VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

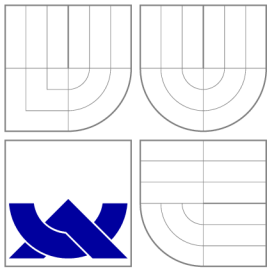
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZACE ALGORITMŮ PRO VYHLEDÁVÁNÍ TRIPLEXŮ V DNA

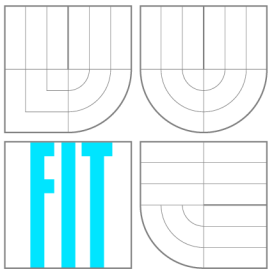
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JIŘÍ HON
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZACE ALGORITMŮ PRO VYHLEDÁVÁNÍ TRIPLEXŮ V DNA

OPTIMIZATION OF ALGORITHMS FOR TRIPLEX DETECTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JIŘÍ HON
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ MARTÍNEK, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Současné studie naznačují, že triplexy hrají důležitou roli v mechanismech regulace transkripce, rekombinace DNA a mutageneze a mají proto velký význam pro biologii, biotechnologii a medicínu. Tato bakalářská práce optimalizuje nedávno publikovaný algoritmus pro vyhledávání potenciálních *intramolekulárních* triplexů na třech úrovních návrhu: uživatelské rozhraní, využití paměti a výpočetní náročnost. V úrovni uživatelského rozhraní byl algoritmus rozšířen o existující vizualizační funkce a transformován do podoby balíčku pro prostředí R/Bioconductor. Optimalizací využití paměti a cache procesoru v kombinaci s redukcí výpočtu na základě analýzy jeho stavu bylo dosaženo více než trojnásobného zrychlení oproti původní implementaci.

Abstract

Triplex-forming DNA sequences have been implicated as important players in several key processes, such as transcriptional regulation, DNA recombination and mutagenesis, which emphasize their importance for biology, biotechnology and medicine. This bachelor thesis optimizes recently publicated dynamic programming algorithm for identification of triplex-forming sequences on three levels of design: user interface, memory usage and computation time. On the level of user interface, the algorithm was extended with existing visualization functions and rewritten into R/Bioconductor package. Memory usage optimization and processor cache analysis in combination with computation time reduction based on current computation state analysis lead to more than three times acceleration.

Klíčová slova

DNA, triplex, H-DNA, vyhledávání, identifikace, vizualizace, optimalizace, R, Bioconductor

Keywords

DNA, triplex, H-DNA, searching, identification, visualization, optimization, R, Bioconductor

Citace

Jiří Hon: Optimalizace algoritmů pro vyhledávání
triplexů v DNA, bakalářská práce, Brno, FIT VUT v Brně, 2013

Optimalizace algoritmů pro vyhledávání triplexů v DNA

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Martínka Ph.D.

.....
Jiří Hon
13. května 2013

Poděkování

Děkuji za pomoc a odborný přístup vedoucímu práce, panu Ing. Tomáši Martínkovi, Ph.D. Velice si vážím času, který mi věnoval při práci na tomto bakalářském projektu.

Obsah

Kapitola 1

Úvod

Analýza DNA a její anotace jsou důležitými kroky k porozumění molekulárním základům života, protože život, jak jej známe a pozorujeme na Zemi, vyplývá ze schopnosti buněk uchovávat, třídit a měnit dědičnou informaci uloženou v molekule DNA. Od objevení dvoušroubovicové struktury DNA v roce 1953 [27] byla pozornost vědců převážně soustředěna na zkoumání genetické informace jako předpisu pro tvorbu proteinů, avšak s příchodem nového tisíciletí se těžiště vědeckých studií posunulo od analýzy genů ke zkoumání mezigenové DNA a strukturních vlastností chromozomů a buněčného jádra. Ačkoliv je dvoušroubovicová DNA obecně přijímána jako kanonická struktura, existují i jiné způsoby uspořádání (A-DNA, Z-DNA), z nichž některé se vyskytují *in vivo*, jiné byly pouze uměle vytvořeny v laboratorních podmínkách. Z pohledu této práce jsou pozoruhodné zejména úseky DNA, které se v důsledku působení Watson-Crickových [27] a Hoogsteenových [14] vazeb skládají do tvaru trojšroubovice – zkráceně triplexu nebo též H-DNA.

Ve vědeckých studiích zkoumajících triplexovou strukturu DNA jsou patrné dva trendy. První vychází ze skutečnosti, že lze *in vitro* syntetizovat oligonukleotidový řetězec, který je schopen vytvořit s dvoušroubovicí DNA v živé buňce tzv. *intermolekulární* triplex. Tyto řetězce jsou známé pod zkratkou TFO (triplex-forming oligonucleotide) a mají obrovský přínos pro biotechnologii a medicínu, neboť jsou schopny selektivně ovlivňovat určitou oblast genomu. Mutageneze vyvolaná jejich navázáním může vést k trvalým dědičným změnám ve specifických genech nebo můžeme pomocí TFO cíleně deaktivovat určitý gen [24]. V současné době existují vhodné nástroje, které jsou schopny identifikovat místa v genomu, kam by se mohly potenciální TFO navázat [9]. Naproti tomu využití *intramolekulárních* triplexů není tak podrobně zmapováno. Některé studie však naznačují, že hrají důležitou roli nejen v mechanismech regulace transkripce, rekombinace DNA a mutageneze [10] ale i při vzniku vážných geneticky podmíněných onemocnění [8, 23].

Cílem této práce bylo identifikovat slabá místa konkrétního algoritmu pro vyhledávání *intramolekulárních* triplexů [19] a implementovat vhodný způsob jeho optimalizace. Pro pochopení problematiky bylo nutné nastudovat podrobné informace o stavbě triplexů, jiných možnostech jejich hledání a důkladně se seznámit s principem výpočtu zvoleného algoritmu, který je založen na metodě dynamického programování.

Na základě získaných znalostí a výstupů předchozích bakalářských a diplomových prací [18, 30, 22] byly navrženy optimalizace na třech úrovních: uživatelské rozhraní, využití paměti a výpočetní náročnost. Optimalizace na úrovni uživatelského rozhraní zahrnuje vytvoření balíčku pro aplikační prostředí R/Bioconductor, který integruje nejen vyhledávací algoritmus [19], ale zároveň i funkcionalitu potřebnou k vizualizaci [30, 22] nalezených triplexů. Na úrovni paměťových optimalizací byly prověřeny a opraveny části vyhledávacího

algoritmu, které jsou výpočetně náročné a zároveň intenzivním způsobem pracují s pamětí. Poslední úroveň optimalizací vychází ze způsobu, jakým probíhá vyhledávání, a parametrům, kterými je omezeno. Díky tomu můžeme výpočet některých částí vyhledávacího algoritmu vynechat, protože v prohledávané oblasti není možné naleznout dostatečně kvalitní triplex.

Text práce je rozdělen do několika tematických celků. Kapitola 2 shrnuje poznatky o struktuře DNA a triplexech, vysvětluje základní principy molekulární biologie a způsoby uložení informace v DNA sekvencích. V kapitole 4 nalezneme stručný přehled o prostředí R a projektu Bioconductor. Následující kapitola 5 kriticky shrnuje výchozí stav použitých algoritmů před vlastní implementací navržených optimalizací a poslední kapitola 6 osvětluje návrh a řešení konkrétních optimalizací a jejich dopad na výkon a použitelnost výchozích algoritmů.

Kapitola 2

Struktura DNA a triplexy

Podle předního světového virologa Erlinga Norrbyho můžeme ve zkratce definovat život jako samoreplikující se chemický systém, který je schopný dělat chyby [26]. Klíčovou strukturu, která tento proces replikace i vznik chyb v genetické informaci živých organismů umožňuje, nazýváme DNA (deoxyribonukleová kyselina). Proto se v této kapitole budeme zabývat nejen způsobem uložení dědičné informace v DNA sekvencích, ale i významem speciálních sekundárních struktur DNA, které hrají důležitou roli při vzniku chyb. Primárním zdrojem pro tuto kapitolu je publikace *Základy molekulární biologie* [1].

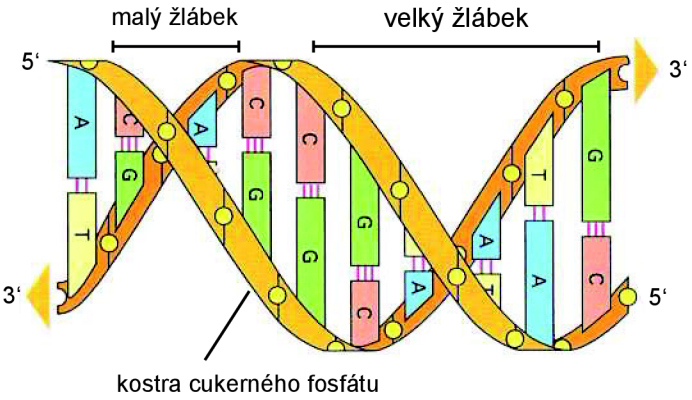
2.1 Deoxyribonukleová kyselina

Dědičná informace je v DNA uložena v podobě *genů* – základních jednotek, které určují vlastnosti jednotlivce i celého druhu. Ačkoliv samotná struktura DNA byla objevena až v roce 1953 [27], bylo již dříve známo, že genetická informace obsahuje instrukce převážně pro tvorbu proteinů (bílkovin), což jsou makromolekuly, které slouží jako stavební kameny buněčných struktur, mohou fungovat jako enzymy, zpětně regulují genovou expresi a umožňují buněčný pohyb i vzájemnou komunikaci. První důkazy, že geny jsou tvořeny právě DNA, pocházejí z roku 1944, kdy bylo prokázáno, že přidání purifikované DNA k bakteriím mění jejich vlastnosti a že tyto vlastnosti jsou předávány na další generace [3].

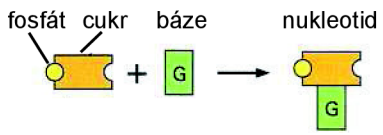
Molekula DNA je složena ze dvou polynukleotidových vláken složených ze čtyř typů nukleotidů. Obě tato vlákna jsou nazývána *řetězce DNA* nebo *vlákna DNA*, jsou vzájemně spojena vodíkovými můstky a stočena do tvaru dvoušroubovice – B-DNA (viz obrázek 2.1).

Jednotlivé nukleotidy jsou tvořeny pětiuhlíkovým sacharidem, na nějž je navázána dusíkatá báze a jedna nebo více fosfátových skupin (viz obrázek 2.2). V případě DNA se jedná o sacharid *deoxyribóza* a jednu fosfátovou skupinu. Bázi může představovat *adenin* (A), *cytosin* (C), *guanin* (G) nebo *thymin* (T). Nukleotidy jsou spojeny v řetězce kovalentními vazbami mezi sacharidy a fosfáty, které tak tvoří cukrfosfátovou kostru.

Tvar a chemická struktura bází umožňují efektivní tvorbu vodíkových můstků pouze mezi A-T a G-C, kde se atomy schopné tvořit vodíkové můstky dostanou blízko sebe bez narušení struktury dvoušroubovice, protože v tomto uspořádání mají oba páry bází podobnou šířku, která umožňuje udržení *stabilní* vzdálenosti cukrfosfátové kostry obou řetězců podél celé délky molekuly DNA. Tuto vlastnost nazýváme *komplementarita bází*. Díky ní jsme schopni pouze na základě znalosti sekvence jednoho vlákna určit i sekvenci vlákna komplementárního. Báze se mohou párovat pouze v případě, že jsou obě vlákna vůči sobě antiparalelní a tento způsob párování nazýváme *Watson-Crickovo* [27] (viz obrázek 2.3).



Obrázek 2.1: *Dvoušroubovice – B-DNA* – pevné kovalentní vazby cukrfosfátové kostry jsou znázorněny tím, jak do sebe jednotlivé části zapadají (kolečko reprezentuje fosfát a jamka patří k sacharidu). Obrázek respektuje dohodnutou konvenci pro označování konců jednotlivých řetězců a počet čar mezi jednotlivými komplementárními bázemi odpovídá počtu vodíkových můstků. Převzato z knihy [1].



Obrázek 2.2: *Stavba nukleotidu* – převzato z knihy [1].

2.2 Konvence

Konce jednotlivých řetězců DNA označujeme dle konvence podle toho, zda končí -OH skupinou respektive fosfátovou skupinou jako *3'* respektive *5'* konec.

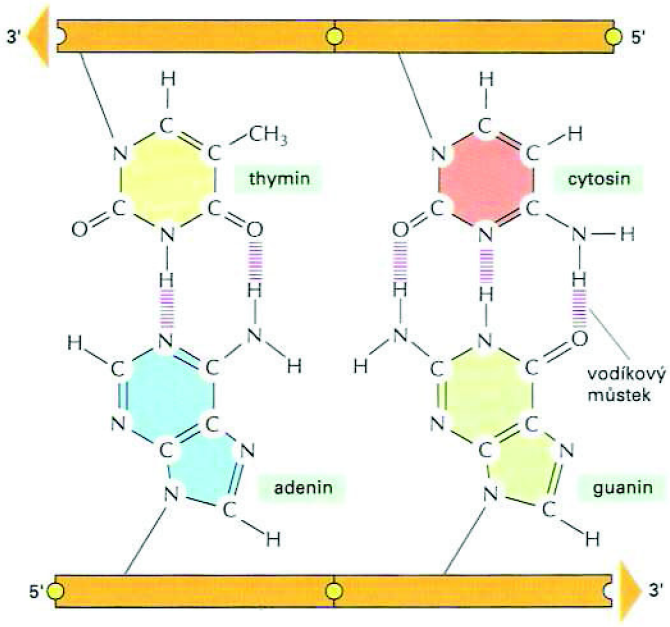
Ve dvoušroubovici DNA mají navzájem komplementární vlákna opačný směr, říkáme, že jsou tzv. antiparalelní a označujeme je jako + respektive −. Skutečnost, které vlákno označíme jako +, tedy přímé, určujeme podle směru, kterým probíhá replikace dané molekuly.

Pro potřeby výkladu v kapitole 2.4 jsou na obrázku 2.1 zakresleny oblasti tzv. velkého a malého žlábku. *Velký* žlábek může obsahovat skupiny donorů a akceptorů schopných vytvořit alternativní vodíkové vazby s další molekulou DNA (viz kapitola 2.4).

2.3 DNA jako informační nosič

Informace kódovaná ve struktuře DNA je dána pořadím nukleotidů v řetězci. Každou z bází – A, C, G a T – si můžeme představit jako jedno písmeno z čtyřpísmenné abecedy, která je používána k uchování biologické informace ve struktuře DNA. Živé organismy se navzájem liší sekvencemi nukleotidů a tím i biologickou informací.

Lineární sekvence nukleotidů v DNA je v rámci buněčných procesů překládána do lineární sekvence aminokyselin potřebných pro tvorbu proteinu, kdy jednu aminokyselinu



Obrázek 2.3: *Watson-Crickovo párování bází* – obě vlákna jsou vůči sobě antiparalelní. Mezi adeninem a thyminem vznikají dva vodíkové můstky, mezi guaninem a cytosinem můstky tři. Převzato z knihy [1].

kóduje sekvence tří po sobě jdoucích nukleotidových bází – tzv. *triplet*.

Dlouhou dobu se pozornost vědců soustředila na zkoumání genů jako hlavního zdroje biologické informace. V důsledku tak opomíjeli obrovské množství DNA, které ač pravděpodobně nekóduje žádnou známou bílkovinu, může přesto hrát významnou roli v životě buňky. Tato mezigenová DNA může mít vliv na regulaci transkripce, rekombinaci či mutagenezi. V současné době se ukazuje, že jsou biologicky zajímavé především ty úseky, jež jsou schopny tvořit alternativní struktury DNA [10].

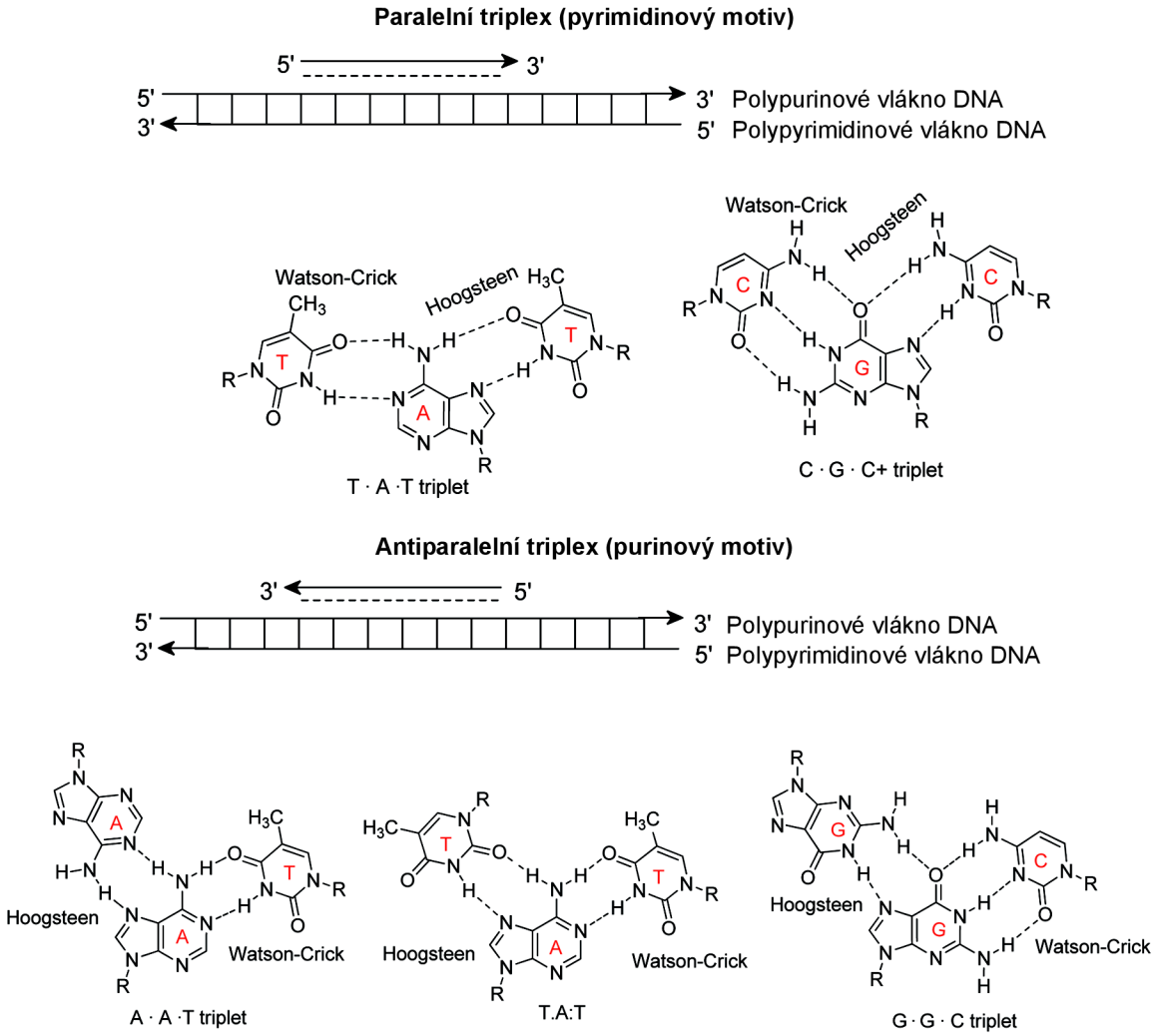
2.4 Triplexy

Tvar dvoušroubovice není jediným známým modelem struktury DNA. Watson a Crick před vydáním slavného článku, který jim později vynesl Nobelovu cenu [27], experimentovali i s modelem trojšroubovice. Definitivní návrh, jak by mohla vypadat trojšroubovicová struktura DNA (zkráceně *triplex*), předložili Felsenfeld, Davies a Rich v roce 1957 [12].

Vznik triplexů

Vznik triplexů souvisí s úseky DNA, které se vyznačují velkým zastoupením nukleotidů buďto s purinovou (A, G), nebo pyrimidinovou (C, T) bází. Takové úseky zkráceně nazýváme polypurinové potažmo polypyrimidinové.

Třetí vlákno se k standardní dvoušroubovici váže v oblasti velkého žlábku (viz obrázek 2.1) pomocí alternativních vodíkových vazeb, které byly podle svého objevitele nazvány jako Hoogsteenovy a reverzní Hoogsteenovy vazby [14]. Nejčastější kombinace Hoogsteenova a Watson-Crickova párování při tvorbě triplexů ilustruje obrázek 2.4.

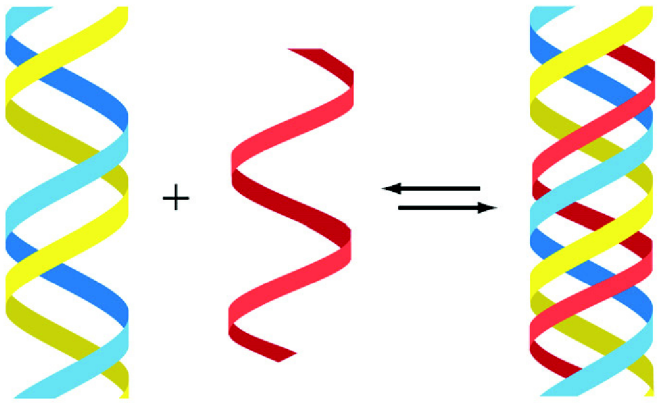


Obrázek 2.4: *Hoogsteenovo párování bází* – v pyrimidinovém motivu (Y) je třetí vlákno složené z pyrimidinových bází paralelně vázané k purinovému vláknu Hoogsteenovým párováním. V purinovém motivu (R) je třetí vlákno purinové a k purinovému vláknu dvoušroubovice je vázáno reverzním Hoogsteenovým párováním. Převzato z článku *New Approaches Toward Recognition of Nucleic Acid Triple Helices* [2].

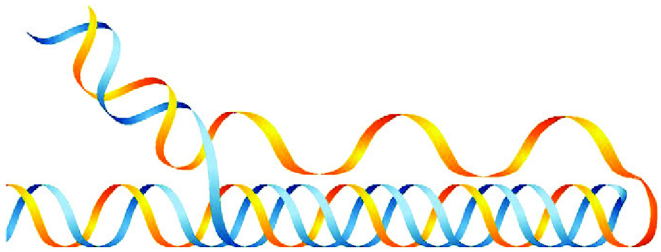
Dělení triplexů

Z hlediska původu třetího vlákna dělíme triplexy do těchto kategorií:

1. *Intermolekulární triplexy* – třetí vlákno pochází z jiné molekuly DNA (obrázek 2.5).
2. *Intramolekulární triplexy* – třetí vlákno pochází ze stejné molekuly DNA. Jsou založeny na dvoušroubovicovém modelu, kdy se navíc jedno z vláken odvine a znovu obtočí kolem dvoušroubovice (obrázek 2.6).
3. *Intravláknové triplexy* – speciální případ intramolekulárního triplexu. V tomto případě je celý triplex tvořen pouze jedním vláknem molekuly DNA (viz kapitola 3.1).



Obrázek 2.5: *Intermolekulární triplex* – převzato z článku *DNA triple helices: Biological consequences and therapeutic potential* [17].



Obrázek 2.6: *Intramolekulární triplex* – převzato z článku [17].

Stabilita triplexů

Ačkoliv vytvoření triplexu v laboratorních podmínkách nepředstavuje složitý problém, v jádře živé buňky existuje mnoho překážek, které musí třetí vlákno překonat, aby se spojilo v triplex. Může se stát, že vazebné místo je zrovna nedostupné kvůli chromatinové bariéře, problém může představovat nedostatečná afinita vazebného místa způsobená koncentrací solí v mezibuněčném prostoru a nevhodným pH. Nicméně experimentálně bylo zjištěno, že například purinový motiv je oproti pyrimidinovému na pH prostředí buňky relativně nezávislý [16].

Všechny známé algoritmy, které slouží k vyhledávání potenciálních triplexů, tyto netriviální vlivy buněčného prostředí na stabilitu triplexů neuvažují.

Využití triplexů

In vitro je možné syntetizovat oligonukleotidový řetězec, který je schopen vytvořit s dvoušroubovicí DNA v živé buňce *intermolekulární* triplex. Tyto řetězce jsou známé pod zkratkou TFO (triplex-forming oligonucleotide) a mají obrovský přínos pro biotechnologii a medicínu, neboť jsou schopny selektivně ovlivňovat určitou oblast genomu. TFO lze navíc snadno chemicky modifikovat, což umožňuje navázání různých sloučenin a v důsledku i cílené poškození popřípadě mutaci vybrané části genomu. Mutageneze vyvolaná jejich navázáním může vést k trvalým dědičným změnám ve specifických genech nebo můžeme pomocí TFO cíleně

deaktivovat určitý gen [24]. V současné době existují vhodné nástroje, které jsou schopny identifikovat místa v genomu, kam by se mohly potenciální TFO navázat [9].

Naproti tomu využití *intramolekulárních* triplexů není tak podrobně zmapováno. Některé studie však naznačují, že hrají důležitou roli nejen v mechanismech regulace transkripce, rekombinace DNA a mutageneze [10] ale i při vzniku vážných geneticky podmíněných onemocnění [8], například Friedreichovy ataxie [23].

Dále se v této práci budeme věnovat principům detekce a vizualizace pouze *intramolekulárních* triplexů.

Kapitola 3

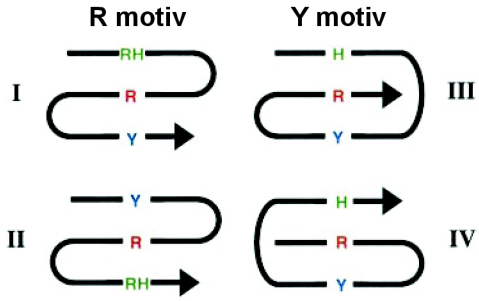
Existující přístupy pro detekci intramolekulárních triplexů

V této kapitole se seznámíme s dvěma rozdílnými algoritmickými přístupy pro detekci intramolekulárních triplexů. První z nich vychází z funkcionálního jazyka *Palingol* a druhý je založen na metodě *dynamického programování*. Text kapitoly navazuje na diplomové práce, které již na téma triplexů vznikly dříve [30, 28].

3.1 Řešení založené na jazyce Palingol

Programovací jazyk Palingol [6] vznikl za účelem popisu sekundárních struktur nukleových kyselin a jejich vyhledávání v databázi sekvencí. Sekundární struktura je plně popsána jako seřazený seznam vlásenek spolu se vzájemnými omezujícími podmínkami.

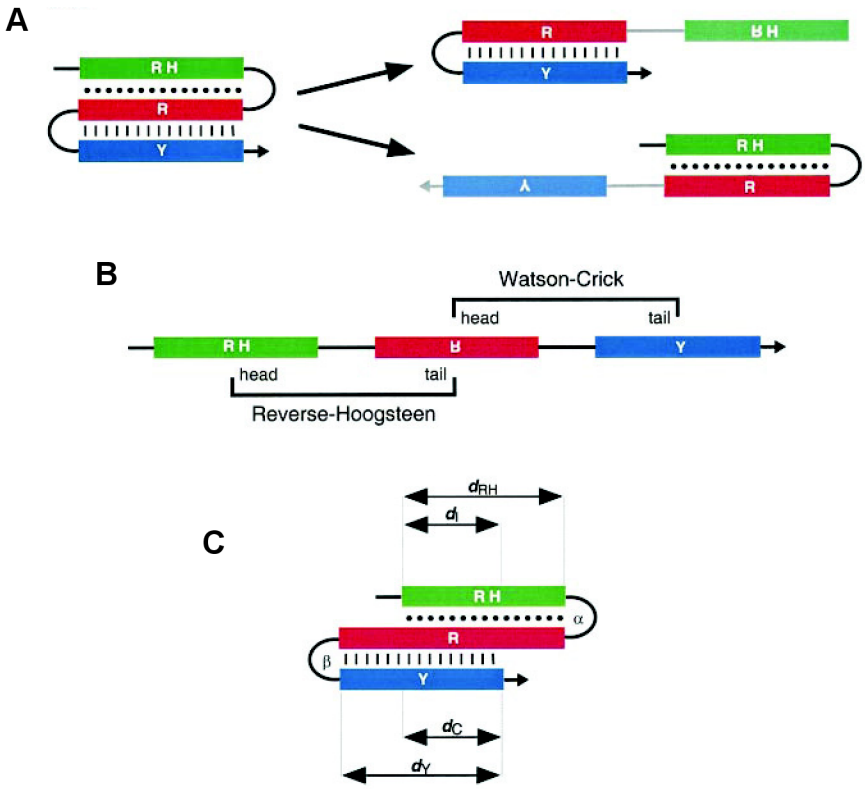
Řešení publikované v článku *Searching genomes for sequences with the potential to form intrastrand triple helices* [15] je založeno na popisu *intravláknového* triplexu jako dvou vlásenek, které sdílejí společné homopurinové vlákno (viz obrázek 3.1).



Obrázek 3.1: *Typy intravláknových triplexů* – orientace vláken čtyř tříd intravláknových triplexů. R – purin (červeně), Y – pyrimidin (modře), H – Hoogsteen (zeleně), RH – reverzní Hoogsteen (zeleně). Převzato z článku [15].

Proces hledání probíhá ve dvou krocích. V prvním kroku se identifikují dvě sady vlásenkových struktur. Každá vlásenka z první sady je ohodnocena odpovídajícím skóre a má-li takovou strukturu, aby nukleotidové páry v ní obsažené vyhovovaly Watsonovu-Crickovu párování bází, a zároveň přesahuje minimální zadanou délku, je uchována. Druhá sada vlásenek se liší v tom, že namísto Watsonova-Crickova párování je vyžadováno Hoogsteenovo

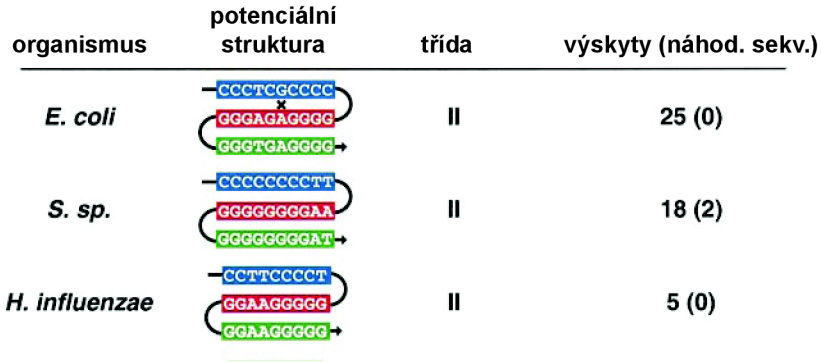
nebo reverzní Hoogsteenovo párování. Druhý krok spočívá v nalezení Watson-Crickových a Hoogsteenových vlásenek, které sdílejí stejné homopurinové vlákno. Taková struktura pravděpodobně má potenciál vytvořit triplex, nicméně aby ji algoritmus definitivně za takovou strukturu označil, musí ověřit další specifické podmínky, které byly zjištěny empiricky, například omezení pro délku těla a hlaviček vlásenek. Jednotlivé fáze algoritmu ilustruje obrázek 3.2.



Obrázek 3.2: *Vyhledávací strategie* – (A) v první fázi jsou zaznamenány všechny výskyty vlásenek vyhovujících Watson-Crickovu, Hoogsteenovu nebo reverznímu Hoogsteenovu párování. (B) v další fázi se identifikují místa kde Watson-Crickovy vlásenky sdílejí homopurinové vlákno s Hoogsteenovými vlásenkami. (C) ilustruje další omezení, která musí nalezený triplex splňovat: d_{RH} – oblast reverzního Hoogsteenova párování, d_y – pyrimidinová oblast, d_c – střední oblast a d_I – oblast průniku. Převzato z článku [15].

Autoři algoritmu analyzovali genomy bakterií *Escherichia coli*, *Synechocystis* a *Haemophilus influenzae* a v každém nalezli více rozptýlených výskytů stejné kopie konkrétní oblasti schopné vytvořit triplex – *PIsT* (potential intrastrand triplex). V každém genomu byly objeveny PIsT některé z uvedených tříd z obrázku 3.1. Tyto výsledky mohou být považovány za významné, protože v náhodných sekvencích se stejným poměrem obsahu nukleotidů nalezeny nebyly (viz obrázek 3.3).

Ačkoliv uvedený algoritmus vyhledává *intravláknové* triplexy, je možné stejným způsobem vyhledávat taktéž *intramolekulární* triplexy. De facto by se tím algoritmus zjednodušil, neboť by stačilo identifikovat pouze jednu sadu vlásenkových struktur odpovídajících Hoogsteenovu párování. Hlavní nevýhoda tohoto přístupu spočívá v tom, že nedovoluje prak-



Obrázek 3.3: *Výsledky hledání* – čísla v závorkách udávají počet výskytů nalezených v náhodné sekvenci se stejným poměrem obsahu nukleotidů. Převzato z článku [15].

ticky žádné strukturní nedokonalosti ve stavbě triplexu, přičemž některé studie ukazují, že i méně dokonalé triplexy jsou schopny vytvořit stabilní triplexovou strukturu [20].

3.2 Algoritmus na principu dynamického programování

Výpočetní postup publikovaný v článku *A dynamic programming algorithm for identification of triplex-forming sequences* [19] se oproti řešení založeném na jazyce Palingol [15] snaží respektovat strukturní nedokonalosti ve stavbě triplexu a zahrnout do procesu vyhledávání i další poznatky o stavbě triplexu.

Jedná se především o *neshodu v párování bází*, v jejímž důsledku není možný vznik silné Hoogsteenovy nebo reverzní Hoogsteenovy vazby, a *geometrickou neshodu* mezi triplety, které nespadají do stejné izomorfní skupiny. Geometrická neshoda zatěžuje cukrfosfátovou kostru třetího vlákna a zabraňuje vzniku vodíkových můstků.

Dále se v této práci budeme zabývat pouze tímto algoritmem a jeho optimalizací.

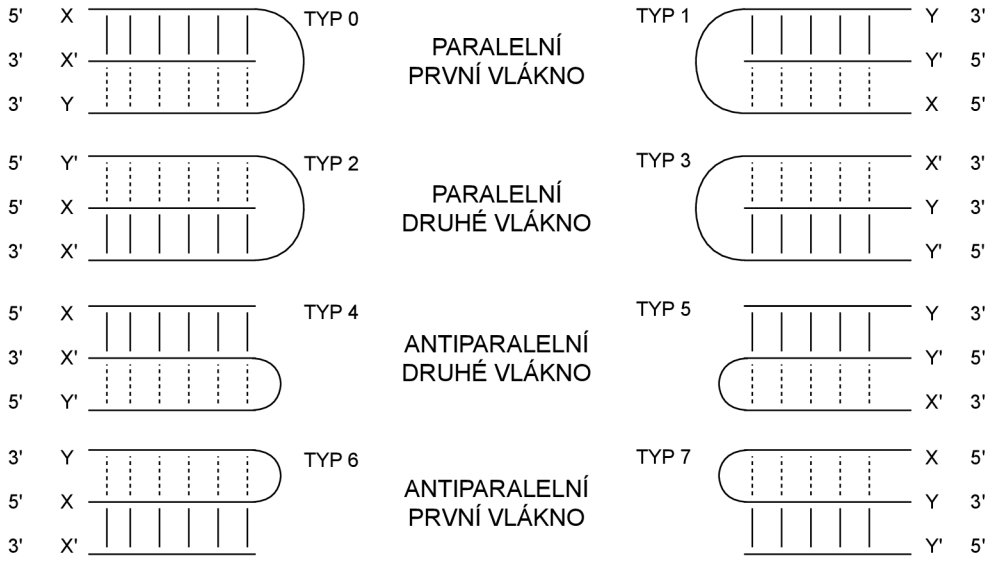
Dynamické programování

Jádro algoritmu vychází z matematické metody *dynamického programování* [5] uzpůsobené pro vyhledávání palindromů, neboť vlásenková struktura palindromů je podobná vlásenkové struktuře třetího vlákna triplexu, uplatňují se však jiná pravidla pro párování bází.

Metoda dynamického programování umožňuje redukovat problém nalezení všech triplexů na výpočet skóre v dvourozměrné matici, která je zkonstruována tak, aby sloupce reprezentovaly prohledávanou sekvenci a řádky tutéž sekvenci zapsanou pozpátku (viz obrázek 3.5). Výpočet začíná na hlavních antidiagonálách, jež jsou inicializovány nulovým skóre, a probíhá po jednotlivých antidiagonálách směrem k pravému dolnímu rohu matice. Skóre na každé pozici $[i, j]$ získáme jako maximum z následujících tří možností:

1. *Prodloužení těla triplexu* podél diagonály s přičtením bonusu za shodu nebo odečtením penalizace za neshodu.
2. *Vložení mezery do původní sekvence* s přičtením penalizace za inzerci.
3. *Vložení mezery do zrcadlové sekvence* s přičtením penalizace za inzerci.

Shoda respektive neshoda v párování se určuje na základě tabulek, které každému možnému tripletu přiřazují skóre, přičemž existuje osm různých skórovacích tabulek upravených pro různé typy triplexů, které ilustruje obrázek 3.4.



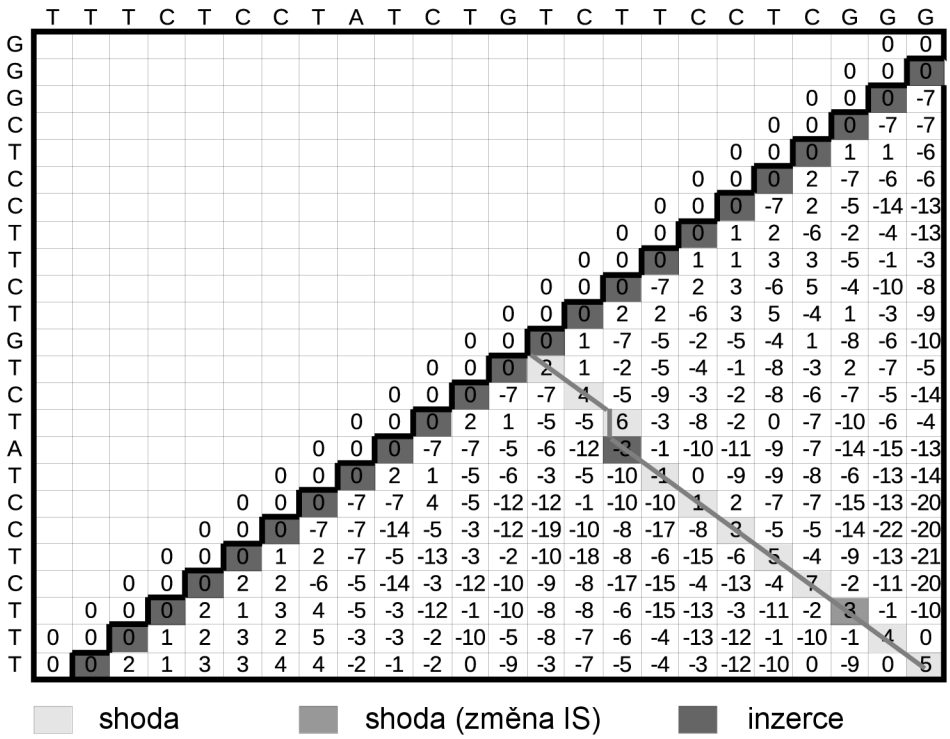
Obrázek 3.4: *Typy triplexů* – X a Y značí části stejného vlákna, které se účastní tvorby triplexu. Watson-Crickovo párování je znázorněno plnou čarou, Hoogsteenovo přerušovanou. Převzato z článku [19].

Aby celkové skóre nalezeného triplexu nebylo zhoršováno úvodní smyčkou triplexu (penalizacemi v důsledku neshody v párování), využívá algoritmus *kombinaci* lokálního a globálního zarovnání. Prvních $2 \cdot max\_loop$ antidiagonál je v duchu lokálního zarovnání korigováno skóre tak, aby nekleslo pod nulovou hodnotu. Parametr max_loop udává maximální velikost smyčky v počtu nukleotidů (viz obrázek 3.6). Na následujících antidiagonálách pokračuje výpočet v duchu globálního zarovnání a skóre tak může klesat do záporných hodnot.

Detekce významných výskytů

Výskyty nejkvalitnějších triplexů v matici dynamického programování můžeme rozpoznat podle vysokého skóre. Pro detekci takových úseků používá algoritmus podobnou techniku jako program BLAST [4] – jakmile skóre přesáhne zvolený práh (minimální skóre), je daná oblast označena za potenciální triplex. Další vývoj skóre je sledován a může průběžně dále růst nebo klesat, dokud opět neklesne pod zvolený práh. Jako výsledný triplex je označena oblast od začátku na hlavní antidiagonále až po lokální maximum ve sledované oblasti (viz obrázek 3.7).

Jeden z problémů, se kterým se musí tento postup vyrovnat, jsou falešné detekce oblastí s vysokým skóre v důsledku přenosu skóre ze sousedních diagonál. Z principu dynamického programování vyplývá, že v případě výskytu kvalitního triplexu přejímají sousední diagonály vysoké skóre snížené o penalizaci za inzerci. Algoritmus tyto případy detekuje a exportuje pouze hlavní výskyty a nikoliv jejich méně kvalitní sousední deriváty.



Obrázek 3.5: *Výpočetní matice dynamického programování* – zkratka IS značí isomorfní skupinu. Převzato z článku [19].

Časová a prostorová složitost

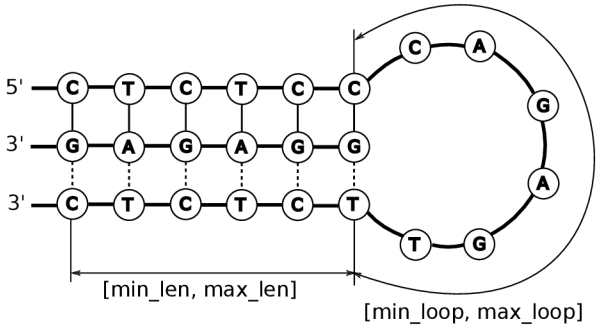
Výpočet trojúhelníkové oblasti trvá $n^2/2$ kroků, kde n reprezentuje délku prohledávané sekvence. Jestliže však analyzujeme reálné nebo náhodné sekvence, pravděpodobnost výskytu exponenciálně klesá s délkou triplexu. Proto algoritmus vypočítává pouze omezené množství antidiagonál podle zvolené maximální délky triplexu. Označíme-li maximální délku triplexu jako l, pak bude výsledná časová složitost algoritmu $O(2ln)$. Omezení počtu antidiagonál je důležitou vlastností pro následnou optimalizaci algoritmu navrženou v kapitole 6.

Vzhledem k závislostem při výpočtu matice dynamického programování, je třeba uchovávat hodnoty buněk pouze dvou posledních antidiagonál, proto prostorová složitost odpovídá $O(2n)$.

Parametry algoritmu

Pro snadnější pochopení problematiky řešené v kapitole 6 je uveden význam hlavních parametrů vyhledávacího algoritmu, které může uživatel nastavit. Především se jedná o tyto volby:

- *Typ triplexu* (`type`) – určuje, které typy triplexů se budou vyhledávat. Jednotlivé typy se značí podle obrázku 3.4.
- *Minimální skóre* (`min_score`) – hodnota prahu pro detekci oblastí s vysokým skóre (viz kapitola 3.2).



Obrázek 3.6: *Schéma délkových parametrů algoritmu.*

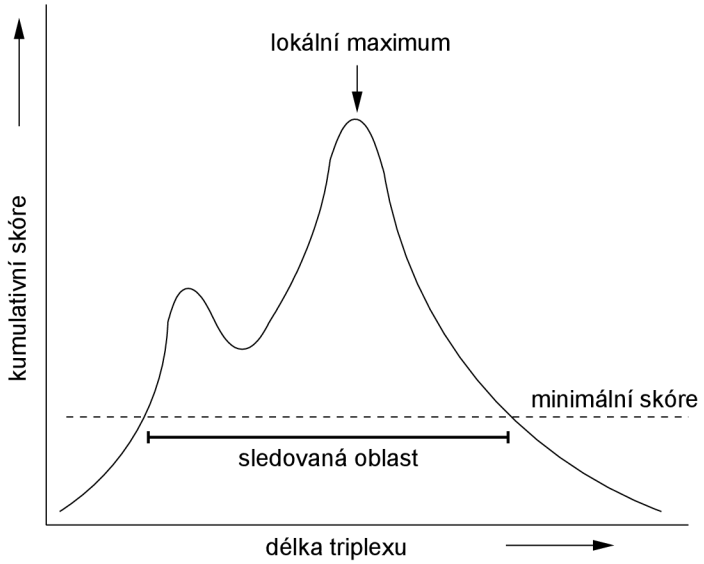
- *P-hodnota* (`p_value`) – umožňuje určit minimální požadovanou statistickou významnost nalezených výsledků.
- *Minimální délka triplexu* (`min_len`) – zdola omezuje délku těla triplexu. Všechna omezení na délku těla i smyčky triplexu ilustruje obrázek 3.6.
- *Maximální délka triplexu* (`max_len`) – shora omezuje délku těla triplexu a podílí se na určení maximálního počtu prohledávaných antidiagonál.
- *Minimální délka smyčky* (`min_loop`) – zdola omezuje délku smyčky triplexu a zároveň udává, kolik úvodních antidiagonál může algoritmus při výpočtu přeskočit.
- *Maximální délka smyčky* (`max_loop`) – shora omezuje délku smyčky triplexu a podílí se spolu s maximální délkou triplexu na omezení maximálního počtu prohledávaných antidiagonál.
- *Penalizace* – uživatel může nastavit některé penalizace, například penalizaci za neshodu, inzerci nebo změnu isomorfní skupiny.

Platnost výsledků

Výstup algoritmu byl ověřen na množině DNA sekvencí o délce 4,7 miliónu bází. Konkrétně se jednalo o genom bakterie *Escherichia coli*, poměrnou část 5. lidského genomu, dále jejich randomizované verze a taktéž randomizovanou sekvenci *E. coli* obohacenou o množinu experimentálně ověřených triplexů. Algoritmus přednostně identifikoval umístění známých triplexů, přestože byly do některých záměrně umístěny umělé inzerce.

Pro srovnání správnosti algoritmu byly výsledky analýzy genomu E. coli porovnány s výstupem algoritmu pro hledání intravláknových triplexů (viz kapitola 3.1). Takový postup je možný, protože z definice potenciální intravláknové triplexy (PIsT) vždy zahrnují potenciální intramolekulární triplex (PImT). Znamená to, že pokud nalezneme výskyty PIsT, musíme odhalit stejné pozice i algoritmem pro hledání PImT. Experimenty ověřily, že pro každý z 25 PIsT publikovaných v článku [15], byl algoritmus schopen identifikovat související intramolekulární triplexy.

Autoři algoritmu provedli taktéž pokus směřující k ověření biologické významnosti nalezených výsledků tak, že hledali souvislost výskytů PImT s pozicemi známých genů E. coli.



Obrázek 3.7: *Detekce oblastí s vysokým skóre* – sledovaná oblast, pro kterou se určuje lokální maximum, začíná vzrůstem skóre nad práh minimálního skóre a končí klesnutím skóre pod tentýž práh. Převzato z článku [19].

Zjistili, že PImT mají sklon se vyskytovat v oblastech od -50 do -160 relativně k umístění genů. Vzhledem k vysoké P-hodnotě, při které byl tento efekt ještě pozorovatelný, je pravděpodobnější, že tento jev nesouvisí ani tak s výskytem skutečných triplexů, jako spíš se společnou sekvenční charakteristikou triplexů a regulačních oblastí, které mívají palindromatickou povahu.

Kapitola 4

Prostředí R a Bioconductor

Cílem této kapitoly je v krátkosti představit programovací jazyk R a prostředí Bioconductor. Tyto pojmy jsou důležité pro následující text práce, který se věnuje optimalizacím zmíněného algoritmu [19] z hlediska uživatelského rozhraní (viz kapitola 6.1).

4.1 Jazyk R

R je open-source *programovací jazyk* a softwarové *prostředí* pro statistické výpočty a vizualizaci [25]. Vychází z jazyka S vytvořeného Johnem Chambersem v Bellových laboratořích a je v podstatě jednou z jeho implementací obohacenou o některé vlastnosti dalších programovacích jazyků. R vytvořili Ross Ithaka a Robert Gentleman a v současné době je vyvíjeno sdružením *R Development Core Team*. Své jméno získalo částečně podle iniciál křestních jmen autorů a částečně jako reakce na jméno jazyka, ze kterého vychází.

Z hlediska této bakalářské práce je podstatné především uživatelské prostředí jazyka R a způsob tvorby rozšiřujících balíčků. Je možné jej používat v tzv. *interaktivním* režimu stejně jako jiná známá výpočetní prostředí, například MATLAB nebo Python. R spolu s velkým množstvím rozšiřujících balíčků a různými grafickými prostředími tvoří ucelený výpočetní systém s integrovanou nápovědou. Jedno z nejrozšířenějších grafických prostředí pro jazyk R je známé pod jménem *RStudio*.

Rozšiřující balíčky je možné implementovat nejen v samotném jazyce R, ale i v jazycích C nebo Fortran formou *sdílených knihoven*. R pro tento účel poskytuje podrobné rozhraní k jeho vnitřním strukturám a umožňuje vytvářet efektivní implementace algoritmů v kompilovaných jazycích tak, aby byly přístupné v interaktivním prostředí.

4.2 Prostředí Bioconductor

Cílem projektu Bioconductor [13] je umožnit širší a efektivnější spolupráci na vývoji softwaru pro výpočetní biologii a bioinformatiku (VBB). Biologie, molekulární biologie především, v současné době prochází dvěma souvisejícími transformacemi. Za prvé roste povědomí o matematické podstatě mnoha biologických procesů a o tom, že lze pro řešení řady problémů s výhodou využít výpočetní a statistické modely. Za druhé rostoucí množství dostupných biologických informací – genomů a anotací – vytváří vyšší nároky na softwarové nástroje v každé fázi biologického výzkumu.

Bioconductor dlouhodobě vytváří flexibilní vývojové prostředí a odstraňuje překážky, které brání společnému vývoji a využívání nejnovějšího softwaru pro VBB. Ve své podstatě

se jedná o určitý typ open-source *repositáře*, který sdružuje softwarové a datové (anotační) balíčky pro prostředí R týkající se práce s biologickými daty.

Projekt vede již přes deset let stabilní tým vývojářů převážně z americké instituce *Fred Hutchinson Cancer Research Center*, kteří zároveň pracují na vývoji společných balíčků obsahujících algoritmy a třídy využitelné napříč celým Bioconductorem. Pro orientaci následuje stručný popis několika takových balíčků a tříd, z nichž některé byly zároveň využity při implementaci uživatelských optimalizací (viz kapitola 6.1).

1. *IRanges* – balíček nabízející efektivní nízkoúrovňové třídy pro ukládání celočíselných rozsahů, rovněž poskytuje implementace pro obecné operace nad rozsahy hodnot, jako jsou sjednocení, rozdíl, průnik apod. Z pohledu ekosystému Bioconductoru se jedná o pomyslný základní kámen, na kterém je založeno velké množství dalších balíčků.

2. *XString* – obecná třída z balíčku *Biostrings*, která usnadňuje manipulaci s rozsáhlými řetězci. Reálně využitelné jsou až její odvozeniny pro konkrétní účel, například pro reprezentaci sekvencí DNA (*DNAString*), RNA (*RNAString*) nebo řetězců aminokyselin (*AAString*).

3. *XStringViews* – opět se jedná o třídu z balíčku *Biostrings* kombinující možnosti balíčku *IRanges* a třídy *XString* za účelem uložení *pohledů* na dlouhé řetězce. Takový pohled na řetězec DNA může například reprezentovat geny nebo úseky tvořící triplexy a od od výše zmíněných rozsahů se liší především tím, že je vázán ke konkrétnímu řetězci (třídě *XString*).

4. *GRanges* – třída z balíčku *GenomicRanges*, která slouží k ukládání anotací. Každá anotace obsahuje název sekvence, interval, označení vlákna a případně další volitelné informace. Jelikož *GRanges* v podstatě reprezentuje formát *GFF3*, jsou součástí balíčku *GenomicRanges* i funkce pro obousměrný převod mezi nimi.

5. *BSgenome* – tento balíček tvoří společnou infrastrukturu pro uložení genomů konkrétních organismů. Bioconductor v současné době obsahuje velké množství datových balíčků vycházejících z *BSgenome*, které usnadňují přístup k velkému množství kompletních genomů z různých zdrojů, například z NCBI[1].

[1]National Center for Biotechnology Information – http://www.ncbi.nlm.nih.gov/

Kapitola 5

Kritika současného stavu

Optimalizace uvedené v následující kapitole 6 navazují na algoritmus dynamického programování pro vyhledávání triplexů v DNA [19] a výsledky několika předešlých bakalářských a diplomových prací [18, 22, 30]. Následující text kriticky shrnuje výchozí stav použitých algoritmů před vlastní implementací navržených optimalizací.

5.1 Výsledky předchozích prací

1. Článek *A dynamic programming algorithm for identification of triplex-forming sequences* [19] – výsledný vyhledávací algoritmus je dostupný ve formě programu v jazyce C. Vstupem je DNA sekvence ve FASTA formátu a výstupem formát GFF3.

2. Bakalářská práce *Rozšíření a optimalizace programu pro vyhledávání triplexů v DNA sekvencích* [18] – autor rozšířil výše uvedený algoritmus o kontrolu izomorfních skupin tripletů a implementoval základní paměťové optimalizace.

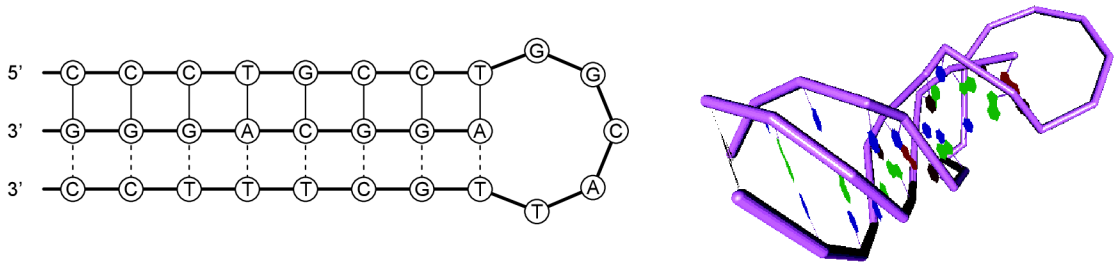
3. Diplomová práce *Vyhledávání triplexů v DNA sekvencích* [30] – výstupem je webové rozhraní pro vyhledávání triplexů a program v jazyce C umožňující *zpětné zarovnání* triplexů nalezených výše uvedeným algoritmem. Zpětné zarovnání umožňuje určit přesné párování jednotlivých tripletů, jehož znalost je nutným předpokladem pro případnou vizualizaci.

4. Bakalářská práce *Funkce pro manipulaci a vizualizaci molekulárních dat v prostředí R* [22] – autor práce implementoval sadu funkcí pro vizualizaci *intramolekulárních* triplexů v prostředí R umožňující mimo jiné zobrazení ve 2D a 3D prostoru (viz obrázek 5.1).

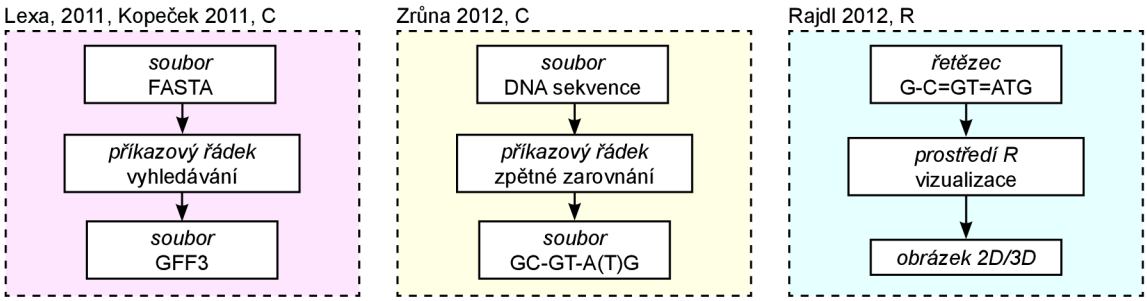
Nahlédneme-li na současný stav nástrojů pro detekci a vizualizaci triplexů pohledem cílového uživatele – *molekulárního biologa* – objevíme překážky, které brání jejich efektivnímu používání v *praxi*:

1. Nástroje jsou k dispozici pouze jako samostatné nespolupracující programy, každý se odlišně používá a některé vyžadují nemálo zkušeností s kompilací programů.

2. Nelze přímo aplikovat sekvenční postup: vyhledávání – zpětné zarovnání – vizualizace, neboť vstupy a výstupy jednotlivých fází nejsou vzájemně *kompatibilní*. Například výstupem vyhledávání je GFF3, kdežto vstupem zpětného zarovnání formát FASTA.



Obrázek 5.1: *Vizualizace triplexu ve 2D a 3D prostoru* – autorem funkcí pro vizualizaci triplexů v prostředí R je Kamil Rajdl [22].



Obrázek 5.2: *Schéma vstupů a výstupů výchozích aplikací* – jednotlivé bloky odpovídají existujícím aplikacím před započetím optimalizačních úprav. U každého z nich je uveden typ uživatelského rozhraní, implementační jazyk a formát vstupů a výstupů. Schéma demonstruje nemožnost přímého použití nástrojů v pořadí: vyhledávání – zpětné zarovnání – vizualizace. Pro srovnání viz obrázek 6.1.

3. Nelze s výsledky přímo pracovat a provádět nad nimi filtraci nebo statistické analýzy.

Z *ekonomického* hlediska je v současných nástrojích stále velký prostor pro snížení výpočetní náročnosti programů a tím i nezanedbatelného snížení finančních nákladů na výpočet nad dlouhými sekvencemi DNA. Pro představu v současné době trvá vyhledávání triplexů v celém lidském genomu (přibližně tři miliardy bází) od desítek minut k několika dnům v závislosti na parametrech a konfiguraci výpočetního stroje.

5.2 Nová koncepce

Cílem této práce je optimalizovat zmíněné programy na třech úrovních: uživatelské rozhraní (viz kapitola 6.1), využití paměti (viz kapitola 6.2) a výpočetní náročnost (viz kapitola 6.3) a tím vyřešit výše uvedené nedostatky.

Novou koncepcí se rozumí vytvoření optimalizovaného softwarového balíčku pro prostředí R/Bioconductor tak, aby integroval všechny uživatelské fáze: vyhledávání, zpětné zarovnání a vizualizaci. Díky tomu se publikované algoritmy stanou součástí rozsáhlého a dobře dokumentovaného softwarového ekosystému Bioconductoru a budou snadněji dostupné cílové skupině uživatelů.

Kapitola 6

Návrh a řešení optimalizací

V následujícím části prezentuji *vlastní přístup* k návrhu a implementaci optimalizací výchozích algoritmů uvedených v kapitole 5.1. Text je rozdělen na tři části odpovídající optimalizacím na různých úrovních návrhu.

6.1 Optimalizace uživatelského rozhraní

Cílem tohoto typu optimalizace je sjednocení zdrojových kódů pro vyhledávání, zpětné zarovnání a vizualizaci do podoby balíčku pro prostředí R/Bioconductor. Molekulární biologové – cílová uživatelská skupina – tak mohou plně využít možností rozsáhlého aplikačního prostředí, se kterým jsou zvyklí pracovat, a bez překážek nad výsledky inovativního algoritmu provádět další analýzy a vytvářet souvislosti s existujícími anotacemi.

Pro potřebu tvorby rozšiřujících balíčků poskytuje tým vývojářů jazyka R podrobnou technickou dokumentaci [29], která předepisuje adresářovou strukturu balíčku (viz příloha A) a techniky pro manipulaci s datovými typy R v jazyce C popřípadě Fortran [21]. Jejich konkrétní popis neuvádím, neboť není pro text této práce podstatný. Další požadavky na obsah balíčku – především na kvalitu dokumentace a využívání existující infrastruktury – kladou vývojáři Bioconductoru [7].

Kompatibilita vstupů a výstupů

Programy pro vyhledávání, zpětné zarovnání a vizualizaci triplexů byly vyvíjeny více méně nezávisle na sobě, což se nepříznivě podepsalo na vzájemné kompatibilitě vstupů a výstupů. Aby mohl vzniknout integrovaný balíček, bylo nutné v první fázi sjednotit výstup vyhledávání se vstupem zarovnání a v druhé fázi výstup zarovnání se vstupem vizualizace. Pro ilustraci problému je uvedena odlišnost formátů v druhé fázi. Písmena T_i značí nukleotidy (A, C, G nebo T), které tvoří tělo triplexu, přičemž mezi T_i a T_i' je vazba na základě Hoogsteenova nebo reverzního Hoogsteenova párování. Písmena S_i značí nepárované nukleotidy tvořící smyčku triplexu.

$$T_1T_2(T_3)T_4 - S_1S_2S_3 - T_4'T_2'T_1' \tag{6.1}$$

$$T_1T_2T_3T_4 = S_1S_2S_3 = T_4' - T_2'T_1' \tag{6.2}$$

Zápis 6.1 ilustruje syntax výstupu zarovnání [30] a zápis 6.2 popisuje formát vstupu vizualizačních funkcí [22]. Liší se za prvé v tom, jakým symbolem označují začátek a konec

smyčky triplexu (pomlčka oproti rovnítku) a za druhé tím, jak zapisují inzerce respektive mezery v těle triplexu. Ze dvou možných variant bylo zvoleno sjednocení výstupu zarovnání tak, aby vyhovoval vstupu vizualizačních funkcí, protože tento lépe odpovídá zavedeným konvencím v reprezentaci zarovnání.

Balíček *triplex* pro prostředí R/Bioconductor

Výsledný balíček s názvem *triplex* je navržen jako kombinace kódu v jazycích R a C, kde R většinou plní funkci *rozhraní* k procedurám v jazyce C[1] – zdrojové kódy v jazyce C jsou během instalace zkompilovány do podoby sdílené knihovny, která je pak za běhu připojena k prostředí R.

Během začleňování existujícího kódu bylo nezbytné učinit rozhodnutí, které části existujících programů pro vyhledávání a zpětné zarovnání zůstanou implementovány v C a které se naopak přesunou do prostředí R:

1. Vzhledem k tomu, že prostředí Bioconductor poskytuje vhodné balíčky pro načítání vstupní DNA sekvence v různých formátech, byla tato funkcionalita z původních algoritmů odstraněna a příslušné úvodní části algoritmů upraveny tak, aby dokázaly dekódovat sekvence uložené v objektu třídy *DNAString* (viz kapitola 4.2).

2. Výstup původních algoritmů směřuje do souboru (formát GFF3), což by výrazně komplikovalo další práci s výsledky v interaktivním prostředí. Proto byla výstupní část obou algoritmů přepsána a nově vrací výsledky v podobě vektorů – základních datových typů jazyka R.

3. Výsledky vyhledávání ovšem nejsou uživateli k dispozici pouze jako vektory číselných hodnot (vektory začátků a konců triplexů, vektor skóre apod.) ale pro jejich reprezentaci a textovou vizualizaci byla v jazyce R vytvořena třída *TriplexViews*. Ta dědí základní část funkcionality od třídy *XStringViews* a nalezené výskyty tak můžeme chápat jako pohled na vstupní sekvenci omezený vektory začátečních a koncových pozic triplexů. *TriplexViews* přidává další informace o jednotlivých položkách pohledu – triplexech (skóre, označení vlákna, P-hodnotu, počet inzercí atd.) a reimplementuje textový výpis objektu pro snadnou orientaci ve výsledcích.

Následuje stručný přehled výsledných rozhraní v jazyce R, s nimiž může koncový uživatel pracovat:

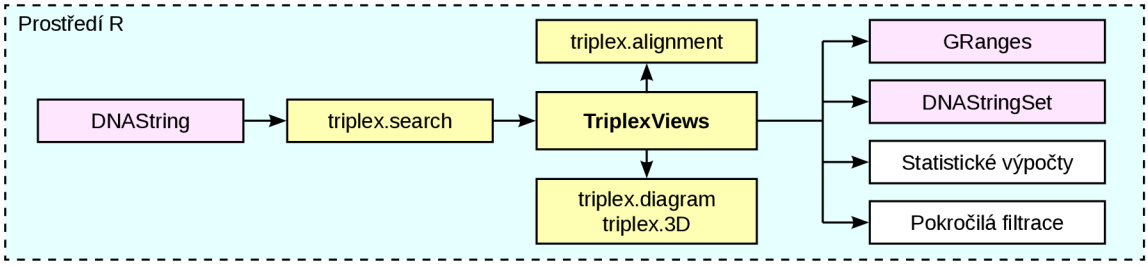
- `triplex.search` – rozhraní pro vyhledávání triplexů. Vstupem je objekt třídy *DNAString*, umožňuje nastavovat parametry algoritmu (viz kapitola 3.2) a vrací výsledky reprezentované třídou *TriplexViews*.

- `triplex.alignment` – rozhraní pro zarovnání triplexů. Výstupem je textová reprezentace zarovnání a na vstupu požaduje objekt třídy *TriplexViews*, protože jsou v něm uloženy parametry s jakými byly triplexy původně vyhledány. Bez jejich znalosti by zpětné zarovnání mohlo proběhnout způsobem neodpovídajícím zvoleným parametrům (délka smyčky by mohla být menší než zadaná minimální délka apod.).

- `triplex.diagram` a `triplex.3D` – vizualizační funkce. Na vstupu požadují objekt třídy *TriplexViews*, přičemž před samotnou vizualizací interně proběhne zarovnání

[1] Podobný přístup můžeme pozorovat i v jiných interpretovaných programovacích jazycích – např. Python.

sekvence. Výstupem je grafická reprezentace a v případě zobrazení v podobě 2D diagramu je možné zvolit výstupní zařízení (X11, PDF, PNG, apod.).



Obrázek 6.1: *Uživatelské schéma výsledného balíčku pro prostředí R/Bioconductor* – žlutě jsou označeny entity pocházející ze samotného balíčku *triplex*, červeně zvýrazněny jsou existující objekty z jiných balíčků Bioconductoru. Schéma naznačuje možnost přímé cesty od vyhledávání k vizualizaci díky objektu *TriplexViews*. Pro srovnání s původním stavem viz obrázek 5.2.

Nové možnosti

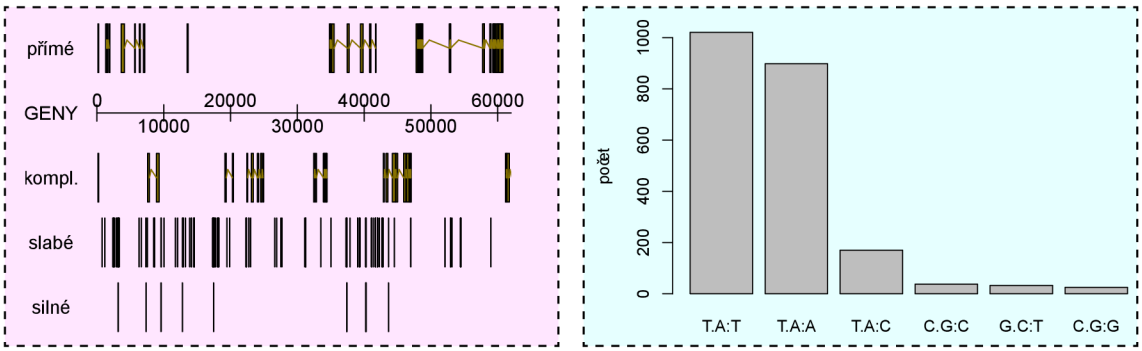
Transformace algoritmů pro vyhledávání, zpětné zarovnání a vizualizaci do podoby jednotného balíčku přináší nové možnosti užitečné pro koncové uživatele – především možnost přímé práce s výsledky vyhledávání – ať už se jedná o filtraci dle různých kritérií (skóre, P-hodnota, pozice, apod.) nebo statistické výpočty (např. procentuální podíl jednotlivých tripletů na tvorbě triplexu).

Následující příklad v prostředí R demonstruje jednoduchost postupu vyhledávání triplexů v chromozomu X organismu *Caenorhabditis elegans* a následné filtrace triplexů s největším skóre. Příklad také ilustruje způsob reprezentace výsledků vyhledávání třídou *TriplexViews* a její textovou vizualizaci. Složitější ukázka použití balíčku je prezentována v příloze C.

```
t <- triplex.search(Celegans$chrX, min_score=17)
t
  Triplex views on a 17718866-letter DNAString subject
subject: CTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCT...GCTTAGGCTTAGGCTTAGGCTTAGGCTTAGG
triplexes:
           start width score  pvalue ins type s
    [1]      762    28    17 6.5e-04   0    4 - [TTAGAAAAAAA...TCTAAAAGACA]
    [2]     1160    26    17 3.7e-04   0    7 + [ACAAAAACTTC...ACAAGAAAAAA]
    ...      ...   ...   ...      ... ...  ... ...
[20033] 17715172    29    17 3.7e-04   0    6 + [AAAAAAAAGTG...CTGAATTTCAT]
[20034] 17718247    27    17 3.7e-04   0    6 + [AAAAAAAAACA...ACATAAAACTA]

t[score(t) == max(score(t))]
  Triplex views on a 17718866-letter DNAString subject
subject: CTAAGCCTAAGCCTAAGCCTAAGCCTAAGCCT...GCTTAGGCTTAGGCTTAGGCTTAGGCTTAGG
triplexes:
```

```
        start width score  pvalue ins type s
  [1]    97738    30    26 1.6e-06   0    5 - [TAAATAATTTTT...AAAAAAAAAAAA]
  [2]   430917    30    26 6.3e-07   0    2 - [TTTTTTTTTTTT...TTTTTTTTTTTT]
  ...      ...   ...   ...     ... ...  ... ...
[171] 17292965    30    26 1.6e-06   0    5 - [TAAAAAATAAAA...AAAAAAAAAAAA]
[172] 17368050    30    26 1.6e-06   0    5 - [AAAAAATTAAAT...AAAAAAAAAAAA]
```



Obrázek 6.2: *Možnosti použití balíčku – levá* část obrázku dává do souvislosti pozice genů s výskytem potenciálního triplexu na chromozomu *X* organismu *C. elegans*. Pomocí balíku *GenomeGraph* jsou vykresleny stopy genů na přímém i reverzním vlákně. Stopa pro triplexy s dosaženým skóre v rozsahu 17–20 bodů je označena jako *slabé*. Triplexy se skóre větším než 20 bodů jsou znázorněny ve stopě *silné* (viz příloha C). *Pravá* část obrázku zobrazuje výstup analýzy četnosti výskytu tripletů tvořících nalezené triplexy na tomtéž chromozomu X organismu C. elegans.

6.2 Paměťové optimalizace

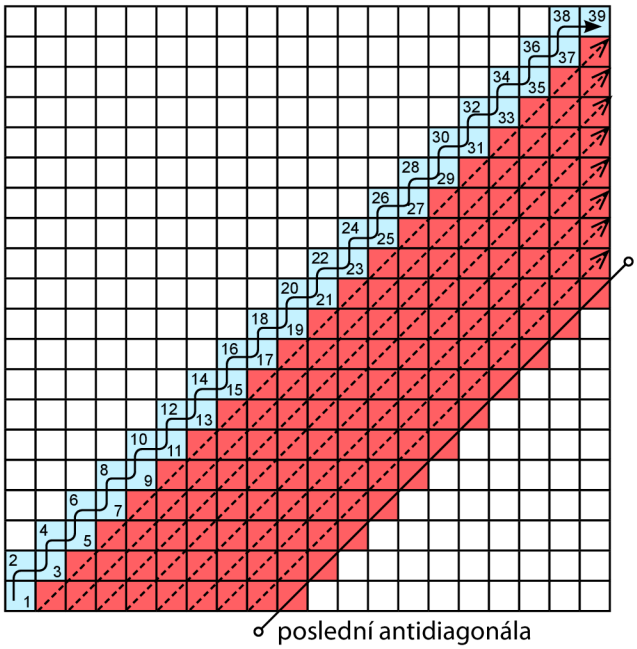
V úrovni paměťových optimalizací byly prověřeny části vyhledávacího algoritmu, které jsou výpočetně náročné a zároveň intenzivním způsobem pracují s pamětí. První prostor pro optimalizaci byl v tzv. *úzkém hrdle* programu – ve funkci `get_max_score`. Ta se propočítává pro každou buňku lichoběžníkové části matice dynamického programování (MDP) – konkrétně se jedná o $2ln$ volání, kde n je přibližně dvojnásobek maximální délky hledaných triplexů a l odpovídá délce vstupní DNA sekvence (viz kapitola 3.2).

Paměťová neefektivita spočívala v principu předávání parametrů k vyhodnocení a ve způsobu ukládání výsledků. Funkce `get_max_score` vypočítává podle principu dynamického programování maximální skóre ze tří možností a potřebuje k tomu znát hodnoty tří sousedních buněk (vlevo, vlevo nad a nad). Hodnota každé buňky je reprezentována strukturou `t_diag`, která v operační paměti zabírá oblast 28 bajtů (bez případného zarovnání). Problém byl v tom, že všechny tři sousední buňky byly do funkce předávány *hodnotou* stejně jako návratová hodnota funkce, což vyžadovalo zbytečné kopírování paměti. Vhodnou úpravou těla funkce bylo dosaženo minimalizování těchto přesunů a zkrácení doby výpočtu přibližně o čtvrtinu.

Procesorová cache

Další prostor pro paměťové optimalizace nabízí analýza využívání procesorových cache pamětí v průběhu algoritmu vyhledávání. Současné procesorové čipy obvykle disponují hned několika úrovněmi cache pamětí, přičemž pro algoritmus vyhledávání je podstatná především poslední úroveň (LL) cache, která maskuje přístupy do fyzické operační paměti. Z analýzy původního algoritmu vyhledávání simulačním nástrojem *cachegrind*[2] vyplynulo, že v LL cache dochází k výpadkům při čtení i zápisu. Ačkoliv se jednalo o hodnoty v řádu desetin procenta (0,6 %), následující experimenty ukázaly, že i takto malý počet výpadků má výkonnostní vliv na průběh výpočtu.

Jako zdroj výpadků byl identifikován způsob, jakým probíhá výpočet MDP. Hodnoty jednotlivých buněk se propočítávají po jednotlivých antidiagonálách a během výpočtu si algoritmus uchovává hodnoty jen posledních dvou antidiagonál, které jsou uloženy v jediném lineárním poli struktur `t_diag` (viz obrázek 6.3). Znamená to, že když je vstupní sekvence DNA příliš dlouhá, zabírá pole pro uložení stavu výpočtu velkou oblast paměti, která se nevejde do LL cache. Při cyklickém průchodu přes tuto oblast (výpočtu dalších antidiagonál) pak dochází k výpadkům cache.

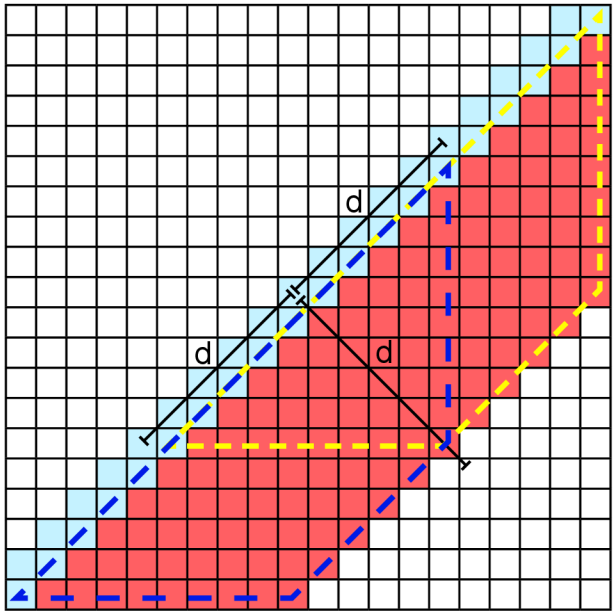


Obrázek 6.3: *Směr výpočtu buněk na antidiagonálách* – modrá políčka označují první dvě hlavní antidiagonály, hodnoty červených buněk musí algoritmus vypočítat postupně ve směru šipek. Klikatá čára naznačuje pořadí v jakém jsou uloženy vždy dvě poslední antidiagonály v jediném lineárním poli struktur `t_diag` a čísla uvedená v buňkách reprezentují diagonální čísla, jež jsou rovna indexu do zmíněného lineárního pole.

Nutno podotknout, že původní algoritmus vyhledávání je vybaven mechanismem dělení výpočtu na části. Je-li vstupní sekvence delší než zvolená konstanta M, rozdělí se výpočet na části s daným minimálním překryvem a proběhne nezávisle pro každou část (viz obrázek

[2]Součást projektu *valgrind*, viz http://valgrind.org/docs/manual/cg-manual.html

6.4). Mechanismus byl primárně vyvinut proto, abychom byli schopni prohledávat dlouhé sekvence DNA, u nichž bychom pro nedostatek operační paměti nebyli schopni uložit kompletní stav výpočtu – $2ln$ buněk matice, ale vhodným snížením konstanty M bychom mohli optimalizovat i využívání cache procesoru.



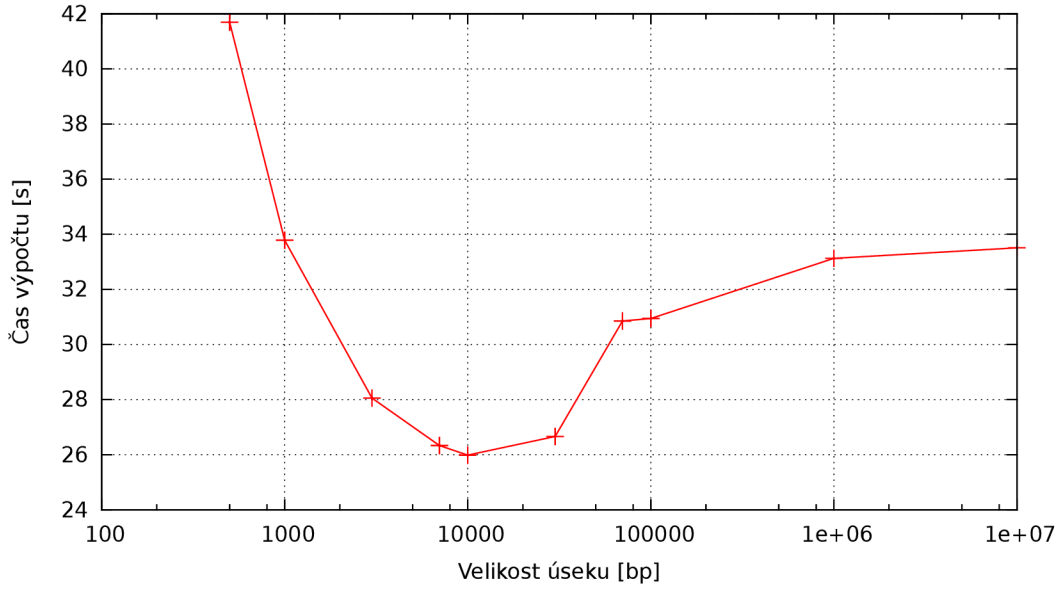
Obrázek 6.4: *Dělení výpočtu na části* – obrázek ilustruje vztah mezi hloubkou výpočtu (počtem antidiagonál) a minimálním nutným překryvem. V tomto případě je výpočet rozdělen na dvě stejné části zvýrazněné žlutou respektive modrou přerušovanou čarou s minimální velikostí překryvu $2d$ diagonál, kde d je celkový počet antidiagonál. Kdyby byl překryv mezi částmi menší než $2d$, mohlo by se stát, že algoritmus nedetekuje některé potenciální výskyty triplexů na hranici mezi dvěma částmi.

Vlastní měření

Pro zjištění vhodné velikosti úseku M bylo provedeno měření[3] na exponenciálním rozsahu hodnot. Z výsledků vyplývá, že optimální hodnota M je přibližně rovna 10000 bp. Naměřenou závislost doby výpočtu na velikosti úseku zobrazuje graf 6.5.

Hodnoty jsou ale platné pouze pro počítače se stejnou nebo větší velikostí cache, než jaká byla dostupná na měřícím stroji (2 MB) a navíc se může lišit pro různě zvolenou hloubku prohledávání (maximální délku triplexu), protože na ní závisí režie dvojitého výpočtu překryvů mezi jednotlivými úseky. V reálných případech použití se ale hloubka prohledávání pohybuje v rozsahu 10–100, což je vzhledem ke zvolené velikosti úseku stále procentuálně zanedbatelná část. Nízká hodnota M navíc přináší podstatné snížení paměťových nároků – místo více než 100 MB paměti pro uložení stavu vyžaduje algoritmus po provedení optimalizace setinový prostor.

[3]Použitá konfigurace: operační systém Linux (64bit), procesor Intel Core 2 Duo T5600 (2 MB LL cache) a 4 GB operační paměti. Stejné údaje platí i pro všechna následující měření.



Obrázek 6.5: *Závislost doby výpočtu na velikosti prohledávaného úseku* – měření bylo provedeno na části pátého lidského chromozomu s výchozími parametry vyhledávacího algoritmu. Osa x je v *logaritmickém* měřítku. Měření bylo nejprve realizováno pro velikosti úseku o mocninách čísla deset a následně byla podrobněji proměřena oblast kolem minima.

6.3 Výpočetní optimalizace

Díky způsobu, jakým probíhá vyhledávání, a parametrům, kterými je omezeno, je ve skutečnosti možné vynechat výpočet některých částí vstupní sekvence, v nichž už není možné naleznout dostatečně kvalitní triplex. Problém spočívá pouze ve vhodném a včasném určení takových úseků a způsobu jejich vyloučení z dalšího výpočtu.

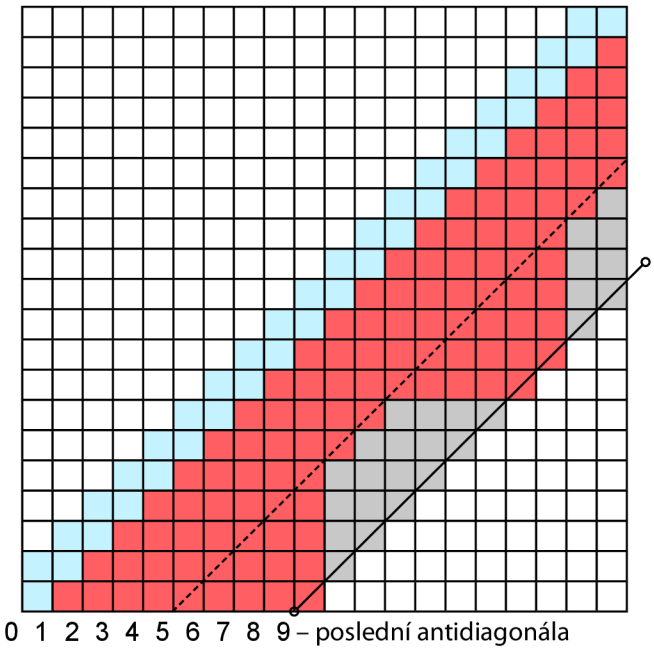
Jak bylo zmíněno již dříve v této práci, výpočet MDP probíhá po jednotlivých antidiagonálách. Je-li však určeno minimální skóre, kterého musí dosáhnout potenciální triplex (viz parametr `min_score`) a maximální počet antidiagonál, které se propočítávají (odvozeno z parametrů `max_len` a `max_loop`), můžeme algoritmus ve vhodnou chvíli zastavit, analyzovat poslední dvě antidiagonály a omezit další výpočet pouze na úseky, které mají do konce výpočtu (na poslední antidiagonále) šanci dosáhnout aspoň minimálního požadovaného skóre. Uvedený princip ilustruje obrázek 6.6.

Prahové skóre

Abychom dokázali vyloučit nepotřebné úseky na každé antidiagonále $i \in \langle 0, n-1 \rangle$, kde n reprezentuje celkový počet antidiagonál, musíme být schopni pro každou z nich analyticky určit minimální prahovou hodnotu skóre P_i, pro kterou má ještě smysl pokračovat ve výpočtu. Na základě znalosti průběhu výpočtu a jeho omezení lze pro tento účel odvodit analytický vzorec 6.3.

$$P_i = S_{min} - B_{max}\frac{n - i\ + 1}{2} \tag{6.3}$$

kde S_{min} je hodnota minimálního požadovaného skóre triplexu a B_{max} maximální možný



Obrázek 6.6: *Optimalizační strategie* – po výpočtu antidiagonály číslo 5 byl algoritmus zastaven a na základě analýzy hodnot dvou předchozích antidiagonál (4 a 5) omezen pouze na oblasti antidiagonály, které mají dostatečné skóre. *Světle modrou* barvou jsou zvýrazněny buňky na prvních dvou hlavních antidiagonálách, *červená* označuje buňky, které je nutné v průběhu vyhledávání spočítat a *šedá* hodnoty, které optimalizační mechanismus na základě analýzy antidiagonál vyloučil z výpočtu.

bonus za shodu, tedy přechod v matici dynamického programování v diagonálním směru. Dělení dvěma je realizováno celočíselným způsobem.

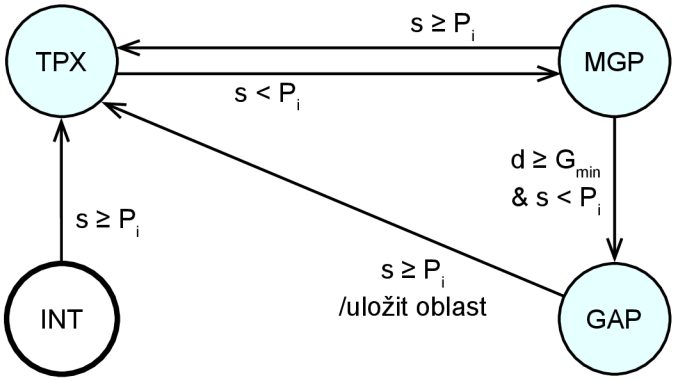
Slovně vyjádřeno se práh P_i rovná rozdílu minimálního skóre a maximálního možného přírůstku skóre do konce výpočtu, což znamená, že pokud je skóre buňky menší, než práh P_i, není možné, aby do konce výpočtu v diagonálním směru vzrostlo na minimální skóre i přes samé shody.

Analýza antidiagonál

Pomocí prahové hodnoty P_i sice dokážeme rozlišit, které *buňky antidiagonály*, či přesněji řečeno které *diagonály* můžeme z dalšího výpočtu vynechat, ale kvůli závislostem v matici dynamického programování na hodnotách třech předcházejících buněk si nemůžeme dovolit vyloučit každou jednotlivou diagonálu. Musíme je sdružit do oblastí, jež jsou minimálně tak dlouhé, jako je dvojnásobek překryvu potřebného ke splnění závislostí.

Proto jsem navrhl konečných automat 6.7, jehož vstupní sekvencí jsou dvě poslední antidiagonály uložené v jediném lineárním poli struktur `t_diag`. Výstupem jsou oblasti (intervaly), v nichž *můžeme* stále najít dostatečně kvalitní triplex a má tedy smysl je dopočítat. Implementaci automatu naleznete ve funkci `get_triplex_regions`.

Za pozornost stojí způsob, jakým jsou výsledné intervaly uloženy. Jedná se o celočíselné vyjádření začátku a konce, ovšem nikoliv indexem první a poslední *diagonály* oblasti, ale



Obrázek 6.7: *Konečný automat pro analýzu antidiagonál* – počáteční stav INT je zvýrazněn silnějším tahem, všechny koncové stavy automatu mají namodralou barvu. s – skóre aktuální buňky, P_i – prahové skóre pro antidiagonálu i, d – aktuální délka oblasti s nízkým skóre, G_{min} – minimální délka oblasti s nízkým skóre.

indexem řádku MDP (počítáno směrem od posledního řádku k prvnímu) a indexem sloupce MDP (viz obrázek 6.8). Tento způsob byl zvolen proto, jelikož vizuálně lépe odpovídá šíření závislostí v MDP při přechodu na další antidiagonály a zároveň není problém na základě znalosti aktuálního čísla antidiagonály dopočítat konkrétní diagonální číslo. Pro snadnější porozumění doporučuji znovu prohlédnout také obrázek 6.3.

Nevyřešenou otázkou zůstává, kdy a jak často spouštět analýzu antidiagonál, protože ačkoliv je konečný automat 6.7 poměrně jednoduchý, jeho výpočetní režie je příliš vysoká na to, aby se vyplatilo spouštět analýzu při každém přechodu na další antidiagonálu.

Spuštění první analýzy

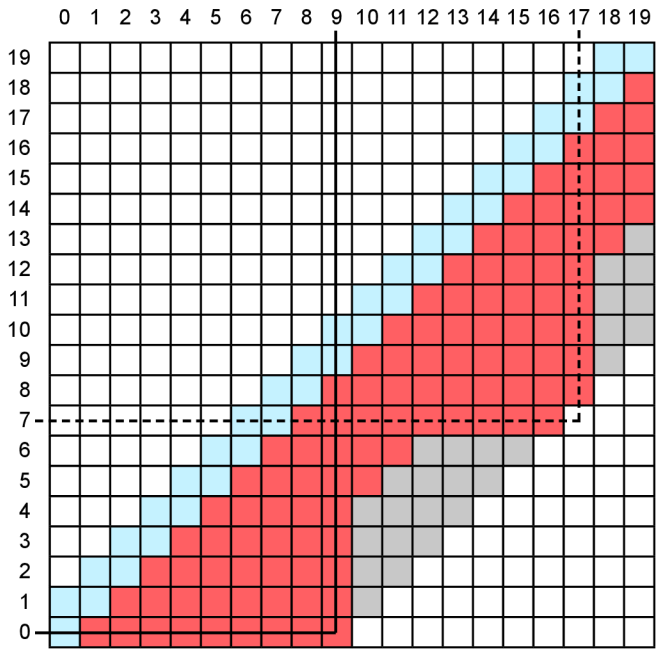
Z principu optimalizační strategie je zřejmé, že v každém případě bude muset vyhledávací algoritmus určitý počet antidiagonál kompletně spočítat, aby kumulativní skóre kvůli neshodám v diagonálním směru postupně kleslo pod daný práh P_i. Pro tyto úvodní antidiagonály samozřejmě nemá smysl provádět analýzu konečným automatem 6.7, protože by nevedla k žádné redukci a jen by přidala zbytečnou výpočetní režii navíc. Kdy tedy spustit první analýzu?

První možné řešení je založeno na analytickém výpočtu čísla první antidiagonály, na níž by *mohly* hodnoty skóre klesnout pod práh P_i. Vychází z logické úvahy, že stejně jako jsme schopni spočítat práh, můžeme spočítat i minimální počet *neshod* x nutných k tomu, aby skóre pod práh kleslo. Nerovnice 6.4 formalizuje zmíněnou úvahu.

$$x(-P_{max}) + (T_{max} - x)B_{max} < S_{min} \tag{6.4}$$

kde P_{max} představuje maximální možnou penalizaci za *neshodu* a T_{max} maximální délku těla triplexu (viz kapitola 3.2 a obrázek 3.6). Vazbu proměnných na poměry v MDP ilustruje obrázek 6.9.

Následné přepočítání počtu neshod na číslo první antidiagonály je triviální záležitost – je v podstatě jeho dvojnásobkem. Problém nastává v praktickém použití. Během experimentování s tímto modelem spouštění první analýzy vyplynulo, že takový odhad není příliš užitečný. Sice zjistíme číslo antidiagonály, na níž může dojít k prvnímu poklesu skóre



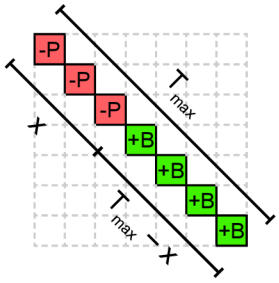
Obrázek 6.8: *Způsob uložení začátku a konce intervalů* – v obrázku jsou naznačeny dvě oblasti, v nichž můžeme nalézt dostatečně kvalitní triplex. První oblast je zakreslena plnými čarami a pro potřeby dalšího výpočtu by byla uložena jako interval $\langle 0, 9\rangle$. Druhá oblast zvýrazněná čarami přerušovanými odpovídá intervalu $\langle 7, 17\rangle$.

pod práh P_i, ale neznamená to, že bude možné další výpočet redukovat. Jak bylo zmíněno dříve, z dalšího výpočtu lze vyloučit pouze celé shluky diagonál minimálně o počtu dvojnásobku překryvu nutného kvůli splnění závislostí v MDP. Například, je-li velikost minimálního překryvu 10 antidiagonál, můžeme vyloučit pouze shluky diagonál o minimálním počtu 20. V nejhorších případech se tedy může stát, že ačkoliv bude například 90 % hodnot na antidiagonále pod prahem P_i, nebudeme moci redukovat jediný interval, protože budou rozloženy rovnoměrně (9 hodnot pod prahem, 1 nad, 9 pod, 1 nad,...). Výsledný způsob rozložení hodnot je navíc výrazně závislý na typu prohledávané sekvence, proto bylo řízení spuštění analýzy antidiagonál navrženo jiným způsobem, který bude uveden v následující části textu.

Odhad redukčního poměru

Reálný redukční poměr (RRP) udává, kolik procent z celkového počtu buněk na antidiagonále bude při výpočtu další antidiagonály vynecháno, ale můžeme jej spočítat jedině na základě výsledků analýzy konečným automatem 6.7. Myšlenka jiného způsobu spouštění analýzy vychází z možnosti RRP přibližně aproximovat s minimální výpočetní režií oproti konečnému automatu. Stačí během výpočtu antidiagonály zaznamenat počet buněk, jejichž skóre je pod hodnotou prahu P_i a podělit jej celkovým počtem buněk na aktuální antidiagonále. Získáme tak horní odhad reálného redukčního poměru (HORRP), který je vhodnější pro řízení spouštění analýzy.

Měření 6.10 ukazuje, jak závisí RRP a HORRP pro konkrétní antidiagonály na typu



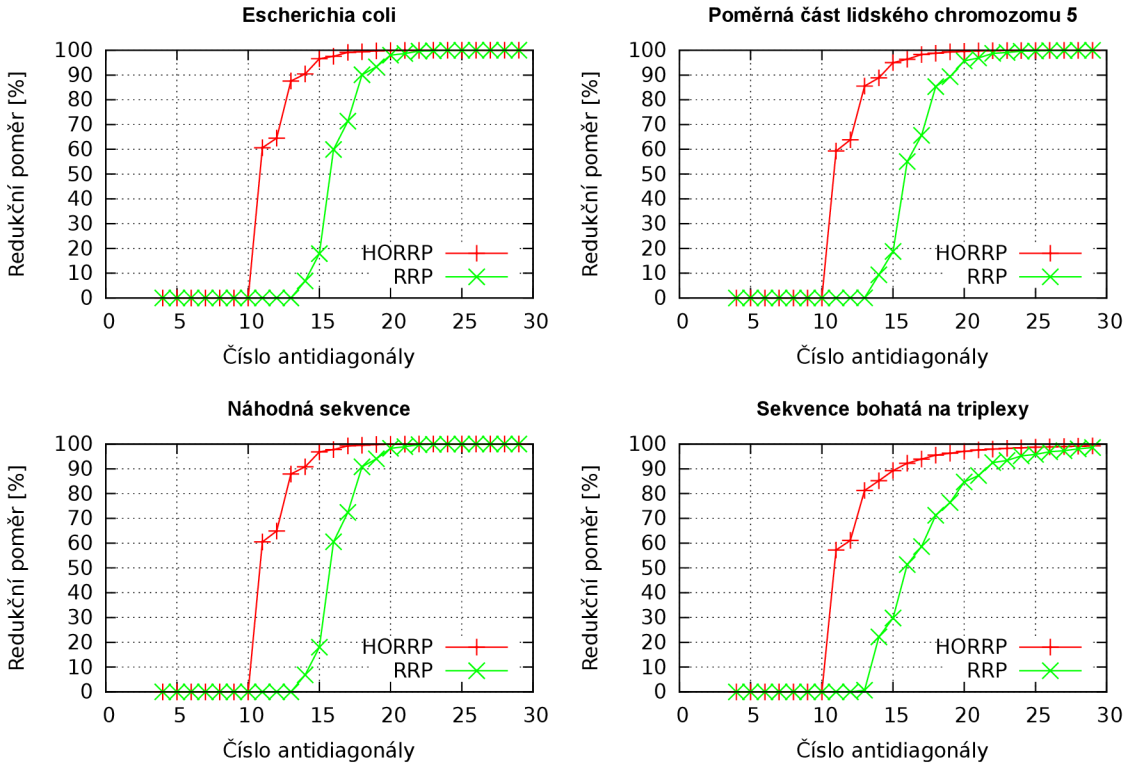
Obrázek 6.9: *Schéma k analytickému výpočtu čísla první antidiagonály* – červeně jsou označeny neshody, zeleně shody. T_{max} je maximální délka těla triplexu a x vyjadřuje minimální počet neshod, aby skóre kleslo pod práh P_i.

prohledávané sekvence. Z grafů lze vyvodit, že optimální prahová hodnota HORRP, nad níž by se vyplatilo spustit analýzu antidiagonál konečným automatem, bude v oblasti 80–100 %. Nicméně prahová hodnota HORRP by mohla být výrazně závislá také na minimálním skóre, protože ústřední optimalizační strategie z hodnoty minimálního skóre vychází. Za účelem ověření této skutečnosti bylo provedeno měření závislosti redukčního poměru na minimálním skóre (viz graf 6.11). Ukázalo se, že čím je minimální skóre menší, tím více se sice blíží hodnoty RRP a HORRP, ale optimální rozsah HORRP přesto zůstává v oblasti 80–100 %.

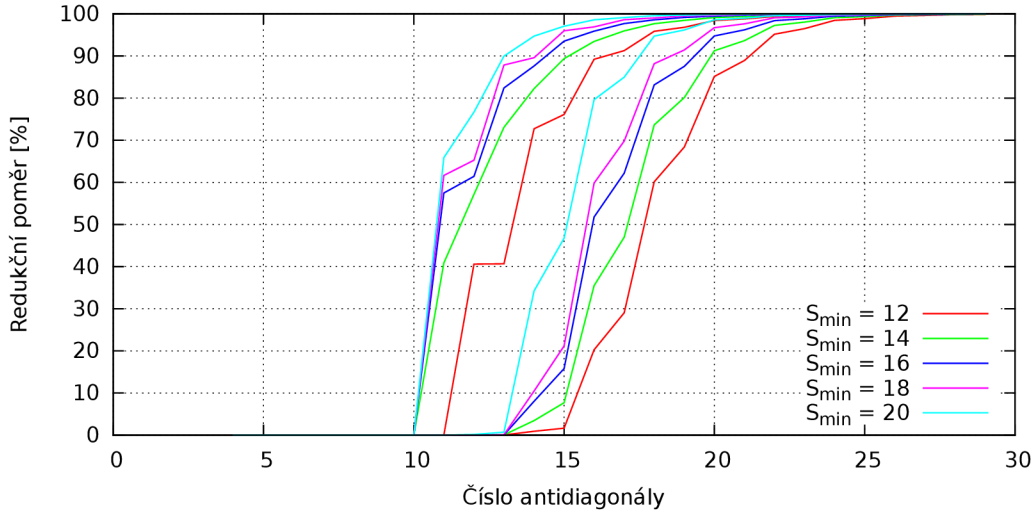
Výsledný redukční práh

Na závěr těchto experimentů byly podrobněji proměřeny prahové hodnoty z rozsahu 80–100 % a jejich vliv na celkovou dobu vyhledávání (viz graf 6.12). Vždy, když hodnota HORRP vzrostla nad daný práh, byla spuštěna analýza antidiagonál konečným automatem a tím i redukce výpočtu. Měření byla provedena zvlášť pro eukaryota a prokaryota na poměrných částech genomů následujících organismů: *Anabaena cylindrica*, *Bacillus anthracis*, *Caenorhabditis elegans*, *Drosophila melanogaster*, *Escherichia coli* a *Homo sapiens*.

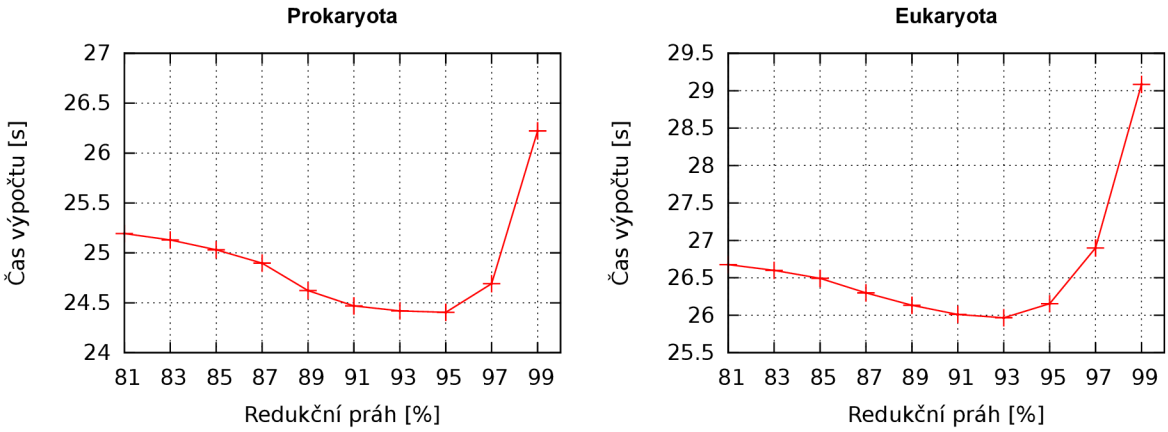
Z grafů vyplývá, že nejvhodnější práh odpovídá hodnotě 93 % a můžeme jej zvolit shodně pro eukaryotní i prokaryotní sekvence DNA.



Obrázek 6.10: *Závislost redukčního poměru na typu sekvence* – grafy demonstrují rozdíl mezi reálným redukční poměrem (RRP) a jeho horním odhadem (HORRP) v závislosti na různých typech sekvence DNA. Měření bylo provedeno na vyhledávacím algoritmu s výchozími parametry. V grafech je patrný nulový RRP i HORRP pro několik úvodních antidiagonál, které musíme vždy spočítat, aby skóre mohlo klesnout pod práh P_i. Hodnoty RRP a HORRP byly naměřeny pro každý typ triplexu zvlášť (celkem osm hodnot) a poté zprůměrovány.



Obrázek 6.11: *Závislost redukčního poměru na minimálním skóre* – dvojice stejně barevných čar odpovídá hodnotám RRP a HORRP pro konkrétní minimální skóre. Hodnoty RRP a HORRP byly naměřeny při konkrétním nastavení minimální skóre pro každý typ triplexu zvlášť (celkem osm hodnot) a poté zprůměrovány.



Obrázek 6.12: *Závislost celkové doby výpočtu na redukčním prahu* – zvlášť pro prokaryotní a eukaryotní DNA. V případě prokaryot se jedná o průměr z měření na genomech *A. cylindrica*, *B. anthracis* a *E. coli*, v případě eukaryot probíhalo měření na genomech *C. elegans*, *D. melanogaster* a pátém lidském chromozomu.

Odvození minimálního skóre

Další prostor pro mírnou výpočetní optimalizaci nabízí revize parametru `p_value`, který umožňuje nastavit minimální statistickou významnost (maximální P-hodnotu) nalezených výsledků. P-hodnotu algoritmus vyhledávání vypočítává pro výskyty potenciálních triplexů v závislosti na jejich skóre x podle vzorce 6.5.

$$P(x) = 1 - e^{-ne^{-e^{-\lambda(x-\mu)}}} \tag{6.5}$$

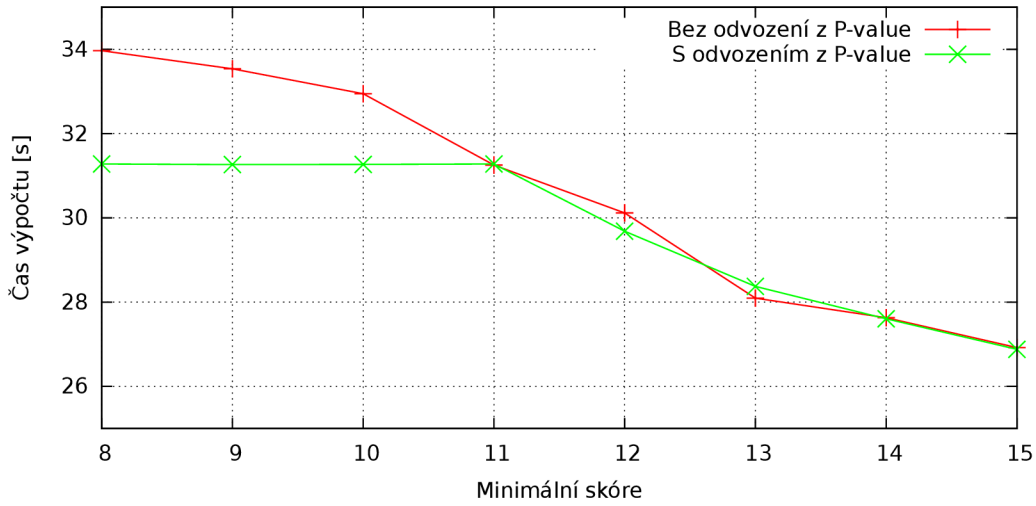
kde λ a μ jsou parametry EVD (extreme value distribution) rozložení podle publikace *Maximum likelihood fitting of extreme value distributions* [11] a n reprezentuje pravděpodobný celkový počet potenciálních triplexů v sekvenci zadané délky.

Pokud bychom byli schopni zjistit hodnotu inverzní funkce $P^{-1}(x)$, mohli bychom odvodit minimální skóre z uživatelem zadané P-hodnoty, což by bylo užitečné v případech, kdy uživatel zadá nízkou P-hodnotu (požaduje velkou statistickou významnost) a zároveň neadekvátně nízké minimální skóre. Se zvýšením minimálního skóre se totiž zlepšuje efektivita hlavní optimalizační strategie, protože může dříve začít redukce. Použijeme tedy maximum ze dvou minimálních skóre: zadaného uživatelem a odvozeného z P-hodnoty.

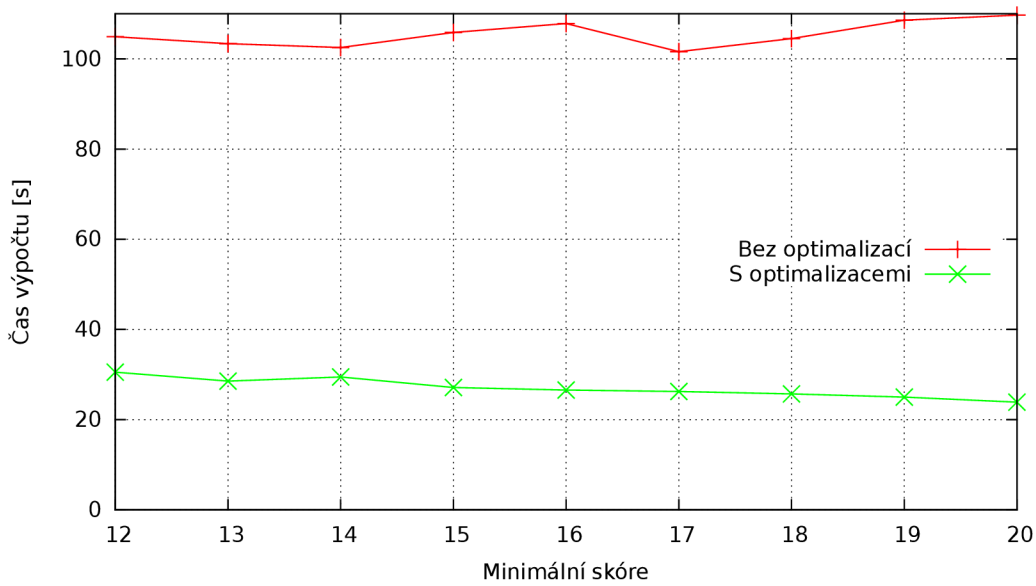
Vzhledem ke složitosti analytického odvození inverzní funkce $P^{-1}(x)$ bylo implementováno numerické řešení. V cyklu se inkrementuje skóre x do té doby, dokud není hodnota funkce $P(x)$ menší nebo rovna zadané P-hodnotě. Je to korektní postup, protože skóre nabývá pouze diskrétních celočíselných hodnot a funkce $P(x)$ je ze své definice monotónní (nerostoucí) blížící se k nule. Pro zvážení výsledného efektu byla změřena doba vyhledávání s odvozením a bez odvození z P-hodnoty napříč různými minimálními skóre (viz graf 6.13).

6.4 Výsledný efekt

Po zahrnutí všech optimalizací bylo dosaženo více než *trojnásobného* zrychlení vyhledávání oproti původnímu algoritmu (viz graf 6.14) a podstatného snížení paměťových nároků na setinu (viz kapitola 6.2). Nejvýznamnější podíl na celkovém zrychlení má výpočetní optimalizace se zrychlením 1,5–1,8, následovaná paměťovou optimalizací funkce `get_max_score` se zrychlením 1,4–1,6. Efektivní využívání procesorové cache přispívá zrychlením v rozsahu 1,2–1,3.



Obrázek 6.13: *Efekt odvození minimálního skóre* – vyhledávání bylo provedeno postupně s různými hodnotami minimálního skóre, avšak vždy se stejnou P-hodnotou. Rozdíl je patrný do doby, než se zadané minimální skóre vyrovná odvozenému z P-hodnoty. Pro každé minimální skóre bylo měření provedeno jednou.



Obrázek 6.14: *Celkový efekt optimalizací* – graf ilustruje i mírné snížení výpočetní doby v závislosti na zvýšení minimálního skóre. Pro každé minimální skóre bylo měření doby výpočtu provedeno jednou.

Kapitola 7

Závěr

Účelem této práce bylo identifikovat slabá místa algoritmu pro vyhledávání *intramolekulárních* triplexů [19] a implementovat vhodné způsoby jeho optimalizace, čehož bylo dosaženo na několika úrovních návrhu: uživatelské rozhraní, využití paměti a výpočetní náročnost.

Na úrovni uživatelského rozhraní (viz kapitola 6.1) byla provedena transformace algoritmu do podoby balíčku pro prostředí R/Bioconductor, do něhož byly zahrnuty i výstupy bakalářských a diplomových prací [18, 30, 22], které na téma intramolekulárních triplexů vznikly dříve. Balíček přináší nové vlastnosti užitečné pro koncové uživatele (molekulární biology) – především možnost přímé práce s výsledky vyhledávání, ať už se jedná o filtraci dle různých kritérií (skóre, P-hodnota, pozice, apod.), statistické výpočty, vytváření souvislostí s existujícími anotacemi nebo propracovanou nápovědu a stručnou uživatelskou příručku. Vznik balíčku představuje výrazný posun směrem k lepší uživatelské přístupnosti výchozích algoritmů, protože uživatel může instalovat balíček přímo z repositářů Bioconductoru a průběžně jej aktualizovat. Balíček úspěšně prošel revizí vývojářů Bioconductoru a je dostupný v aktuální stabilní verzi 2.12[1] z dubna 2012 a také ve vývojové verzi 2.13[2], která oproti stabilní obsahuje implementace paměťových a výpočetních optimalizací.

Z hlediska využití paměti byl optimalizován průběh výpočtu funkce maximálního skóre ze tří buněk matice dynamického programování (MDP) a eliminovány výpadky v cache paměti procesoru, což dohromady přineslo zrychlení v rozsahu 1,6–2 (viz kapitola 6.2).

Výpočetní optimalizace spočívá ve vyloučení částí MDP, ve kterých kvůli omezujícím parametrům – maximální délce triplexu, maximální délce smyčky a minimálnímu skóre – nemůže do konce výpočtu vzniknout dostatečně kvalitní triplex. Pro účely analýzy stavu výpočtu a vyloučení částí MDP byl implementován konečný automat 6.7, který omezuje další výpočet pouze na oblasti, jež mají potenciál vytvořit triplex s požadovaným skóre. Analýza je spouštěna ve vhodnou chvíli na základě horního odhadu reálného redukčního poměru tak, aby byl přínos redukce vzhledem k režii automatu co největší (viz kapitola 6.3). Tato optimalizační strategie je závislá na typu prohledávané sekvence a nastavených omezujících parametrech a celkově přináší zrychlení v rozsahu 1,5–1,8.

Další možné pokračování optimalizací by mohlo spočívat v eliminaci parametru vyhledávacího algoritmu, jenž omezuje maximální délku těla triplexu, protože se předpokládá, že molekulární biologové budou spíše požadovat vyhledávání všech kvalitních triplexů ve vstupní sekvenci nezávisle na jejich délce. Pokud bychom však měli spočítat všechny antidiagonály MDP, znamenalo by to extrémní nárůst výpočetní doby. Mohli bychom však

[1] http://www.bioconductor.org/packages/2.12/bioc/html/triplex.html
[2] http://www.bioconductor.org/packages/2.13/bioc/html/triplex.html

využít existující infrastrukturu pro redukci výpočtu (automat 6.7) a stanovit konstantní práh skóre P, který by vyjadřoval maximální tolerované množství neshod popřípadě inzercí. I když by pak výpočet probíhal pro všechny antidiagonály MDP, postupně by se redukoval jen na ty části, ve kterých skóre stále ještě roste. Získali bychom tak celé výskyty všech kvalitních triplexů za cenu minimálního nárůstu výpočetní doby. Problém tohoto přístupu ale spočívá ve vhodném určení minimálního počtu diagonál, které lze z výpočtu vyloučit, vzhledem k neomezené hloubce (viz obrázek 6.4). V praxi by to znamenalo zvolit dostatečně velkou hodnotu odpovídající maximální délce laboratorně ověřených stabilních triplexů.

Nabízí se také možnost paralelizace výpočtu, ať už na úrovni jednotlivých typů vyhledávaných triplexů nebo na úrovni překrývajících se částí MDP. V aktuální verzi prostředí R již existují balíčky (například *doParallel* nebo *foreach*), které umožňují spouštění paralelních výpočtů v *klastru*. Proto je třeba zvážit, jakým způsobem implementovat podporu pro tyto balíčky, aby uživatel mohl úlohu rozdělit mezi zvolené výpočetní stroje nebo mezi více jader jediného počítače a následně výsledky paralelních výpočtů spojit.

Literatura

[1] ALBERTS, B.; BRAY, D.; JOHNSON, A.; aj.: *Základy buněčné biologie – Úvod do molekulární biologie buňky*. Espero Publishing, druhé vydání, 2005, ISBN 80-902906-2-0.

[2] ARYA, D. P.: New Approaches Toward Recognition of Nucleic Acid Triple Helices. *Accounts of Chemical Research*, ročník 44, č. 2, 2011: s. 134–146, doi:10.1021/ar100113q.

[3] AVERY, O. T.; MACLEOD, C. M.; MCCARTY, M.: Studies on the Chemical Nature of the Substance Inducing Transformation of Pneumococcal Types : Induction of Transformation by a Desoxyribonucleic Acid Fraction Isolated From Penumococcus Type III. *J Exp Med*, ročník 79, č. 2, 1996: s. 137–158, doi:10.1084/jem.79.2.137.

[4] Basic Local Alignment Search Tool. [online], [cit. 2013-05-01].
URL http://blast.ncbi.nlm.nih.gov/

[5] BELLMAN, R.: The theory of dynamic programming. *Bulletin of the American Mathematical Society*, ročník 60, 1954: str. 503–516, doi:10.1090/S0002-9904-1954-09848-8.

[6] BILLOUD, B.; KONTIC, M.; VIARI, A.: Palingol: a declarative programming language to describe nucleic acids' secondary structures and to scan sequence database. *Nucleic Acids Res*, ročník 24, č. 8, 1996: s. 1395–1403.

[7] *Bioconductor – Package Guidelines*. [online], [cit. 2013-04-26].
URL http://www.bioconductor.org/developers/package-guidelines/

[8] BISSLER, J. J.: Triplex DNA and human disease. *Front Biosci*, ročník 12, 2007: s. 4536–4546.

[9] BUSKE, F. A.; BAUER, D. C.; MATTICK, J. S.; aj.: Triplexator: detecting nucleic acid triple helices in genomic and transcriptomic data. *Genome research*, ročník 22, č. 7, 2012: s. 1372–1381, doi:10.1101/gr.130237.111.

[10] DU, X.; WOJTOWICZ, D.; BOWERS, A. A.; aj.: The genome-wide distribution of non-B DNA motifs is shaped by operon structure and suggests the transcriptional importance of non-B DNA structures in Escherichia coli. *Nucleic Acids Res*, 2013, doi:10.1093/nar/gkt308.

[11] EDDY, S. R.: *Maximum likelihood fitting of extreme value distributions*. 1997, [online], [cit. 2013-05-08].
URL
http://selab.janelia.org/publications/Eddy97b/Eddy97b-techreport.pdf

[12] FELSENFELD, G.; DAVIES, D. R.; RICH, A.: Formation of a Three-stranded Polynucleotide Molecule. *Journal of the American Chemical Society*, ročník 79, č. 8, 1957: s. 2023–2024, doi:10.1021/ja01565a074.

[13] GENTLEMAN, R. C.; CAREY, V. J.; BATES, D. M.; aj.: Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, ročník 5, č. 10, 2004, doi:10.1186/gb-2004-5-10-r80.

[14] HOOGSTEEN, K.: The crystal and molecular structure of a hydrogen-bonded complex between 1-methylthymine and 9-methyladenine. *Acta Crystallographica*, ročník 16, č. 9, 1963: s. 907–916, doi:10.1107/S0365110X63002437.

[15] HOYNE, P. R.; EDWARDS, L. M.; VIARI, A.; aj.: Searching genomes for sequences with the potential to form intrastrand triple helices. *Journal of Molecular Biology*, ročník 302, č. 4, 2000: s. 797–809, ISSN 0022-2836, doi:10.1006/jmbi.2000.4502.

[16] HOYNE, P. R.; GACY, A. M.; MCMURRAY, C. T.; aj.: Stabilities of intrastrand pyrimidine motif DNA and RNA triple helices. *Nucleic Acids Res*, ročník 28, č. 3, 2000: s. 770–775.

[17] JAIN, A.; WANG, G.; VASQUEZ, K. M.: DNA triple helices: biological consequences and therapeutic potential. *Biochimie*, ročník 90, č. 8, 2008: s. 1117–1130, doi:10.1016/j.biochi.2008.02.011.

[18] KOPEČEK, D.: *Rozšíření a optimalizace programu pro vyhledávání triplexů v DNA sekvencích*. Bakalářská práce, Masarykova univerzita, Fakulta informatiky, Brno, 2011.

[19] LEXA, M.; MARTÍNEK, T.; BURGETOVÁ, I.; aj.: A dynamic programming algorithm for identification of triplex-forming sequences. *Bioinformatics*, ročník 27, č. 18, 2011: s. 2510–2517, ISSN 1367-4803, doi:10.1093/bioinformatics/btr439.

[20] MERGNY, J. L.; SUN, J. S.; ROUGEE, M.; aj.: Sequence specificity in triple-helix formation: experimental and theoretical studies of the effect of mismatches on triplex stability. *Biochemistry*, ročník 30, č. 40, 1991: s. 9791–9798.

[21] *R Internals*. [online], [cit. 2013-04-26].
URL http://cran.r-project.org/doc/manuals/r-release/R-ints.html

[22] RAJDL, K.: *Funkce pro manipulaci a vizualizaci molekulárních dat v prostředí R*. Bakalářská práce, Masarykova univerzita, Fakulta informatiky, Brno, 2012.

[23] RAJESWARI, M. R.: DNA triplex structures in neurodegenerative disorder, Friedreich's ataxia. *J Biosci*, ročník 37, č. 3, 2012: s. 519–532.

[24] SCHLEIFMAN, E. B.; CHIN, J. Y.; GLAZER, P. M.: Triplex-mediated gene modification. *Methods Mol Biol*, ročník 435, 2008: s. 175–190, doi:10.1007/978-1-59745-232-8˙13.

[25] The R Project for Statistical Computing. [online], [cit. 2013-04-26].
URL http://www.r-project.org/

[26] UHLÍŘ, M.: Na infekce už umíme reagovat rychle. *Respekt*, ročník 24, č. 15, 2013: s. 44–47, ISSN 0862-6545.

[27] WATSON, J. D.; CRICK, F. H. C.: Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature*, ročník 171, 1953: s. 737–738, doi:10.1038/171737a0.

[28] WEISER, M.: *Akcelerace algoritmů pro hledání triplexů v DNA sekvencích.* Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, 2012.

[29] *Writing R Extensions.* [online], [cit. 2013-04-26].
URL http://cran.r-project.org/doc/manuals/r-release/R-exts.html

[30] ZRŮNA, M.: *Vyhledávání triplexů v DNA sekvencích.* Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, 2012.

Příloha A

Adresářová struktura balíčku

- `man/` – složka obsahuje dokumentační stránky ve speciálním formátu *Rd* [29], z něhož jsou během instalace vytvářeny tři verze dokumentace: čistě textová, HTML a PDF.
- `R/` – zde nalezneme všechny zdrojové kódy v jazyce *R*, jedná se především o implementaci třídy *TriplexViews*, vizualizační funkce a rozhraní k vyhledávání a zpětnému zarovnání.
- `src/` – obsahuje zdrojové soubory v jazyce *C*, tedy vlastní implementace algoritmů pro vyhledávání a zpětné zarovnání a taktéž implementaci paměťových a výpočetních optimalizací.
- `vignettes/` – obsahuje uživatelskou příručky k balíčku. Tzv. *vignetty* oproti dokumentačním stránkám slouží pro vysvětlení složitější problematiky, která vyžaduje ucelenější souvislý text.
- `DESCRIPTION` – soubor se všemi důležitými informacemi o balíčku – obsahuje například název, verzi, krátký popis, zařazení a závislosti balíčku.
- `NAMESPACE` – tento soubor určuje veřejné rozhraní balíčku (názvy funkcí a tříd), které se exportuje do uživatelského jmenného prostoru.
- `NEWS` – soubor informující o posledních novinkách a opravách chyb v balíčku.

Příloha B

Postup instalace balíčku *triplex* z Bioconductoru

K instalaci aktuálních balíčků z Bioconductoru slouží skript `biocLite.R`. Pro instalaci stabilní verze balíčku *triplex*[1] stačí v aktuálním prostředí R (verze 3.0) zadat následující příkazy:

```
source("http://bioconductor.org/biocLite.R")
biocLite("triplex")
```

Z repositářů Bioconductoru se automaticky stáhnou a nainstalují i všechny potřebné balíčky uvedené v závislostech. Vývojová verze balíčku *triplex*[2] se instaluje stejným způsobem jako stabilní s tím rozdílem, že je nutné příkazy spustit ve vývojové verzi prostředí R. Alternativně lze zdrojový archiv balíčku stáhnout přímo ze stránek http://bioconductor.org.

[1] http://www.bioconductor.org/packages/release/bioc/html/triplex.html
[2] http://www.bioconductor.org/packages/devel/bioc/html/triplex.html

Příloha C

Příklad použití balíčku

Následující příklad demonstruje vyhledávání triplexů v prostředí R/Bioconductor a vykreslení jejich rozložení v rámci chromozomu.

1. Načteme potřebné balíčky.

```
library(triplex)
library(rtracklayer)
library(GenomeGraphs)
library(BSgenome.Celegans.UCSC.ce10)
```

2. Vyhledáme potenciální intramolekulární triplexy na začátku chromozomu X organismu *Caenorhabditis elegans*.

```
t <- triplex.search(Celegans[["chrX"]][1:62000], min_score=17, min_len=8)
```

3. Stáhneme pozice genů z databáze Ensembl.

```
mart <- useMart("ensembl", dataset = "celegans_gene_ensembl")
```

4. Vytvoříme osu a oblasti genů na přímém a komplementárním vlákně.

```
genomeAxis <- makeGenomeAxis()
genesplus <- makeGeneRegion(start = 0, end = 62000, strand = "+",
   chromosome = "X", biomart = mart)
genesminus <- makeGeneRegion(start = 0, end = 62000, strand = "-",
   chromosome = "X", biomart = mart)
```

5. Výsledky vyhledávání převedeme na anotační stopy. První stopa bude zobrazovat slabé triplexy (skóre 17–20) a druhá silné (skóre nad 20).

```
tall <- as(t,"GRanges")
ta <- makeAnnotationTrack(
  start = start(tall),
  end = end(tall),
  feature = "gene_model",
  dp = DisplayPars(gene_model = "grey")
)
```

```
ta1 <- makeAnnotationTrack(
  start = start(tall[which(score(tall) > 20)]),
  end = end(tall[which(score(tall) > 20)]),
  feature = "gene_model",
  dp = DisplayPars(gene_model = "darkblue")
)
```

6. Přesměrujeme výstup do PDF souboru a všechny připravené stopy vykreslíme.

```
pdf("annotation.pdf", 4.5, 3)
gdPlot(list("přímé" = genesplus, GENY = genomeAxis, "kompl." = genesminus,
  "slabé" = ta, "silné" = ta1), minBase = 0,
  maxBase = 62000, labelRot = 0)
dev.off()
```