_{s,o})$ е поголем од резултатот на оптимално порамнување до поместување s, $f(A_o)$. Ако $f(A_{s,o}) > f(A_o)$, содржината на векторот v_o се заменува со содржината на векторот $v_{s,o}$, $v_o \leftarrow v_{s,o}$. Освен содржината на векторот v_o се менува и резултатот на оптимално порамнување, $f(A_o) \leftarrow f(A_{s,o})$.

По $n - m + 1$ единечни поместувања на пократката секвенца b долж подолгата секвенца a, оптималното локално порамнување за секвенците a и b го определува векторот v_o. Со примена на Теорема 3.1, бројот на поместувања може да се намали од $n - m + 1$ на $n + m - \left[\frac{f_{max}}{\mu}\right]$, а сепак да се најде оптималното порамнување, со што се подобрува временската комплексност на алгоритмот.

Теорема 3.1. Не постои порамнување во рамки на преклопување за поместување s, $s \geq n + m - \left\lceil \frac{f_{max}}{\mu} \right\rceil$, со поголем резултат на порамнување од f_{max}, каде f_{max} е резултат на оптимално локално порамнување до поместување s.

Доказ: Неравенството $s \geq n + m - \left\lceil \frac{f_{max}}{\mu} \right\rceil$ може да се запише како $\left\lceil \frac{f_{max}}{\mu} \right\rceil \geq n + m - s$. Ако левата и десната страна на неравенството се помножат со $\mu, \mu > 0$ се добива $\mu \left\lceil \frac{f_{max}}{\mu} \right\rceil \geq \mu(n + m - s)$. Од претходното неравенство и неравенството $f_{max} = \mu \frac{f_{max}}{\mu} \geq \mu \left\lceil \frac{f_{max}}{\mu} \right\rceil$ следи $f_{max} = \mu \frac{f_{max}}{\mu} \geq \mu \left\lceil \frac{f_{max}}{\mu} \right\rceil \geq \mu(n + m - s) = f$, каде f е максимален можен резултат на порамнување за поместување s, $s \geq n + m - \left\lceil \frac{f_{max}}{\mu} \right\rceil$.

Согласно Теорема 3.1, последните $\left\lceil \frac{f_{max}}{\mu} \right\rceil$ поместувања на пократката секвенца долж подолгата се редундантни и како такви истите не е неопходно да се извршат, бидејќи должините на фрагментите на преклопување се помали или еднакви на $\left\lceil \frac{f_{max}}{\mu} \right\rceil$, каде во никој случај не може да се најде локално порамнување со резултат на порамнување поголем од f_{max}.

Со намалување на бројот на поместувања, се намалува просторот каде се бара оптимално решение, односно се намалува бројот на генерирани локални порамнувања – кандидати за оптимално локално порамнување, со што се намалува и времето на извршување на алгоритмот.

СТУДИЈА НА СЛУЧАЈ: Нека a: CTCGGAC и b:GTAGGT се две пример секвенци, за кои се бара оптимално локално порамнување. За метрика на порамнување се избира: $\mu = +2$ (награда за порамнување на еквивалентни нуклеотиди) и $\delta = -1$ (казна за порамнување на различни нуклеотиди).

Во согласност со концепциската поставеност на алгоритмот, пократката секвенца се поместува долж подолгата секвенца, со што се образуваат фрагменти на преклопување со должини помеѓу 1 и $m = |b| = 6$, Табела 3.1. Споредувајќи ги паралелните нуклеотиди во рамки на фрагментите на преклопување, се бараат фрагменти на совпаѓање, кои учествуваат во конструкцијата на локалните порамнувања.

Почетно векторот на оптимално локално порамнување е празен, $v_0 = []$, односно $f(A_o) = 0$. Ако се споредат паралелните бази за првото поместување, се утврдува Цитозин-Тимин (C-T) базно несовпаѓање, односно за векторот на оптимално порамнување за првото поместување и резултатот на порамнување важи: $v_{1,o} = []$ и $f(A_{1,o}) = 0$.

Базното совпаѓањето за второто поместување Табела 3.1 се бележи со податочна тројка (2,6,1), каде 2 е почетната положба на совпаѓањето во секвенцата a, 6 е почетната положба на совпаѓањето во секвенцата b и 1 е должина на совпаѓање. Оптималното локално порамнување за второто поместување е определено со векторот $v_{2,o} = [(2,6,1)]$, со резултат на

порамнување 2, бидејќи е порамнет пар на совпаѓачки нуклеотиди. Поради тоа што резултатот на оптимално порамнување за второто поместување е поголем од резултатот на оптимално порамнување до второто поместување ($f(A_{2,o}) = 2 > 0 = f(A_o)$), содржината на векторот на оптимално порамнување v_o се ажурира со содржината на векторот $v_{2,o}$: $v_o \leftarrow v_{2,o} = [(2,6,1)]$. Резултатот на оптимално порамнување сега изнесува $f(A_o) \leftarrow f(A_{2,o}) = 2$.

Непостоењето на базни совпаѓања кај третото и четвртото поместување, нема да предизвика промена на оптималното порамнување и резултатот на оптимално порамнување. Кај петтото поместување се утврдува Гванин базно совпаѓање, кое се бележи со податочна тројка (5,4,1), Табела 3.1. Повторно станува збор за порамнување на базен пар, чиј резултат на порамнување не е поголем од резултатот на оптимално порамнување до петтото порамнување (T-T порамнувањето), односно структурата на векторот на оптимално порамнување и резултатот на оптимално порамнување нема да се променат.

Со споредба на паралелните нуклеотиди за шестото поместување се утврдуваат два фрагменти на совпаѓање: T-T и GG-GG. Фрагментите на совпаѓање се бележат со податочните тројки: (2,2,1) и (4,4,2) соодветно, Табела 3.1. Од $k = 2$ фрагменти на совпаѓање се образуваат $\frac{k(k+1)}{2} = \frac{2\times3}{2} = 3$ различни порамнувања: T-T, GG-GG и TCGG-TAGG, од кои за оптимално порамнување за поместување се избира порамнувањето со максимален резултат на порамнување. За претходните порамнувања со резултати на порамнување: 2, 4 и 5 соодветно, оптимално порамнување е порамнувањето TCGG-TAGG со резултат на порамнување 5, кое се репрезентира со податочен вектор: $v_{6,o} = [(2,2,1),(4,4,2)]$.

Бидејќи резултатот на оптимално порамнување за шестото поместување е поголем од резултатот на оптимално порамнување до шестото поместување, $5 = f(A_{6,o}) > f(A_o) = 2$, податочната структура на векторот на оптимално порамнување v_o се заменува со податочната структура на векторот $v_{6,o}$, $v_o \leftarrow v_{6,o} = [(2,2,1),(4,4,2)]$. Новата вредност на резултатот на оптимално порамнување изнесува $f(A_o) \leftarrow f(A_{6,o}) = 5$.

Имајќи предвид дека резултатите на оптималните порамнувања за наредните четири поместувања се помали од резултатот на оптимално порамнување $f(A_o) = 5$, Табела 3.1, векторот на оптимално локално порамнување и резултатот на оптимално локално порамнување до десетто поместување остануваат непроменети, $v_o = [(2,2,1),(4,4,2)]$, $f(A_o) = 5$.

Согласно Теорема 3.1, последните $\left\lceil \frac{f_o}{\mu} \right\rceil = \left\lceil \frac{5}{2} \right\rceil = 2$ поместувања (поместување број 11 и поместување број 12) не се извршуваат, бидејќи таму во никој случај не може да се најде порамнување со резултат на порамнување поголем од 5, од каде за оптимално локално порамнување за пример секвенците a:CTCGGAC и b:GTAGGT се добива: TCGG-TAGG.

Табела 3.1. Чекори на извршување на алгоритмот за пример секвенците a и b
Table 3.1. Execution of the algorithm for the sequences a and b

Поместување s	**Преклопување**	**Порамнувања**	$v_o, v_{s,o}, f(A_o), f(A_{s,o})$
1	CTCGGAC GTAGGT	/	$v_{1,o} = [\,], f(A_{1,o}) = 0,$ $v_o = 0, f(A_o) = 0$
2	CTCGGAC GTAGGT	T T	$v_{2,o} = [(2,6,1)], f(A_{2,o}) = 2,$ $v_o = [(2,6,1)], f(A_o) = 2$
3	CTCGGAC GTAGGT	/	$v_{3,o} = [\,], f(A_{3,o}) = 0,$ $v_o = [(2,6,1)], f(A_o) = 2$
4	CTCGGAC GTAGGT	/	$v_{4,o} = [\,], f(A_{4,o}) = 0,$ $v_o = [(2,6,1)], f(A_o) = 2$
5	CTCGGAC GTAGGT	G G	$v_{5,o} = [(4,5,1)], f(A_{5,o}) = 2,$ $v_o = [(2,6,1)], f(A_o) = 2$
6	CTCGGAC GTAGGT	T T GG GG **TCGG** **TAGG**	$v_{6,o} = [(2,2,1),(4,4,2)\,], f(A_{6,o}) = 5,$ $v_o = [(2,2,1),(4,4,2)], f(A_o) = 5$
7	CTCGGAC GTAGGT	G G	$v_{7,o} = [(5,4,1)], f(A_{7,o}) = 2,$ $v_o = [(2,2,1),(4,4,2)], f(A_o) = 5$
8	CTCGGAC GTAGGT	/	$v_{8,o} = [\,], f(A_{8,o}) = 0,$ $v_o = [(2,2,1),(4,4,2)], f(A_o) = 5$
9	CTCGGAC GTAGGT	G G A A GGA GTA	$v_{9,o} = [(4,1,1),(6,3,1)], f(A_{9,o}) = 3,$ $v_o = [(2,2,1),(4,4,2)], f(A_o) = 5$
10	CTCGGAC GTAGGT	G G	$v_{10,o} = [(5,1,1)], f(A_{10,o}) = 2,$ $v_o = [(2,2,1),(4,4,2)], f(A_o) = 5$

ДИСКУСИЈА: Бројот на извршени споредби за да се најде оптимално порамнување изнесува: 1+2+3+4+5+6+6+5+4+3=39 и истиот зависи од процентот на идентичност помеѓу секвенците PID и метриката на порамнување.

Процентот на идентичност помеѓу две ДНК секвенци се пресметува според формулата: $PID = \frac{hits}{averageLength} \times 100$, каде $hits$ е број на совпаѓања и $averageLength$ е средна должина на порамнетите секвенци.

За пример секвенците a:CTCGGAC и b:GTAGGT, $PID = \frac{3}{\frac{6+7}{2}} \times 100 = \frac{6}{13} \times 100 = 46{,}2\%$. Ако нуклеотидот на положба 3 во рамки на секвенцата b се замени

со цитозин (C), процентот на идентичност помеѓу секвенците ќе изнесува $PID = \frac{4}{\frac{6+7}{2}} \times 100 = \frac{8}{13} \times 100 = 61{,}5\%$, бидејќи во рамки на оптималното порамнување TCGG-TCGG со резултат на порамнување 8, наместо три ќе бидат вклучени четири базни совпаѓања. Согласно Теорема 3.1, последните $\left[\frac{8}{2}\right] = 4$ поместувања не се извршуваат, што резултира со намалување на бројот на извршени споредби на базни парови од 39 на: 1+2+3+4+5+6+6+5=32.

Како заклучок се наметнува дека бројот на извршени споредби на парови на нуклеотиди се намалува со зголемување на процентот на идентичност помеѓу порамнетите ДНК секвенци, односно колку процентот на идентичност помеѓу ДНК секвенците е повисок, дотолку порамнувањето е побрзо.

Изборот на конкретна метрика за порамнување, исто така може да влијае на временската комплексност на алгоритмот. Со избор на поголема вредност за казна δ, се зголемува и бројот на извршени споредби. Така, на пример, ако како вредности за доделување награда μ и казна δ наместо +2 и -1 се изберат +2 и -3 соодветно, тогаш резултатот на оптималното порамнување: TCGG-TAGG, наместо 5 ќе изнесува 3. Според Теорема 3.1, последното поместување ($\left[\frac{3}{2}\right] = 1$) не се извршува, со што бројот на споредени базни парови се зголемува од 39 на 41. Во принцип, за доделување на помала казна се намалува бројот на извршени споредби, односно се подобруваат временските аспекти на извршување на алгоритмот.

Примената на Теорема 3.1, овозможува утврдување на оптимално локално порамнување за две ДНК секвенци со помалку од $n \times m$ споредби, каде n е должина на секвенцата a и m е должина секвенцата b. За пример секвенците a и b со должини 7 и 6 соодветно, бројот на извршени споредби изнесува 39, што е помалку од $n \times m = 6 \times 7 = 42$ споредби, колку што извршуваат алгоритмите базирани на динамичко програмирање. Со избор на оптимална метрика за порамнување на ДНК секвенци со повисок процент на идентичност уште повеќе се намалува бројот на извршени споредби.

Од мемориски аспект, во фаза на извршување, во секој чекор, во меморија се чуваат два податочни вектори, и тоа: вектор $v_{s,o}$ кој го бележи оптималното порамнување за поместување s и вектор v_o кој го бележи оптималното порамнување до поместување s. Содржината на векторите $v_{s,o}$ и v_o динамички се ажурира, во зависност од структурата и резултатот на оптималното порамнување за поместување.

За пример секвенците a:CTCGGAC и b:GTAGGT, во просек се зафаќа меморија за $\frac{22 \times 3}{10} = 6{,}6$ целобројни вредности. Мемориските трошок е максимален кај шестото и деветтото поместување, каде векторот на оптимално порамнување за поместување и векторот на оптимално порамнување содржат по шест целобројни вредности.

Мемориската побарувачка на алгоритмот по вектор на порамнување во фаза на извршување може да се менува во интервалот: $0 \leq S \leq 3 \times \left(\frac{m}{2} + 1\right) \approx 1{,}5 \times m$, каде m е должина на пократката ДНК секвенца. Теоретски,

максималниот мемориски трошок по вектор на порамнување изнесува: $3 \times \left(\frac{m}{2}+1\right)$, што претствува случај на репрезентација на порамнување од тип: match|*mismatch*|match|*mismatch*|..., кога во рамки на порамнување со должина m, $\frac{m}{2}+1$ нуклеотиди се совпаѓачки, додека останатите се несовпаѓачки. На пример, за репрезентација на: A,T,G совпаѓањата од порамнувањето $A{:}\,ACTGA - AGTCA$ се зафаќаат 4×9 =36 В за $3 \times \left(\frac{5}{2}+1\right) = 9$ целобројни вредности (по три целобројни вредности за репрезентација на секое совпаѓање). Минимален мемориски трошок се забележува во случај кога не постои ниту едно базно совпаѓање во рамки на преклопување, односно кога векторот на порамнување е празен.

При порамнување на ДНК секвенци со висок процент на идентичност, максималната мемориска побарувачка на алгоритмот во фаза на извршување се очекува да изнесува: $k \times m, k < 1$, односно максималниот број на целобројни вредности во меморија да биде помал од должината на пократката ДНК секвенца – m. Очекувањето се базира на фактот дека со зголемување на процентот на базна идентичност се зголемува и должината на совпаѓањата, што резултира со помал број совпаѓања, односно помала мемориска побарувачка за репрезентација на истите во фаза на извршување. Како последица на тоа, бројот на податочни тројки по вектор на порамнување се намалува.

Алгоритам со мемориска побарувачка од тип: $k \times m, k < 1$ за порамнување на ДНК секвенци со висок процент на идентичност е просторно пооптимален како во однос на алгоритмите за порамнување на ДНК секвенци базирани на динамичко програмирање (*Needleman-Wunsch*, *Gotoh*, *Smith-Waterman*, *Sellers* и др.), каде мемориската побарувачка изнесува $O(n \times m)$, така и во однос на мемориски-линеарните имплементации на истите (*Hirschberg*, *Myers и Miller*, *Huang et al*. и др.), каде во најдобар случај резервираниот мемориски простор е пропорционален на должината на пократката секвенца m.

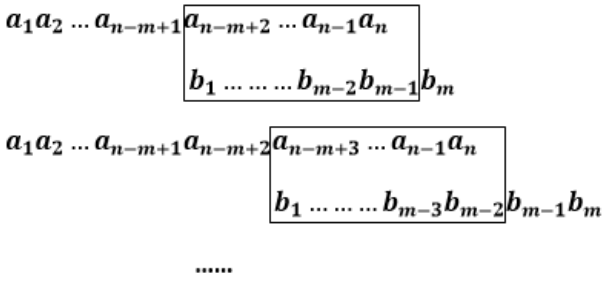
Бројот на извршени споредби може да се намали ако првин се извршат поместувањата со кои се образуваат фрагменти на преклопувања со должина еднаква на должината на пократката ДНК секвенца – m, по што во зависност од резултатот на оптимално порамнување се ограничуваат левите и десни поместувања. Претходната теза се базира на фактот дека резултатот на порамнување зависи од бројот на совпаѓања, односно колку е поголема должината на преклопување помеѓу ДНК секвенците, дотолку повеќе базни совпаѓања можат да се пронајдат, што значи дека веројатноста да се утврди оптимално порамнување во рамки на преклопување со должина m е поголема од веројатноста да се најде оптимално порамнување во рамки на преклопување со должина помала од m. Всушност, како се намалува должината на преклопување, така се намалува и веројатноста за идентификација на оптимално порамнување.

Во првата фаза се извршуваат $n - m + 1$ поместувања на пократката секвенца b долж подолгата a, за кои должината на преклопување е еднаква на должината на пократката ДНК секвенца m, Сл. 3.4. Со споредба на паралелните нуклеотиди од фрагментите на преклопување, за секое поместување s може да

се утврди оптимално порамнување за поместување $A_{s,o}$, со резултат на порамнување $f(A_{s,o})$.

Слика 3.4. Поместувања на секвенцата b долж секвенцата a, за кои должината на фрагментите на преклопување изнесува m.
Figure 3.4. Shifting the sequence b over the sequence a, such as the length of the overlapping fragments equals m.

Ако максималниот резултат на порамнување по извршување на првите $n-m+1$ поместувања е f_{max}^m, $f_{max}^m = \max\{f(A_{1,o}), f(A_{2,o}), \dots, f(A_{n-m,o}), f(A_{n-m+1,o})\}$, тогаш бројот на десни поместувања на секвенцата b надвор од опсегот на секвенцата a, Сл. 3.5, може да се ограничи според Теорема 3.2.



Слика 3.5. Десни поместувања на секвенцата b, надвор од опсегот на секвенцата a.
Figure 3.5. Right shifts of the sequence b, out of the range of the sequence a.

Теорема 3.2. Порамнување со поголем резултат на порамнување од f_{max}^m, може да се утврди само во рамки на фрагмент на преклопување со должина помеѓу: $m-1$ и $\left[\frac{f_{max}^m}{\mu}\right]+1$.

Доказ: Во рамки на преклопување со должина l, $l \leq \left[\frac{f_{max}^m}{\mu}\right]$, за резултатот на секое порамнување $f(A)$ важи: $f(A) \leq \mu\left[\frac{f_{max}^m}{\mu}\right] \leq \mu\frac{f_{max}^m}{\mu} = f_{max}^m$, каде μ е награда за порамнување на пар на еквивалентни нуклеотиди.

По извршување на првите $n-m+1$ поместувања, со секое десно поместување на секвенцата b, се образува фрагмент на преклопување со должина помала од m, Сл. 3.5. Должините на фрагментите на преклопување се намалуваат за 1 при секое наредно поместување, од $m-1$ до 1. Кога би се извршиле сите $m-1$ поместувања, би требало да се споредат $(m-1)+(m-2)+\cdots+2+1=\frac{m(m-1)}{2}$ базни парови.

Согласно Теорема 3.2, поместувањата за кои должината на преклопување е помала или еднаква на $\left[\frac{f_{max}^{m}}{\mu}\right]$ не се извршуваат. Неизвршувањето на овие поместувања не влијае на оптималноста на решението, бидејќи и во случај на совпаѓање на сите нуклеотиди, резултатот на порамнување не може да биде поголем од f_{max}^{m}, со што бројот на извршени споредби се намалува за $\frac{\left[\frac{f_{max}^{m}}{\mu}\right]\left(\left[\frac{f_{max}^{m}}{\mu}\right]+1\right)}{2}$.

Ако $f_{max}^{m,r}$ е максимален резултат на порамнување по извршување на централните поместувањата со кои образуваат фрагменти на преклопувања со должина $l=m$ и десни поместувања со кои се образуваат фрагменти на преклопувања со должина $l<m$ согласно Теорема 3.2, со лево поместување на секвенцата b надвор од опсегот на секвенцата a, Сл. 3.6, а по аналогија на Теорема 3.2, порамнување со поголем резултат на порамнување од $f_{max}^{m,r}$ може да се постои само во рамки на фрагмент на преклопување со должина помеѓу $m-1$ и $\left[\frac{f_{max}^{m,r}}{\mu}\right]+1$.

$$\begin{array}{l} \boxed{a_1 a_2 \dots a_{m-2} a_{m-1}} a_m \dots a_{n-1} a_n \\ b_1 \boxed{b_2 b_3 \dots b_{m-1} b_m} \end{array}$$

$$\begin{array}{l} \boxed{a_1 a_2 \dots a_{m-3} a_{m-2}} a_{m-1} \dots a_{n-1} a_n \\ b_1 b_2 \boxed{b_3 b_4 \dots b_{m-1} b_m} \end{array}$$

.....

Слика 3.6. Леви поместувања на секвенцата b, надвор од опсегот на секвенцата a.

Figure 3.6. Left shifts of the sequence b, out of the range of the sequence a.

За разлика од алгоритмите базирани на динамичко програмирање, кои за утврдување на оптимално порамнување извршуваат nm споредби, предлог алгоритмот го намалува тој број за $\frac{\left[\frac{f_{max}^{m}}{\mu}\right]\left(\left[\frac{f_{max}^{m}}{\mu}\right]+1\right)}{2}+\frac{\left[\frac{f_{max}^{m,r}}{\mu}\right]\left(\left[\frac{f_{max}^{m,r}}{\mu}\right]+1\right)}{2}$. Ако се усвои овој редослед на поместување на пократката ДНК секвенца, бројот на извршени споредби дополнително се намалува за $\frac{\left[\frac{f_{max}^{m,r}}{\mu}\right]\left(\left[\frac{f_{max}^{m,r}}{\mu}\right]+1\right)}{2}$ во споредба со првичниот редослед на поместување.

Степенот на подобрување зависи од: *процентот на базна идентичност помеѓу ДНК секвенците*, *разликата помеѓу должините на ДНК секвенците* и *метриката на порамнување*.

При порамнување на ДНК секвенци со висок процент на базна идентичност, f_{max}^{m} и $f_{max}^{m,r}$ имат високи вредности. Како резултат на тоа, се извршуваат само мал број на поместувања на секвенцата b надвор од опсегот на секвенцата a, односно се намалува бројот на споредени нуклеотиди. Помалиот број на споредби го намалува времето на извршување на алгоритмот.

Влијание врз временските аспекти на алгоритмот има и разликата помеѓу должините на ДНК секвенците кои се порамнуваат. Ако разликата $n - m$ е мала, каде n е должина на подолгата ДНК секвенца a и m е должина на пократката ДНК секвенца b, во согласност со концепциската поставеност на алгоритмот, мал е и бројот на поместувања на пократката ДНК секвенца долж подолгата, со кои се образуваат фрагменти на преклопувања со должина еднаква на должината на пократката ДНК секвенца m. Се очекува времето на извршување на алгоримот да биде пократко за ДНК секвенци со приближно еднаква должина.

Метриката на порамнување, односно вредностите кои се избираат како награда за порамнување на еквивалентни нуклеотиди и казна за порамнување на различни нуклеотиди, влијае на резултатите на порамнување: f_{max}^{m} и $f_{max}^{m,r}$. Метрика на порамнување која ги максимизира: $\left[\frac{f_{max}^{m}}{\mu}\right]$ и $\left[\frac{f_{max}^{m,r}}{\mu}\right]$ за иста структура на порамнување, го намалува бројот на поместувања, со што се подобруваат временските аспекти на алгоритмот.

Оптималните порамнувања за поместувањата се добиваат на начин така што најдолгиот фрагмент на совпаѓање од $k, k > 0$ последователни фрагменти на совпаѓање $R_1R_2 \dots R_{k-1}R_k$, иницијално се избира за *оптимално* и *издолжувачко* порамнување за поместување. Со последователно додавање на фрагменти на совпаѓање на издолжувачкото порамнување, за секое издолжување се пресметува резултатот на порамнување. Ако резултатот на порамнување за издолжување е поголем од резултатот на оптимално порамнување, оптималното порамнување, како и резултатот на оптимално порамнување за поместување, се заменуваат со издолжувачкото порамнување и резултатот на издолжувачко порамнување.

Со примена на овој концепт за формирање на порамнувања, бројот на кандидати за оптимално порамнување за поместување се намалува од $\frac{k(k+1)}{2}$ на k, што додатно ја подобрува временската комплексност на алгоритмот.

Во зависност од положбата на најдолгиот фрагмент на совпаѓање за поместување s, се разгледуваат три случаи:

Случај 1: Ако најдолгиот фрагмент на совпаѓање е R_1, односно $R_1 = \max len\{R_1, R_2, \dots, R_{k-1}, R_k\}$, за оптимално и издолжувачко порамнување за поместување s на почеток се избира R_1, односно: $A_{s,o} = R_1, A_{s,E} = R_1$. Додавајќи ги фрагмените на совпаѓање $R_2R_3 \dots R_{k-1}R_k$ на $A_{s,E}$, за секое издолжување $A_{s,E} = A_{s,E} \bowtie R_i$ за $i = 2,3, \dots, k-1, k$ се проверува дали $f(A_{s,E}) > f(A_{s,o})$. Во потврден случај, оптималното порамнување за поместување s, $A_{s,o}$ се заменува со

издолжувачкото порамнување $A_{s,E}$, $A_{s,o} \leftarrow A_{s,E}$ и резултатот на оптимално порамнување за поместување се заменува со резултатот на издолжувачко порамнување $f(A_{s,o}) \leftarrow f(A_{s,E})$.

Случај 2: Ако најдолгиот фрагмент на совпаѓање е R_k, односно $R_k = \max len\{R_1, R_2, \dots, R_{k-1}, R_k\}$, за оптимално и издолжувачко порамнување за поместување s на почеток се избира R_k, односно: $A_{s,o} = R_k, A_{s,E} = R_k$. Додавајќи ги фрагментите на совпаѓање $R_{k-1}R_{k-2} \dots R_2R_1$ на $A_{s,E}$, за секое издолжување $A_{s,E} = R_i \bowtie A_{s,E}$ за $i = k-1, k-2, \dots, 2, 1$, се проверува дали $f(A_{s,E}) > f(A_{s,o})$. Ако неравенството важи, оптималното порамнување за поместување s, $A_{s,o}$се заменува со издолжувачкото порамнување $A_{s,E}$, $A_{s,o} \leftarrow A_{s,E}$ и резултатот на оптимално порамнување за поместување s се заменува со резултатот на издолжувачко порамнување, $f(A_{s,o}) \leftarrow f(A_{s,E})$.

Случај 3: Ако најдолгиот фрагмент на совпаѓање е $R_\xi, 2 \le \xi \le k-1$, за оптимално и издолжувачко порамнување за поместување на почеток се избира R_ξ, $A_{s,o} \leftarrow R_\xi, A_{s,E} \leftarrow R_\xi$. Со додавање на фрагментите на совпаѓање од левата страна на R_ξ, за секое издолжување $A_{s,E} = R_i \bowtie A_{s,E}$ за $i = \xi - 1, \xi - 2, \dots, 2, 1$ се проверува дали $f(A_{s,E}) > f(A_{s,o})$. Во потврден случај, $A_{s,o} \leftarrow A_{s,E}$ и $f(A_{s,o}) \leftarrow f(A_{s,E})$.

Пред да се започне со додавање на фрагментите на совпаѓање од десната страна на R_ξ, оптималното порамнување за поместување се избира за издолжувачко порамнување, $A_{s,E} \leftarrow A_{s,o}$ и $f(A_{s,E}) \leftarrow f(A_{s,o})$. Со додавање на фрагментите на совпаѓање од десната страна на R_ξ на $A_{s,E}$, за секое издолжување $A_{s,E} = A_{s,E} \bowtie R_i$ за $i = \xi + 1, \xi + 2, \dots, k-1, k$ се проверува дали $f(A_{s,E}) > f(A_{s,o})$. Како и претходно ако $f(A_{s,E}) > f(A_{s,o})$, тогаш $A_{s,o} \leftarrow A_{s,E}$ и $f(A_{s,o}) \leftarrow f(A_{s,E})$.

Ако се задржи пристапот за репрезентација на порамнувањата: A_o и $A_{s,o}$ со вектори v_o и $v_{s,o}$, каде A_o е оптимално порамнување до поместување s, кое го бележи векторот v_o и $A_{s,o}$ е оптимално порамнување за поместување s кое го бележи векторот $v_{s,o}$, промената на редоследот на поместување не ја загрозува оптималноста на предлог пристапот за минимизација на мемориската комплексност во фаза на извршување.

Како и претходно, пред секое наредно поместување се проверува дали $f(A_{s,o}) > f(A_o)$. Во потврден случај, векторот на оптимално порамнување v_o се заменува со векторот на оптимално порамнување за поместување s, $v_o \leftarrow v_{s,o}$ и $f(A_o) \leftarrow f(A_{s,o})$. За потсетување, векторите v_o и $v_{s,o}$ се множества на податочни тројки од тип: (p_a, p_b, l), каде p_a и p_b ги определуваат почетните положби на фрагментите на совпаѓања од A_o и $A_{s,o}$ во a и b, додека l е должина на фрагмент на совпаѓање.

СТУДИЈА НА СЛУЧАЈ: Влијанието на промената на редоследот на поместување врз временските аспекти на алгоритмот, ќе ја разгледаме за пример секвенците: a:CTCGGAC и b:GTAGGT, доделувајќи награда $\mu = +2$ за порамнување на еквивалентни нуклеотиди и казна $\delta = -1$ за порамнување на различни нуклеотиди.

Во согласност со концепциската поставеност на алгоритмот, првин се извршуваат поместувањата со кои се образуваат фрагменти на преклопувања со должина еднаква на должината на пократката ДНК секвенца b. Бројот на такви поместувања изнесува $n - m + 1 = 7 - 6 + 1 = 2$, Табела 3.2.

Најдолгиот фрагмент на совпаѓање за првото поместување е GG совпаѓањето, кое се бележи со податочна тројка $R_{1,2}: (4,4,2)$. Нотацијата $R_{s,j}$ означува j-ти фрагмент на совпаѓање за поместување s. Гванин-Гванин совпаѓањето на почеток се избира за оптимално и издолжувачко порамнување за првото поместување, односно: $A_{1,o} = R_{1,2} = GG: (4,4,2), A_{1,E} = R_{1,2} = GG: (4,4,2)$. Резултатите на порамнувањата $A_{1,o}$ и $A_{1,E}$, $f(A_{1,o})$ и $f(A_{1,E})$ изнесуваат 4, бидејќи порамнувањата вклучуваат по две базни совпаѓања.

Со лево издолжување на $A_{1,E}$ се додава Тимин фрагмент на совпаѓање $R_{1,1} = T: (2,2,1)$, со што се образува издолжувачко порамнување $A_{1,E} = R_{1,1} \bowtie A_{1,E} = R_{1,1}R_{1,2}: (2,2,1)(4,4,2)$, со резултат на порамнување 5 (порамнувањето вклучува три базни совпаѓања и едно базно несовпаѓање, Табела 3.2).

Бидејќи $5 = f(A_{1,E}) > f(A_{1,o}) = 4$, односно резултатот на порамнување за издолжување $f(A_{1,E})$ е поголем од резултатот на оптимално порамнување за поместување $f(A_{1,o})$, оптималното порамнување за поместување се заменува со издолжувачкото порамнување, односно: $A_{1,o} \leftarrow A_{1,E}: R_{1,1}R_{1,2}$. Во ваков случај резултатот на оптимално порамнување за поместување се ажурира со резултатот на издолжувачко порамнување, односно: $f(A_{1,o}) \leftarrow f(A_{1,E}) = 5$.

На почеток, векторот на оптимално порамнување v_o е празен, $v_o = []$ и резултатот на оптимално порамнување $f(A_o) = 0$. Пред секое наредно поместување се проверува дали $f(A_{s,o}) > f(A_o)$. Во потврден случај, векторот на оптимално порамнување v_o се ажурира со содржината на векторот на оптимално порамнување за поместување $v_{s,o}$, $v_o \leftarrow v_{s,o}$ и $f(A_o) \leftarrow f(A_{s,o})$.

Пред да се изврши второто поместување со кое исто така се образува фрагмент на преклопување кој вклучува по шест нуклеотиди, v_o се заменува со $v_{1,o}: ([2,2,1], [4,4,2])$, односно $v_o \leftarrow v_{1,o}: ([2,2,1], [4,4,2])$, бидејќи $f(A_{1,o}) = 5 > 0 = f(A_o)$. Покрај векторот на оптимално порамнување се менува и резултатот на оптимално порамнување, односно $f(A_o) \leftarrow f(A_{1,o}) = 5$.

Гванин совпаѓањето $R_{2,1}: (5,4,1)$ е оптимално порамнување за поместување број 2. Бидејќи резултатот на оптимално порамнување за второто поместување изнесува 2 и истиот е помал од резултатот на оптимално порамнување $f(A_o) = 5$, оптималното порамнување A_o и резултатот на оптимално порамнување по извршување на второто поместување остануваат непроменти, односно оптималното порамнување и понатаму е определено со векторот $v_o = [(2,2,1), (4,4,2)]$, со резултат на порамнување $f(A_o) = 5$.

Според Теорема 3.2, десните поместувања со кои се образуваат фрагменти на преклопувања со должина помала или еднаква на $\left[\frac{f(A_o)}{\mu}\right] = \left[\frac{5}{2}\right] = 2$

не се извршуваат, бидејќи и во случај на совпаѓање на сите нуклеотиди, резултатот на порамнување не може да биде поголем од $f(A_o) = 5$. Ова е неоспорно точно, бидејќи за поместувањата со кои се образуваат фрагменти на преклопувања со должина помала или еднаква на 2, максималниот можен резултат на порамнување изнесува: $\mu \times 2 = 2 \times 2 = 4 < 5 = f(A_o)$.

Ако сите фрагментите на совпаѓање имаат еднаква должина како кај четвртото поместување, Табела 3.2, алгоритмот почетно го избира првиот фрагмент на совпаѓање за оптимално и издолжувачко порамнување за поместување, односно $A_{4,o} = R_{4,1} = G\colon(4,1,1)$ и $A_{4,E} = R_{4,1} = G\colon(4,1,1)$.

Бидејќи резултатот на порамнување за издолжувањето $A_{4,E} = A_{4,E} \bowtie R_{4,2} = R_{4,1}R_{4,2}\colon(4,1,1)(6,3,1)$ е поголем од резултатот на оптимално порамнување $A_{4,o}$, односно $f(A_{4,E}) = 3 > 2 = f(A_{4,o})$, оптималното порамнување и резултатот на оптимално порамнување за четвртото поместување: $A_{4,o}$ и $f(A_{4,o})$ се ажуираат со издолжувачкото порамнување и резултатот на издолжувачко порамнување: $A_{4,E}$ и $f(A_{4,E})$, односно: $A_{4,o} \leftarrow A_{4,E}\colon R_{4,1}R_{4,2}$ и $f(A_{4,o}) \leftarrow f(A_{4,E}) = 3$.

Пред да се изврши последното десно поместување, со кое се образува фрагмент на преклопување со должина 3, се проверува дали $f(A_{4,o}) = 3 > 5 = f(A_o)$. Поради невистинитоста на претходното неравенство, векторот на оптимално порамнување $v_o = [(2,2,1),(4,4,2)]$ и резултатот на оптимално порамнување $f(A_o) = 5$ остануваат непроменети.

Гванин совпаѓањето за петтото поместување, со резултат на порамнување 2, исто така нема да предизвика промена на оптималното порамнување A_o и резултатот на оптимално порамнување $f(A_o)$.

По извршување на левите поместувања на пократката секвенца b, со кои се образуваат фрагменти на преклопувања со должина поголема или еднаква на 3, Табела 3.2, не е утврдено порамнување со резултат на порамнување поголем од $f(A_o) = 5$, од каде може да се заклучи дека оптималното порамнување за пример секвенците a:CTCGGAC и b:GTAGGT е определено со векторот $v_o = [(2,2,1),(4,4,2)]$, со резултат на порамнување $f(A_o) = 5$.

По аналогија на Теорема 3.2, левите поместувања со кои се образуваат фрагменти на преклопувања со должина помала или еднаква на $\left\lceil \frac{f(A_o)}{\mu} \right\rceil = \left\lceil \frac{5}{2} \right\rceil = 2$ не се извршуваат, бидејќи во никој случај резултатот на порамнување во фрагмент на преклопување со должина помала или еднаква на 2 не може да биде поголем од $f(A_o) = 5$.

Табела 3.2. Чекори на извршување на алгоритмот со променет редослед на поместување на пократката ДНК секвенца
Table 3.2. Steps of execution of the algorithm with modified order of shifting of the shorter DNA sequence

Поместување s	Преклопување	Фрагменти на совпаѓања / оптимално порамнување за поместување	$v_{s,o}, f(v_{s,o}), v_o, f(v_o)$ оптимално порамнување за поместување
1	CTCGGAC GTAGGT	$R_{1,1}$: T T, (2,2,1) $R_{1,2}$: GG GG, (4,4,2) TCGG TAGG	$v_{1,o} = [(2,2,1),(4,4,2)]$, $f(A_{1,o}) = 5$ $v_o = [(2,2,1),(4,4,2)]$ $f(A_o) = 5$
2	CTCGGAC GTAGGT	$R_{2,1}$: G G, (5,4,1) G G	$v_{2,o} = [(5,4,1)]$ $f(A_{2,o}) = 2$ $v_o = [(2,2,1),(4,4,2)]$ $f(A_o) = 5$
3	CTCGGAC GTAGGT	/	$v_{3,o} = []$ $f(A_{3,o}) = 0$ $v_o = [(2,2,1),(4,4,2)]$ $f(A_o) = 5$
4	CTCGGAC GTAGGT	$R_{4,1}$: G G, (4,1,1) $R_{4,2}$: A A, (6,3,1) GGA GTA	$v_{4,o} = [(4,1,1),(6,3,1)]$ $f(A_{4,o}) = 3$ $v_o = [(2,2,1),(4,4,2)]$ $f(A_o) = 5$
5	CTCGGAC GTAGGT	G G, (5,1,1) G G	$v_{5,o} = [(5,1,1)]$ $f(A_{5,o}) = 2$ $v_o = [(2,2,1),(4,4,2)]$ $f(A_o) = 5$
6	CTCGGAC GTAGGT	G G, (4,5,1) G G	$v_{6,o} = [(4,5,1)]$ $f(A_{6,o}) = 2$ $v_o = [(2,2,1),(4,4,2)]$ $f(A_o) = 5$
7	CTCGGAC GTAGGT	/	$v_{7,o} = []$ $f(A_{7,o}) = 0$ $v_o = [(2,2,1),(4,4,2)]$ $f(A_o) = 5$
8	CTCGGAC GTAGGT	/	$v_{8,o} = []$ $f(A_{8,o}) = 0$ $v_o = [(2,2,1),(4,4,2)]$ $f(A_o) = 5$

ДИСКУСИЈА: Во споредба со првопредложениот редослед на поместување, каде за да се порамнат секвенците a:CTCGGAC и b:GTAGGT алгоритмот извршува 39 споредби, со промената на редоследот на поместување тој број се намалува на 36. Бројот на неизвршени споредби се удвојува поради неизвршувањето на десните и леви поместувања со кои се образуваат фрагменти на преклопувања каде во никој случај не може да се најде пооптимално решение.

За пример секвенците a и b оптимално порамнување е порамнувањето A_o:TCGG-TAGG со резултат на порамнување $f(A_o) = 5$. Резултатот на оптимално порамнување $f(A_o) = 5$ ја ограничува должината на фрагментите на преклопувања $l > \left\lceil \frac{f(A_o)}{\mu} \right\rceil = \left\lceil \frac{5}{2} \right\rceil = 2$. Неизвршувањето на десните поместувања со кои се образуваат фрагменти на преклопувања со должина $l, l \leq \left\lceil \frac{f(A_o)}{\mu} \right\rceil = 2$ резултира со заштеда од 3 споредби, колку што се заштедуваат и од неизвршувањето на левите поместувања со кои се образуваат фрагменти на преклопувања со должина $l, l \leq \left\lceil \frac{f(A_o)}{\mu} \right\rceil = 2$. Вкупната заштеда при порамнување на пример секвенците a и b во споредба со алгоритмите базирани на динамичко програмирање изнесува 6. Во споредба со првопредложениот редослед на поместување, бројот на споредби се намалува за 3.

Влијанието на промената на редоследот на поместување во насока на подобрување на временските аспекти на алгоритмот, доаѓа уште повеќе до израз при порамнување на ДНК секвенци со висок процент на базна идентичност. Со зголемување на процентот на базна идентичност се зголемува резултатот на оптимално порамнување, а колку резултатот на оптимално порамнување е поголем, дотолку помалку поместувања треба да се извршат, со што се намалува бројот на извршени споредби, без да се влијае на оптималноста на решението.

На пример, ако нуклеотидот на положба 3 во секвенцата b се замени со Цитозин, според првопредложениот редослед на поместување се извршуваат 32 споредби. За потсетување, намалувањето на бројот на извршени споредби се должи на зголемувањето на резултатот на оптимално порамнување (зголемениот процент на базна идентичност) од $f(A_o) = 5$ на $f(A_o) = 8$ (од $PID = 46{,}1\%$ на $PID = 61{,}5\%$), односно на неизвршувањето на последните $\left\lceil \frac{f(A_o)}{\mu} \right\rceil = \left\lceil \frac{8}{2} \right\rceil = 4$ поместувања наместо $\left\lceil \frac{f(A_o)}{\mu} \right\rceil = \left\lceil \frac{5}{2} \right\rceil = 2$ поместувања.

Со промената на редоследот на поместување, бројот на извршени споредби се намалува за 10, односно алгоритмот за утврдување на оптимално порамнување за секвенците a:CTCGGAC и b:GTCGGT извршува 22 споредби. Заштедата од 10 базни споредби во споредба со првопредложениот редослед на поместување е резултат на неизвршувањето на левите поместувања, покрај десните, со кои се образуваат фрагменти на преклопувања со должина l, $l \leq \left\lceil \frac{f(A_o)}{\mu} \right\rceil = \left\lceil \frac{8}{2} \right\rceil = 4$.

Неизвршувањето на овие поместувања не влијае на оптималноста на решението, бидејќи во никој случај во рамки на фрагмент на преклопување со

должина $l \leq \left\lceil \frac{f(A_o)}{\mu} \right\rceil = \left\lceil \frac{8}{2} \right\rceil = 4$, независно од тоа дали фрагментот е добиен со десно или лево поместување на пократката секвенца b, резултатот на порамнување не може да биде поголем од $\mu \times 4 = 2 \times 4 = 8$.

Разликата помеѓу должините на ДНК секвенците кои се порамнуваат влијае на времето на извршување на алгоритмот. Колку е помала таа разлика, дотолку помало е времето на извршување на алгоритмот и обратно. На пример, ако се избрише нуклеотидот Цитозин на крајот од секвенцата a, a:CTCGGA, разликата помеѓу должините на ДНК секвенците се намалува од 7-6=1 на 7-7=0. Сега наместо две преклопувања во должина од шест нуклеотиди ќе постои само едно и за иста метрика на порамнување ($\mu = +2, \delta = -1$), алгоритмот наместо 36 ќе изврши 30 споредби.

Метриката на порамнување исто така влијае на времето на извршување на алгоритмот. Во принцип, повисока негативна казна делува негативно на временската комплексност на алгоритмот и обратно. На пример, за да се утврди оптимално порамнување за пример секвенците a:CTCGGAC и b:GTAGGT, алгоритмот извршува 36 споредби, под услов дека за секое базно совпаѓање се доделува награда $\mu = +2$, додека казната за несовпаѓање изнесува $\delta = -1$. Ако казната се промени од $\delta = -1$ на $\delta = -3$, алгоритмот наместо 36 споредби ќе изврши 40 споредби. Зголемувањето на бројот на споредби се должи на намалувањето на резултатот на оптимално порамнување: TCGG-TAGG, кој наместо 5, за метрика на порамнување $\mu = +2, \delta = -3$ изнесува: $3 \times 2 - 1 \times 3 = 3$. Според Теорема 3.2, може да се допушти да не се извршат само поместувањата со кои се образуваат фрагменти на преклопувања со должина $l, l = \left\lceil \frac{f(A_o)}{\mu} \right\rceil = \left\lceil \frac{3}{2} \right\rceil = 1$, без да се влијае на оптималноста на решението.

Задржувајќи го пристапот за динамичко ажурирање на податочните вектори: v_o и $v_{s,o}$ во фаза на извршување, теоретски максималниот мемориски трошок по податочен вектор изнесува: $3\left(\frac{m}{2} + 1\right)$, што е случај на репрезентација на порамнување од тип: match|*mismatch*|match|*mismatch*|... За пример секвенците a и b, максималниот мемориски трошок во фаза на извршување изнесува 12×4 B=48 B. Промената на редоследот на поместување може да влијае само на просечната мемориска потрошувачка. За пример секвенците, промената на редоследот на поместување резултира со зголемување на просечниот мемориски трошок од 6,6 на $\frac{12+9+6+12+9+9+6+6}{8} = \frac{69}{8} = 8{,}6$ целобројни вредности (секој собирок претставува вкупен мемориски трошок за поместување $s, 1 \leq s \leq 8$).

Од пресметковен аспект, со примена на предложените алгоритми се намалува времето на извршување (бројот на споредби) и меморискиот трошок во фаза на извршување во споредба со алгоритмите базирани на динамичко програмирање (*Needleman-Wunsch*, *Wagner-Fischer*, *Gotoh*, *Smith-Waterman*, *Sellers*, *Waterman-Eggert*, *Ficket* и др.) и нивните просторно линеарни имплементации. За разлика од алгоритмите базирани на динамичко програмирање кои се извршуваат во $O(nm)$ време, каде n и m се должини на секвенците кои се порамнуваат, времето на извршување на алгоритмот со примена на второпредложениот редослед на поместување на пократката

секвенца изнесува $O(kn)$, каде вредноста на параметарот k е помала од должината на пократката секвенца m. Степенот на подобрување зависи од вредноста на параметарот k, односно колку истиот е помал дотолку позначајно е подобрувањето и обратно. По аналогија на претходните дискусии, фактори од кои зависи вредноста на параметарот k се: *процентот на базна идентичност*, *разликата помеѓу должините на секвенците кои се порамнуваат* и *усвоената метрика на порамнување*. Зголемувањето на процентот на базна идентичност ја намалува вредноста на параметарот k. Помалата разлика помеѓу должините на секвенците кои се порамнуваат и изборот на помала казна резултира исто така со намалување на вредноста на параметарот k. Во фаза на извршување за секвенци со задоволителен процент на базна идентичност, меморискиот трошок е помал од должината на пократката секвенца m, што укажува на фактот дека предлог алгоритмите се мемориски поефикасни како во однос на стандардните алгоритми за порамнување базирани на динамичко програмирање, така и во однос на нивните просторно линеарни оптимизации (*Hirschberg*, *Myers и Miller*, *Huang et al*., и др).

3.2. АЛГОРИТАМ ЗА ПРАЗНИНСКО ПОРАМНУВАЊЕ

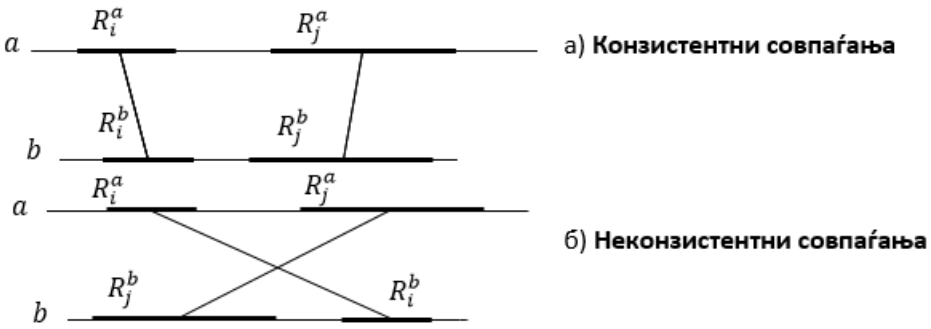
За разлика од предложените алгоритми со чија примена може да се утврди оптимална безпразнинска хомологија помеѓу две секвенци, понекогаш потребно е да утврди егзактната структурна зависност, со која покрај фрагментите на совпаѓања и несовпаѓања, еднозначно се определени и положбите на додавања (бришења) на нуклеотиди. За таа цел предложен е нов алгоритам кој во споредба со алгоритмите базирани на динамичко програмирање е побрз, односно извршува помалку од $O(nm)$ споредби за да најде решение и ја подобрува точноста на порамнување – *бројот на совпаѓања вклучени во порамнувањето*, како во однос на алгоритмите базирани на динамичко програмирање за избор на некои метрики на порамнување, така и во однос на хевристичните алгоритми (*MUMmer*, *AVID*, *GLASS*, *YASS* и др.).

Предлог алгоритмот за празнинско порамнување на ДНК секвенци: $a = a_1a_2 \dots a_{n-1}a_n$ и $b = b_1b_2 \dots b_{m-1}b_m, m \leq n$, се извршува во две фази:

Фаза 1: Се утврдуваат конзистентни парови на совпаѓања: $(R_\xi^a, R_\xi^b), 1 \leq \xi \leq k$ за ДНК секвенците a и b.

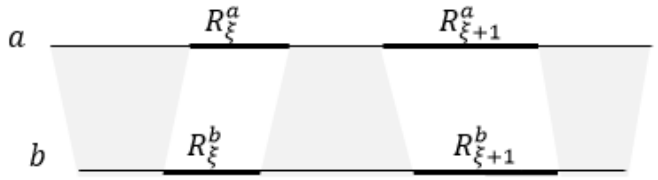
Дефиниција 12: Паровите на совпаѓања: (R_i^a, R_i^b) и (R_j^a, R_j^b), $i, j \in Z^+$ каде R_x^y означува фрагмент на совпаѓање R_x во рамки на ДНК секвенца y, се козистентни ако и само ако важи: $endingPositions(R_i^a, R_i^b) < startingPositions(R_j^a, R_j^b)$, односно крајните положби на фрагментот на совпаѓање R_i во a и b претходат на почетните положби на фрагментот на совпаѓање R_j во a и b.

На Сл. 3.7, се дадени примери за конзистентни и неконзистентни парови на совпаѓања.



Слика 3.7. Приказ на конзистентни и неконзистентни совпаѓања
Figure 3.7. Consistent and inconsistent matches

Секој пар на совпаѓање $(R_\xi^a, R_\xi^b), 1 \le \xi \le k$ од множеството на конзистентни парови на совпаѓања: $\{(R_1^a, R_1^b), (R_2^a, R_2^b), \dots, (R_{k-1}^a, R_{k-1}^b), (R_k^a, R_k^b)\}$ се издвојува од *конзистентен опсег на пребарување*. Конзистентен опсег на пребарување е пар на фрагменти од ДНК секвенците a и b кој одделува две последователни совпаѓања: (R_ξ^a, R_ξ^b) и $(R_{\xi+1}^a, R_{\xi+1}^b)$ или се наоѓа пред (после) најлевото (најдесното) совпаѓање, Сл. 3.8.



Слика 3.8. Приказ на три конзистентни опсези на пребарување и две последователни совпаѓања: (R_ξ^a, R_ξ^b) и $(R_{\xi+1}^a, R_{\xi+1}^b)$
Figure 3.8. Three consistent searching ranges and two consecutive hits: (R_ξ^a, R_ξ^b) and $(R_{\xi+1}^a, R_{\xi+1}^b)$

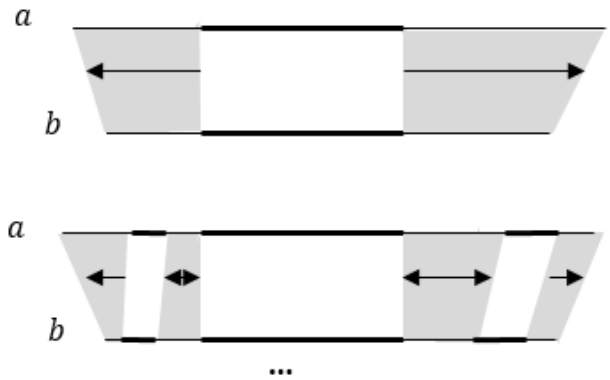
Ако опсегот на пребарување е определен со ДНК фрагментите: $a_f, 1 \le |a_f| \le n, a_f \in A$ и $b_f, 1 \le |b_f| \le m, b_f \in b$, совпаѓањето со максимална должина во рамки на опсегот се утврдува со последователна споредба на парови на зборови: $w_a, w_a \in a_f$ и $w_b, w_b \in b_f$ со должина $|w_a| = |w_b| = |b_f| - k$, ако $|a_f| > |b_f|$, односно со должина $|w_a| = |w_b| = |a_f| - k$, ако $|a_f| \le |b_f|$, зголемувајќи го k за 1, започнувајќи од 0, се до утврдување на првото совпаѓање, кое воедно е и совпаѓање со максимална должина во рамки на опсегот на пребарување.

Паровите на нуклеотиди од зборовите: $w_a, w_a \in a_f$ и $w_b, w_b \in b_f$, $|w_a| = |w_b|$ се споредуваат се до првото базно несовпаѓање. Тоа значи дека во најдобар случај, нееднаквоста на зборовите w_a и w_b се утврдува само со една споредба (првиот нуклеотид од w_a не соодветствува на првиот нуклеотид од w_b), додека во најлош случај, кога сите нуклеотиди освен последните се совпаѓаат, алгоритмот извршува $|w_a|$ споредби. Исто толку споредби алгоритмот извршува

и во случај на еднаквост на зборовите w_a и w_b, што претставува случај на утврдување на конзистентен пар на совпаѓање: $(R_\xi^a, R_\xi^b), (R_\xi^a, R_\xi^b) = (w_a, w_b)$.

За да се утврди најдолгото совпаѓање во рамки на опсегот на пребарување определен со ДНК фрагментите: a_f: ACTG и b_f: ACG алгоритмот извршува вкупно 6 споредби. На почеток се проверува дали постои совпаѓање со должина $|b_f| - k = 3 - 0$ за $k = 0$, односно се споредуваат зборовите: $w_{a,1} = a_{f,1-3}$: ACT и $w_{a,2} = a_{f,2-4}$:CTG од a_f со $w_{b,1} = b_f$:ACG. Совпаѓањето на почетните нуклеотиди од $w_{a,1}$: ACT и $w_{b,1}$: ACG, имплицира споредба на наредниот пар на нуклеотиди, на положба 2. Цитозин совпаѓањето на положба 2, имплицира споредба на нуклеотидите на положба 3, Тимин и Гванин. Различноста на зборовите $w_{a,1}$ и $w_{b,1}$ се утврдува врз основа на нееднаквоста на нуклеотидите на положба 3, што преставува случај на утврдување на нееднаквост на два збора со максимален број на споредби. Врз основа на нееднаквоста на почетните нуклеотиди од $w_{a,2}$: CTG и $w_{b,1}$: ACG, различноста на зборовите се утврдува само со една базна споредба, што претставува случај на утврдување на несовпаѓање со минимален број на споредби. Бидејќи не постои совпаѓање со должина 3, алгоритмот во наредниот чекор проверува дали постои совпаѓање со должина $|b_f| - k = |b_f| - 1 = 2$, за $k = 1$, односно последователно се споредуваат зборовите: AC, CT, TG од a_f со зборовите: AC и CG од b_f се до утврдување на првото совпаѓање. Првиот пар на зборови предмет на споредба е парот: (AC,AC), чија еквивалентност се утврдува со 2 базни споредби. За утврдување на совпаѓањето со максимална должина: AC во рамки на опсегот на пребарување определен со ДНК фрагментите a_f:ACTG и b_f:ACG, алгоритмот извршува 3+1+2=6 споредби.

Иницијалниот опсег на пребарување ги вклучува ДНК секвенците a и b во целост. По утврдување на совпаѓањето со максимална должина за a и b, се бараат совпаѓања во рамки на левиот и десниот опсег на пребарување, Сл. 3.9. Најдолгото совпаѓање за ДНК секвенците a и b и совпаѓањата во рамки на левиот и десниот опсег на пребарување, Сл. 3.9, одделуваат четири нови конзистентни опсези на пребарување итн. На овој начин може да се издвои комплетно множество на конзистентни совпаѓања: $\{(R_1^a, R_1^b), (R_2^a, R_2^b), \dots, (R_{k-1}^a, R_{k-1}^b), (R_k^a, R_k^b)\}$ за ДНК секвенците a и b.



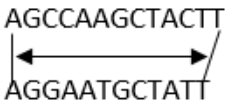
Слика 3.9. Редослед на утврдување на конзистентни совпаѓања
Figure 3.9. Order of identification of consistent hits

Секој пар на совпаѓање (R_ξ^a, R_ξ^b) се бележи со податочна четворка: $(p_{a,s}, p_{a,f}, p_{b,s}, p_{b.f})$, каде $p_{a,s}$ ја означува почетна положба на фрагментот на совпаѓање R_ξ^a, $p_{a,f}$ ја означува крајната положба на фрагментот на совпаѓање R_ξ^a, $p_{b,s}$ ја означува почетната положба на фрагментот на совпаѓање R_ξ^b и $p_{b.f}$ ја означува крајната положба на фрагментот на совпаѓање R_ξ^b. Бележејќи го секој пар на совпаѓање (R_ξ^a, R_ξ^b) со податочна четворка од тип: $\left(p_{a,s}, p_{a,f}, p_{b,s}, p_{b.f}\right) = (v_{4\xi-3}, v_{4\xi-2}, v_{4\xi-1}, v_{4\xi})$, множеството на конзистентни совпаѓања: $\left\{\left(R_1^a, R_1^b\right), \left(R_2^a, R_2^b\right), \dots, \left(R_{k-1}^a, R_{k-1}^b\right), \left(R_k^a, R_k^b\right)\right\}$ се репрезентира со вектор $v = [(v_1, v_2, v_3, v_4), (v_5, v_6, v_7, v_8), \dots, (v_{4k-3}, v_{4k-2}, v_{4k-1}, v_{4k})]$.

Почетно векторот v е празен. Неговата содржина динамички се ажурира во фаза на извршување, односно за секое новооткриено совпаѓање (R_ξ^a, R_ξ^b) се додава по една податочна четворка: $(p_{a,s}, p_{a,f}, p_{b,s}, p_{b.f})$ во v.

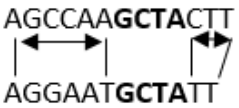
По утврдување на сите парови на совпаѓања, векторот v се сортира во растечки редослед, по почетните положби на паровите на совпаѓања: (R_ξ^a, R_ξ^b), од лево на десно.

За пример секвенците a: AGCCAAGCTACTT и b: AGGAATGCTATT, алгоритмот на почеток го наоѓа најдолгото заедничко совпаѓање: GCTA, кое се бележи со (6,9,6,9). Положбите на нуклеотидите се означуваат започнувајќи од 0. Најдолгото заедничко совпаѓање GCTA се издвојува од иницијалниот опсег на пребарување, кој ги вклучува пример секвенците a и b во целост, односно на почеток важи: $a_f = a$ и $b_f = b$, Сл. 3.10. Целобројната четворка: (6,9,6,9), која ги определува почетните и крајните положби на совпаѓањето GCTA во рамки на a и b, е првата четворка од тип: $(p_{a,s}, p_{a,f}, p_{b,s}, p_{b.f})$, која се додава во v, $v = [(6,9,6,9)]$.



Слика 3.10. Иницијален опсег на пребарување
Figure 3.10. Initial searching range

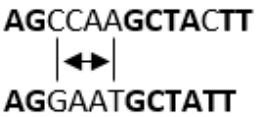
Во понатамошниот дел од извршувањето, алгоритмот бара совпаѓања со максимална должина во рамки на конзистентните опсези на пребарување, лево и десно од совпаѓањето: (GCTA,GCTA), кои се определени со паровите на ДНК фрагменти: (AGCCAA;AGGAAT) и (CTT;TT) соодветно, Сл. 3.11.



Слика 3.11. Опсези на пребарување, лево и десно од GCTA совпаѓањето
Figure 3.11. Left and right searching ranges regarding the GCTA hit

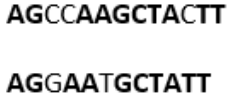
Најдолгото совпаѓање во рамки на левиот опсег на пребарување, определен со фрагментите: (AGCCAA,AGGAAT) е AG совпаѓањето кое се бележи со: (0,1,0,1), Сл. 3.12. Најдолгото совпаѓање во рамки на десниот опсег на пребарување, определен со фрагментите: (CTT,TT) е TT совпаѓањето кое се бележи со: (11,12,10,11), Сл. 3.12. Целобројната вредност 11 ја означува почетната положба на фрагментот на совпаѓање TT во рамки на секвенцата a, 12 е крајна положба на истиот фрагмент во a, 10 е почетна положба на TT совпаѓањето во рамки на секвенцата b и 11 е крајна положба на TT во b.

По утврдување на AG и TT совпаѓањата, структурата на векторот v се менува, односно $v = [(6,9,6,9), (0,1,0,1), (11,12,10,11)]$.



Слика 3.12. Опсег на пребарување помеѓу AG и GCTA совпаѓањата
Figure 3.12. Searching range between AG and GCTA hits

Во завршната фаза од пребарувањето, алгоритмот го утврдува совпаѓањето: AA во опсегот на пребарување определен со фрагментите: CCAA и GAAT, Сл. 3.13. Податочната четворка која го бележи AA совпаѓањето е: (4,5,3,4) и истата се додава во векторот v, $v = [(6,9,6,9), (0,1,0,1), (11,12,10,11), (4,5,3,4)]$.



Слика 3.13. Множество на издвоени конзистентни совпаѓања за пример секвенците a и b
Figure 3.13. Set of consistent hits for the sample sequences a and b

На крај, векторот v се сортира во растечки редослед, по почетните положби на паровите на совпаѓања (R_ξ^a, R_ξ^b), од лево па надесно, односно $v = [(0,10,1), (4,5,3,4), (6,9,6,9), (11,12,10,11)]$.

Фаза 2: Генерирање на празнинско порамнување, врз основа на релативната поместеност на паровите на последователни совпаѓања: (R_ξ^a, R_ξ^b) и $(R_{\xi+1}^a, R_{\xi+1}^b)$ за $\xi = 1,2, \dots, k-1$.

За секои две последователни совпаѓања: (R_ξ^a, R_ξ^b) и $(R_{\xi+1}^a, R_{\xi+1}^b)$, определени со: $(v_{4\xi-3}, v_{4\xi-2}, v_{4\xi-1}, v_{4\xi})$ и $(v_{4\xi+1}, v_{4\xi+2}, v_{4\xi+3}, v_{4\xi+4})$, се пресметува разликата: $d_\xi = v_{4\xi+1} - v_{4\xi-2} - (v_{4\xi+3} - v_{4\xi})$. Ако $d_\xi > 0$, R_ξ^b се издолжува за $|d_\xi|$ празнини. Инаку, ако $d_\xi < 0$, R_ξ^a се издолжува за $|d_\xi|$ празнини. Ако $|d_\xi| = 0$, последователните совпаѓања: (R_ξ^a, R_ξ^b) и $(R_{\xi+1}^a, R_{\xi+1}^b)$ се порамнуваат без да се додадат празнини.

Издолжувањето на R_ξ^a за $|d_\xi|$ празнини, резултира со поместување на совпаѓањата: $R_{\xi+1}^a, R_{\xi+2}^a, \dots, R_{k-1}^a R_k^a$ за $|d_\xi|$ положби на десно, односно паровите целобројни вредности од векторот v: $(v_{4\xi+1}, v_{4\xi+2}), (v_{4\xi+5}, v_{4\xi+6}), \dots, (v_{4k-3}, v_{4k-2})$ се зголемуваат за $|d_\xi|$. Издолжувањето на R_ξ^b за исто толку празнини, резултира со поместување на совпаѓањата: $R_{\xi+1}^b, R_{\xi+2}^b, \dots, R_{k-1}^b R_k^b$ за $|d_\xi|$ положби на десно, односно паровите целобројни вредности од векторот v, $(v_{4\xi+3}, v_{4\xi+4}), (v_{4\xi+7}, v_{4\xi+8}), \dots, (v_{4k-1}, v_{4k})$ се зголемуваат за $|d_\xi|$.

За пример секвенците a и b, последователните совпаѓања: AG и AA се определени со податочните четворки: (0,1,0,1) и (4,5,3,4) од векторот v. Бидејќи разликата $d_1 = v_5 - v_2 - (v_7 - v_4) = 4 - 1 - (3 - 1) = 3 - 2 = 1 > 0$, AG совпаѓањето во b се издолжува за $|d_1| = 1$ (една) празнина, со што се порамнуваат АА совпаѓањата, Сл.3.14. Со додавање на една празнина во b, по фрагментот на совпаѓање АА, наредните совпаѓања: АА, GCTA и TT во истата секвенца се поместуваат за едно место надесно, односно нивните почетни и крајни положби се зголемуваат за $|d_1| = 1$, на што соодветствува вектор $v = [(0,1,0,1), (4,5,3+1,4+1), (6,9,6+1,9+1), (11,12,10+1,11+1)] = [(0,1,0,1), (4,5,4,5), (6,9,7,10), (11,12,11,12)]$.

```
AGCCAAGCTACTT
| |   | |
AG_GAATGCTATT
```

Слика 3.14. Издолжување на AG за $|d_1| = 1$ (една) празнина
Figure 3.14. Extending AG for $|d_1| = 1$ gap

Следниот пар на податочни четворки кој го обработува алгоритмот е парот: (4,5,4,5) и (6,9,7,10), кој ги бележи совпаѓањата: АА и GCTA. Бидејќи: $d_2 = v_9 - v_6 - (v_{11} - v_8) = (6 - 5) - (7 - 5) = 1 - 2 = -1 < 0$, GCTA совпаѓањата се порамнуваат со додавање на $|d_2| = |-1| = 1$ (една) празнина во a по фрагментот на совпаѓање АА, Сл.3.15. Бидејќи разликата d_2 во овој случај има негативен предзнак, издолжувањето се врши во однос на АА совпаѓањето во секвенцата a. Додавањето на една празнина во a ќе резултира со поместување на совпаѓањата кои следуваат по АА за едно место на десно, односно почетните и крајни положби на совпаѓањата: GCTA и TT во a се зголемуваат за $|d_2| = 1$, на што соодветствува вектор $v = [(0,1,0,1), (4,5,4,5), (6+1,9+1,7,10), (11+1,12+1,11,12)] = [(0,1,0,1), (4,5,4,5), (7,10,7,10), (12,13,11,12)]$.

```
AGCCAA_GCTACTT
| |   | |  ||||
AG_GAATGCTATT
```

Слика 3.15. Издолжување на АА за $|d_2| = 1$ (една) празнина
Figure 3.15. Extending AA for $|d_2| = 1$ (one) gap

Последниот пар на последователни податочни четворки од векторот v е парот: (7,10,7,10) и (12,13,11,12), кој ги определува совпаѓањата: GCTA и TT.

Бидејќи: $d_3 = v_{13} - v_{10} - (v_{15} - v_{12}) = 12 - 10 - (11 - 10) = 2 - 1 > 0$, GCTA совпаѓањето во b се издолжува за $|d_3| = |1| = 1$ (една) празнина, со што се порамнуваат TT совпаѓањата, Сл. 3.16. Додавањето на празнина ќе предизвика поместување на TT совпаѓањето во b за една положба на десно, односно почетната и крајна положба на TT во b се зголемуваат за $|d_3| = 1$, $v = [(0,1,0,1), (4,5,4,5), (7,10,7,10), (12,13,11+1,12+1)] =$
$[(0,1,0,1), (4,5,4,5), (7,10,7,10), (12,13,12,13)]$. На Сл. 3.16 е дадено конченото порамнување за пример секвенците a: AGCCAAGCTACTT и b: AGGAATGCTATT со примена на предлог алгоритмот за празнинско порамнување.

```
AGCCAA_GCTACTT
||  ||  ||||  ||
AG_GAATGCTA_TT
```

Слика 3.16. Конечен резултат од примената на предлог алгоритмот за празнинско порамнување за пример секвенците a: AGCCAAGCTACTT и b: AGGAATGCTATT
Figure 3.16. Output from the proposed algorithm for the sample sequences a: AGCCAAGCTACTT and b: AGGAATGCTATT

Најголемиот дел од времето на извршување на алгоритмот се троши на фазата на издвојување на множество на конзистентни совпаѓања. Бројот на извршени споредби, за да се издвои множество на конзистентни совпаѓања, зависи од процентот на базна идентичност. За ДНК секвенци со висок процент на базна идентичност, множеството на конзистентни совпаѓања се издвојува со помал број на споредби и обратно.

За да се издвои множество на конзистентни совпаѓања за секвенците: ACTGATG и ACTGAG, за кои $PID = \frac{6}{\frac{6+7}{2}} \times 100 = \frac{12}{13} \times 100 = 92\%$, алгоритмот извршува 14 споредби, Табела 3.3. Ако се намали процентот на базна идентичност PID, кој се пресметува како количник помеѓу збирот од должините на совпаѓањата и средната должина на ДНК секвенците кои се порамнуваат, бројот на споредби се зголемува. Така, за да се издвои множество на конзистентни совпаѓања за: ACTGAAG и ACTGTG, за кои $PID = \frac{5}{\frac{6+7}{2}} \times 100 = \frac{10}{13} \times 100 = 77\%$, бројот на извршени споредби изнесува 31, Табела 3.4.

Во согласност со концепциската поставеност на алгоритмот, долгите совпаѓања, кои се локализирани на почетоците од опсезите на пребарување се утврдуваат со помал број на споредби, отколку кратките совпаѓања, кои се локализирани на краевите од опсезите на пребарување.

За да се утврди ACTGA (1,5,1,5) совпаѓањето во рамки на опсегот на пребарување определен со: a_{1-7} =ACTGATG и b_{1-6} =ACTGAG се извршуваат 12 споредби, Табела 3.3, додека за да се утврди ACTG (1,4,1,4) совпаѓањето, во рамки на опсегот на пребарување определен со: a_{1-7} =ACTGAAG и b_{1-6} =ACTGTG, алгоритмот извршува 23 споредби, Табела 3.4 (фрагментите од ДНК зборовите кои се споредуваат се до утврдување на базно несовпаѓање или

утврдување на совпаѓачки пар на ДНК зборови се означени со испрекинати правоаголници).

G-G совпаѓањето во рамки на опсегот на пребарување: a_{5-7} =AAG, b_{6-7} =TG се утврдува со 8 споредби, Табела 3.4. Ако нуклеотидите Тимин и Гванин си ги заменат положбите во b_{6-7}, b_{6-7} =GT, G-G совпаѓањето може да се утврди со 5, наместо 8 базни споредби.

Табела 3.3. Издвојување на множество на конзистентни совпаѓања за пример секвенците: a =ACTGATG и b =ACTGAG.
Table 3.3. Identification of set of consistent hits for the sample sequences a =ACTGATG and b =ACTGAG.

ДНК секвенци: a =ACTGATG, b =ACTGAG		
Опсег на пребарување:	a_{1-7} =ACTGATG, b_{1-6} =ACTGAG	Број на споредби:
Споредбени ДНК зборови:	a_{1-6} =ACTGAT b_{1-6} =ACTGAG	6
	a_{2-7} =CTGATG b_{1-6} =ACTGAG	1
	a_{1-5} =**ACTGA** b_{1-5} =**ACTGA** (1,5,1,5)	5
Опсег на пребарување:	a_{6-7} =TG, b_{6-6} =G	Број на споредби:
Споредбени ДНК зборови:	a_{6-6} =T b_{6-6} =G	1
	a_{7-7} =**G** b_{6-6} =**G** (7,7,6,6)	1
Вкупен број на споредби:		14

Табела 3.4. Издвојување на множество на конзистентни совпаѓања за пример секвенците: a =ACTGAAG и b =ACTGTG.
Table 3.4. Identification of set of consistent hits for the sample sequences a =ACTGAAG and b =ACTGTG.

ДНК секвенци: a =ACTGAAG и b =ACTGTG		
Опсег на пребарување:	**a_{1-7} =ACTGAAG, b_{1-6} =ACTGTG**	**Број на споредби:**
Споредбени ДНК зборови:	a_{1-6} =ACTGAA b_{1-6} =ACTGTG	5
	a_{2-7} =CTGAAG b_{1-6} =ACTGTG	1
	a_{1-5} =ACTGA b_{1-5} =ACTGT	5
	a_{2-6} =CTGAA b_{1-5} =ACTGT	1
	a_{3-7} =TGAAG b_{1-5} =ACTGT	1
	a_{1-5} =ACTGA b_{2-6} =CTGTG	1
	a_{2-6} =CTGAA b_{2-6} =CTGTG	4
	a_{3-7} =TGAAG b_{2-6} =CTGTG	1
	a_{1-4} =**ACTG** b_{1-4} =**ACTG** **(1,4,1,4)**	4
Опсег на пребарување:	**a_{5-7} =AAG, b_{5-6} =TG**	**Број на споредби:**
Споредбени ДНК зборови:	a_{5-6} =AA b_{5-6} =TG	1
	a_{6-7} =AG b_{5-6} =TG	1
	a_{5-5} =A b_{5-5} =T	1
	a_{6-6} =A b_{5-5} =T	1
	a_{7-7} =G b_{5-5} =T	1

	a_{5-5} =A b_{6-6} =G	1
	a_{6-6} =A b_{6-6} =G	1
	a_{7-7} =G b_{6-6} =G **(7,7,6,6)**	1
Вкупен број на споредби:		**31**

Гледано од пресметковен аспект, предложениот алгоритам за празнинско порамнување на ДНК секвенци извршува помалку споредби во споредба со алгоритмите базирани на динамичко програмирање. Исклучоци, во однос на подобрувањето на временските аспекти на извршување, може да се јават при порамнување на ДНК секвенци со мал процент на базна идентичност, кога за да се издвои множество на конзистентни совпаѓања се споредуваат поголем број на ДНК зборови. Во глобала, подобрувањето се должи на примената на концептот на парцијална споредба на ДНК зборови, се до утврдување на прв несовпаѓачки пар на нуклеотиди, што во основа овозможува идентификација на несовпаѓања, без да се споредуваат ДНК зборовите во целост.

Анализата на бројот на извршени споредби за издвојување на множество на конзистентни совпаѓања за претходните пример секвенци го потврдува подобрувањето на временските аспекти на извршување на предложениот алгоритмот во споредба со алгоритмите базирани на динамичко програмирање, при порамнување на ДНК секвенци со задоволителен процент на базна идентичност. За издвојување на множеството на конзистентни совпаѓања за првиот пар на ДНК секвенци: (a =ACTGATG, b =ACTGAG), за кои PID=92 %, алгоритмот извршува 14 споредби. За празнинско порамнување на парот на ДНК секвенци: (a =ACTGAAG, b =ACTGTG), за кои PID=77%, бројот на извршени споредби изнесува 31. Ако претходните секвенци се порамнат со примена на алгоритмите базирани на динамичко програмирање, бројот на извршени споредби изнесува: $n \times m = 7 \times 6 = 42$ по порамнет пар на ДНК секвенци и истиот не зависи од процентот на идентичност помеѓу ДНК секвенците.

За разлика од алгоритмите базирани на динамичко програмирање, кои генерираат решение врз основа на максимизација на резултат на порамнување, што во некои случаи, во зависност од изборот на метриката на порамнување, може да доведе до отфрлање на дел од совпаѓањата, со чие вклучување во решението би се намалил резултатот на порамнување, предложениот алгоритам за празнинско порамнување секогаш генерира решение за кое важи дека бројот на совпаѓања вклучени во порамнувањето е максимален, не отфрлувајќи ниту едно конзистентно совпаѓање.

Во споредба со хевристичните алгоритми (*MUMmer*, *AVID*, *GLASS*, *YASS* и др.) кај кои постои тенденција на отфрлање на кратките совпаѓања (т.н. *помалку значајни совпаѓања*) кои се наоѓаат на поголема далечина во однос на ДНК фрагментите со висока густина на совпаѓања, со примена на предложениот алгоритам се генерира порамнување кое покрај значајните совпаѓања ги вклучува и помалку значајните совпаѓања, со што се зголемува точноста на порамнување (бројот на совпаѓања вклучени во решението), што од своја страна

нуди можност за идентификација на блиска и далечна хомологија, односно се генерира модел на порамнување со кој во целост, а не парцијално, е претставена структурната и еволуциската врска помеѓу порамнетите секвенци.

Формално-математичката поставеност на алгоритмите базирани на динамичко програмирање, овозможува генерирање на оптимално решение (порамнување) кое го максимизира резултатот на порамнување, изразен како функција од награди и казни. Награда $match = s(a_i, b_j) > 0$ се доделува ако $a_i = b_j$, додека казна се доделува за порамнување на различни нуклеотиди: $mismatch = s(a_i, b_j) < 0,\ a_i \neq b_j$ и порамнување на база со празнина: $gap = s(a_i, -) < 0$ или $gap = s(-, b_j) < 0$ (вообичаено се избира $gap < mismatch$). Изборот на конкретни вредности за: $match, mismatch$ и gap може да влијае на структурата на добиеното решение, односно за различни вредности на: $match, mismatch$ и gap се добиваат решенија со различна структура.

Локалното порамнување базирано на динамичко програмирање за ДНК секвенците: a =ACTGATGA и b =ACTGAGC, ако метриката на порамнување вклучува: додела на награда $match = 1$ за порамнување на еквивалентни нуклеотиди, додела на казна: $mismatch = -1$ за порамнување на различни нуклеотиди и $gap = -2$ за порамнување на база со празнина, резултира со решение кое го вклучува само фрагментот на совпаѓање: ACTGA, Сл. 3.17, Сл. 3.18. Гванин-Гванин (G-G) совпаѓањето не е вклучено во решението, бидејќи тоа резултира со намалување на резултатот на оптимално порамнување од: $score(ACTGA; ACTGA) = 5 \times 1 = 5$ на $score(ACTGATG; ACTGA - G) = 5 \times 1 - 2 + 1 = 4$, Сл. 3.19.

Локалното порамнување, кое го генерира предложениот алгоритам го вклучува и G-G совпаѓањето, Сл. 3.19. Ако се отфрли G-G совпаѓањето, во име на максимизација на резултатот на порамнување, бришењето на нуклеотидот на положба 6 во рамки на ACTGAGC секвенцата не се детектира, Сл. 3.18. Можноста за детекција на базни мутации, независно од предефинирана метрика на порамнување, кои со примена на алгоритмите базирани на динамичко програмирање, за конкретна метрика на порамнување, останууваат недетектирани, овозможува генерирање на попрецизен и посеопфатен модел на празнинско порамнување од аспект на структурна и еволуциска меѓусебна поврзаност.

		A	C	T	G	A	G	C
	0	0	0	0	0	0	0	0
A	0	1	0	0	0	1	0	0
C	0	0	2	0	0	0	0	1
T	0	0	0	3	1	0	0	0
G	0	0	0	1	4	2	1	0
A	0	1	0	0	2	5	3	1
T	0	0	0	1	0	3	4	2
G	0	0	0	0	2	1	4	3
A	0	1	0	0	0	3	2	3

Слика 3.17. Матрица на динамичко програмирање при локално порамнување на пример секвенците: a =ACTGATGA и b =ACTGAGC
Figure 3.17. Dynamic programming matrix for local alignment of the sample sequences: a =ACTGATGA и b =ACTGAGC

```
ACTGA
|||||
ACTGA
```

Слика 3.18. Решение со примена на динамичко програмирање
Figure 3.18. Dynamic programming solution

```
ACTGATG
|||||  |
ACTGA_G
```

Слика 3.19. Решение со примена на предлог алгоритмот за празнинско порамнување
Figure 3.19. Solution generated by execution of the proposed algorithm

Со примена на хевристичните алгоритми (*MUMmer*, *AVID*, *GLASS*, *YASS* и др.), во некои случаи исто така се добива парцијално решение, во кое не се вклучени т.н. *помалку значајни совпаѓања*, односно кратките совпаѓања кои се наоѓаат на поголема оддалеченост во однос на ДНК фрагментите со највисока густина на распределба на совпаѓања. Така, на пример, ако секвенцата a =ACTGATGA се издолжи со додавање на CCCCC фрагмент по нуклеотидот T (Тимин) и секвенцата b =ACTGAGC се издолжи со додавање на AAAAA фрагмент веднаш по заедничкото ACTGA совпаѓање, дел од хевристичните алгоритми го класифицираат G-G (Гванин-Гванин) конзистентното совпаѓање како *помалку значајно совпаѓање* при порамнување на издолжените секвенци: a =ACTGATCCCCCGA и b =ACTGAAAAAAGC, што доведува до ситуација на генерирање на решение како на Сл. 3.18, во кое G-G (Гванин-Гванин) совпаѓањето исто така не е вклучено. Со примена на предложениот алгоритам за празнинско порамнување, G-G (Гванин-Гванин) конзистентното совпаѓање се детектира и истото се зема како дел од порамнувањето, Сл. 3.20, со кое покрај блиската и значајна ACTGA хомологија помеѓу пример секвенците a и b, утврдена е и далечната и помалку значајна G-G (Гванин-Гванин) хомологија, која

се зема како индикатор за далечна структурна, функционална и еволуциска поврзаност помеѓу порамнетите секвенци. Со споредба со хевристичните алгоритми, бројот на утврдени совпаѓања вклучени во порамнувањето за пример секвенците a и b се зголемува од 4 на 5, односно се потврдува придобивката од примената на предложениот алгоритам за празнинско порамнување, а тоа е зголемување на точноста на порамнување, што овозможува конструкција на модел на порамнување со кој е определена секоја конзистентна хомологија, неотфрлувајќи ги во никој случај од порамнувањето далечните и помалку значајни совпаѓања.

```
ACTGATCCCCCG
|||||         |
ACTGAAAAAA_G
```

Слика 3.20. Решение со примена на предлог алгоритмот за празнинско порамнување за издолжените секвенци a =ACTGATCCCCCGA и b =ACTGAAAAAAGC

Figure 3.20. Solution generated by execution of the proposed algorithm for the extended sequences a =ACTGATCCCCCGA and b =ACTGAAAAAAGC

Додавањето на празнини во фазата на конструкција на порамнување може да се реализира на два начини. Првиот модел за конструкција на празнинско порамнување вклучува последователно додавање на празнини по совпаѓање, не земајќи ги предвид фреквенциите на замена на нуклеотид X со нуклеотид Y во рамки на секвенцата во која се врши додавањето. Вториот модел на празнинско порамнување ја утврдува положбата на која се додава празнина врз основа на фреквенциите на замена на нуклеотид X со нуклеотид Y. Според овој модел на празнинско порамнување, празнина се додава на положба помеѓу пар на нуклеотиди XY за кој фреквенцијата на базна замена $f(XY)$ е минимална.

Фреквенцијата на базна замена $f(XY)$ се пресметува како број на преклопувачки и непреклопувачки појавувања на парот нуклеотиди XY, $X,Y = \{A,C,T,G\}$, одделно за секоја ДНК секвенца. На пример, $f(AA) = 0$ за пример секвенцата a =ACACTGGCTTGTTTAC, Сл. 3.21. За пример секвенцата b =ACACAAGCACACCAC, $f(AC) = 5$, Сл. 3.22, бидејќи парот на нуклеотиди AC во b се појавува пет пати. На идентичен начин може да се пресметаат фреквенциите на базни замени $f(XY)$ и за останатите комбинации на парови на нуклеотиди, со тоа што една матрица на базни замени се конструира за пример секвенцата a, додека друга за пример секвенцата b, Сл. 3.21, Сл. 3.22.

	A	C	T	G
A	f(AA)=0	f(AC)=3	f(AT)=0	f(AG)=0
C	f(CA)=1	f(CC)=0	f(CT)=2	f(CG)=0
T	f(TA)=1	f(TC)=0	f(TT)=3	f(TG)=2
G	f(GA)=0	f(GC)=1	f(GT)=1	f(GG)=1

Слика 3.21. Матрица на замени за секвенцата a: ACACTGGCTTGTTTAC

Figure 3.21. Substitution matrix for the sequence a: ACACTGGCTTGTTTAC

	A	C	T	G
A	f(AA)=1	f(AC)=5	f(AT)=0	f(AG)=1
C	f(CA)=5	f(CC)=1	f(CT)=0	f(CG)=0
T	f(TA)=0	f(TC)=0	f(TT)=0	f(TG)=0
G	f(GA)=0	f(GC)=1	f(GT)=0	f(GG)=0

Слика 3.22. Матрица на замени за секвенцата b: ACACAAGCACACCAC
Figure 3.22. Substitution matrix for the sequence b: ACACAAGCACACCAC

На истите пример секвенци: a =ACACTGGCTTGTTTAC и b =ACACAAGCACACCAC може да се покаже разликата помеѓу двата модели на порамнување. Во првата фаза од извршувањето на алгоритмот се издвојува множеството на конзистентни совпаѓања, кое вклучува максимален број на порамнети парови на нуклеотиди: $v = [ACAC: (0,3,0,3), GC: (6,7,6,7), AC: (14,15,13,14)]$. Во согласност со концепциската поставеност на алгоритмот, GC совпаѓањата се порамнуваат без да се додадат празнини, додека за да се порамнат AC совпаѓањата, 14-7-(13-7)=7-6=1 (една) празнина треба да се додаде во секвенцата b помеѓу: GC и AC.

Првиот модел на празнинско порамнување ја додава празнината по GC совпаѓањето, Сл. 3.23. Но таа положба не е единствената положба на која може да се додаде празнина. Празнина помеѓу GC и AC совпаѓањата во b може да се додаде на шест различни положби : _ACACC, A_CACC, AC_ACC, ACA_CC, ACAC_C и ACACC_.

```
ACACTGGCTTGTTTAC
||||  ||      ||
ACACAAGC_ACACCAC
```

Слика 3.23. Порамнување на секвенците a и b со последователно додавање на празнини
Figure 3.23. Alignment of the sequences a and b with consecutive insertion of gaps

За разлика од моделот базиран на последователно додавање на празнини, во вториот случај, алгоритмот во секој чекор го избира парот на нуклеотиди XY од опсег каде се додава една или повеќе празнини за кој фреквенцијата на базна замена $f(XY)$ е минимална. Таквиот пар на нуклеотиди XY се одделува со додавање на празнина (X_Y). Опсегот од каде се избира нуклеотиден пар со минимална фреквенција на базна замена, покрај ДНК фрагментот кој ги одделува последователните совпаѓања: $R_\xi^a(R_\xi^b)$ и $R_{\xi+1}^a(R_{\xi+1}^b)$, ги вклучува и последниот нуклеотид од $R_\xi^a(R_\xi^b)$ и првиот нуклеотид од $R_{\xi+1}^a(R_{\xi+1}^b)$.

За пример секвенците, опсегот од каде се избира нуклеотиден пар со минимална фреквенција на базна замена е: CACACCA од секвенцата b. Опсегот се добива со издолжување на ACACC фрагментот помеѓу GC и AC за еден нуклеотид на лево и на десно. Со издолжување на лево се додава последниот нуклеотид од GC совпаѓањето. Со издолжување на десно за една базна положба се додава првиот нуклеотид од AC совпаѓањето. Врз основа на фреквенциите на базна замена за паровите на нуклеотиди: CA, $f(CA) = 5$, AC, $f(AC) = 5$ и CC,

$f(CC) = 1$ од опсегот CACACCA во секвенцата b, алгоритмот избира да ја додаде празнината помеѓу CC нуклеотидниот пар, Сл. 3.24, со што се овозможува конструкција на празнинско порамнување кое покрај тоа што вклучува максимален број на порамнувања на еквивалентни нуклеотиди, ги зема предвид и веројатностите на замена на нуклеотид X со нуклеотид Y, врз основа на што се утврдуваат положбите на кои се додаваат празнини.

```
ACACTGGCTTGTTTAC
||||   ||       ||
ACACAAGCACAC_CAC
```

Слика 3.24. Порамнување на секвенците a и b врз основа на фрекфенциите на замена на нуклеотид X со нуклеотид Y
Figure 3.24. Alignment of the sequences a and b on the basis of substitution frequencies of nucleotide X with nucleotide Y

Од биолошки аспект, моделот за празнинско порамнување врз основа на фреквенциите на базна замена е повеќе издржан, бидејќи веројатноста за бришење на база од нуклеотиден пар со висока фреквенција на појавување е помала од веројатноста за бришење на база од нуклеотиден пар со мала фреквенција на појавување.

3.3. АЛГОРИТАМ ЗА ИНДЕКСИРАЊЕ И ПРЕБАРУВАЊЕ НА ДНК БАЗА НА ПОДАТОЦИ

Освен предложените алгоритми за паролво порамнување на ДНК секвенци, со чија примена се подобрува временската и мемориската комплексност во споредба со алгоритмите базирани на динамичко програмирање и се зголемува точноста на порамнување во споредба со хевристичните алгоритми, предложен е и нов хеш-базиран алгоритам за индексирање и пребарување на ДНК база на податоци, со чија примена се забрзува процесот на индексирање на масивни ДНК бази на податоци, се намалува трошокот за меморирање и пребарување на индексираната генетска содржина и се зголемува точноста на пребарување во споредба со современиците на предложениот алгоритам, како SSAHA и алгоритмот на Reneker и Shyu. Предложениот алгоритам се извршува во две фази и тоа: *фаза на индексирање* на ДНК базата на податоци и *фаза на пребарување* на индексираната ДНК содржина по ДНК прашалник.

ФАЗА НА ИНДЕКСИРАЊЕ НА ДНК БАЗА НА ПОДАТОЦИ

Нека S_i е *i*-та ДНК секвенца од база на податоци за ξ ДНК секвенци, $S = \{S_1, S_2, ..., S_{\xi-1}, S_\xi\}$, $1 \le i \le \xi$. Ако S_i вклучува *n* нуклеотиди, тогаш од S_i може да се издвојат *n-k+1* преклопувачки зборови *w* со должина *k, k<<n*. Секој збор $w: b_1 b_2 \dots b_{k-1} b_k, b_j \in \{A, C, T, G\}, 1 \le j \le k$ може да се преслика во број со примена на формулата (3.5).

$$f(w: b_1 b_2 \dots b_{k-1} b_k) = \sum_{j=1}^{k} f(b_j) \times 4^{j-1} \quad (3.5)$$

Формулата (3.5) за пресликување (индексирање) на ДНК зборови се користи во SSAHA и алгоритмот на Reneker и Shyu. Изборот на вредности за базно пресликување $f(b_j): f(A), f(C), f(T), f(G)$ е различен кај SSAHA и алгоритмот на Reneker и Shyu. Нултото пресликување на нуклеотидот Аденин (A) кај SSAHA, $f(A) = 0$, оневозможува дистинкција на различни зборови, кои завршуваат со различен број на аденини. Неконзистентноста во однос на избор на вредности за базно пресликување кај SSAHA може да се покаже за зборовите: $w_1: CAA$ и $w_2: CAAA$, кои поради еднаквите вредности на пресликување: $f(w_1: CAA) = f(C) \times 4^0 + f(A) \times 4^1 + f(A) \times 4^2 = f(C) + 0 \times 4^1 + 0 \times 4^2 = f(C)$ и $f(w_2: CAAA) = f(C) \times 4^0 + f(A) \times 4^1 + f(A) \times 4^2 + f(A) \times 4^3 = f(C) + 0 \times 4^1 + 0 \times 4^2 + 0 \times 4^3 = f(C)$ се земаат за еквивалентни, што во никој случај не смее да се дозволи.

Различна вредност на пресликување $f(w)$ за различен збор w се добива со избор на ненултни вредности на пресликување за четирите ДНК нуклеотиди. Изборот на вредности за базно пресликување како кај Reneker и Shyu: $f(A) = 1, f(T) = 2, f(G) = 3, f(C) = 4$ овозможува дистинкција на различни ДНК зборови кои завршуваат со различен број на аденини, со што се гарантира различна вредност на пресликување $f(w)$ за различен ДНК збор w.

Процесот на пресликување на ДНК зборови може да се забрза за фактор *k* (*k* е должина на ДНК зборови кои се индексираат) во споредба со SSAHA и алгоритмот на Reneker и Shyu ако секој нареден збор $w_{i,p+1}: b_{i,p+1} b_{i,p+2} \dots b_{i,p+k-1} b_{i,p+k}$, со исклучок на првиот $w_{i,1}: b_{i,1} b_{i,2} \dots b_{i,k-1} b_{i,k}$, се изведе од претходниот збор $w_{i,p}: b_{i,p} b_{i,p+1} \dots b_{i,p+k-2} b_{i,p+k-1}$ со примена на формулата (3.6). Нотацијата $w_{i,p}$ означува ДНК збор составен од *k* нуклеотиди од ДНК секвенца S_i на почетна положба p во рамки на S_i.

$$f(w_{i,p+1}) = \frac{f(w_{i,p}) - f(b_{i,p})}{4} + f(b_{i,p+k}) \times 4^{k-1} \quad (3.6)$$

Равенството (3.6) се базира на фактот дека за секои два последователни и преклопувачки ДНК зборови $w_{i,p}$ и $w_{i,p+1}$ од S_i постојат *k-1* заеднички нуклеотиди. Зборот $w_{i,p+1}$ може да се изведе од зборот $w_{i,p}$ ако: од $w_{i,p}$ се исклучи првиот нуклеотид $b_{i,p}$ ($f(w_{i,p}) - f(b_{i,p})$ во (3.6)), заедничките *k-1* нуклеотиди се поместат за едно место на лево (делење со 4 во (3.6)) и се додаде нова база $b_{i,p+k}$($+f(b_{i,p+k}) \times 4^{k-1}$ во (3.6)).

Нека S_i=AACTT… е пример ДНК секвенца и *k=4*. Откако ќе се преслика првиот збор од S_i според (3.5), $f(w_{i,1}: AACT) = f(A) \times 4^0 + f(A) \times 4^1 + f(C) \times 4^2 + f(T) \times 4^3 = 197$, секој наредент збор, кој вклучува *k=4* нуклеотиди, може да се преслика со примена на (3.6), $f(w_{i,2}: ACTT) = \frac{f(w_{i,1}) - f(A)}{4} + f(T) \times 4^3 = 49 + 2 \times 64 = 177$ итн.

За да се преслика еден збор со примена на формулата за индексирање (3.5), која ја користат SSAHA и алгоритмот на Reneker и Shyu, се извршуваат *4k* операции, односно се читаат и пресликуваат *k* нуклеотиди ($f(b_1), f(b_2), \dots, f(b_k)$), се пресметуваат *k* степени на 4 ($4^0, 4^1, \dots, 4^{k-1}$), се извршуваат *k* множења ($f(b_1) \times 4^0, f(b_2) \times 4^1, \dots, f(b_k) \times 4^{k-1}$) и исто толку додавања ($f(w: b_1 b_2 \dots b_{k-1} b_k) \leftarrow f(b_1) \times 4^0, f(w: b_1 b_2 \dots b_{k-1} b_k) \leftarrow f(w: b_1 b_2 \dots b_{k-1} b_k) + f(b_2) \times 4^1, \dots, f(w: b_1 b_2 \dots b_{k-1} b_k) \leftarrow f(w: b_1 b_2 \dots b_{k-1} b_k) + f(b_k) \times 4^{k-1}$.

За да се преслика ДНК секвенца S_i од *n* нуклеотиди, потребно е да се извршат *4k(n-k+1)* операции. Бидејќи должината на ДНК секвенцата е многу поголема од должината на ДНК зборовите кои се пресликуваат, *n>>k*, приближно *4nk* операции се извршуваат за да се пресликаат сите зборови кои вклучуваат по *k* нуклеотиди од S_i.

Ако индексирањето се врши со примена на формулата за пресликување (3.6), *4k* операции се извршуваат за да се преслика првиот збор $w_{i,1}$ од S_i. Бидејќи извршувањето на (3.6) бара 4 операции по пресликан збор: одземање ($f(w_{i,p}) - f(b_{i,p})$), делење со 4 ($\frac{f(w_{i,p}) - f(b_{i,p})}{4}$), множење ($f(b_{i,p+k}) \times 4^{k-1}$) и собирање ($\frac{f(w_{i,p}) - f(b_{i,p})}{4} + f(b_{i,p+k}) \times 4^{k-1}$) и постојат *n-k* последователни и преклопувачки зборови од *k* нуклеотиди во S_i, вкупно се извршуваат *4k+4(n-k)=4n* операции, со што се забрзува процесот на индексирање на ДНК базата на податоци за фактор *k* (*k*-пати) во споредба со SSAHA и алгоритмот на Reneker и Shyu.

Резултатот од индексирањето на ДНК базата на податоци S се чува на различен начин кај SSAHA и алгоритмот на Reneker и Shyu. SSAHA на почеток генерира хеш-табела со 4^k клучеви, каде секое појаување на ДНК збор $w_{i,p}$ е определено со податочна двојка (i, p) кон која покажува клуч $f(w_{i,p})$. Бидејќи хеш-табелата се чува во главната меморија, а капацитетот на истата е ограничен, SSAHA најчесто се користи за индексирање на мали ДНК бази на податоци. Директната комуникација помеѓу процесорот и главната меморија, овозможува брзо индексирање и пребарување на индексираните податоци.

За разлика од SSAHA, резултатот од индексирањето може да чува и во рамки на датотека, поставена на диск или сервер, со што се овозможува индексирање на долги ДНК секвенци, како што е човечкиот геном. Но додатниот податочен трансфер помеѓу главната меморијата и дискот не влијае позитивно на временските аспекти на индексирање и пребарување.

Со примена на пристапот за индексирање кај SSAHA, базиран на генерирање на хеш-табела со 4^k клучеви, пред да се започне со читање и пресликување на ДНК зборовите $w_{i,p}$, каде секое појавување на ДНК збор $w_{i,p}$ е определено со податочна двојка (i, p) кон која покажува клуч $f(w_{i,p})$, постои

можност дел од клучевите генерирани на почеток да покажуваат кон 0 податочни двојки (i, p), бидејќи истите соодветствуваат на пресликувања за ДНК зборови кои не постојат во ДНК базата на податоци.

Непотребното трошење на меморија за записи, каде клучевите кои соодветствуваат на пресликувања за ДНК зборови кои не постојат во базата на податоци може да се одбегне ако индексираната податочна структура се конструира динамички, со читање и пресликување на ДНК зборовите кои постојат во базата на податоци.

На пример, ако должината на индексирање *k* изнесува 8 и зборот *TTTGTTAA* не постои во ДНК базата на податоци, тогаш клучот $f(TTTGTTAA)$ е непотребно генериран и чуван во хеш-табелата. Ако индексираната податочна структура се генерира динамички, со додавање на записи само за ДНК зборовите кои постојат во базата на податоци, запис на кој одговара клуч $f(TTTGTTAA)$ нема да постои, со што се овозможува секој клуч да покажува кон најмалку една податочна двојка (i, p).

Со исклучување на редундантните записи од индексираната податочна структура се намалува мемориската побарувачка, а помалиот број на записи овозможува побрзо пребарување.

ФАЗА НА ПРЕБАРУВАЊЕ НА ИНДЕКСИРАНА БАЗА НА ПОДАТОЦИ ПО ДНК ПРАШАЛНИК

Од биолошки аспект, ДНК базата на податоци се пребарува за: *наоѓање на гени*, *идентификација на регулирачки секвенци*, *предвидување на интрони и егзони во рамки на гени, анализа на кратки тандемски повторувања*, *препознавање на псевдогени*, *наоѓање на рестрикциски страни* итн. Во некои од случаите, како што е предвидувањето на промотор секвенца, врз основа на познат консензус елемент, пребарувањето на ДНК базата на податоци по познат ДНК темплејт (прашалник) може да резултира со информација за функционална ДНК (множество на потенцијални гени).

За да се забрза процесот на пребарување на индексираната ДНК база на податоци, која се чува во главната меморија, наместо хеш-табела може да се употреби сортиран речник. Имајќи предвид дека записите во рамки на сортираниот речник се подредени во растечки редослед, по вредност на клуч, сите појавувања на ДНК прашалникот во базата на податоци можат да се најдат без да се проверуваат сите записи, што во секој случај ќе ги подобри временските аспекти на пребарување.

Ако е познато множеството на ДНК прашалници $Q = \{q_1, q_2, \dots, q_{n-1}, q_n\}$ кое се пребарува во ДНК база на податоци, тогаш зборовите составени од $k = |q_{max}| + 1$ нуклеотиди од ДНК базата на податоци се пресликуваат во клучеви со примена на формулата (3.5) за секој прв пресликан збор од $S_i, 1 \leq i \leq \xi$ и примена на формулата (3.6) за сите останати, каде $|q_{max}|$ е должина на најдолгиот прашалник од Q. Секое пресликување на збор $w_{i,p}$, кој се наоѓа на

почетна положба p во рамки на ДНК секвенцата S_i, се бележи со податочна двојка (i,p), кон која покажува клуч $f(w_{i,p})$. Во вакви услови, сите појавувања на ДНК прашалник q можат да се најдат како суфикси или префикси во рамки на дел од пресликаните зборови.

Сите појавувања на ДНК прашалник q на почетни положби $p > k - |q|$ во ДНК секвенца S_i, каде $|q|$ е должина на ДНК прашалник, се наоѓаат како суфикси во рамки на дел од пресликаните зборови од S_i. На пример, ДНК прашалникот q:AA е суфикс во зборот TTAA, кој е вториот пресликан збор од ДНК секвенцата S_2: CTTAAC…, за *k*=4. Почетната положба на ДНК прашалникот q:AA во ДНК секвенцата S_2: CTTAAC… може да се изведе од податочната двојка $(i,p) = (2,2)$ кон која покажува клучот $f(TTAA) = 90$ од сортираниот речник ако p се зголеми за $k - |q| = 4 - 2 = 2$, односно податочната двојка која го определува појавувањето на ДНК прашалникот q:AA во ДНК секвенцата S_2: CTTAAC… е $(i, p + k - |q|) = (2,2 + 2) = (2,4)$.

Издолжување на ДНК прашалникот q до *k* нуклеотиди со минимална вредност на пресликување се добива со додавање на $k - |q|$ A (Аденини) на почетокот на q, $q^{suffix,min} = \underbrace{A \dots A}_{k-|q|} q_1 \dots q_{|q|}$, $f(A) = 1, f(A) = \min\{f(A) = 1, f(T) = 2, f(G) = 3, f(C) = 4\}$.

Со додавање на $k - |q|$ C (Цитозини), исто така на почетокот на q, се генерира издолжување со иста должина, но со максимална вредност на пресликување $q^{suffix,max} = \underbrace{C \dots C}_{k-|q|} q_1 \dots q_{|q|}$, $f(C) = 4, f(C) = max\{f(A) = 1, f(T) = 2, f(G) = 3, f(C) = 4\}$.

Секој клуч од сортираниот речник за кој важи неравенството (3.7) покажува кон една или повеќе податочни двојки (i,p) кои бележат ДНК зборови кои завршуваат со пребаруваниот ДНК прашалник. Совпаѓањата се генерираат во облик на податочни двојки од тип $(i, p + k - |q|)$, каде (i,p) е податочна двојка кон која покажува клуч за кој важи неравенството (3.7).

$$f(q^{suffix,min}) \leq key \leq f(q^{suffix,max}) \quad (3.7)$$

Суфикс-пребарувањето на совпаѓања по ДНК прашалник во рамки на индексираната ДНК база на податоци се извршува се додека не се прочита првиот клуч за кој важи $key > f(q^{suffix,max})$. Ниту еден од клучевите за кои важи $key > f(q^{suffix,max})$ не соодветствува на пресликан ДНК збор кој го содржи пребаруваниот ДНК прашалник како суфикс. Следствено, таквите записи не се обработуваат, со што во споредба со SSAHA се подобруваат временските аспекти на пребарување.

Преклопувачките совпаѓања на ДНК прашалник q и совпаѓањата со должина помала од должината на индексирање *k*, кои не можат да се утврдат со примена на SSAHA, се дел од недостатоците на SSAHA кои ги решава алгоритмот на Reneker и Shyu.

Суфикс-базираната имплементација на алгоритмот на Reneker и Shyu при пребарување на ДНК прашалник q, чија должина е помала од должината на

индексирање k, оневозможува идентификација на совпаѓања на почетни положби $p \leq k - |q|$ во индексирана ДНК секвенца $S_i, 1 \leq i \leq \xi$. Овој недостаток може да резултира со биолошки последици како, на пример, нецелосна идентификација на повторувачки нуклеотидни секвенци во рамки на хромозомски завршетоци (теломери) при пребарување во инверзна насока и проблеми при идентификација на клучни ДНК шаблони како, на пример, познат промотор консензус елемент во кратки исчитувања на ДНК.

Овој недостаток може да се реши со примена на формулите (3.8) и (3.9). Секој клуч од сортираниот речник кој ги задоволува ограничувањата (3.8) и (3.9), односно се наоѓа во опсегот помеѓу $f(q^{prefix,min})$ и $f(q^{prefix,max})$, каде $f(q^{prefix,min})$ и $f(q^{prefix,max})$ се добиени со додавање на $k - |q|$ A (Аденини) и C (Цитозини) на крајот од ДНК прашалникот q ($q^{prefix,min} = q_1 \dots q_{|q|} \underbrace{A \dots A}_{k-|q|}$, $q^{prefix,max} = q_1 \dots q_{|q|} \underbrace{C \dots C}_{k-|q|}$) и остатокот при делење на разликата $key - f(q)$ со $4^{|q|}$ е еднаква на 0, покажува кон една или повеќе податочни двојки (i, p) кои определуваат еден или повеќе ДНК зборови кои го содржат ДНК прашалникот q како префикс.

$$f(q^{prefix,min}) \leq key \leq f(q^{prefix,max}) \quad (3.8)$$

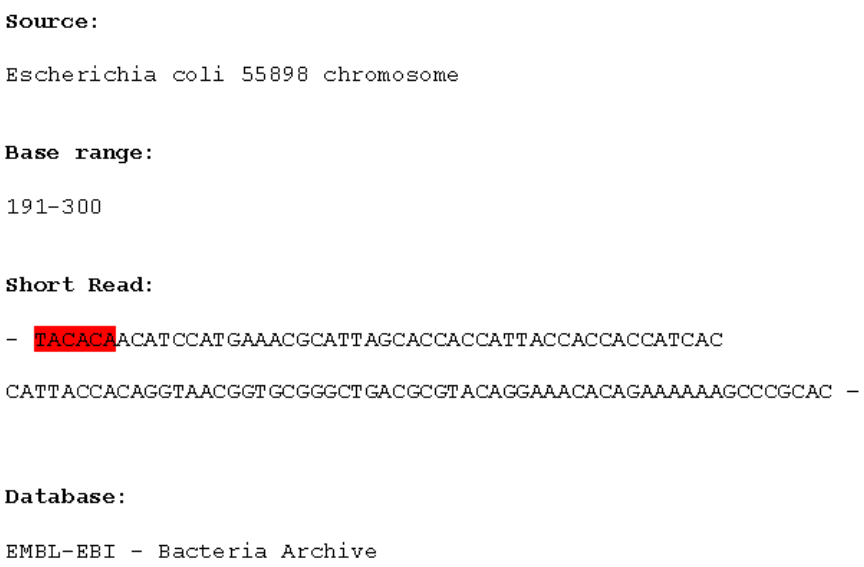
$$mod\left(key - f(q), 4^{|q|}\right) = 0 \quad (3.9)$$

Ако со примена на алгоритмот на Reneker и Shyu се пребарува краткото читање од *E. coli 55989* хромозомот во базен опсег помеѓу 191 и 300 по *E. coli thrL* ген промотор консензус TACACA (на положба -10 релативно во однос на транскрипцискиот почеток), за должина на индексирање *k*=8 не може да се утврди TACACA совпаѓањето на почетокот од читањето, Сл. 3.25.

Тоа се должи на суфикс пребарувањето по ДНК прашалник, со чија примена се утврдуваат совпаѓањата како завршетоци во рамки на дел од пресликаните ДНК зборови. Консензус елементот TACACA е лоциран на почеток од хромозомското исчитување и ниту еден од пресликаните ДНК зборови кои вклучуваат по *k*=8 нуклеотиди: TACACAAC, ACACAACA, CACAACAT… ,Сл. 3.25, не го содржи бараниот ДНК прашалник TACACA како суфикс. Затоа со примена на алгоритмот на Reneker и Shyu, совпаѓањето на почетна положба 0 не може да се утврди.

Ако освен суфикс пребарување се примени и префикс ДНК пребарување, кое се базира на примена на (3.8) и (3.9), совпаѓањето TACACA на почеток од хромозомското читање може да се утврди. Кога ќе се прочита клуч со вредност 71814, кој одговара на првиот пресликан збор TACACAAC, ограничувањата: (3.8) и (3.9) се задоволени, односно $22662 = f\left(q^{prefix,min}\right) \leq key = 71814 \leq f\left(q^{prefix,max}\right) = 84102$ и $mod\left(key - f(q), 4^{|q|}\right) = mod(71814 - 2182, 4^6) = mod(69632{,}4096) = 0$. Пронајденото совпаѓање е определено со податочната двојка $(i, 0)$, под претпоставка дека краткото исчитување од *E. coli 55989 хромозомот* е i-тата индексирана ДНК секвенца од ДНК базата на податоци S. Уште пред да се започне со префикс пребарување се пресметуваат: $f(q) =$

$f(TACACA) = 2182$, $f(q^{prefix,min}) = f(TACACAAA) = 22662$ и $f(q^{prefix,max}) = f(TACACACC) = 84102$.



Слика 3.25. Кратко исчитување од *E. coli 55989* хромозом, базен опсег: 191-300
Figure 3.25. Short read from *E. coli 55989* chromosome, base range: 191-300

СТУДИЈА НА СЛУЧАЈ И ДИСКУСИЈА: Нека S е ДНК база на податоци, која ги содржи секвенците: S_1:AACTTG, S_2:CTTAAC и S_3: AAACTG. Ако должината на најдолгиот ДНК прашалник кој се пребарува во S изнесува 3, алгоритмот во фазата на индексирање ги пресликува сите ДНК зборови составени од *k*=3+1=4 нуклеотиди. Пресликувањата се извршуваат со примена на формулата (3.5) за секој прв пресликан збор од $S_i, 1 \leq i \leq 3$ и примена на формулата (3.6) за сите останати зборови.

Примената на формулата (3.6) го забрзува процесот на индексирање во споредба со SSAHA и алгоритмот на Reneker и Shyu, кај кои податочното индексирање се врши исклучиво со примена на формулата (3.5).

За да се преслика првиот збор од S_1, AACT со примена на формулата (3.5) се извршуваат $4 \times k = 4 \times 4 = 16$ операции (се читаат и пресликуваат четири нуклеотиди: $f(A) = 1, f(A) = 1, f(C) = 4, f(T) = 2$, се пресметуваат четири степени на 4: $4^0, 4^1, 4^2, 4^3$, се извршуваат четири множења: $f(A) \times 4^0 = 1, f(A) \times 4^1 = 4, f(C) \times 4^2 = 64, f(T) \times 4^3 = 128$ и четири додавања: $f(AACT) = f(A) \times 4^0 = 1$, $f(AACT) = f(AACT) + f(A) \times 4^1 = 1 + 4 = 5$, $f(AACT) = f(AACT) + f(C) \times 4^2 = 5 + 64 = 69$, $f(AACT) = f(AACT) + f(T) \times 4^3 = 69 + 128 = 197$. Ако формулата (3.5) се примени за пресликување на вториот и третиот збор од S_1: ACTT и CTTG, вкупниот број на извршени операции за да се преслика секвенцата S_1 изнесува: $3 \times 4 \times k = 3 \times 4 \times 4 = 48$.

Ако вториот и третиот збор од S_1: ACTT и CTTG се пресликаат со примена на формулата (3.6), за секое пресликување се извршуваат по *k*=4 операции. На пример, за да се преслика вториот збор ACTT од S_1, од претходниот збор AACT се одзема првиот нуклеотид A (Аденин) ($f(AACT) - f(A) = 197 - 1 = 196$), заедничките ACT нуклеотиди се поместуваат за една положба на лево

($\frac{f(AACT)-f(A)}{4} = \frac{196}{4} = 49$), се додава нуклеотид Т (Тимин) на положба *k* ($f(T) \times 4^3 = 2 \times 64 = 128$) и се пресметува збирот $\frac{f(AACT)-f(A)}{4} + f(T) \times 4^3 = 49 + 128 = 177 = f(ACTT)$. Со тоа бројот на извршени операции се намалува од 48 на 16+4+4=24.

Постапката за пресликување на сите зборови од ДНК базата на податоци S е дадена во продолжение.

$$f(w_{1,1}: AACT) = f(A) \times 4^0 + f(A) \times 4^1 + f(C) \times 4^2 + f(T) \times 4^3$$
$$= 1 \times 4^0 + 1 \times 4^1 + 4 \times 4^2 + 2 \times 4^3 = 197$$

$$f(w_{1,2}: ACTT) = \frac{f(w_{1,1}) - f(A)}{4} + f(T) \times 4^3 = 49 + 2 \times 64 = 177$$

$$f(w_{1,3}: CTTG) = \frac{f(w_{1,2}) - f(A)}{4} + f(G) \times 4^3 = 44 + 3 \times 4^3 = 236$$

$$f(w_{2,1}: CTTA) = f(C) \times 4^0 + f(T) \times 4^1 + f(T) \times 4^2 + f(A) \times 4^3 = 108$$

$$f(w_{2,2}: TTAA) = \frac{f(w_{2,1})-f(C)}{4} + f(A) \times 4^3 = 26 + 64 = 90$$

$$f(w_{2,3}: TAAC) = \frac{f(w_{2,2}) - f(T)}{4} + f(C) \times 4^3 = 22 + 256 = 278$$

$$f(w_{3,1}: AAAC) = f(A) \times 4^0 + f(A) \times 4^1 + f(A) \times 4^2 + f(C) \times 4^3 = 277$$

$$f(w_{3,2}: AACT) = \frac{f(w_{3,1})-f(A)}{4} + f(T) \times 4^3 = 69 + 128 = 197$$

$$f(w_{3,3}: ACTG) = \frac{f(w_{3,2}) - f(A)}{4} + f(G) \times 4^3 = 49 + 192 = 241$$

Секое пресликување на ДНК збор $w_{i,p}$ од базата на податоци S, $f(w_{i,p})$ се додава како клуч во рамки на сортиран речник, кој покажува кон една или повеќе податочни двојки (i, p), каде i е индекс на ДНК секвенца (S_i) од каде зборот $w_{i,p}$ е пресликан и p е почетна положба на зборот $w_{i,p}$ во ДНК секвенца S_i.

На пример, клучот 90 покажува кон податочна двојка (2,2) и истиот се генерира со пресликување на ДНК зборот ТТАА од ДНК секвенцата S_2: СТТААС, кој се наоѓа на почетна положба 2 во S_2, Сл. 3.26.

Ако индексираната податочна структура се генерира динамички, со додавање на записи само за ДНК зборови кои постојат во ДНК базата на податоци S, тогаш може да се генерира индексирана податочна структура од тип

сортиран речник, Сл. 3.26, кој вклучува 8 записи од тип <*Клуч*=$f(w)$,*Вредност*=(i,p), ...>.

За иста должина на индексирање *k*=4, SSAHA генерира хеш-табела со $4^k = 4^4 = 256$ клучеви, од $f(AAAA) = 0$ до $f(TTTT) = 255$, Сл. 3.27, од кои само 8 покажуваат кон барем една (i,p) податочна двојка, под претпоставка дека се индексирани сите преклопувачки ДНК зборови од базата на податоци S. Кај SSAHA вредноста на пресликување за нуклеотидот Аденин $f(A) = 0$ е минимална, додека вредноста на пресликување на нуклеотидот Тимин $f(T) = 3$ е максимална.

Бројот на редундантни записи во рамки на хеш-табелата изнесува: 256-8=248 и истите претставуваат мемориско оптоварување, кое го отежнува процесот на пребарување во втората фаза од извршувањето на алгоритмот.

Клуч	Вредност
90	(2,2)
108	(2,1)
177	(1,2)
197	(1,1), (3,2)
236	(1,3)
241	(3,3)
277	(3,1)
278	(2,3)

Слика 3.26. Динамичка конструкција на индексирана податочна структура
Figure 3.26. Dynamic construction of the indexed data structure

Клуч	Вредност
0 | $f(AAAA)$	
	….
	….
	….
	….
	….
	….
255 | $f(TTTT)$	

Слика 3.27. Хеш-табела според SSAHA
Figure 3.27. Hash-table according SSAHA

Индексираната податочна структура, во конкретниот случај сортиран речник, се чува во главната меморија и истата може да се пребарува по ДНК прашалник q. Појавувањата на ДНК прашалник q на почетни положби: $p > k - |q|$ во индексирани ДНК секвенци $S_i, 1 \leq i \leq \xi$ од ДНК базата на податоци S, каде: *k* е должина на индексирање и $|q|$ е должина на ДНК прашалник, се наоѓаат како суфикси во рамки на $h, h \in 0 \cup Z^+$ пресликани ДНК зборови $w_{i,p}$.

Ако пресликан ДНК збор $w_{i,p}$ (определен со податочна двојка (i,p)) го содржи ДНК прашалникот q како суфикс, тогаш појавувањето на q во индексирана ДНК секвенца S_i е определено со податочна двојка $(i,p+k-|q|)$. Секоја податочна двојка $(i,p+k-|q|)$ се добива од податочна двојка (i,p) кон која покажува клуч, чија вредност е во опсегот помеѓу: $q^{suffix,min}$($q^{suffix,min}=\underbrace{A\ldots A}_{k-|q|}q_1\ldots q_{|q|}$) и $q^{suffix,max}$($q^{suffix,max}=\underbrace{C\ldots C}_{k-|q|}q_1\ldots q_{|q|}$), со зголемување на p за $k-|q|$.

Зголемувањето се должи на фактот дека пред појавувањето на q во $w_{i,p}$ постојат $k-|q|$ нуклеотиди, па ако почетната положба на ДНК збор $w_{i,p}$ во индексирана ДНК секвенца S_i е p, тогаш почетната положба на ДНК прашалникот q во истата секвенца (S_i) е $p+k-|q|$.

Нека ДНК прашалникот q: AA се пребарува во индексираната база на податоци S: {S_1: AACTTG, S_2: CTTAAC и S_3: AAACTG}, Сл. 3.26. За да се најдат појавувањата на ДНК прашалникот на почетни положби $p>k-|q|=4-2=2$, пред да се започне со пребарување се пресметуваат: $f\left(q^{suffix,min}\right)=f(AAAA)=85$ и $f\left(q^{suffix,max}\right)=f(CCAA)=100$. Од клучевите во сортираниот речник за базата на податоци S, Сл. 3.26, единствено клучот со вредност 90 се наоѓа во опсегот помеѓу: $f\left(q^{suffix,min}\right)=f(AAAA)=85$ и $f\left(q^{suffix,max}\right)=f(CCAA)=100$ и истиот покажува кон податочната двојка$(i,p)=(2,2)$. Целобројната вредност $i=2$ го определува индексот на ДНК секвенцата (S_2) која го содржи ДНК прашалникот, додека $p=2$ е почетна положба на ДНК зборот $w_{2,2}$: TTAA во S_2, каде q:AA се појавува како суфикс. Ако $p=2$ се зголеми за $k-|q|=4-2=2$, тогаш единственото појавување на ДНК прашалникот q:AA во ДНК базата на податоци S на почетна положба $p>k-|q|=4-2=2$ е определено со податочната двојка $(i,p+k-|q|)=(2,2+4-2)$.

За разлика од SSAHA, каде во фаза на пребарување се обработуваат сите записи, предложениот алгоритам ги наоѓа сите совпаѓања на ДНК прашалник во ДНК база на податоци на почетни положби $p>k-|q|$ се додека не се прочита првиот клуч, чија вредност е поголема од $q^{suffix,max}$.

За пример базата на податоци S пребарувањето по ДНК прашалник „AA“ на почетни положби $p>k-|q|=4-2=2$ ќе заврши откако ќе се прочита клучот 108 од сортираниот речник, кој воедно е и прв клуч поголем од $q^{suffix,max}=100$. Бидејќи не постои пресликување за ДНК збор со вредност поголема од 100, каде ДНК прашалникот „AA“ е содржан како суфикс, наместо да се обработуваат 8 записи тој број се намалува на 2, со што додатно се подобруваат временските аспекти на пребарување.

За разлика од алгоритмот на Reneker и Shyu, со чија примена не можат да се утврдат совпаѓања на ДНК прашалник q во индексирани ДНК секвенци $S_i, 1\leq i\leq\xi$ на почетни положби $p<k-|q|$ (при пребарување по ДНК прашалник q чија должина $|q|$ е помала од должината на индексирање *k*), овие совпаѓања можат да се утврдат со примена на предложениот алгоритам, кој покрај суфикс пребарување вклучува и префикс пребарување по ДНК прашалник.

Секое совпаѓање на почетна положба $p \leq k - |q|$ е определено со податочна двојка $(i, p), p \leq k - |q|$ кон која покажува клуч кој се наоѓа во опсегот помеѓу: $q^{prefix,min}(q^{prefix,min}: q_1 \dots q_{|q|} \underbrace{A \dots A}_{k-|q|})$ и $q^{prefix,max}(q^{prefix,max}: q_1 \dots q_{|q|} \underbrace{C \dots C}_{k-|q|})$ и за кој важи дека разликата помеѓу вредноста на клучот и пресликувањето на прашалникот $f(q)$ е делива со $4^{|q|}$, $mod(key - f(q), 4^{|q|}) = 0$.

Пред да се започне со пребарување на ДНК базата на податоци S по ДНК прашалник q:AA на почетни положби $p \leq k - |q| = 4 - 2 = 2$, се пресметуваат: $f(q^{prefix,min}) = AAAA = 85$, $f(q^{prefix,max}) = AACC = 325$ и $f(q: AA) = 1 \times 4^0 + 1 \times 4^1 = 5$.

Клучот 197 е првиот клуч од сортираниот речник за кој важи дека се наоѓа во опсегот помеѓу: $f(q^{prefix,min}) = AAAA = 85$ и $f(q^{prefix,max}) = AACC = 325$ и за кој важи дека разликата помеѓу вредноста на клучот и пресликувањето на прашалникот $f(q)$ е делива со $4^{|q|}$, $85 = f(q^{prefix,min}) \leq key = 197 \leq f(q^{prefix,max}) = 325$, $mod(key - f(q), 4^{|q|}) = mod(197 - 5, 4^2) = mod(192,16) = 0$.

Истиот покажува кон податочните двојки: (1,1) и (3,2) со кои се определени две појавувања на ДНК прашалникот q:AA на почетни положби $p \leq k - |q| = 4 - 2 = 2$. Податочната двојка (1,1) определува појавување на ДНК прашалникот „AA“ на почетна положба 1 во ДНК секвенцата S_1:AACTTG, додека втората податочна двојка (3,2) определува појавување на ДНК прашалникот на почетна положба 2 во секвенцата S_3:AAACTG.

Претходните ограничувања, освен за клучот 197, се задоволени и за клучот 277: $85 = f(q^{prefix,min}) \leq key = 277 \leq f(q^{prefix,max}) = 325, mod(key - f(q), 4^{|q|}) = mod(277 - 5, 4^2) = mod(272,16) = 0$, кој покажува кон податочна двојка (3,1) со кое е определено совпаѓање на почетна положба 1 во секвенцата S_3: AAACTG.

Вкупниот број на пронајдени совпаѓања на ДНК прашалникот q:AA во индексираната ДНК база на податоци изнесува 4 (совпаѓањето (2,2) на почетна положба $p > k - |q| = 4 - 2 = 2$ е пронајдено со суфикс пребарување, додека останатите три совпаѓања: (1,1), (3,2) и (3,1) на почетни положби $p \leq k - |q| = 4 - 2 = 2$ се пронајдени со префикс пребарување на сортираниот речник).

Ако се примени алгоритмот на Reneker и Shyu за пример базата на податоци S и ДНК прашалник q: AA, тогаш може да се утврди само едно совпаѓање, совпаѓањето (2,2). Совпаѓањата на почетни положби $p \leq k - |q| = 4 - 2 = 2$: (1,1), (3,2) и (3,1) не можат да се утврдат со примена на алгоритмот на Reneker и Shyu, бидејќи истите не фигурираат како суфикси во рамки на пресликани ДНК зборови.

Предложениот алгоритам освен суфикс пребарување по ДНК прашалник, вклучува и префикс пребарување, со што може да се утврди секое совпаѓање, вклучувајќи ги и (1,1), (3,2), (3,1) совпаѓањата во S, со што се зголемува точноста на пребарување во споредба со алгоритмот на Reneker и Shyu.

4. СОФТВЕРСКА БИБЛИОТЕКА

Предложените алгоритми за порамнување (беспразнинско порамнување и двата модели на празнинско порамнување) и индексирање – пребарување на ДНК секвенци се имплементирани во C#/C++ хибридна програмска околина. Почетниот кориснички интерфејс на интегрираната софтверска библиотека е прикажан на Сл. 4.1. На корисникот му е овозможено да избере една од понудените опции: „*FLAG Ungapped Alignment*“, „*DNA Gapped Pairwise Alignment*“ и „*DNA Database indexing and search*“.

Слика 4.1. Почетен кориснички интерфејс на интегрираната софтверската библиотека за порамнување и пребарување на ДНК секвенци
Figure 4.1. Starting user interface of the integrated software library for DNA alignment and DNA searching

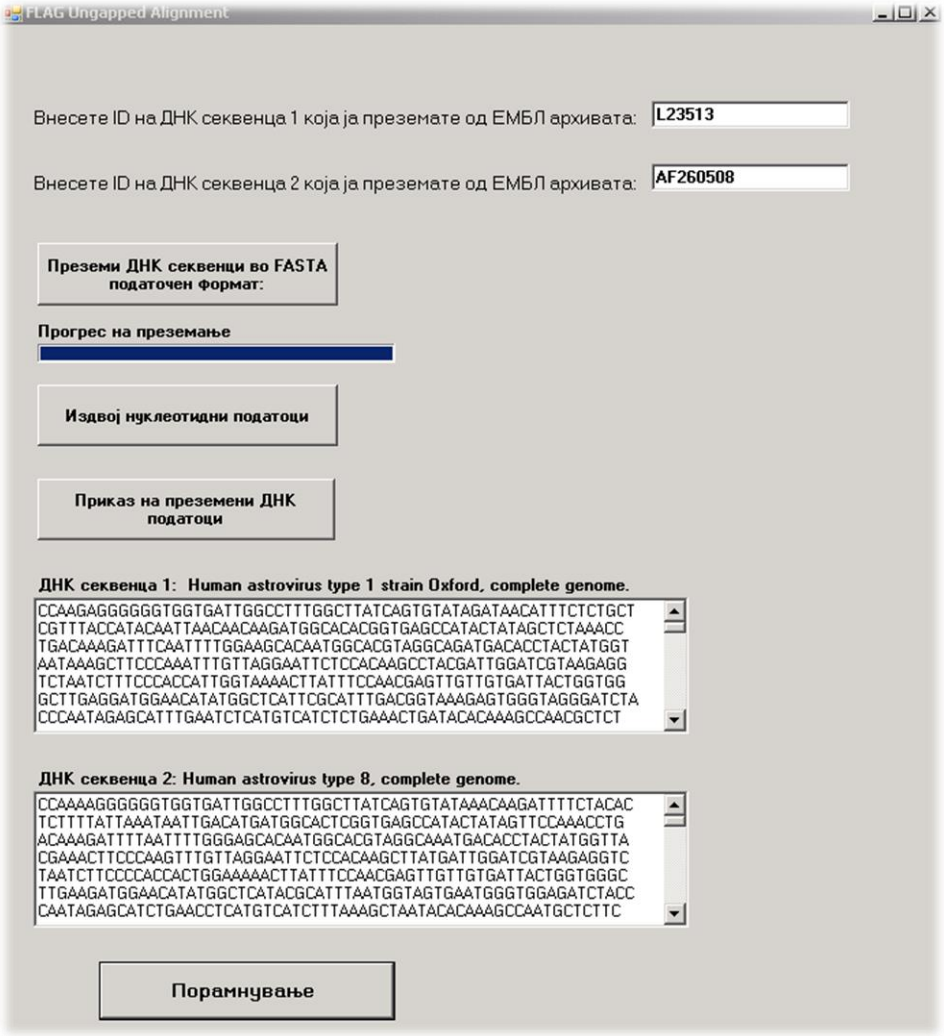
Со избор на опцијата „*FLAG Ungapped Alignment*“ се активира C#/C++ програмски модул – имплементација на предложениот алгоритам за брза идентификација на оптимална беспразнинска хомологија помеѓу две ДНК секвенци за кориснички дефинирана метрика на порамнување.

Со избор на опцијата „*DNA Gapped Pairwise Alignment*“ се активира C# програмски модул за празнинско порамнување врз основа на предложените модели за последователно додавање на празнини по совпаѓање и додавање на празнини на положби во зависност од фреквенциите на базни замени.

Со избор на опцијата „*DNA Database indexing and search*“ се подига C# хеш-базирана имплементација на предложениот алгоритам за индексирање на ДНК база на податоци и пребарување на индексираната содржина по функционално анотиран ДНК прашалник.

Изборот на опцијата „*FLAG Ungapped Alignment*“ активира C# форма во која корисникот треба да ги наведе идентификаторите на ДНК секвенците кои се преземаат од ENA, Европската Нуклеотидна Архива (ENA, European Nucleotide

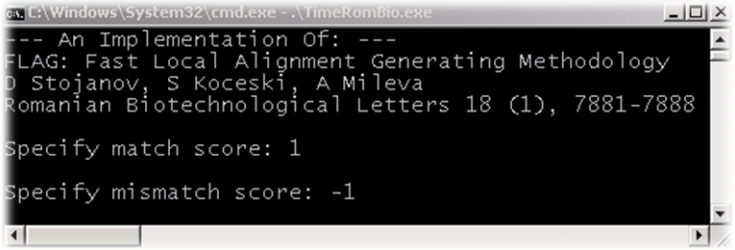
Archive), Сл. 4.2. Преземените податоци се зачувуваат во локални текстуални датотеки на апликацијата, во кои освен податоци за редоследот на нуклеотиди, во заглавја, на почеток од датотеките се чуваат податоци и за имињата на преземените секвенци. Податоците за ДНК секвенците во FASTA податочен облик (*име на секвенца + редослед на нуклеотиди*) се преземаат со кориснички избор на опцијата „*Преземи ДНК секвенци во FASTA податочен формат*“, Сл. 4.2. Опцијата „*Издвој нуклеотидни податоци*“ ги издвојува податоците за нуклеотидната содржина од заглавјата и истите ги зачувува во одделни апликациски датотеки, Сл. 4.2. Дополнително, опцијата „*Приказ на преземени ДНК податоци*“ му овозможува на корисникот приказ на редоследот на нуклеотиди кај преземените секвенци во две одделни текстуални контроли, Сл. 4.2.



Слика 4.2. C# форма за преземање, филтрирање и приказ на преземени ДНК секвенци од Европската Нуклеотидна Архива
Figure 4.2. C# form for DNA data retrieval, filtering and display obtained from the European Nucleotide Archive

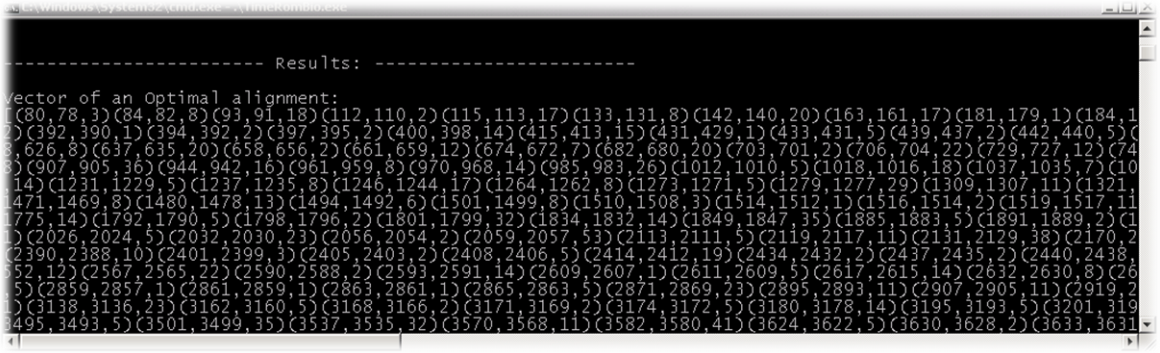
Преземените секвенци се порамнуваат со избор на опцијата „*Порамнување*“, Сл. 4.2, по што се активира C++ апликација, каде корисникот ја дефинира метриката на порамнување, односно специфицира награда за

порамнување на еквивалентни нуклеотиди (*match score*) и казна за порамнување на различни нуклеотиди (*mismatch score*), Сл. 4.3.



Слика 4.3. Кориснички-дефинирана метрика на порамнување
Figure 4.3. User-defined alignment metrics

По извршување, апликацијата ги печати: векторот на оптимално беспразнинско порамнување, Сл. 4.4, резултатот на порамнување и структурата на оптималното беспразнинско порамнување за кориснички дефинираната метрика, Сл. 4.5.

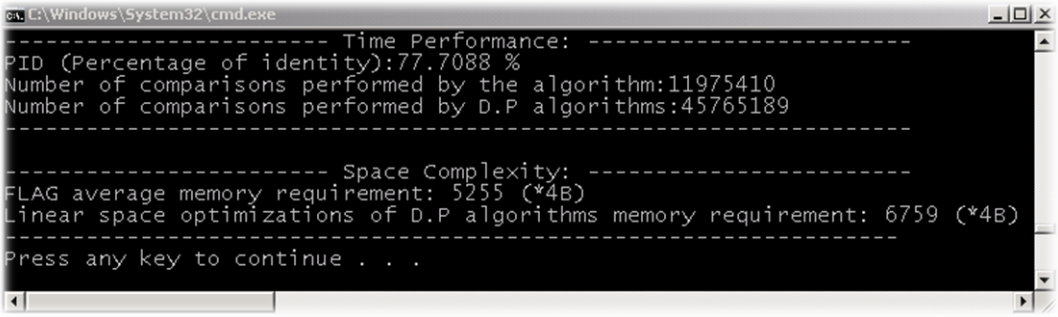


Слика 4.4. Вектор на оптимално порамнување
Figure 4.4. Vector of an optimal alignment

Слика 4.5. Резултат и структура на оптимално порамнување
Figure 4.5. Score and structure of an optimal alignment

Дополнително се печатат и детали за: процентот на идентичност (PID), временската комплексност – изразена во број на извршени споредби и просечната мемориска побарувачка на алгоритмот во фаза на извршување, Сл. 4.6.

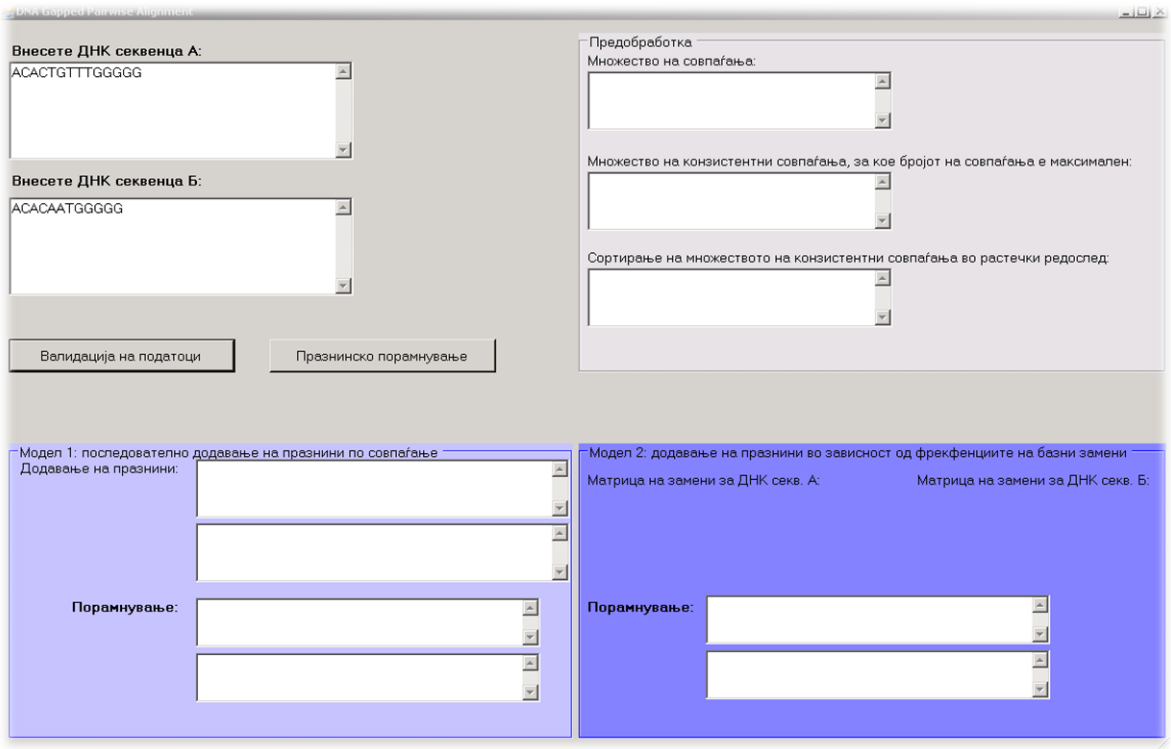
За да се овозможи компаративна анализа на перформансите на предложениот алгоритам со алгоритмите базирани на динамичко програмирање, се печатат и детали за временската и мемориската комплексност кога решението би се генерирало со примена на динамичко програмирање и тоа во најоптималните случаи на примена на динамичко програмирање, Сл. 4.6.



Слика 4.6. Процент на идентичност, споредба на пресметковните и мемориски перформанси на предложениот алгоритам со пресметковните и мемориски перформанси на алгоритмите базирани на динамичко програмирање
Figure 4.6. Percent of identity, comparison of the computational and memory requirements of the proposed algorithm to the computational and memory requirements of dynamic-programming based algorithms

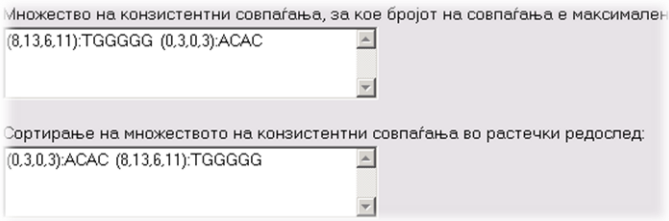
Се очекува со примена на предложениот алгоритам на исти тест секвенци да се намали бројот на извршени споредби и мемориските трошок во фаза на извршување во споредба со алгоритмите базирани на динамичко програмирање.

Опцијата „*DNA Gapped Pairwise Alignment*“ активира C# програмски модул за празнинско порамнување на две ДНК секвенци врз основа на предложените модели за последователно додавање на празнини по совпаѓање и утврдување на положбите каде се додаваат празнини врз основа на фреквенциите на базни замени, Сл. 4.7. Податоците за ДНК секвенците кои се порамнуваат се внесуваат во две одделни текстуални контроли, Сл. 4.7. Пред порамнување, се врши валидација на кориснички-внесените податоци над азбуката $\Sigma = \{A, C, T, G\}$, со избор на опцијата „*Валидација на податоци*“, Сл. 4.7. Порамнувањето се извршува со избор на опцијата „*Празнинско порамнување*“, Сл. 4.7.



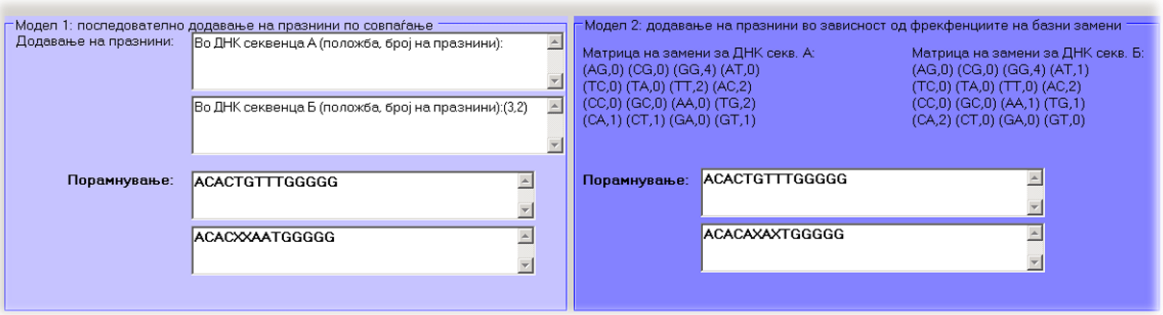
Слика 4.7. Апликација за празнинско порамнување на ДНК секвенци
Figure 4.7. Application for gapped DNA alignment

Опцијата „*Празнинско порамнување*“, Сл. 4.7, повикува C# функција со која се утврдуваат структурите на порамнувањата врз основа на предложените модели за празнинско порамнување на ДНК секвенци. За да се применат моделите, во согласност со концепциската поставеност на алгоритмот, на почеток се утврдува множество на конзистентни совпаѓања, кое вклучува максимален број на единични совпаѓања. Совпаѓањата се утврдуваат со споредба на зборови со еднаква должина, се до утврдување на прво базно несовпаѓање во рамки на споредбен пар на зборови. Со примена на овој пристап се минимизира бројот на извршени базни споредби. Множеството на конзистентни совпаѓања и неговиот сортиран еквивалент, по редослед на појавување на совпаѓањата, се печатат во горниот десен панел на апликацијата, Сл. 4.7, Сл. 4.8.



Слика 4.8. Множество на конзистентни совпаѓања и негов сортиран еквивалент
Figure 4.8. Set of consistent hits and its sorted equivalent

Структурата на порамнувањето добиено со примена на моделот за последователно додавање на празнини по совпаѓање и структурата на порамнувањето добиено со примена на моделот за додавање на празнини помеѓу несовпаѓачки базни парови XY со најмала фреквенција на појавување, се печатат во долниот панел на апликацијата, Сл. 4.9. Положбите на кои се додаваат празнини се означени со знакот „X“, додека над панелот каде е прикажан резултатот од примената на вториот модел на порамнување, се печатат и фреквенциите на базни замени за секвенците кои се порамнуваат.



Слика 4.9. Резултати од примената на предложените модели за празнинско порамнување на ДНК секвенци

Figure 4.9. Results of the application of the proposed models for gapped DNA alignment

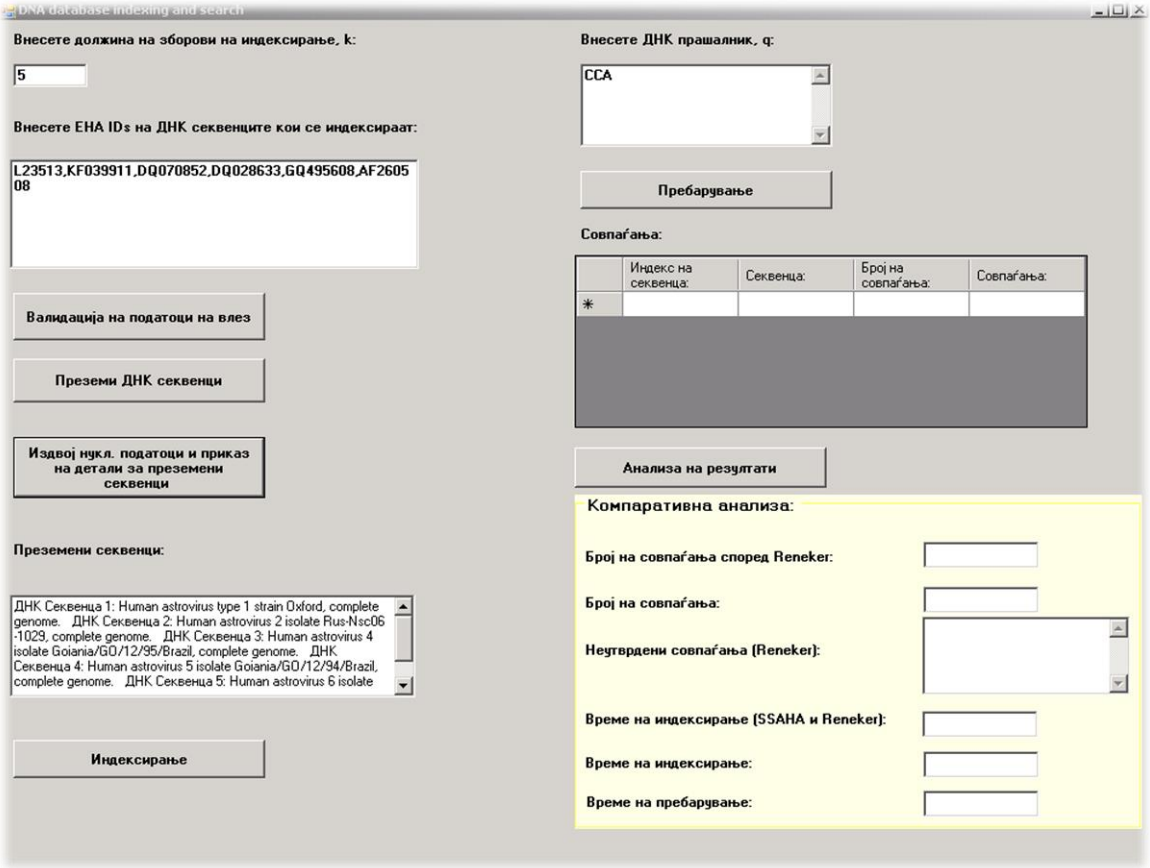
Откако ќе се испечатат порамнувањата, предмет на компаративна анализа се бројот на базни совпаѓања вклучени во решенијата добиени со примена на предложените модели и бројот на базни совпаѓања вклучени во решенијата добиени со примена на хевристичните алгоритми и алгоритмите базирани на динамичко програмирање.

Се очекува со примена на предложените модели да се зголеми точноста на порамнување, односно за исти тест секвенци бројот на базни совпаѓања вклучени во решенијата добиени со примена на предложените модели да биде поголем од бројот на базни совпаѓања вклучени во решенијата добиени со примена на хевристичните алгоритми и алгоритмите базирани на динамичко програмирање.

Имплементацијата на предложениот алгоритам за индексирање и пребарување на база на ДНК секвенци во C# се подига со избор на опцијата „*DNA Database indexing and search*“ од почетниот кориснички интерфејс, Сл. 4.1.

На влез, корисникот ја задава должината на индексирање k, ги наведува во низа идентификаторите на ДНК секвенците кои се преземаат од Европската Нуклеотидна Архива и ја задава структурата на ДНК прашалникот, Сл. 4.10. Влезните податоци се валидираат. Валидацијата се извршува со избор на опцијата „*Валидација на податоци на влез*“. Предмет на валидација се: должината на индексирање k, која треба да биде позитивна целобројна вредност, низата со идентификатори на ДНК секвенци од Европската Нуклеотидна Архива, кои меѓусебно треба да бидат одделени со знакот (,) и структурата на ДНК прашалникот, кој треба да биде зададен како збор над азбуката $\Sigma = \{A, C, T, G\}$. Опциите за преземање на ДНК секвенци, индексирање

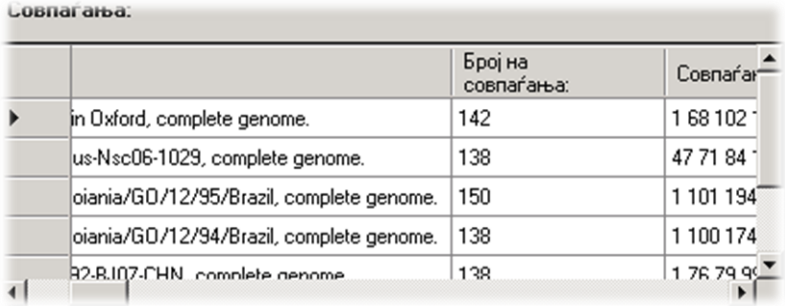
на преземените секвенци – конституенти на базата на податоци и пребарување на индексираната содржина по ДНК прашалник се овозможени само во случај на валиден внес на податоци на влез. Апликацијата не дозволува понатамошно извршување ако барем еден влезните податоци не е наведен или пак е наведен во погрешен податочен облик. Во таков случај на корисникот му се генерира порака дека еден или повеќе влезни параметри не се наведени или пак се наведени во погрешен податочен облик. Само во случај на валиден внес на податоци на влез може да се изврши преземање на ДНК секвенци од Европската Нуклеотидна Архива со избор на опцијата „*Преземи ДНК секвенци*“, Сл 4.10. Преземените секвенци во FASTA податочен облик (*име на секвенца+нуклеотидна содржина*) се зачувуваат во локални апликациски датотеки. За да се оддели нуклеотидната содржина од заглавјето, кое го содржи името на преземената секвенца, корисникот треба да ја избере опцијата „*Издвој нукл. податоци и приказ на детали за преземени секвенци*“ , Сл 4.10. Претходната опција ги печати имињата на преземените секвенци во одделна текстуална контрола.



Слика 4.10. Апликација за индексирање (пребарување) на база на ДНК секвенци

Figure 4.10. Application for DNA database indexing (searching)

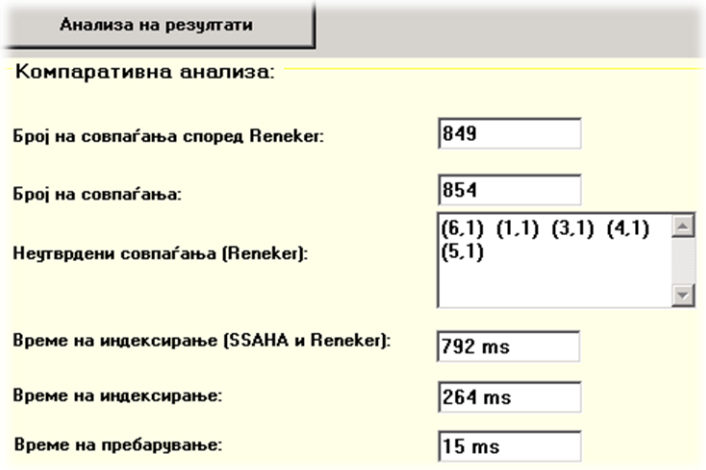
Индексирањето на преземената ДНК содржина, врз основа на формулата на Reneker и Shyu за единично базно пресликување и примена на предложената формула за брзо пресликување на преклопувачки зборови од ДНК секвенца, се извршува со избор на опцијата „*Индексирање*“, Сл 4.10. Совпаѓањата на ДНК прашалникот во индексираната база на податоци се наоѓаат преку опцијата „*Пребарување*“ и истите се печатат во DataGrid контрола по број на совпаѓања и почетни положби на совпаѓања во секвенца, Сл 4.11.



Слика 4.11. Број на совпаѓања и почетни положби на совпаѓања на прашалник во база на податоци

Figure 4.11. Number of query hits and its starting positions in the database

За споредба на перформансите на предложениот алгоритам со SSAHA и алгоритмот на Reneker и Shyu, во долниот десен панел се печати бројот на совпаѓања пронајдени со примена на предложениот алгоритам и бројот на совпаѓања кои се пронајдени со примена на алгоритмот на Reneker и Shyu, Сл 4.12. Предмет на споредбена анализа се и времињата на индексирање и пребарување кај SSAHA и алгоритмот на Reneker и Shyu и предложениот алгоритам, Сл 4.12.



Слика 4.12. Споредба на перформанси на извршување и точност на пребарување

Figure 4.12. Comparing running performances and hits’ detection accuracy

Се очекува со примена на предложениот алгоритам да се намали времето на индексирање и пребарување во споредба со SSAHA и алгоритмот на Reneker и Shyu, но и да се подобри точноста на пребарување во споредба со овие алгоритми, односно да се зголеми бројот на утврдени совпаѓања.

5. ЕКСПЕРИМЕНТАЛНИ РЕЗУЛТАТИ

Резултатите се добиени со извршување на софтверската библиотека на компјутер со двојно-јадрен *Intel E4400* процесор, кој работи на фреквенција од *2GHz* и RAM меморија со капацитет од *4GB*. Сите секвенци врз кои се извршени мерењата се преземени од Европската Нуклеотидна Архива (http://www.ebi.ac.uk/). За тест секвенци земени се: *астровирус геноми*, *хепатитис Б вирус секвенци*, *албумин ген секвенци* и *фрагменти од E. coli хромозоми*.

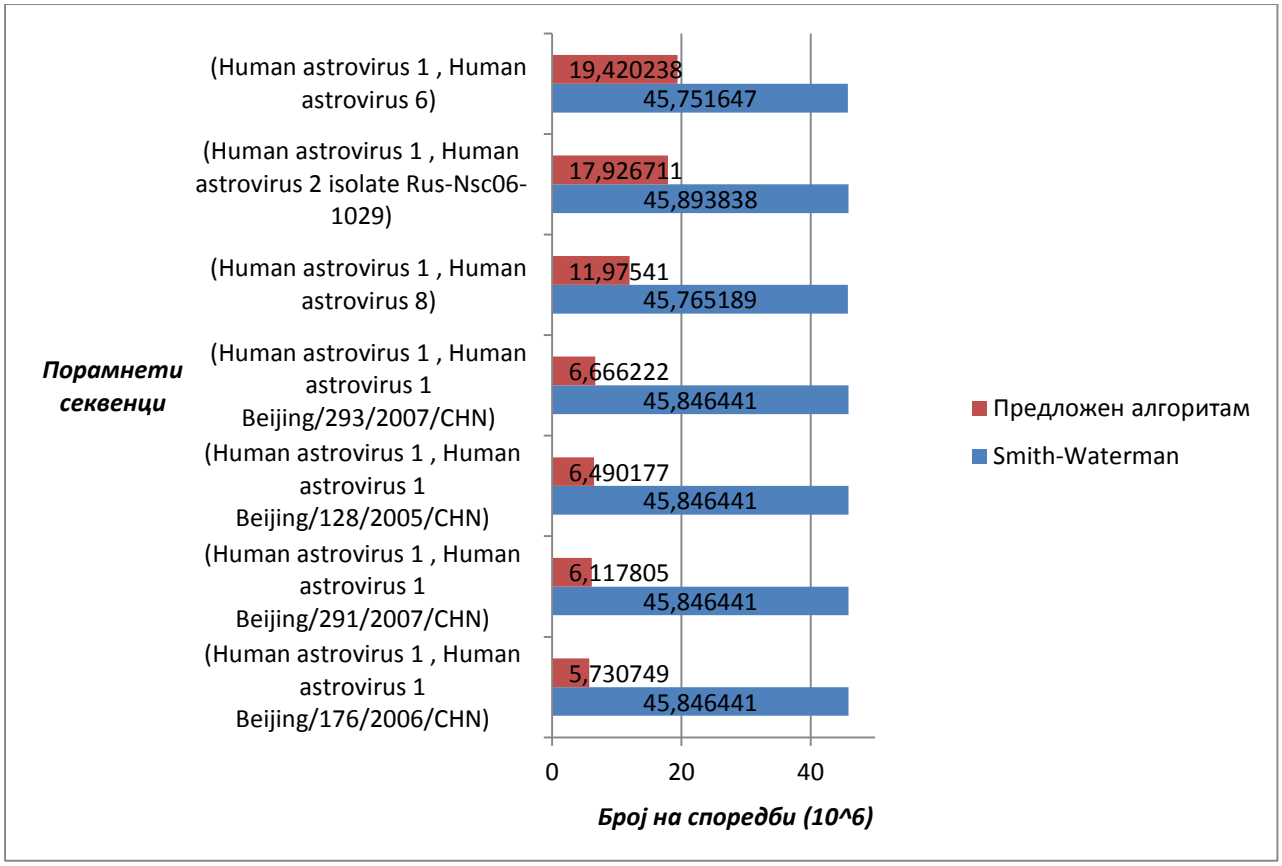
Имплементацијата на алгоритмот за брза идентификација на оптимална безпразнинска хомологија помеѓу две ДНК секвенци во C#/C++ е тестирана на парови геноми од тип: човечки астровирус и хепатитис Б вирус. За секое порамнување се пресметува временската и мемориската комплексност. Временската комплексност се мери во број на извршени базни споредби, додека мемориската комплексност се мери во број на резервирани целобројни мемориски единици во фаза на извршување. За исти тест секвенци се пресметува бројот на извршени базни споредби со примена на предложениот алгоритам и бројот на извршени базни споредби со примена на алгоритмот на Smith и Waterman. За истите секвенци се пресметува и мемориската комплексност во фаза на извршување на предложениот алгоритам и мемориската комплексност на мемориски-линеарните имплементации на алгоритмите базирани на динамичко програмирање, како: Hirschberg, Myers и Miller, Huang et al.

Се очекува со примена на предложениот алгоритам да се намали бројот на извршени споредби во споредба со алгоритмот на Smith и Waterman, но исто така и да се намали меморискиот трошок во фаза на извршување во споредба со мемориски-линеарните имплементации на алгоритмите базирани на динамичко програмирање: Hirschberg, Myers и Miller, Huang et al.

Во Табела 5.1 се дадени резултатите за временската комплексност (број на извршени базни споредби) на предложениот алгоритам и алгоритмот на Smith и Waterman при порамнување на парови на геноми од тип човечки астровирус. Врз основа на овие податоци може да се пресмета и забрзувањето од примената на предложениот алгоритам, кое се пресметува како количник помеѓу број на извршени споредби со примена на алгоритмот на Smith и Waterman и број на извршени споредби со примена на предложениот алгоритам. Во истата табела, Табела 5.1, се дадени и податоци за процентот на идентичност (PID) за секое порамнување, кој се пресметува како количник помеѓу бројот на совпаѓања вклучени во порамнувањето и средната должина на порамнетите секвенци. Се очекува за повисок процент на идентичност забрзувањето да биде поголемо и обратно. Во Табела 5.2 се дадени резултатите за мемориската побарувачка на предложениот алгоритам во фаза на извршување и мемориската побарувачка на мемориски-линеарните имплементации на динамичко програмирање.

Табела 5.1. Резултати за временска комплексност на порамнувања на астровирусни геноми
Table 5.1. Time complexity results for alignments of astrovirus genomes

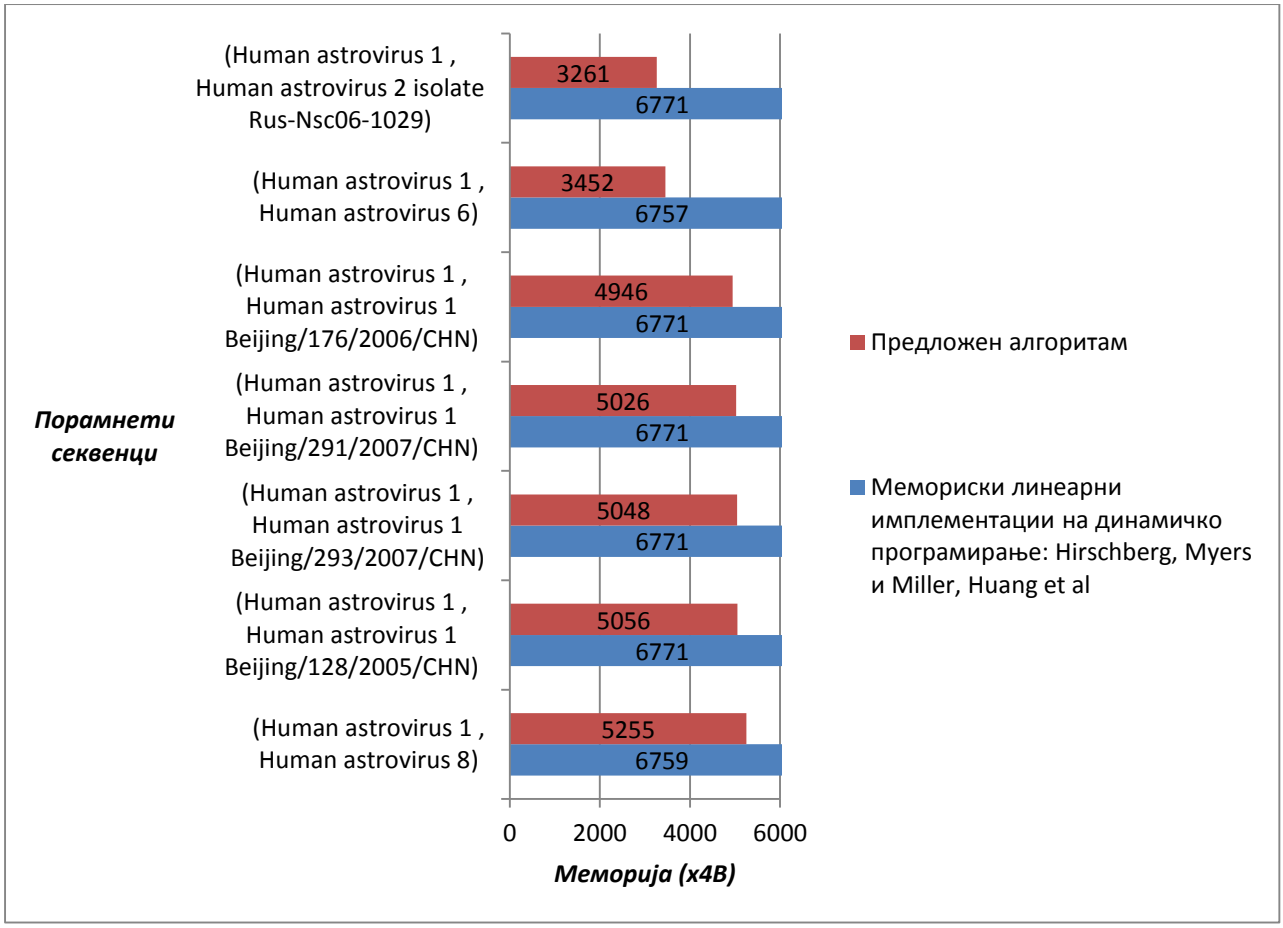
Порамнети секвенци	**Број на споредби: Smith-Waterman**	**Број на споредби: Предлог алгоритам**	**(PID) %**	**Забрзување**
(Human astrovirus 1 , Human astrovirus 1 Beijing/176/2006/CHN)	45846441	5730749	92,35	8,000078349
(Human astrovirus 1 , Human astrovirus 1 Beijing/291/2007/CHN)	45846441	6117805	91,77	7,493936306
(Human astrovirus 1 , Human astrovirus 1 Beijing/128/2005/CHN)	45846441	6490177	90,64	7,06397391
(Human astrovirus 1 , Human astrovirus 1 Beijing/293/2007/CHN)	45846441	6666222	90,53	6,877424874
(Human astrovirus 1 , Human astrovirus 8)	45765189	11975410	77,71	3,821596839
(Human astrovirus 1 , Human astrovirus 2 isolate Rus-Nsc06-1029)	45893838	17926711	64,11	2,560081322
(Human astrovirus 1 , Human astrovirus 6)	45751647	19420238	26,85	2,355874681



Слика 5.1. Графички приказ на добиените резултати за временска комплексност од Табела 5.1
Figure 5.1. Bar graph for time complexity results from Table 5.1

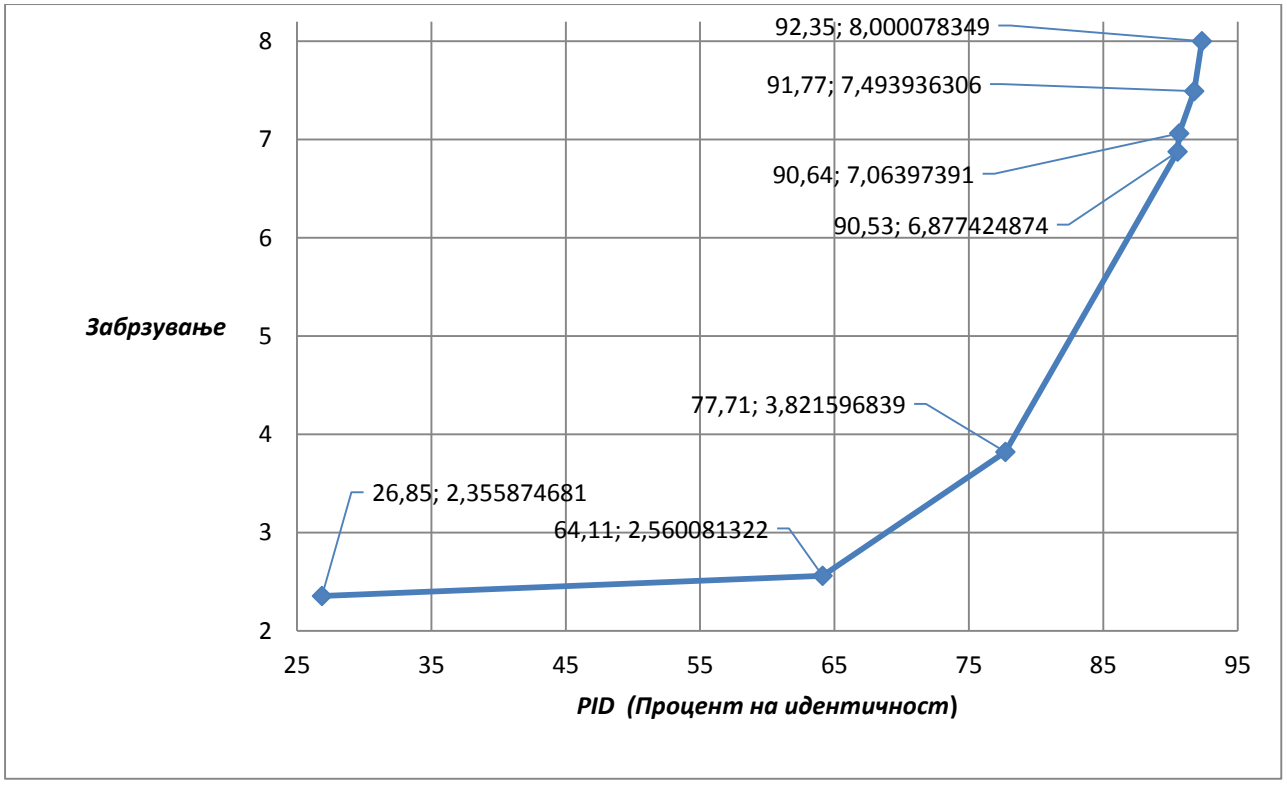
Табела 5.2. Резултати за мемориска комплексност на порамнувања на астровирусни геноми
Table 5.2. Memory complexity results for alignments of astrovirus genomes

Порамнети секвенци	**Меморија (x4B): Мемориски линеарни имплементации на динамичко програмирање: *Hirschberg, Myers u Miller, Huang et al***	**Меморија (x4B): Предложен алгоритам**	**Намалување на меморијата (%)**
(Human astrovirus 1 , Human astrovirus 8)	6759	5255	22,25181
(Human astrovirus 1 , Human astrovirus 1 Beijing/128/2005/CHN)	6771	5056	25,32861
(Human astrovirus 1 , Human astrovirus 1 Beijing/293/2007/CHN)	6771	5048	25,44676
(Human astrovirus 1 , Human astrovirus 1 Beijing/291/2007/CHN)	6771	5026	25,77167
(Human astrovirus 1 , Human astrovirus 1 Beijing/176/2006/CHN)	6771	4946	26,95318
(Human astrovirus 1 , Human astrovirus 6)	6757	3452	48,91224
(Human astrovirus 1 , Human astrovirus 2 isolate Rus-Nsc06-1029)	6771	3261	51,83872



Слика 5.2. Графички приказ на добиените резултати за мемориска комплексност од Табела 5.2
Figure 5.2. Bar graph for memory complexity results from Table 5.2

Во сите случаи, Табела 5.1, Сл. 5.1, предложениот алгоритам извршува помалку споредби од Smith-Waterman. Намалувањето на бројот на извршени споредби може да се квантифицира преку забрзувањето на предложениот алгоритам во однос на алгоритмот на Smith и Waterman, кое е поголемо за секвенци во повисок процент на идентичност и обратно. Најголемо намалување на бројот на извршени споредби може да се забележи кај порамнувањето на секвенците: {Human astrovirus 1, Human astrovirus 1 Beijing/176/2006/CHN}, каде за да се добие решение предложениот алгоритам извршува осумпати помалку споредби во однос на алгоритмот на Smith и Waterman, Табела 5.1, Сл. 5.1. Најмало намалување на бројот на извршени споредби може да се забележи кај порамнувањето на секвенците: {Human astrovirus 1, Human astrovirus 6}, каде предложениот алгоритам извршува речиси двојно (2,35) помалку споредби во однос на алгоритмот на Smith и Waterman, Табела 5.1, Сл. 5.1. Во првиот случај забрзувањето е најголемо бидејќи се порамнуваат секвенци со највисок процент на идентичност (PID=92,35%) од множеството на порамнети геноми. Во вториот случај, забрзувањето е најмало бидејќи се порамнуваат секвенци со најмал процент на идентичност (PID=26,85%). Трендот на пораст на забрзувањето на извршување со зголемување на процентот на идентичност го следат и останатите порамнувања во Табела 5.1, Сл. 5.3.



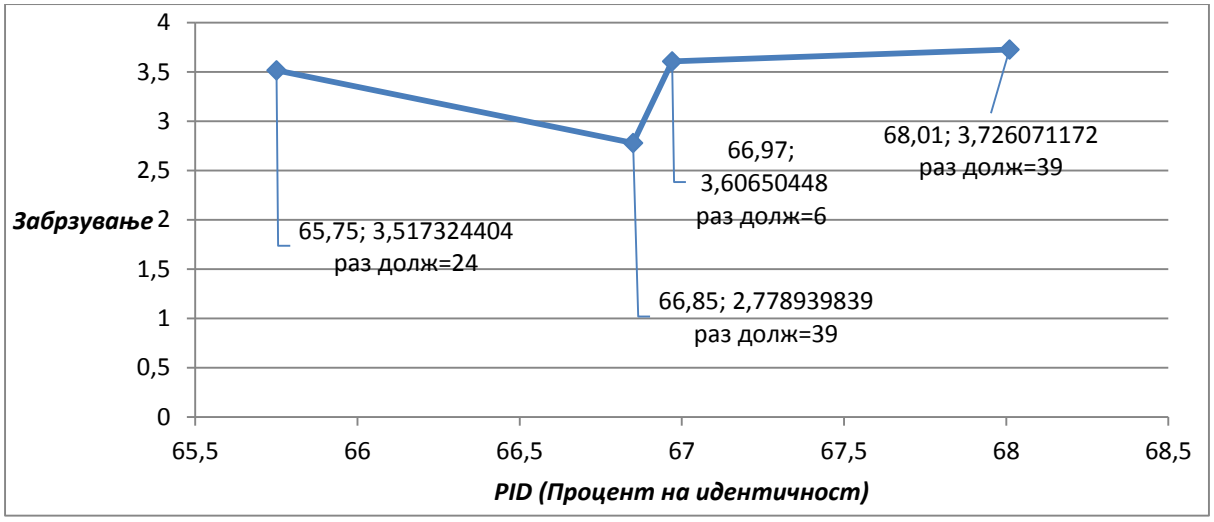
Слика 5.3. Зависност помеѓу процентот на идентичност (PID) и забрзувањето
Figure 5.3. Relation between the percent of identity (PID) and the speed up

Параметар кој може да го наруши трендот на пораст на забрзувањето во однос на процентот на идентичност е разликата помеѓу должините на секвенците кои се порамнуваат. Имено, при порамнување на хепатитис Б вирусни секвенци, Табела 5.3, Сл. 5.4, утврден е порамнет пар секвенци: {Hepatitis B virus (HBV 991) complete genome, Hepatitis B virus genotype C} за кој забрзувањето на извршување на предложениот алгоритам во однос на алгоритмот на Smith и Waterman е помало од забрзувањето за порамнетиот пар секвенци: {Hepatitis B virus (HBV 991) complete genome, Hepatitis B virus isolate CH109}, иако процентот на идентичност за првиот порамнет пар (PID=66,85%) е поголем од процентот на идентичност за вториот порамнет пар (PID=65,75%), Табела 5.3, Сл. 5.4. Ваквата отстапка се должи на зголемувањето на разликата помеѓу должините на секвенците кои се порамнуваат. Разликата помеѓу процентите на идентичност во двата случаи на порамнување е незначителна (1,1 %), но не и разликата помеѓу должините на секвенците кои се порамнуваат. Во првиот случај, кога се порамнуваат секвенците: {Hepatitis B virus (HBV 991) complete genome, Hepatitis B virus genotype C}, разликата помеѓу должините на секвенците изнесува 39, додека во вториот случај кога се порамнуваат секвенците: {Hepatitis B virus (HBV 991) complete genome, Hepatitis B virus isolate CH109}, разликата помеѓу должините на секвенците изнесува 24, Табела 5.3, Сл. 5.4. Во согласност со концепциската поставеност на предложениот алгоритам, колку е поголема разликата помеѓу должните на секвенците, толку повеќе поместувања на пократката секвенца долж подолгата се извршуваат, со што во основа се зголемува бројот на извршени споредби, односно се намалува факторот на забрзување на предложениот алгоритам во однос на алгоритмот на Smith и Waterman. И покрај тоа што процентот на идентичност за порамнетиот

пар секвенци: {Hepatitis B virus (HBV 991) complete genome , Hepatitis B virus genotype C} е поголем од процентот на идентичност за порамнетиот пар секвенци: {Hepatitis B virus (HBV 991) complete genome, Hepatitis B virus isolate CH109}, забрзувањето во првиот случај е помало бидејќи алгоритмот во првиот случај извршува $m \times (n - m + 1) = 3182 \times (39 + 1) = 3182 \times 40 = 127280$ споредби долж опсегот на подолгата секвенца, додека во вториот случај бројот на извршени споредби долж опсегот на подолгата секвенца изнесува $m \times (n - m + 1)$=3197$\times (24 + 1) = 3197 \times 25 = 79925$.

Табела 5.3. Резултати за временска комплексност на порамнувања на хепатитис Б вирусни секвенци
Table 5.3. Time complexity results for alignments of hepatitis B virus sequences

Порамнети секвенци	**Број на споредби: Smith-Waterman**	**Број на споредби: Предложен алгоритам**	**(PID) %**	**Забрзување**	**Разлика помеѓу должини на секвенци**
(Hepatitis B virus (HBV 991) complete genome , Hepatitis B virus, genotype A, isolate X104)	10374841	350453	68,01	3,726071	39
(Hepatitis B virus (HBV 991) complete genome , Hepatitis B virus, genotype D)	10249222	2750678	66,97	3,606504	6
(Hepatitis B virus (HBV 991) complete genome , Hepatitis B virus genotype C)	10355515	2871344	66,85	2,77894	39
(Hepatitis B virus (HBV 991) complete genome , Hepatitis B virus isolate CH109)	10249222	3688177	65,75	3,517324	24



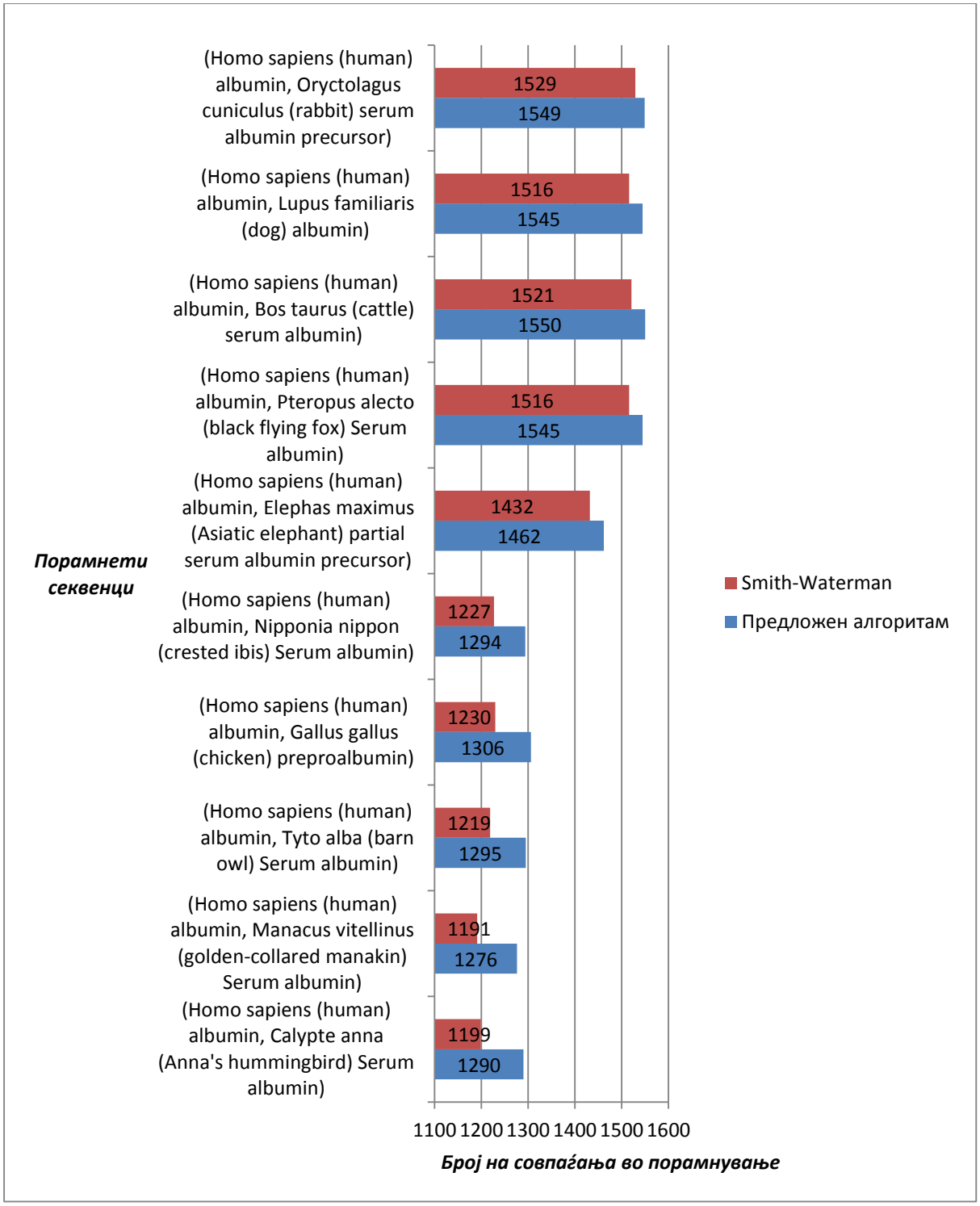
Слика 5.4. Зависност помеѓу процентот на идентичност (PID) и забрзувањето, каде е вклучена и разликата помеѓу должините на порамнетите секвенци
Figure 5.4. Relation between the percent of identity (PID) and the speed up, where the difference between the aligned sequences is also included

Ако се споредат податоците за мемориската зафатнина во фаза на извршување на предложениот алгоритам со мемориски-линеарните имплементации на динамичко програмирање (Hirschberg, Myers и Miller, Huang et al), може да се забележи дека предложениот алгоритам во сите тест случаи на порамнувања на парови астровирусни геноми, Табела 5.2, Сл. 5.2, зафаќа помалку меморија од мемориски-линеарните имплементации на динамичко програмирање. Процентот на намалување на мемориското трошок варира од 22% до 51%. Најмал процент на намалување на мемориската побарувачка во фаза на извршување на предложениот алгоритам во однос на мемориски-линеарните имплементации на динамичко програмирање може да се забележи кај порамнувањето на секвенците: {Human astrovirus 1, Human astrovirus 8}, додека најголем процент на мемориска заштеда може да се забележи кај порамнувањето на секвенците {Human astrovirus 1, Human astrovirus 2 isolate Rus-Nsc06-1029}.

Имплементацијата на предложениот алгоритам за празнинско порамнување на ДНК секвенци во C# е тестирана на парови Албумин протеин – кодирачки ДНК секвенци од различни видови. Предмет на компаративна анализа е бројот на базни совпаѓања вклучени во порамнувањата добиени со примена на предложениот алгоритам и бројот на базни совпаѓања вклучени во порамнувањата добиени со примена на алгоритмот на Smith и Waterman. Се очекува со примена на предложениот алгоритам да се зголеми бројот на совпаѓања по порамнување, односно да се отстрани можноста за отфрлање на конзистентни совпаѓања од порамнување поради намалување на глобален резултат на порамнување кај алгоритмот на Smith и Waterman за избор на неповолна метрика на порамнување. Во Табела 5.4 се дадени податоци за број на совпаѓања по порамнување на пар Албумин – кодирачки секвенци со примена на предложениот алгоритам и алгоритмот на Smith и Waterman (казната за отворање на празнина = -5, казна за издолжување на отворена празнина = -1).

Табела 5.4. Број на совпаѓања по порамнување со примена на: предложен алгоритам и Smith-Waterman
Table 5.4. Number of hits in alignment by applying: the proposed algorithm and Smith-Waterman

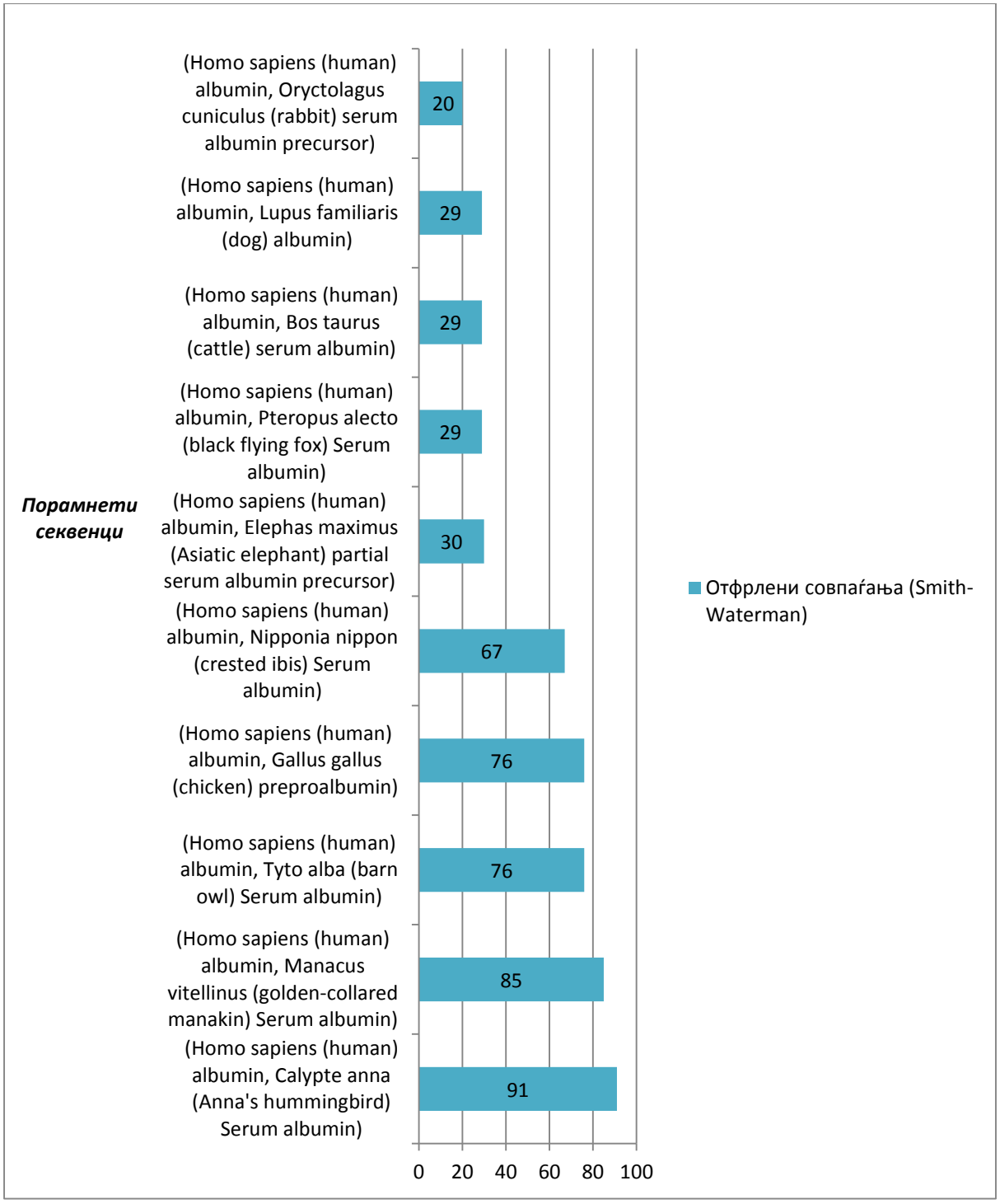
Порамнети секвенци:	**Предложен алгоритам**	**Smith-Waterman**
(Homo sapiens (human) albumin, Calypte anna (Anna's hummingbird) Serum albumin)	1290	1199
(Homo sapiens (human) albumin, Manacus vitellinus (golden-collared manakin) Serum albumin)	1276	1191
(Homo sapiens (human) albumin, Tyto alba (barn owl) Serum albumin)	1295	1219
(Homo sapiens (human) albumin, Gallus gallus (chicken) preproalbumin)	1306	1230
(Homo sapiens (human) albumin, Nipponia nippon (crested ibis) Serum albumin)	1294	1227
(Homo sapiens (human) albumin, Elephas maximus (Asiatic elephant) partial serum albumin precursor)	1462	1432
(Homo sapiens (human) albumin, Pteropus alecto (black flying fox) Serum albumin)	1545	1516
(Homo sapiens (human) albumin, Bos taurus (cattle) serum albumin)	1550	1521
(Homo sapiens (human) albumin, Lupus familiaris (dog) albumin)	1545	1516
(Homo sapiens (human) albumin, Oryctolagus cuniculus (rabbit) serum albumin precursor)	1549	1529



Слика 5.5. Графичка репрезентација на податоците за број на совпаѓања во Табела 5.4

Figure 5.5. Bar graph for the number of hits in Table 5.4

Во сите тест случаи, Табела 5.4, Сл. 5.5, предложениот алгоритам генерира порамнувања со поголем број на совпаѓања во однос на алгоритмот на Smith и Waterman. Со примена на предложениот алгоритам, порамнувањата кои се добиени со примена на Smith-Waterman алгоритмот се издолжуваат за 20 до 91 базно совпаѓање, Сл. 5.6. Совпаѓањата кои ги отфрла алгоритмот на Smith и Waterman, Сл. 5.6, се совпаѓања со чие вклучување во решението би се намалил резултатот на порамнување. Отфрлувањето на совпаѓања во вакви случаи може да доведе до генерирање на порамнувања во кои остатоците на идентичност од подалечен предок нема да бидат вклучени, со што се губи можноста за детекција на подалечна хомологија. Овој недостаток го решава предложениот алгоритам, со тоа што генерира порамнување во кое е вклучено секое базно совпаѓање кое би можело да се вклучи, односно се нуди можност за детекција како на блиска, така и на подалечна хомологија. Дополнително, вториот предложен модел ги зема предвид и предиспозициите на зачуваност на базните парови, врз основа на што се предвидуваат положбите на кои се има случено настан од тип *бришење на нуклеотид*, со што се добива модел на порамнување, кој доследно ја прикажува структурната и еволуциската зависност помеѓу порамнетите секвенци.



Слика 5.6. Отфрлени совпаѓања со примена на Smith-Waterman
Figure 5.6. Rejected hits by applying Smith-Waterman

Имплементацијата на предложениот алгоритам за индексирање и пребарување на ДНК база на секвенци во C# е тестирана на база на фрагменти од *E. Coli* секвенци со големина од $0{,}1\ \mathrm{Gb}$ ($1\ \mathrm{Gb} = 1\,000\,000\,000$ нуклеотиди). Предмет на анализа се: времето на индексирање, мемориската зафатнина на индексираната податочна структура, времето на пребарување и точноста на пребарување.

Придобивките од примената на предложената формула за брзо пресликување на зборови од ДНК база на секвенци ги потврдуваат времињата за индексирање на *E. coli 55989* хромозом, со и без примена на примена на предложената формула. За да се пресликаат сите преклопувачки зборови во должина од $k = 8$ нуклеотиди од *E. coli 55989* хромозом, кој вклучува приближно $5\ \mathrm{Mb}$ ($1\ \mathrm{Mb} = 1\ 000\ 000$ нуклеотиди), потребни се осум минути без примена на предложената формула. Со примена на предложената формула на истиот хромозом (*E. coli 55989*) и за истата должина на индексирање ($k = 8$), времето на индексирање се намалува од осум минути на една минута.

Имајќи предвид дека временската комплексност на предложената формула за брзо индексирање на ДНК зборови е линеарна, времето за индексирање на тест базата на *E. Coli* секвенци со големина од $0{,}1\ \mathrm{Gb}$ со примена на предложената формула изнесува 20 минути. Времето за индексирање на истата база на секвенци со директна примена на формулата на Reneker и Shyu, без да се вклучи предложената надградба, изнесува околу 160 минути, врз основа на што може да се заклучи дека со примена на предложената формула процесот на индексирање се забрзува за фактор k, каде k е должина на индексирање.

Со употреба на истите ресурси и за иста должина на индексирање ($k = 8$), времето за индексирање на човечки геном, кој содржи околу $3\ \mathrm{Gb}$, би изнесувало 600 минути (10 часа) со примена на предложената формула, додека за истата цел со примена на формулата на Reneker и Shyu би биле потребни 3,3 денови.

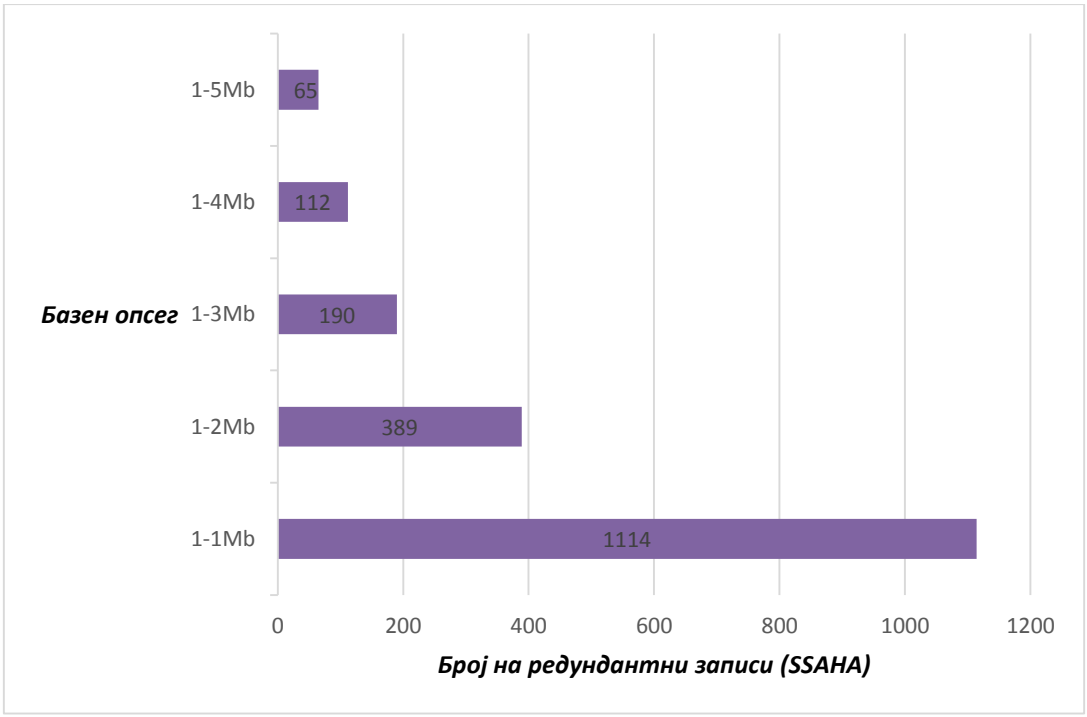
Со примена на пристапот за додавање на записи во индексираната податочна структура само за зборови кои постојат во базата на податоци, се намалува мемориската зафатнина на индексираната структура во споредба со SSAHA. Големината на хеш-табелата кај SSAHA зависи исклучиво од должината на индексирање, k. Така, на пример, за должина на индексирање $k = 8$, SSAHA конструира хеш-табела со $4^8 = 65536$ клучеви (записи). Постои можност дел од овие клучеви да соодветствуваат на пресликувања за зборови кои не постојат во базата на податоци, што во основа претставува непотребно мемориско оптоварување, кое може да го забави процесот на пребарување. Ако записи се додаваат само за зборовите кои ќе бидат прочитани од базата на секвенци, во тој случај нема да постојат записи во индексираната структура кои би соодветствувале на непостоечки зборови. Во такви услови, бројот на записи во индексираната податочна структура ќе зависи од обемот на индексираната содржина, при што се очекува тој број да расте со зголемување на обемот на индексираната ДНК содржина.

Во Табела 5.5 се дадени податоци за бројот на записи во индексираната податочна структура при индексирање на: $1\ \mathrm{Mb}$, $2\ \mathrm{Mb}$, $3\ \mathrm{Mb}$, $4\ \mathrm{Mb}$ и $5\ \mathrm{Mb}$ од *E. Coli 55989* хромозом. Како што може да се забележи во Табела 5.5, бројот на записи во индексираната структура со примена на предложениот пристап расте со зголемувањето на обемот на ДНК податоците, за разлика од SSAHA каде независно од обемот на ДНК податоците кои се индексираат, мемориската зафатнина на индексираната структура е константна. Мемориската заштеда со примена на предложениот пристап се движи помеѓу: 1114 и 65, Сл. 5.7. Најголема заштеда се добива при индексирање на ДНК содржина со најмал обем

(1 Mb), додека најмала е заштедата при индексирање на ДНК содржина со најголем обем (5 Mb).

Табела 5.5. Број на записи во индексираната податочна структура
Table 5.5. Number of records in the indexed data structure

Базен опсег:	**Број на записи (SSAHA)**	**Број на записи (предложен алгоритам)**	**Број на редундантни записи (SSAHA)**
1-1Mb	65536	64422	1114
1-2Mb	65536	65147	389
1-3Mb	65536	65346	190
1-4Mb	65536	65424	112
1-5Mb	65536	65471	65



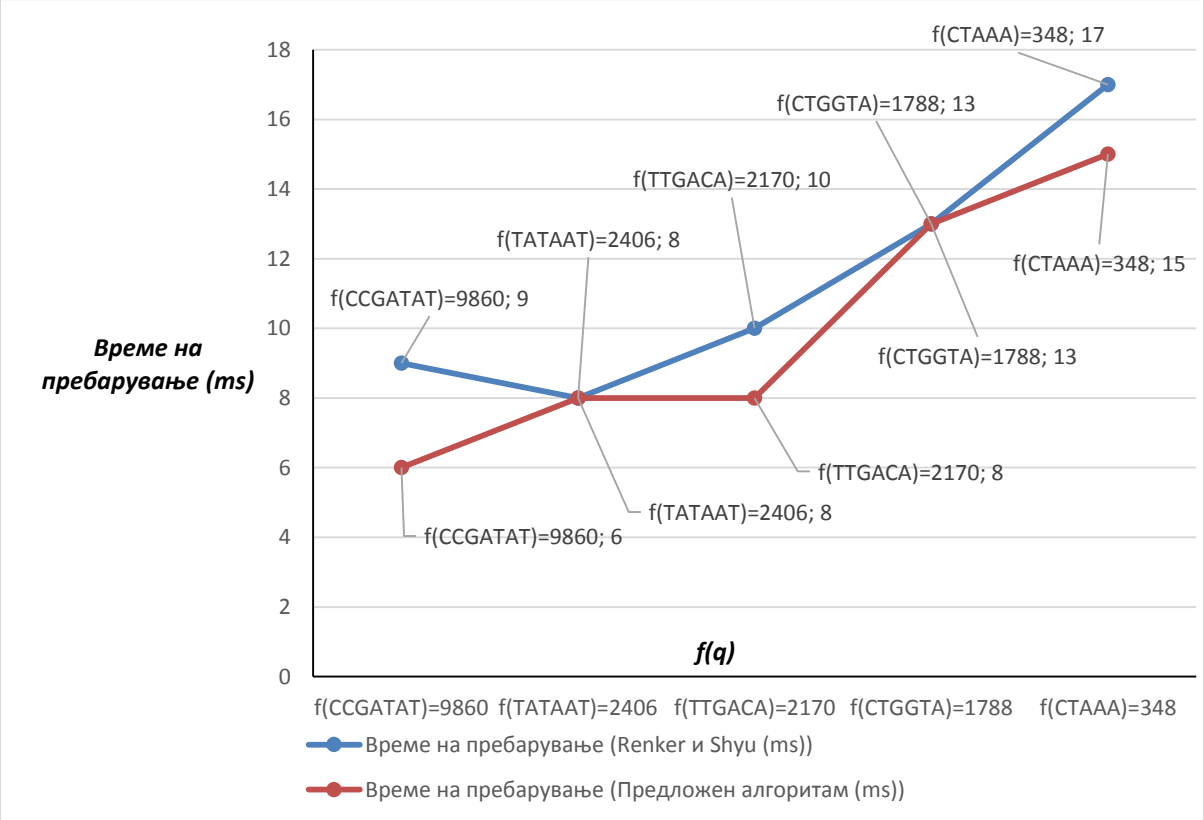
Слика 5.7. Број на редундантни записи (SSAHA)
Figure 5.7. Number of redundant records (SSAHA)

За да се намали времето на пребарување во споредба со алгоритмот на Reneker и Shyu, предложениот алгоритам употребува сортиран речник, наместо хеш-табела или индексирана датотека. Со употреба на сортиран речник се овозможува идентификација на сите совпаѓања на ДНК прашалник во база на ДНК секвенци без да се обработат сите записи, на што во принцип се должи подобрувањето. Во Табела 5.6 се дадени податоци за измерените времиња на пребарување на тест индексираната база на секвенци по различни промотор консензус секвенци, кои ги препознаваат: σ^{28}, σ^{54} и σ^{70} транскрипциските фактори. Како што може да се забележи од Табела 5.6, Сл. 5.8, во три од пет случаи на пребарување на базата на податоци со капацитет од 0,1 Gb за исти промотор консензус секвенци, времето на пребарување со примена на

предложениот алгоритам е пократко од времето на пребарување со примена на алгоритмот Reneker и Shyu. Исто така може да се забележи дека колку е поголемо пресликувањето на прашалникот, дотолку пократко е времето на пребарување и обратно, Табела 5.6, Сл. 5.8.

Табела 5.6. Времиња на пребарување по различни консензус секвенци
Table 5.6. Run time samples for searching different promoter consensus sequences

Сигма фактор	**q: консензус прашалник**	**f(q): пресликување на прашалник**	**Време на пребарување (Reneker и Shyu (ms))**	**Време на пребарување (Предложен алгоритам (ms))**
σ^{28}	CCGATAT	f(CCGATAT)=9860	9	6
σ^{70}	TATAAT	f(TATAAT)=2406	8	8
σ^{70}	TTGACA	f(TTGACA)=2170	10	8
σ^{54}	CTGGTA	f(CTGGTA)=1788	13	13
σ^{28}	CTAAA	f(CTAAA)=348	17	15



Слика 5.8. Споредба на временските интервали на пребарување
Figure 5.8. Comparison of run time samples

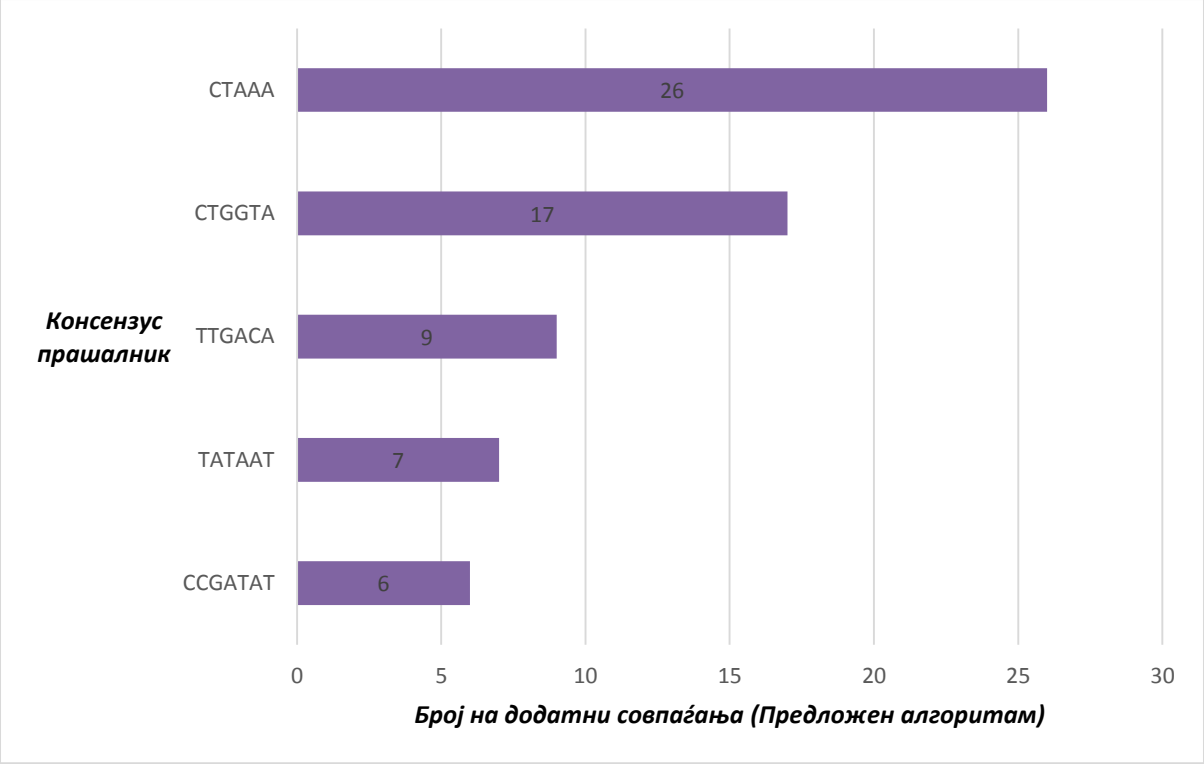
Настрана од предложените пресметковни подобрувања, главното подобрување на предложениот алгоритам од биолошки аспект е можноста за детекција на сите совпаѓања, без оглед на нивната почетна положба во секвенците од базата на податоци.

Имено, алгоритмот на Reneker и Shyu не може да пронајде совпаѓања на ДНК прашалник на почетни положби: $p \leq k - |q|$ во рамки на индексирани секвенци, каде: k е должина на индексирање и $|q|$ е должина на ДНК прашалник. Последиците од тоа можат да бидат најразлични, како на пример: неможност за детекција на почеток на ген, парцијална детекција на кратки тандемски повторувања и неможност за точна детекција на структура на теломер на почеток на хромозом. Предложениот алгоритам за разлика од алгоритмот на Reneker и Shyu може да ги детектира сите совпаѓања на ДНК прашалник во база на податоци, независно од нивната почетна положба во индексираните секвенци, со што се зголемува точноста на пребарување, односно се елиминира можноста за постоење на неутврдени или парцијално утврдени ДНК шаблони. Клучот за решавање на недостатокот е во интеграцијата на суфикс и префикс базираниот пристап на пребарување.

Во Табела 5.7 се дадени податоци за бројот на совпаѓања кои се утврдени со примена на алгоритмот на Reneker и Shyu и бројот на совпаѓања кои се утврдени со примена на предложениот алгоритам, при пребарување на тест индексираната база на податоци по консензус елементите: CCGATAT, TATAAT, TTGACA, CTGGTA, CTAAA. Како што може да се забележи од Табела 5.7, Сл. 5.9, бројот на совпаѓања кои дополнително се утврдени со примена на предложениот алгоритам варира помеѓу: 6 и 26. Вкупниот број на совпаѓања кои не може да ги детектира алгоритмот на Reneker и Shyu, а ги детектира предложениот алгоритам изнесува 65, Табела 5.7, Сл. 5.9.

Табела 5.7. Број на пронајдени совпаѓања
Table 5.7. Number of detected hits

Сигма фактор	**q: консензус прашалник**	**f(q): пресликување на прашалник**	**Број на совпаѓања (Reneker)**	**Број на совпаѓања (Предложен алгоритам)**	**Број на додатни совпаѓања (Предложен алгоритам)**
σ^{28}	CCGATAT	f(CCGATAT)=9860	2520	2526	6
σ^{70}	TATAAT	f(TATAAT)=2406	3812	3819	7
σ^{70}	TTGACA	f(TTGACA)=2170	4010	4019	9
σ^{54}	CTGGTA	f(CTGGTA)=1788	12001	12018	17
σ^{28}	CTAAA	f(CTAAA)=348	18140	18166	26



Слика 5.9. Број на додатни совпаѓања кои се пронајдени со примена на предложениот алгоритам
Figure 5.9. Number of hits being additionally detected by applying the proposed algorithm

6. ДИСКУСИЈА

Заклучоците за карактеристиките и предностите на предложените алгоритми во однос на постојните и најчесто употребувани алгоритми за порамнување и пребарување на ДНК секвенци се наведени во Табела 6.1. Како потврда за веродостојноста на предложените заклучоци можат да се земат резултатите од извршените мерења во претходното поглавје.

Табела 6.1. Карактеристики и предности на предложените алгоритми
Table 6.1. Characteristics and advantages of the proposed algorithms

Алгоритам	**Параметар**	**Дискусија**
Smith-Waterman	Временска комплексност: $O(nm)$	Параметарот k варира во зависност од процентот на идентичност и разликата помеѓу должините на секвенците кои се порамнуваат.
Предлог алгоритам за безпразнинско порамнување	Временска комплексност: $O(km), k < n$	
Мемориски линеарни имплементации на динамичко програмирање: Hirschberg, Myers и Miller, Huang et al	Мемориска комплексност: $O(m)$	k е должина на вектор на порамнување
Предлог алгоритам за безпразнинско порамнување	Мемориска комплексност: $O(k), k < m$	
Smith-Waterman	Број на совпаѓања по порамнување: h_{SW}	$h_p > h_{SW}$, за избор на висока казна кај Smith-Waterman
Предложен алгоритам за празнинско порамнување	Број на совпаѓања по порамнување: h_p	
SSAHA и алгоритам на Reneker и Shyu	Време на индексирање: $O(\|D\| \times k)$	Процесот на индексирање на ДНК база на податоци D се забрзува за фактор k.
Предложен алгоритам за индексирање и пребарување на ДНК база на податоци	Време на индексирање: $O(\|D\|)$	
SSAHA	Број на записи во индексирана податочна структура: $\|HT\|$	Се намалува мемориската зафатнина на индексираната податочна структура.
Предложен алгоритам за индексирање и пребарување на ДНК база на секвенци	Број на записи во индексирана податочна структура: $\|SD\|, \|SD\| < \|HT\|$	

Reneker и Shyu	Време на пребарување: s_{RS}	Се намалува времето на пребарување, поради тоа што не се обработуваат сите записи од индексираната податочна структура.
Предложен алгоритам за индексирање и пребарување на ДНК база на секвенци	Време на пребарување: $s_p, s_p \leq s_{RS}$	
Reneker и Shyu	Број на совпаѓања: h_{RS}	Се зголемува точноста на пребарување, односно може да се утврди секое совпаѓање на ДНК прашалник q во ДНК база на податоци D.
Предложен алгоритам за индексирање и пребарување на ДНК база на секвенци	Број на совпаѓања: $h_p, h_p \geq h_{RS}$	

7. ЗАКЛУЧОК

Во оваа истражување предложени се три нови алгоритми со примена за порамнување и пребарување на ДНК секвенци. Како прв предложен алгоритам е алгоритмот за брза идентификација на оптимална безпразнинска хомологија помеѓу две секвенци, со чија примена се намалува временскиот и меморискиот трошок во споредба со алгоритмот на Smith и Waterman. Подобрувањето на пресметковните аспекти во фаза на извршување го прави предложениот алгоритам применлив како за кратки секвенци, така и за подолги секвенци како, на пример: масивни прокариотски геноми или еукариотски хромозоми, каде поради квадратната временска и мемориска комплексност, примената на алгоритмите базирани на динамичко програмирање не секогаш е изводлива. Парцијалноста на решението кое се добива со примена на алгоритмот на Smith и Waterman за избор на конкретна метрика на порамнување е мотивација за да се предложи нов алгоритам за празнинско порамнување, со чија примена нема да се отфрли ниту едно конзистентно совпаѓање поради намалување на вредност на резултат на порамнување (Smith и Waterman). Освен блиска, со примена на предложениот алгоритам, може да се детектира и далечна хомологија помеѓу две ДНК секвенци, а со додавањето на празнини помеѓу базни парови со минимални преференции на задржаност, на добиеното решение му се задава додатна биолошка тежина.

Фактот дека за секои два последователни и преклопувачки зборови постои потсеквенца на заеднички елементи е земен како идеја за да се предложи нова формула за индексирање на зборови од ДНК база на податоци, со чија примена процесот на индексирање на содржината од базата на податоци повеќекратно се забрзува во однос на SSAHA и алгоритмот на Reneker и Shyu. Со додавање на записи во индексираната структура само за зборови кои фигурираат во базата на податоци, се намалува мемориската зафатнина на индексираната структура во споредба со SSAHA, додека со употребата на сортиран речник, наместо хеш-табела (SSAHA) или индексирана датотека (Reneker и Shyu), се намалува времето на пребарување на ДНК базата на податоци по ДНК прашалник во споредба со Reneker и Shyu, поради тоа што не мора да се обработат сите записи во индексираната структура за да се пронајдат сите совпаѓања на прашалникот во базата на податоци. Со интеграција на суфикс и префикс базиран концепт на пребарување, предложениот алгоритам може да ги детектира сите совпаѓања на ДНК прашалник во база на податоци, без оглед на положбата на која се наоѓаат, што не претставува случај кај Reneker и Shyu, каде не можат да бидат детектирани совпаѓањата кои се наоѓаат на почетоците од индексираните ДНК секвенци. Сите подобрувања се експериментално докажани на податоци за реални ДНК секвенци преземени од Европската Нуклеотидна Архива.

8. ДОДАТОК – ЛИСТА НА АЛГОРИТМИ

Алгоритам за брзо и мемориски ефикасно порамнување на ДНК секвенци
Влез:
ДНК секвенца 1: $a = a_1a_2 \dots a_{n-1}a_n$, ДНК секвенца 2: $b = b_1b_2 \dots b_{m-1}b_m$ $(n \geq m)$, награда за порамнување на идентични елементи μ , казна за порамнување на различни елементи δ
Излез:
Оптимално безпразнинско порамнување A_0

Променливи:
A_o: оптимално порамнување
R_ξ: $(p_{a,\xi}, p_{b,\xi}, l_\xi)$: фрагмент на совпаѓање R_ξ со должина l_ξ и почетни положби во $a(b)$ $p_{a,\xi}(p_{b,\xi})$
$A_{i,o}$: оптимално порамнување за поместување i: долж опсегот на b, на десно (надвор од опсегот на b) или на лево (надвор од опсегот на b)
A_e: издолжувачко порамнување
$f(A)$: резултат на порамнување A
s: променлива за привремено чување на резултат на порамнување
Ознаки и оператори:
$|\ |$: должина на фрагмент на совпаѓање R_ξ
$\emptyset$: празно порамнување
$\bowtie$: соединување на фрагмент на совпаѓање R_ξ
$\leftarrow$: додела на вредност или структура
$=$: проверка на еднаквост
$++$: оператор за зголемување на вредност на променлива за 1
$--$: оператор за намалување на вредност на променлива за 1

$A_o \leftarrow \emptyset$
$f(A_o) \leftarrow 0$

за$(i \leftarrow 1; i \leq n - m + 1; i++)$ {
најди χ совпаѓања R_ξ: $(p_{a,\xi}, p_{b,\xi}, l_\xi)$, $|R_\xi| \geq 1$
во $(a_ia_{i+1} \dots a_{i+m-2}a_{i+m-1}; b_1b_2 \dots b_{m-1}b_m)$
Скок на S
}

$i \leftarrow 1$
додека$\big((m - i) > f(A_o)/\mu\big)$
најди χ совпаѓања R_ξ: $(p_{a,\xi}, p_{b,\xi}, l_\xi)$, $|R_\xi| \geq 1$
во $(a_{n-m+1+i} \dots a_{n-1}a_n; b_1 \dots b_{m-i-1}b_{m-i})${
Скок на S
$i \leftarrow i + 1$
}

$i \leftarrow 1$
додека$\big((m - i) > f(A_o)/\mu\big)$
најди χ совпаѓања R_ξ: $(p_{a,\xi}, p_{b,\xi}, l_\xi)$, $|R_\xi| \geq 1$ во $(a_1 \dots a_{m-i-1}a_{m-i}; b_{i+1} \dots b_{m-1}b_m)${

```
Скок на S
i ← i + 1
}

S: ако(χ = 0){
    A_{i,o} ← ∅
   f(A_{i,o}) ← 0
    врати се од S
    }
инаку {
A_{i,o} ← FLAG(R_1, R_2, … , R_{χ−1}, R_χ)
 ако (f(A_{i,o}) > f(A_o)){
A_o ← A_{i,o}
f(A_o) ← f(A_{i,o})
  }
врати се од S
 }


FLAG(влез: {R_1, R_2, … , R_{χ−1}, R_χ}, излез: A){
  ако(χ = 1)
    A ← R_1
   инаку
   {
   R_σ ← макс. долж {R_1, R_2, … , R_{χ−1}R_χ}
    A ← R_σ
     A_e ← R_σ
      f(A) ← μ × долж(A)

     ако (σ=1)
       за (ξ ←2; ξ ≤ χ; ξ++)
       {
        A_e ← A_e ⋈ R_ξ
         ако(f(A_e) > f(A)){
           f(A) ← f(A_e)
            A ← A_e
           }
         }

      инаку ако (σ=χ)
        за (ξ ← χ − 1; ξ ≥ 1; ξ − −)
        {
         A_e ← R_ξ ⋈ A_e
         ако (f(A_e) > f(A)){
           f(A) ← f(A_e)
            A ← A_e
         }
       }
```

```
  инаку ако (2 ≤ σ ≤ χ − 1)
   за (ξ ← σ − 1; ε ≥ 1; ε − −)
    {
    A_e ← R_ξ ⋈ A_e
    ако (f(A_e) > f(A)){
    f(A) ← f(A_e)
     A ← A_e
    }
   }
A_e ← A
    за (ξ ← σ + 1; ε ≤ χ; ε + +)
     {
    A_e ← A_e ⋈ R_ξ
    ако (f(A_e) > f(A)){
     f(A) ← f(A_e)
     A ← A_e
      }
    }
   }
```

Алгоритам за издвојување на множество на конзистентни совпаѓања
Влез:
ДНК секвенца 1: $a = a_1 a_2 \dots a_{n-1} a_n$, ДНК секвенца 2: $b = b_1 b_2 \dots b_{m-1} b_m$ $(n \geq m)$
Излез:
SetOfConsistentHits:
Множество на конзистентни совпаѓања од тип: $\left(R_\xi^a, R_\xi^b\right) =$
$(p_{a,s}|R_\xi^a, p_{a,f}|R_\xi^a, p_{b,s}|R_\xi^b, p_{b,f}|R_\xi^b)$

Променливи:
$p_{a,s}\left(p_{b,s}\right)$: почетна положба во $a(b)$
$p_{a,f}\left(p_{b,f}\right)$: крајна положба во $a(b)$
R_ξ^a: Совпаѓање R_ξ во a, R_ξ^b: Совпаѓање R_ξ во b
$p_{a,s}|R_\xi^a$: почетна положба на R_ξ во a, $p_{a,f}|R_\xi^a$: крајна положба на R_ξ во a
$p_{b,s}|R_\xi^b$: почетна положба на R_ξ во b, $p_{b,f}|R_\xi^b$: крајна положба на R_ξ во b
CSR: конзистентен опсег на пребарување во a и b
hit: 1(постои совпаѓање во CSR), 0(не постои совпаѓање во CSR)
w_a: збор во a_f
w_b: збор во b_f
Ознаки и оператори:
||: должина на множество на совпаѓања (број на податочни четворки), должина на збор (број на карактери)
∅: празно множество
←: додела на вредност, додавање кон податочна структура
=: ознака за еднаквост, оператор за проверка на еднаквост
Функции:
Sort(): функција за сортирање на множество на совпаѓања

SetOfConsistentHits ← ∅

додека се зголемува |SetOfConsistentHits|{
за секој CSR $= \left(a_f: a_{i+1}a_{i+2} \dots a_{i+|a_f|}\ ;\ b_f: b_{j+1}b_{j+2} \dots b_{j+|b_f|}, |a_f| \geq 1, |b_f| \geq 1\right)$ каде:
ако $(|\text{SetOfConsistentHits}| = 0)$: $i = 0, |a_f| = n, j = 0, |b_f| = m$
ако $(|\text{SetOfConsistentHits}| = 1)$:
$\left(i = 0, |a_f| = p_{a,s}\left|R_1^a - 1, j = 0, |b_f| = p_{b,s}\right|R_1^b - 1\right)$ или $(i = p_{a,f}|R_1^a, |a_f| = n - p_{a,f}|R_1^a, j = p_{b,f}|R_1^b, |b_f| = m - p_{b,f}|R_1^b)$
ако $(|\text{SetOfConsistentHits}| = r > 1)$:
$\left(i = 0, |a_f| = p_{a,s}\left|R_1^a - 1, j = 0, |b_f| = p_{b,s}\right|R_1^b - 1\right)$
или
$\left(i = p_{a,f}\left|R_\xi^a, |a_f| = p_{a,s}|R_{\xi+1}^a - p_{a,f}\right|R_\xi^a - 1, j = p_{b,f}\left|R_\xi^b, |b_f| = p_{b,s}|R_{\xi+1}^b - p_{b,f}\right|R_\xi^b - 1\right)$, $\xi = 1,2, \dots, r-1$
или
$(i = p_{a,f}|R_r^a, |a_f| = n - p_{a,f}|R_r^a, j = p_{b,f}|R_r^b, |b_f| = m - p_{b,f}|R_r^b)$
{
$k \leftarrow 0$
$hit \leftarrow 0$
ако $(|a_f| > |b_f|)$){
S1: за секој пар на зборови (w_a, w_b) каде: $w_b \in b_f, |w_b| = |b_f| - k \geq 1, w_a \in a_f, |w_a| = |w_b|$
ако $(\text{Compare}(w_a, w_b) = \text{вистина})$ {

SetOfConsistentHits ← $\left(R^{a}_{r+1} = w_a : p_{a,s}, p_{a,f}, R^{b}_{r+1} = w_b : p_{b,s}, p_{b,f}\right)$
r ← r + 1
hit ← 1
напушти циклус: за секој пар на зборови
}
ако (hit = 0){
k ← k + 1
Скок на: S1
}
}
инаку{
S2: за секој пар на зборови (w_a, w_b) каде: $w_a \in a_f, |w_a| = |a_f| - k \geq 1, w_b \in b_f, |w_a| = |w_b|$
ако (Compare(w_a, w_b) = вистина){
SetOfConsistentHits ← $\left(R^{a}_{r+1} = w_a : p_{a,s}, p_{a,f}, R^{b}_{r+1} = w_b : p_{b,s}, p_{b,f}\right)$
r ← r + 1
hit ← 1
напушти циклус: за секој пар на зборови
}
ако(hit = 0){
k ← k + 1
Скок на: S2
}
}
}
}

SetOfConsistentHits. Sort(податочни четворки: $\left(p_{a,s}, p_{a,f}, p_{b,s}, p_{b,f}\right)$, критериум: $p_{a,s}$, p растечки)

Compare(влез: збор x: $x_1x_2 \ldots x_t$, збор y: $y_1y_2 \ldots y_t$, излез: вистина(невистина)){
ако$\left((x_1 = y_1) \text{ и } |x| > 1\right)$ Compare(x: $x_2x_3 \ldots x_t$; y: $y_2y_3 \ldots y_t$)
инаку ако$\left((x_1 = y_1) \text{ and } |x| = 1\right)$ врати вистина
инаку врати невистина
}

Алгоритам за порамнување, базиран на моделот за последователно додавање на празнини по совпаѓање
Влез:
ДНК секвенца 1: $a = a_1 a_2 \dots a_{n-1} a_n$, ДНК секвенца 2: $b = b_1 b_2 \dots b_{m-1} b_m$ $(n \geq m)$, множество на конзистентни совпаѓања: SetOfConsistentHits
Излез:
AlignedSeqA: празнинска модификација на ДНК секвенца a
AlignedSeqB: празнинска модификација на ДНК секвенца b

Променливи:
numberOfGaps: број на празнини кои треба да се додадат во $a(b)$, за да се порамнат две последователни совпаѓања: (R_ξ^a, R_ξ^b) и $(R_{\xi+1}^a, R_{\xi+1}^b)$
Ознаки и оператори:
←: додела на вредност
Функции:
append(): функција за соединување на карактер или збор

$AlignedSeqA = ""$
$AlignedSeqB = ""$
за секои две совпаѓања: (R_ξ^a, R_ξ^b) и $(R_{\xi+1}^a, R_{\xi+1}^b)$ во SetOfConsistentHits
{
$numberOfGaps \leftarrow p_{a,s}|R_{\xi+1}^a - p_{a,f}|R_\xi^a - (p_{b,s}|R_{\xi+1}^b - p_{b,f}|R_\xi^b)$
ако$(numberOfGaps > 0)$
{
$AlignedSeqA.append(a_{p_{a,s}|R_\xi^a} \dots a_{p_{a,s}|R_{\xi+1}^a - 1})$
$AlignedSeqB.append(b_{p_{b,s}|R_\xi^b} \dots b_{p_{b,f}|R_\xi^b})$
додека$(numberOfGaps > 0)$
{
$AlignedSeqB.append("\_")$
$numberOfGaps \leftarrow numberOfGaps - 1$
}
$AlignedSeqB.append(b_{p_{b,f}|R_\xi^b + 1} \dots b_{p_{b,s}|R_{\xi+1}^b - 1})$
}
инаку ако$(numberOfGaps < 0)$
{
$numberOfGaps \leftarrow -numberOfGaps$
$AlignedSeqB.append(b_{p_{b,s}|R_\xi^b} \dots b_{p_{b,s}|R_{\xi+1}^b - 1})$
$AlignedSeqA.append(a_{p_{a,s}|R_\xi^a} \dots a_{p_{a,f}|R_\xi^a})$

додека$(numberOfGaps > 0)$
{
$AlignedSeqA.append("\_")$
$numberOfGaps \leftarrow numberOfGaps - 1$
}
$AlignedSeqA.append(a_{p_{a,f}|R_\xi^a + 1} \dots a_{p_{a,s}|R_{\xi+1}^a - 1})$
}
инаку
{

AlignedSeqA. append($a_{p_{a,s}|R^a_\xi} \dots a_{p_{a,s}|R^a_{\xi+1}-1}$)
AlignedSeqB. append($b_{p_{b,s}|R^b_\xi} \dots b_{p_{b,s}|R^b_{\xi+1}-1}$)
}
}
AlignedA. append($a_{p_{a,s}|R^a_{|SetOfConsistentHits|}} \dots a_{p_{a,f}|R^a_{|SetOfConsistentHits|}}$)
AlignedB. append($b_{p_{b,s}|R^b_{|SetOfConsistentHits|}} \dots b_{p_{b,f}|R^b_{|SetOfConsistentHits|}}$)

Алгоритам за порамнување, базиран на моделот за додавање на празнини на положби во зависност од фреквенциите на базни замени
Влез:
ДНК секвенца 1: $a = a_1 a_2 \dots a_{n-1} a_n$, ДНК секвенца 2: $b = b_1 b_2 \dots b_{m-1} b_m$ $(n \geq m)$, множество на конзистентни совпаѓања: SetOfConsistentHits
Излез:
AlignedSeqA: празнинска модификација на ДНК секвенца a
AlignedSeqB: празнинска модификација на ДНК секвенца b

Променливи:
SubstitutionMatrixA: Матрица на базни замени за a
SubstitutionMatrixB: Матрица на базни замени за b
substringInB: празнински модифициран збор во b (на почеток substringInB е збор помеѓу две последователни совпаѓања во b, во кој се вклучени последниот и првиот елемент од последователните совпаѓања)
substringInA: празнински модифициран збор во a (на почеток substringInA е збор помеѓу две последователни совпаѓања во a, во кој се вклучени последниот и првиот елемент од последователните совпаѓања)
Ознаки и оператори:
←: додела на вредност
! =: ознака за различност
+: оператор за соединување на карактер или збор
Функции:
append(): функција за соединување на карактер или збор
CountOf(): функција која враќа број на базни парови XY
min(): функција за наоѓање на најмал елемент

AlignedSeqA ← ""
AlignedSeqB ← ""
SubstitutionMatrixA[0(A), 1(C), 2(T), 3(G) ; 0(A), 1(C), 2(T), 3(G)]
SubstitutionMatrixB[0(A), 1(C), 2(T), 3(G) ; 0(A), 1(C), 2(T), 3(G)]

за секој пар XY во a, каде X, Y ∈ {A, C, T, G}
SubstitutionMatrixA[X, Y] ← број на појавувања на XY во a
за секој пар XY во b, каде X, Y ∈ {A, C, T, G}
SubstitutionMatrixB[X, Y] ← број на појавувања на XY во b

за секои две совпаѓања: $\left(R_\xi^a, R_\xi^b\right)$ и $\left(R_{\xi+1}^a, R_{\xi+1}^b\right)$ во SetOfConsistentHits
{
numberOfGaps ← $p_{a,s}\left|R_{\xi+1}^a - p_{a,f}\right|R_\xi^a - (p_{b,s}\left|R_{\xi+1}^b - p_{b,f}\right|R_\xi^b)$
ако(numberOfGaps > 0)
{
substringInB ← $b_{p_{b,f}|R_\xi^b} b_{p_{b,f}|R_\xi^b+1} \dots b_{p_{b,s}|R_{\xi+1}^b}$
додека(numberOfGaps > 0)
{
ако(CountOf(XY базни парови во substringInB, каде: X! = " _ " and Y! = " _ ") > 1)
substringInB ← $\text{substringInB}_{1\dots t}$.append(" _ ").$\text{substringInB}_{(t+1)\dots|\text{substringInB}|}$, каде:
$\text{SubstitutionMatrixB}[\text{substringInB}_t, \text{substringInB}_{t+1}] =$
min(SubstitutionMatrixB[X, Y], XY ∈ substringInB)

инаку ако(CountOf(XY базни парови во substringInB, каде: X! = " _ " and Y! = " _ ") = 1)
substringInB ← $substringInB_{1...t}$.append(" _ ").$substringInB_{(t+1)...|substringInB|}$, каде:
$substringInB_t$! = "_" and $substringInB_{t+1}$! = "_"

инаку
substringInB ← $substringInB_1$.append(" _ ").$substringInB_{2...|sustringInB|}$
numberOfGaps ← numberOfGaps − 1
}
AlignedSeqA.append($a_{p_{a,s}|R_\xi^a} \dots a_{p_{a,s}|R_{\xi+1}^a-1}$)
AlignedSeqB.append($b_{p_{b,s}|R_\xi^b} \dots b_{p_{b,f}|R_\xi^b} + substringInB_{2...|substringInB|-1}$)
}

инаку ако(numberOfGaps < 0)
{
numberOfGaps ← −numberOfGaps
substringInA ← $a_{p_{a,f}|R_\xi^a} a_{p_{a,f}|R_\xi^a+1} \dots a_{p_{a,s}|R_{\xi+1}^a}$
додека(numberOfGaps > 0)
{
ако(CountOf(XY базни парови во substringInA, каде: X! = " _ " and Y! = " _ ") > 1)
substringInA ← $substringInA_{1...t}$.append(" _ ").$substringInA_{(t+1)...|substringInA|}$, каде:
SubstitutionMatrixA[$substringInA_t$, $substringInA_{t+1}$] =
min(SubstitutionMatrixA[X, Y], XY ∈ substringInA)

инаку ако(CountOf(XY базни парови во substringInA, каде: X! = " _ " and Y! = " _ ") = 1)
substringInA ← $substringInA_{1...t}$.append(" _ ").$substringInA_{(t+1)...|substringInA|}$, каде:
$substringInA_t$! = "_" and $substringInA_{t+1}$! = "_"

инаку
substringInA ← $substringInA_1$.append(" _ "). $substringInA_{2...|substringInA|}$
numberOfGaps ← numberOfGaps − 1
}
AlignedSeqB.append($b_{p_{b,s}|R_\xi^b} \dots b_{p_{b,s}|R_{\xi+1}^b-1}$)
AlignedSeqA.append($a_{p_{a,s}|R_\xi^a} \dots a_{p_{a,f}|R_\xi^a} + substringInA_{2...|substringInA|-1}$)
}
инаку{
AlignedSeqA.append($a_{p_{a,s}|R_\xi^a} \dots a_{p_{a,s}|R_{\xi+1}^a-1}$)
AlignedSeqB.append($b_{p_{b,s}|R_\xi^b} \dots b_{p_{b,s}|R_{\xi+1}^b-1}$)
}
}
AlignedA.append($a_{p_{a,s}|R^a_{|SetOfConsistentHits|}} \dots a_{p_{a,f}|R^a_{|SetOfConsistentHits|}}$)
AlignedB.append($b_{p_{b,s}|R^b_{|SetOfConsistentHits|}} \dots b_{p_{b,f}|R^b_{|SetOfConsistentHits|}}$)

Алгоритам за индексирање на ДНК база на податоци
Влез:
База на ДНК секвенци: $D = \{S_1, \dots, S_{|D|}\}$, максимална должина на ДНК прашалник: $|q_{max}|$
Излез:
Сортиран речник: SD

Променливи:
k: должина на индексирање
i: индекс на секвенца во D
S_i: i – та секвенца во D
p: почетна положба на збор во секвенца S_i
$w_{i,p}$: збор во должина од k нуклеотиди, кој се наоѓа на почетна положба p во секвенца S_i
$b_{i,p}$: нуклеотид на положба p во секвенца S_i
Ознаки и оператори:
$\leftarrow$: додела на вредност
$||$: должина на збор (секвенца) - број на карактери
$=$: проверка на еднаквост
$++$: зголемување на вредност на променлива за 1
Функции:
$add()$: функција за додавање на клуч: $f(w_{i,p}) \rightarrow$ податочен пар: (i, p) во сортиран речник SD
$f()$: функција за пресликување на нуклеотид b или збор w:
$f(A) = 1, f(T) = 2, f(G) = 3, f(C) = 4$, $f(w: b_1 b_2 \dots b_{k-1} b_k) = \sum_{j=1}^{k} f(b_j) \times 4^{j-1}, b_j \in \{A, T, G, C\}$

$k \leftarrow |q_{max}| + 1$
за секоја секвенца S_i во D
$за(p \leftarrow 1; p \leq |S_i| - k + 1; p++)$
$ако(p = 1)$

$$f(w_{i,1}: b_{i,1} b_{i,2} \dots b_{i,k-1} b_{i,k}) \leftarrow \sum_{j=1}^{k} f(b_{i,j}) \times 4^{j-1}$$

$$SD.add\left(f(w_{i,1}), (i, 1)\right)$$

инаку

$$f(w_{i,p}: b_{i,p} b_{i,p+1} \dots b_{i,p+k-1}) \leftarrow \frac{f(w_{i,p-1}: b_{i,p-1} b_{i,p} \dots b_{i,p+k-2}) - f(b_{i,p-1})}{4} + f(b_{i,p+k-1}) \times 4^{k-1}$$

$$SD.add\left(f(w_{i,p}), (i, p)\right)$$

Алгоритам за пребарување на ДНК база на податоци
Влез:
Сортиран речник: SD, ДНК прашалник: q
Излез:
Множество на совпаѓања на q во база на ДНК секвенци D: Hits

Променливи:
q^{x}: издолжување на q до k нуклеотиди
q_{j}: нуклеотид на положба j во q
$\mathrm{q}_{\mathrm{j}}^{\mathrm{x}}$: нуклеотид на положба j во q^{x}
s_{z}: запис во сортиран речник SD
$\mathrm{s}_{\mathrm{z}}(\mathrm{клуч})$: клуч на запис s_{z}
$\mathrm{s}_{\mathrm{z}}(\mathrm{вредност})$: вредност на запис s_{z} од тип множество на податочни парови: (i, p)
Ознаки и оператори:
$\leftarrow$: додела на вредност или податочна структура
$||$: должина на збор (број на карактери)
$\emptyset$: празно множество
$=$: ознака за еднаквост, оператор за проверка на еднаквост
Функции:
$\mathrm{mod}()$: функција за пресметка на остаток при целобројно делење
$\mathrm{f}()$: функција за пресликување на нуклеотид b или збор w:
$\mathrm{f}(\mathrm{A}) = 1, \mathrm{f}(\mathrm{T}) = 2, \mathrm{f}(\mathrm{G}) = 3, \mathrm{f}(\mathrm{C}) = 4,\ \mathrm{f}(\mathrm{w}: \mathrm{b}_1\mathrm{b}_2 \dots \mathrm{b}_{\mathrm{k}-1}\mathrm{b}_{\mathrm{k}}) = \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{f}(\mathrm{b}_{\mathrm{j}}) \times 4^{\mathrm{j}-1}, \mathrm{b}_{\mathrm{j}} \in \{\mathrm{A}, \mathrm{T}, \mathrm{G}, \mathrm{C}\}$

$\mathrm{Hits} \leftarrow \emptyset$

$$\mathrm{f}\left(\mathrm{q}: \mathrm{q}_1 \dots \mathrm{q}_{|\mathrm{q}|}\right) \leftarrow \sum_{\mathrm{j}=1}^{|\mathrm{q}|} \mathrm{f}(\mathrm{q}_{\mathrm{j}}) \times 4^{\mathrm{j}-1}$$

$$\mathrm{f}\left(\mathrm{q}^{\mathrm{suffix,min}} = \underbrace{\mathrm{A} \dots \mathrm{A}}_{\mathrm{k}-|\mathrm{q}|}\, \mathrm{q}_1 \dots \mathrm{q}_{|\mathrm{q}|}\right) \leftarrow \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{f}(\mathrm{q}_{\mathrm{j}}^{\mathrm{suffix,min}}) \times 4^{\mathrm{j}-1}$$

$$\mathrm{f}\left(\mathrm{q}^{\mathrm{suffix,max}} = \underbrace{\mathrm{C} \dots \mathrm{C}}_{\mathrm{k}-|\mathrm{q}|}\, \mathrm{q}_1 \dots \mathrm{q}_{|\mathrm{q}|}\right) \leftarrow \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{f}(\mathrm{q}_{\mathrm{j}}^{\mathrm{suffix,max}}) \times 4^{\mathrm{j}-1}$$

$$\mathrm{f}\left(\mathrm{q}^{\mathrm{prefix,min}} = \mathrm{q}_1 \dots \mathrm{q}_{|\mathrm{q}|}\, \underbrace{\mathrm{A} \dots \mathrm{A}}_{\mathrm{k}-|\mathrm{q}|}\right) \leftarrow \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{f}(\mathrm{q}_{\mathrm{j}}^{\mathrm{prefix,min}}) \times 4^{\mathrm{j}-1}$$

$$\mathrm{f}\left(\mathrm{q}^{\mathrm{prefix,max}} = \mathrm{q}_1 \dots \mathrm{q}_{|\mathrm{q}|}\, \underbrace{\mathrm{C} \dots \mathrm{C}}_{\mathrm{k}-|\mathrm{q}|}\right) \leftarrow \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{f}(\mathrm{q}_{\mathrm{j}}^{\mathrm{prefix,max}}) \times 4^{\mathrm{j}-1}$$

за секој запис s_{z} во SD
ако $\left(\left(\mathrm{s}_{\mathrm{z}}(\mathrm{клуч}) > f(\mathrm{q}^{\mathrm{suffix,max}})\right)$ и $\left(\mathrm{s}_{\mathrm{z}}(\mathrm{клуч}) > f(\mathrm{q}^{\mathrm{prefix,max}})\right)\right)$
напушти циклус: за секој запис s_{z} во SD

ако $\left(\mathrm{f}\left(\mathrm{q}^{\mathrm{suffix,min}}\right) \le \mathrm{s}_{\mathrm{z}}(\mathrm{клуч}) \le \mathrm{f}\left(\mathrm{q}^{\mathrm{suffix,max}}\right)\right)$
$\mathrm{Hits} \leftarrow \mathrm{s}_{\mathrm{z}}(\mathrm{вредност}) = \text{множество на податочни парови: } (\mathrm{i}, \mathrm{p} + \mathrm{k} - |\mathrm{q}|)$

ако $\left(\left(f\left(q^{prefix,min}\right) \leq s_z(\text{клуч}) \leq f\left(q^{prefix,max}\right)\right)\right.$ и $\left.\left(\text{mod}\left(s_z(\text{клуч}) - f(q), 4^{|q|}\right) = 0\right)\right)$
Hits $\leftarrow s_z(\text{вредност}) =$ множество на податочни парови: $(i, p), p \leq k - |q|$

РЕФЕРЕНЦИ

[1] Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of molecular biology, 48(3), 443-453.

[2] Sellers, P. H. (1974). An algorithm for the distance between two finite sequences. Journal of Combinatorial Theory, Series A, 16(2), 253-258.

[3] Ulam, S. M. (1972). Some combinatorial problems studied experimentally on computing machines. Zaremba SK, Applications of Number Theory to Numerical Analysis, 1-3.

[4] Wagner, R. A., & Fischer, M. J. (1974). The string-to-string correction problem.Journal of the ACM (JACM), 21(1), 168-173.

[5] Waterman, M. S., Smith, T. F., & Beyer, W. A. (1976). Some biological sequence metrics. Advances in Mathematics, 20(3), 367-387.

[6] Sellers, P. H. (1979). Pattern recognition in genetic sequences. Proc. Natl. Acad. Sci. USA, 76(7), 3041.

[7] Smith, T. F., Waterman, M. S., & Fitch, W. M. (1981). Comparative biosequence metrics. Journal of Molecular Evolution, 18(1), 38-46.

[8] Sellers, P. H. (1980). The theory and computation of evolutionary distances: pattern recognition. Journal of algorithms, 1(4), 359-373.

[9] Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. Journal of molecular biology, 147(1), 195-197.

[10] Goad, W. B., & Kanehisa, M. I. (1982). Pattern recognition in nucleic acid sequences. I. A general method for finding local homologies and symmetries.Nucleic Acids Research, 10(1), 247-263.

[11] Sellers, P. H. (1984). Pattern recognition in genetic sequences by mismatch density. Bulletin of Mathematical Biology, 46(4), 501-514.

[12] Fitch, W. M., & Smith, T. F. (1983). Optimal sequence alignments. Proceedings of the National Academy of Sciences, 80(5), 1382-1386.

[13] Gotoh, O. (1982). An improved algorithm for matching biological sequences. Journal of molecular biology, 162(3), 705-708.

[14] Gotoh, O. (1987). Pattern matching of biological sequences with limited storage. Computer applications in the biosciences: CABIOS, 3(1), 17-20.

[15] Waterman, M. S., & Eggert, M. (1987). A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. Journal of molecular biology, 197(4), 723-728.

[16] Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. Communications of the ACM, 18(6), 341-343.

[17] Myers, E. W. and Miller, W. (1988) Optimal alignments in linear space. CABIOS, 4, 11-17.

[18] Miller, W. and Myers, E. W. (1988) Sequence comparison with concave weighting functions. Bull. Math. Biol., 50, 97-120.

[19] Huang, X., Hardison, R. C. and Miller, W. (1990) A space-efficient algorithm for local similarities. CABIOS, 6, 373-381.

[20] Huang, X., & Miller, W. (1991). A time-efficient, linear-space local similarity algorithm. Advances in Applied Mathematics, 12(3), 337-357.

[21] Huang, X., & Miller, W. (1991). A time-efficient, linear-space local similarity algorithm. Advances in Applied Mathematics, 12(3), 337-357.

[22] Sankoff, D., & Kruskal, J. B. (1983). Time warps, string edits, and macromolecules: the theory and practice of sequence comparison. Reading: Addison-Wesley Publication, 1983, edited by Sankoff, David; Kruskal, Joseph B., 1.

[23] Fickett, J. W. (1984). Fast optimal alignment. Nucleic acids research,12(1Part1), 175-179.

[24] Ukkonen, E. (1985). Algorithms for approximate string matching. Information and control, 64(1), 100-118.

[25] Chao, K. M., Pearson, W. R., & Miller, W. (1992). Aligning two sequences within a specified diagonal band. Computer applications in the biosciences: CABIOS, 8(5), 481-487.

[26] Spouge, J. L. (1991) Fast optimal alignment. CABIOS, 7, 1-7.

[27] Lipman, D. J., & Pearson, W. R. (1985). Rapid and sensitive protein similarity searches. Science, 227(4693), 1435-1441.

[28] Pearson, W. R., & Lipman, D. J. (1988). Improved tools for biological sequence comparison. Proceedings of the National Academy of Sciences, 85(8), 2444-2448.

[29] S.F. Altschul, W. Gish, W. Miller, E.W. Myers and D.J. Lipman. Basic local alignment search tool, J. Molecular Biology, 215:403-410 (1990).

[30] Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., & Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic acids research, 25(17), 3389-3402.

[31] Ma, B., Tromp, J., & Li, M. (2002). PatternHunter: faster and more sensitive homology search. Bioinformatics, 18(3), 440-445.

[32] Noé, L., & Kucherov, G. (2005). YASS: enhancing the sensitivity of DNA similarity search. Nucleic acids research, 33(suppl 2), W540-W543.

[33] Califano, A., & Rigoutsos, I. (1993, June). FLASH: A fast look-up algorithm for string homology. In Computer Vision and Pattern Recognition, 1993. Proceedings CVPR'93., 1993 IEEE Computer Society Conference on (pp. 353-359). IEEE.

[35] Delcher, A. L., Kasif, S., Fleischmann, R. D., Peterson, J., White, O., & Salzberg, S. L. (1999). Alignment of whole genomes. Nucleic Acids Research, 27(11), 2369-2376.

[36] Batzoglou, S., Pachter, L., Mesirov, J. P., Berger, B., & Lander, E. S. (2000). Human and mouse gene structure: comparative analysis and application to exon prediction. Genome Research, 10(7), 950-958.

[37] Morgenstern, B., Frech, K., Dress, A., & Werner, T. (1998). DIALIGN: finding local similarities by multiple sequence alignment. Bioinformatics, 14(3), 290-294.

[38] Bray, N., Dubchak, I., & Pachter, L. (2003). AVID: A global alignment program. Genome research, 13(1), 97-102.

[39] Gusfield, D. (1997). Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge University Press.

[40] Brudno, M., Do, C. B., Cooper, G. M., Kim, M. F., Davydov, E., Green, E. D., ... & NISC Comparative Sequencing Program. (2003). LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. Genome research, 13(4), 721-731.

[41] Brudno, M., & Morgenstern, B. (2002). Fast and sensitive alignment of large genomic sequences. In Bioinformatics Conference, 2002. Proceedings. IEEE Computer Society (pp. 138-147). IEEE.

[42] Shen, S. Y., Yang, J., Yao, A., & Hwang, P. I. (2002). Super pairwise alignment (SPA): an efficient approach to global alignment for homologous sequences. Journal of Computational Biology, 9(3), 477-486.

[44] Stokes, W. A., & Glick, B. S. (2006). MICA: desktop software for comprehensive searching of DNA databases. BMC bioinformatics, 7(1), 427.

[45] Benson, G. (1999). Tandem repeats finder: a program to analyze DNA sequences. Nucleic acids research, 27(2), 573.

[46] Rigoutsos, I., & Floratos, A. (1998). Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm. Bioinformatics, 14(1), 55-67.

[47] Giladi, E., Walker, M. G., Wang, J. Z., & Volkmuth, W. (2002). SST: an algorithm for finding near-exact sequence matches in time proportional to the logarithm of the database size. Bioinformatics, 18(6), 873-877.

[48] MacQueen, J. (1967, June). Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability (Vol. 1, No. 14, pp. 281-297).

[49] Meek, C., Patel, J. M., & Kasetty, S. (2003, September). Oasis: An online and accurate technique for local-alignment searches on biological sequences. In Proceedings of the 29th international conference on Very large data bases-Volume 29 (pp. 910-921). VLDB Endowment.

[50] Delcher, A. L., Kasif, S., Fleischmann, R. D., Peterson, J., White, O., & Salzberg, S. L. (1999). Alignment of whole genomes. Nucleic Acids Research, 27(11), 2369-2376.

[51] Delcher, A. L., Phillippy, A., Carlton, J., & Salzberg, S. L. (2002). Fast algorithms for large-scale genome alignment and comparison. Nucleic acids research, 30(11), 2478-2483.

[52] Kurtz, S., Phillippy, A., Delcher, A. L., Smoot, M., Shumway, M., Antonescu, C., & Salzberg, S. L. (2004). Versatile and open software for comparing large genomes. Genome biology, 5(2), R12.

[53] Abouelhoda, M. I., Kurtz, S., & Ohlebusch, E. (2004). Replacing suffix trees with enhanced suffix arrays. Journal of Discrete Algorithms, 2(1), 53-86.

[54] Kurtz, S. (1999). Reducing the space requirement of suffix trees. Software-Practice and Experience, 29(13), 1149-71.

[55] Khan, Z., Bloom, J. S., Kruglyak, L., & Singh, M. (2009). A practical algorithm for finding maximal exact matches in large sequence datasets using sparse suffix arrays. Bioinformatics, 25(13), 1609-1616.

[56] Vyverman, M., De Baets, B., Fack, V., & Dawyndt, P. (2013). essaMEM: finding maximal exact matches using enhanced sparse suffix arrays. Bioinformatics, 29(6), 802-804.

[57] Ferragina, P., & Manzini, G. (2000). Opportunistic data structures with applications. In Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on (pp. 390-398). IEEE.

[58] Grossi, R., & Vitter, J. S. (2005). Compressed suffix arrays and suffix trees with applications to text indexing and string matching. SIAM Journal on Computing, 35(2), 378-407.

[59] Burrows, M., & Wheeler, D. J. (1994). A block-sorting lossless data compression algorithm.

[60] Lippert, R. A., Mobarry, C. M., & Walenz, B. P. (2005). A space-efficient construction of the Burrows-Wheeler transform for genomic data. Journal of Computational Biology, 12(7), 943-951.

[61] Lippert, R. A. (2005). Space-efficient whole genome comparisons with Burrows-Wheeler transforms. Journal of Computational Biology, 12(4), 407-415.

[62] Ning, Z., Cox, A. J., & Mullikin, J. C. (2001). SSAHA: a fast search method for large DNA databases. Genome research, 11(10), 1725-1729.

[63] Reneker, J., & Shyu, C. R. (2005). Refined repetitive sequence searches utilizing a fast hash function and cross species information retrievals. BMC bioinformatics, 6(1), 111.

[64] Mäkinen, V., Navarro, G., Sirén, J., & Välimäki, N. (2009, January). Storage and retrieval of individual genomes. In Research in Computational Molecular Biology (pp. 121-137). Springer Berlin Heidelberg.

[65] Li, R., Li, Y., Zheng, H., Luo, R., Zhu, H., Li, Q., ... & Wang, J. (2010). Building the sequence map of the human pan-genome. Nature biotechnology, 28(1), 57-63.

[66] Langmead, B., Schatz, M. C., Lin, J., Pop, M., & Salzberg, S. L. (2009). Searching for SNPs with cloud computing. Genome Biol, 10(11), R134.

[67] Carver, T., Böhme, U., Otto, T. D., Parkhill, J., & Berriman, M. (2010). BamView: viewing mapped read alignment data in the context of the reference sequence. Bioinformatics, 26(5), 676-677.

[68] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., ... & Durbin, R. (2009). The sequence alignment/map format and SAMtools.Bioinformatics, 25(16), 2078-2079.

[69] De Bona, F., Ossowski, S., Schneeberger, K., & Rätsch, G. (2008). Optimal spliced alignments of short sequence reads. BMC Bioinformatics, 9(Suppl 10), O7.

[70] Mortazavi, A., Williams, B. A., McCue, K., Schaeffer, L., & Wold, B. (2008). Mapping and quantifying mammalian transcriptomes by RNA-Seq. Nature methods, 5(7), 621-628.

[71] Xi, Y., & Li, W. (2009). BSMAP: whole genome bisulfite sequence MAPping program. BMC bioinformatics, 10(1), 232.

[72] Homer, N., Merriman, B., & Nelson, S. F. (2009). Local alignment of two-base encoded DNA sequence. BMC bioinformatics, 10(1), 175.

[73] Chen, K., Wallis, J. W., McLellan, M. D., Larson, D. E., Kalicki, J. M., Pohl, C. S., ... & Mardis, E. R. (2009). BreakDancer: an algorithm for high-resolution mapping of genomic structural variation. Nature methods, 6(9), 677-681.

[74] Blumer, A., Blumer, J., Haussler, D., Ehrenfeucht, A., Chen, M. T., & Seiferas, J. (1985). The smallest automation recognizing the subwords of a text.Theoretical Computer Science, 40, 31-55.

[75] Lam, T. W., Sung, W. K., Tam, S. L., Wong, C. K., & Yiu, S. M. (2008). Compressed indexing and local alignment of DNA. Bioinformatics, 24(6), 791-797.

[76] Li, R., Yu, C., Li, Y., Lam, T. W., Yiu, S. M., Kristiansen, K., & Wang, J. (2009). SOAP2: an improved ultrafast tool for short read alignment.Bioinformatics, 25(15), 1966-1967.

[77] Langmead, B., Trapnell, C., Pop, M., & Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome.Genome Biol, 10(3), R25.

[78] Navarro, G. (2001). A guided tour to approximate string matching. ACM computing surveys (CSUR), 33(1), 31-88.

[79] Munro, J. I., Raman, V., & Rao, S. S. (2001). Space efficient suffix trees.Journal of Algorithms, 39(2), 205-222.

[80] Slater, G. S., & Birney, E. (2005). Automated generation of heuristics for biological sequence comparison. BMC bioinformatics, 6(1), 31.

[81] Farrar, M. (2007). Striped Smith–Waterman speeds database searches six times over other SIMD implementations. Bioinformatics, 23(2), 156-161.

[82] Burkhardt, S., & Kärkkäinen, J. (2003). Better filtering with gapped q-grams.Fundamenta informaticae, 56(1), 51-70.

[83] Cao, X., Li, S. C., & Tung, A. K. (2005, January). Indexing dna sequences using q-grams. In Database Systems for Advanced Applications (pp. 4-16). Springer Berlin Heidelberg.

[84] Weese, D., Emde, A. K., Rausch, T., Döring, A., & Reinert, K. (2009). RazerS—fast read mapping with sensitivity control. Genome research, 19(9), 1646-1654.

[85] Chen, Y., Souaiaia, T., & Chen, T. (2009). PerM: efficient mapping of short sequencing reads with periodic full sensitive spaced seeds. Bioinformatics,25(19), 2514-2521.

[86] Schatz, M. C. (2009). CloudBurst: highly sensitive read mapping with MapReduce. Bioinformatics, 25(11), 1363-1369.

[87] Homer, N., Merriman, B., & Nelson, S. F. (2009). BFAST: an alignment tool for large scale genome resequencing. PloS one, 4(11), e7767.

[88] Lin, H., Zhang, Z., Zhang, M. Q., Ma, B., & Li, M. (2008). ZOOM! Zillions of oligos mapped. Bioinformatics, 24(21), 2431-2437.

[89] Smith, A. D., Chung, W. Y., Hodges, E., Kendall, J., Hannon, G., Hicks, J., ... & Zhang, M. Q. (2009). Updates to the RMAP short-read mapping software.Bioinformatics, 25(21), 2841-2842.

[90] Jiang, H., & Wong, W. H. (2008). SeqMap: mapping massive amount of oligonucleotides to the genome. Bioinformatics, 24(20), 2395-2396.

[91] Malhis, N., Butterfield, Y. S., Ester, M., & Jones, S. J. (2009). Slider—maximum use of probability information for alignment of short sequence reads and SNP detection. Bioinformatics, 25(1), 6-13.

[92] Metzker, M. L. (2010). Sequencing technologies—the next generation. Nature Reviews Genetics, 11(1), 31-46.

[93] Flicek, P., & Birney, E. (2009). Sense from sequence reads: methods for alignment and assembly. Nature methods, 6, S6-S12.

[94] Dalca, A. V., & Brudno, M. (2010). Genome variation discovery with high-throughput sequencing data. Briefings in Bioinformatics, 11(1), 3-14.

[95] Koboldt, D. C., Chen, K., Wylie, T., Larson, D. E., McLellan, M. D., Mardis, E. R., ... & Ding, L. (2009). VarScan: variant detection in massively parallel sequencing of individual and pooled samples. Bioinformatics, 25(17), 2283-2285.

[96] Anson, E. L., & Myers, E. W. (1997). ReAligner: a program for refining DNA sequence multi-alignments. Journal of Computational Biology, 4(3), 369-383.

[97] Mortazavi, A., Williams, B. A., McCue, K., Schaeffer, L., & Wold, B. (2008). Mapping and quantifying mammalian transcriptomes by RNA-Seq. Nature methods, 5(7), 621-628.

[98] Hoffmann, S., Otto, C., Kurtz, S., Sharma, C. M., Khaitovich, P., Vogel, J., ... & Hackermüller, J. (2009). Fast mapping of short sequences with mismatches, insertions and deletions using index structures. PLoS computational biology,5(9), e1000502.

[99] Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., Jones, S. J., & Birol, I. (2009). ABySS: a parallel assembler for short read sequence data. Genome research, 19(6), 1117-1123.

[100] Trapnell, C., Pachter, L., & Salzberg, S. L. (2009). TopHat: discovering splice junctions with RNA-Seq. Bioinformatics, 25(9), 1105-1111.

УНИВЕРЗИТЕТ ,,Гоце Делчев“ – ШТИП
ФАКУЛТЕТ ЗА ИНФОРМАТИКА
Катедра за компјутерски технологии и интелигентни системи
ШТИП

м-р Доне Стојанов

АЛГОРИТМИ ЗА ПОРАМНУВАЊЕ И ПРЕБАРУВАЊЕ НА ДНК СЕКВЕНЦИ

- ДОКТОРСКИ ТРУД -

Штип, 2015