



Zadání diplomové práce

Název:	MassSpecBlocks: Databáze sekvencí a stavebních bloků mikrobiálních metabolitů pro analýzu hmotnostních spekter
Student:	Bc. Jan Přívratský
Vedoucí:	Ing. Jiří Novák, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat aplikaci pro správu sekvencí a stavebních bloků mikrobiálních metabolitů pro analýzu hmotnostních spekter [1, 2]. Aplikace bude rozšířením a reimplementací generátoru stavebních bloků vytvořeného v rámci bakalářské práce [3]. Nově bude implementována podpora uživatelských účtů. Každý uživatel bude mít možnost spravovat své privátní záznamy, administrátor navíc i veřejné, položky bude možné třídit na základě strukturní podobnosti látek. V rámci reimplementace bude navržena nová architektura aplikace. Bude proveden přechod z PHP frameworku CodeIgniter na Symfony, oddělen front-end (React/Angular) a vytvořeno REST API pro dotazování na back-end. Nová technologie na front-endu bude využita k zefektivnění práce s aplikací. Dále bude proveden přechod z SQLite na MySQL. Schéma databáze bude rozšířeno s ohledem na nové funkcionality a databáze bude naplněna zkušebními daty. Aplikace bude otestována s využitím jednotkových testů a z hlediska rychlosti.

[1] Novák J. et al., CycloBranch: De Novo Sequencing of Nonribosomal Peptides from Accurate Product Ion Mass Spectra, *J. Am. Soc. Mass Spectrom.*, 26(10):1780-1786, 2015.

[2] Novák J., Havlíček V. Řešené příklady interpretace produktových spekter peptidů, *Chem. Listy*, 114(3):200-208, 2020.

[3] Přívratský J. Generátor databáze stavebních bloků přírodních látek. Bakalářská práce. ČVUT FIT, Praha, 2019.



**FAKULTA
INFORMAČNÍCH
TECHNOLÓGIÍ
ČVUT V PRAZE**

Diplomová práce

**MassSpecBlocks: Databáze sekvencí
a stavebních bloků mikrobiálních
metabolitů pro analýzu hmotnostních
spekter**

Bc. Jan Přívratský

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Novák, Ph.D.

30.04.2021

Poděkování

Rád bych vyjádřil své poděkování Ing. Jiřímu Novákovi Ph.D. za výborné vedení mé práce a za všechny přínosné rady, kterých se mi dostalo. Dále bych rád poděkoval svým rodičům za jejich trpělivost a podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (buť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 30.04.2021

.....

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague 30.04.2021

.....

České vysoké učení technické v Praze
Fakulta informačních technologií

© 2021 Jan Přívratský. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Přívratský, Jan. *MassSpecBlocks: Databáze sekvencí a stavebních bloků mikrobiálních metabolitů pro analýzu hmotnostních spekter*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Pro potřeby charakterizace neribozomálních peptidů z hmotnostních spekter vznikla nová aplikace MassSpecBlocks – webová open-source aplikace vycházející z aplikace „Building Blocks Database Generator of Natural Compounds“ (Bbdgnc) s možností vyhledávání chemických struktur ve veřejných databázích jako PubChem, ChemSpider, ChEBI a Norine. Struktury se mohou rozpadat na bloky v místech peptidových a esterových vazeb s možností editace. MassSpecBlocks může fungovat jako samostatná databáze, kam si mohou uživatelé přidávat a kde mohou spravovat své vlastní struktury, pro jejich potřeby například v hmotnostní spektrometrii.

Klíčová slova MassSpecBlocks, Bbdgnc, Generátor stavebních bloků, SMILES, CycloBranch, SmilesDrawer, PubChem, ChemSpider, Norine, sekvence, molekuly, stavební bloky, koncové modifikace, monomery, peptidy, podobnost molekul, tanimoto

Abstract

A new open-source and web-based application MassSpecBlocks has been developed to characterize nonribosomal peptides from mass spectra. The application is based on the "Building Blocks Database Generator of Natural Compounds" (Bbdgnc) and can search chemical structures in public databases like PubChem, ChemSpider, ChEBI, and Norine. Structures can be crumbled into blocks in places like peptide or ester bonds, with the possibility to edit these places. MassSpecBlocks can work as a standalone database, where users can add and manage their structures for mass spectrometry experiments or other purposes.

Keywords MassSpecBlocks, Bbdgnc, Building block generator, SMILES, CycloBranch, SmilesDrawer, PubChem, ChemSpider, Norine, sequences, molecules, building blocks, modifications, monomers, peptides, molecule similarity, tanimoto

Obsah

1	Cíl práce	1
2	Analýza	3
2.1	Bbdgnc	3
2.2	Frontend a Backend	4
2.3	REST	4
2.4	React	5
2.5	Angular	5
2.6	Typescript a frontend technologie	5
2.7	Smiles Drawer	6
2.8	JSME	7
2.9	Symfony	7
2.10	Databáze	7
2.11	CycloBranch	8
2.12	Podobné projekty	8
3	Návrh a realizace	11
3.1	Uživatelé a jejich databáze	11
3.2	Model aplikace	12
3.3	Rodiny bloků a sekvencí	15
3.4	Zjednodušení rozhraní	22
3.5	Naplnění daty	22
3.6	Identifikace polyketidů	23
3.7	Zápis sekvence	24
3.8	Import a export	26
3.9	API chemických databází	28
3.10	Frontend a Backend	30
4	Testování	31
4.1	Paralelizace	31

4.2	Měření podobnosti	33
4.3	Testy	34
5	Deployment	35
5.1	Deploy backend	35
5.2	Deploy frontend	36
6	Co tedy aplikace umí?	39
6.1	Popis funkcionalit	39
6.2	Ukázka	41
6.3	Nápady do budoucna	51
	Závěr	53
	Bibliografie	55
A	Seznam použitých zkratk	61
B	Obsah příloženého CD	63

Seznam obrázků

2.1	Jak to celé dohromady funguje	9
3.1	Databázový model	13
3.2	MassSpecBlocks detail	14
3.3	Beauverolide A	16
3.4	Beauverolide B	16
3.5	Tanimoto - SQL ukázka	21
3.6	Desferri-ferrioxamine B	24
3.7	Pseudacyclin A	25
3.8	Acv	28
6.1	Kontejnery ukázka	43
6.2	Bloky ukázka	44
6.3	Acv výsledky hledání	45
6.4	Acv ukázka	46
6.5	Fusarinin výsledky hledání	47
6.6	Putrebactin ukázka	48
6.7	Gramicidin C ukázka	49
6.8	Gramicidin C ukázka - bloky	50
6.9	Ferrichrome	52
6.10	Neocoprogen	52

Seznam tabulek

2.1	SMILES a Valin	6
2.2	InChI a Valin	7
3.1	Struktura B2s pro pseudacyclin A	25
3.2	Možnosti vyhledávání na API třetích stran	29
3.3	API MassSpecBlocks	29
4.1	Měření vyhledávání po rozpadu bloků	32
4.2	Měření rychlosti podobnosti	33

Seznam algoritmů

1	SQL dotaz s výpočtem tanimoto koeficientu	20
2	Detekce polyketidů	23
3	SQL dotaz s generováním zápisu sekvence	27

Cíl práce

Cílem diplomové práce je navrhnout a implementovat aplikaci pro správu sekvencí a stavebních bloků mikrobiálních metabolitů pro analýzu hmotnostních spekter s názvem MassSpecBlocks. Aplikace bude rozšířením a reimplementací generátoru stavebních bloků (Bbdgnc) vytvořeného v rámci mé bakalářské práce. Převážně jde o převod Bbdgnc do jiných technologií a rozšíření o nové funkce. Nově do aplikace MassSpecBlocks implementuji podporu uživatelských účtů. Každý uživatel bude mít možnost spravovat své privátní záznamy, administrátor navíc i záznamy veřejné. Položky v aplikaci bude možné třídit na základě strukturní podobnosti látek - rozdělení na rodiny a přiřazení rodin dle podobnosti. V rámci reimplementace navrhnu novou architekturu aplikace. Provedu přechod z PHP frameworku CodeIgniter na Symfony, oddělím frontend (React/Angular) a vytvořím REST API pro dotazování na backend. Novou technologii na frontendu využiji k zefektivnění práce s aplikací. Dále provedu přechod z databáze SQLite na MySQL. Schéma databáze rozšířím s ohledem na nové funkcionality a databázi naplním zkušebními daty. Průběžně budu aplikaci testovat pomocí unit testů a také z hlediska rychlosti. Zachovám vyhledávání struktur ve veřejných databázích a rozpad sekvence na stavební bloky, stejně tak import/export a další funkce.

Analýza

2.1 Bbdgnc

Bbdgnc [1] (Building Blocks Database Generator of Natural Compounds) je aplikace, kterou jsem vytvořil v rámci bakalářské práce. Nová aplikace MassSpecBlocks vychází z Bbdgnc a bere si mnoho poznatků právě z této aplikace. Aplikace MassSpecBlocks je napsaná v jiných technologiích a od začátku. Vizualně jde o rozšíření Bbdgnc o nové funkce a zjednodušení rozhraní. Bbdgnc staví na vyhledávání, rozpadu sekvencí a importu/exportu do programu CycloBranch. Původní aplikace Bbdgnc je napsaná v PHP a frameworku CodeIgniter s databází SQLite, používá knihovny SmilesDrawer a JSME pro vykreslování a editaci struktur.

Konkrétní odlišnosti mezi aplikacemi jsou: rychlejší vyhledávání bloků na Pubchemu po rozpadu bloků díky paralelizaci na frontendu, rozšíření vyhledávání o ChemSpider databázi, detekce polyketidů, detekce N-C směru sekvence, rozšíření o uživatele (registrace, přihlášení, nastavení) a jejich databáze (přístupová práva uživatelů). Nově je aplikace naplněna sekvencemi z neribozomálních peptidů a siderophorů. Dále je zde nová možnost přiřazovat k sekvencím a blokům rodiny (shlukování podobných struktur), klonování sekvencí, kontejnerů a změny v uživatelském rozhraní (snažím se držet podobnosti Bbdgnc, ale zjednodušit rozhraní). Pro sekvence je také nově implementována podobnost sekvencí, která navrhne rodinu nebo se dá použít pro vyhledávání v databázi pomocí SMILES. Díky rozdělení na frontend a backend by některý uživatel mohl využít i REST API na backendu a dotazovat se na struktury v aplikaci příměji.

2.2 Frontend a Backend

Původní aplikace nemá oddělený frontend a backend. Rozdělením aplikace na dvě části získám mnohé výhody.

V našem případě přenesu zátěž dotazování na databáze třetích stran jako Pubchem a Norine na klienta. Takové API třetích stran mají určitá omezení k dotazování, jako je například počet dotazů za určitý časový úsek a podobně. Tím, že dotazování bude probíhat na frontendu u klienta, budou se tato omezení počítat pro každého uživatele zvlášť, což je velká výhoda oproti tomu, když se tato omezení sčítala pro všechny uživatele používající starou aplikaci.

Další výhodou je také paralelizace dotazování. Paralelizace by šla provést i ve staré aplikaci, ale musel bych pracovat s vlákny v PHP, což není obvyklý způsob. Na frontednu se typicky používá javascript, kde je paralelizace velmi intuitivní pomocí asynchronních volání.

Oddělení také pomůže ke zjednodušení rozhraní aplikace, zde bude záležet již na konkrétní použité technologii. Tím, že oddělím grafické prvky od logiky a databáze, bude se mi v aplikaci lépe pracovat se zdrojovým kódem. Rozdělením také získávám možnost obě části aplikace nasadit na různých serverech.

2.3 REST

Rest je zkratka z Representational State Transfer, což je architektonický styl pro distribuované hypermediální systémy. [2]

Pomocí REST bude komunikovat frontend s backendem. REST používá pro přesnost dat HTTP protokol a hlavně jeho metody POST, GET, PATCH, PUT a DELETE spojené s CRUD operacemi, díky kterým definuje chování pro daný zdroj. V aplikaci existuje několik struktur jako jsou sekvence, bloky, modifikace a další. Nově přibudou uživatelé a jejich databáze. Pojem databáze jsem nahradil slovem kontejner, aby nedocházelo k záměně s databází pro celou aplikaci a chemickými databázemi třetích stran. Výše zmíněné struktury představují zdroje pro REST a jako jejich URI použiji hierarchické URI, protože většina ze struktur bude závislá na kontejneru.

Například: `:endpoint:/rest/container/:containerId:/block`.

2.4 React

React je javascriptová knihovna (není to framework jako například Angular) pro vývoj uživatelských rozhraní na frontendu. [3]

Je to open-source projekt vyvíjený firmou Facebook. React je view vrstva z MVC aplikace. Nejdůležitějším aspektem Reactu je, že lze vytvářet komponenty, což jsou přepoužitelné HTML elementy pro rychlé a efektivní vytváření rozhraní. React také používá efektivní ukládání a zpracování dat pomocí state a props. [4]

React se na problémy dívá minimalistickým přístupem. Nepoužívá Dependency Injection, namísto klasických šablon používá JSX (rozšíření javascriptu), nástroje pro unit testing, XSS protection. Je zde velká svoboda ve výběru dalších knihoven. Nejčastěji používané knihovny s Reactem jsou: React-router pro routing, Fetch API nebo Axios pro HTTP requesty, Enzyme pro unit testing, atd. [5]

2.5 Angular

Angular je framework a vývojová platforma pro výkonné a sofistikované aplikace, který je vyvíjen firmou Google. [6]

Angular je napsán v jazyku Typescript (javascript s typovou nadstavbou). Architektura aplikace napsané v Angularu staví na základních konceptech. Základním blokem jsou NgModules, které poskytují kompilační kontext pro komponenty. NgModules shromažďují související kód do funkčních sad. Aplikace je pak definována několika sadami. Aplikace má vždy jeden kořenový modul a několik dalších modulů.

Komponenty definují view, což jsou množiny elementů, které Angular vybírá a modifikuje dle logiky a dat aplikace. Komponenty používají služby (services). Ty poskytují specifickou funkčnost pro view. Poskytovatelé služeb jsou injektovány do komponent jako závislosti, což dělá kód modulárním, přepoužitelným a efektivním.

Služby i komponenty jsou jednoduché třídy s dekorátory, které označují jejich typ a poskytují metadata. Metadata služeb slouží pro DI (Dependency Injection). [5]

2.6 Typescript a frontend technologie

Typescript [7] rozšiřuje javascript o podporu typů. Zároveň ale lze v typescriptovém kódu použít javascriptový zápis. Pokud přesunu javascriptový kód do typescriptového souboru, nejspíš se zobrazí chyby. Může se jednat o chyby v kódu nebo pouze restriktivní typescript chyby. Toto může pomoci odhalit nežádoucí chyby v kódu. V každém případě půjde kód vždy zkompileovat a spustit jako javascript.

Tabulka 2.1: SMILES a Valin

Generic	Isomeric	Unique
<chem>CC(C)C(C(=O)O)N</chem>	<chem>CC(C)[C@@H](C(=O)O)N</chem>	<chem>CC(C)C(N)C(O)=O</chem>

Mezi Reactem a Angularem jsem zvolil React, protože neřeší vše tak robustně a celkově mi přišel jednodušší na pochopení a naučení. Spolu s Reactem použiji balíčkovací systém npm pro instalaci dalších závislostí jako je například JSME, SmilesDrawer nebo třeba Sass pro obohacené CSS.

Velmi vhodná je také Fetch API [8] a metoda `fetch()` pro HTTP requesty na backend. Fetch je asynchronní, tedy na odpověď nečeká a kód pokračuje dál. Co se má stát po přijetí HTTP response se specifikuje pomocí metody `then()`, nebo se musí fetch synchronizovat pomocí klíčových slov `async` a `await`.

2.7 Smiles Drawer

Smiles Drawer [9] je javascriptová knihovna, kterou používá i Bbdgnc. Jak již napovídá název, knihovna vykresluje struktury ve formátu SMILES. SMILES [10] je jednoduchý zápis struktury do textového řetězce. Existuje více typů SMILES, jsou to generic, isomeric a unique. V tabulce 2.1 je vidět jejich zápis. Generic je základní zápis, isomeric přidává určité prvky navíc a protože několika různými zápisy SMILES lze popsat tu samou strukturu, existuje ještě unique SMILES. Pro tvorbu unique SMILES existuje algoritmus [11], kterému se věnuji v bakalářské práci, nutno podotknout, že algoritmus nefunguje pro všechny vstupy, ale ve většině případů stačí.

V nové verzi přibylo renderování SMILES jako svg, oproti mnou používanému canvasu. V aplikaci používám můj fork původního Smiles Draweru, kvůli implementaci rozpadávání sekvence na stavební bloky. Samotný rozpad na bloky je podrobně popsán v mé předchozí práci [1], přesto je zde potřeba alespoň přiblížit, o co se jedná. Při vykreslení struktury ve SmilesDraweru se automaticky označí body rozpadu, dle kterých se bude sekvence rozpadat. Pokud nestačí automaticky nalezené body rozpadu, pak je možnost přidat/odstranit body rozpadu kliknutím myši na požadovanou hranu. Jakmile je vstup připraven, začne se procházet struktura od všech bodů rozpadu pomocí DFS se zakázaným přechodem přes hranu rozpadu. Výstupem jsou jednotlivé SMILES stavebních bloků ze sekvence. Navíc algoritmus rozezná typ struktury, zda je lineární, cyklická, větvená apod. Nově implementuji do SmilesDraweru také identifikaci polyketidů a detekci N-C směru sekvence, těmto tématům se věnuji podrobněji v sekci 3.6 a 3.7.

Tabulka 2.2: InChI a Valin

	Valine
InChI	1S/C5H11NO2/c1-3(2)4(6)5(7)8/h3-4H,6H2,1-2H3,(H,7,8)/t4-/m0/s1
Key	KZSNJWFQEVHDMF-BYPYZUCNSA-N

2.8 JSME

JSME [12] je knihovna pro vykreslování, ale především editaci SMILES. Bbdgnc ji také používá. Využil jsem balíček, který ji obaluje jako React komponentu [13]. Její použití je velmi jednoduché a intuitivní. Stačí použít tag `Jsme` a nadefinovat metodu pro získání SMILES při změně.

Kromě SMILES umí samotný JSME vyexportovat upravovanou strukturu do formátu MOL nebo vygenerovat InChIKey. InChIKey je formát vycházející z InChI (IUPAC International Chemical Identifier), je to hash z InChI. Pro struktury by měl být unikátní. InChI je zápis struktury poměrně odlišný od SMILES. MassSpecBlocks s těmito identifikátory přímo nepracuje, ale pomocí JSME editoru je lze získat. Ukázku InChI lze vidět v tabulce 2.2. [14]

2.9 Symfony

Symfony [15] je open-source množina PHP komponent a webový framework. Umí dobře pracovat s HTTP response a podporuje routování. Pro správu dependencí používá composer [16]. K připojení databáze používá ORM přístup s knihovnou Doctrine [17]. Pro Symfony existuje mnoho doplňků, takzvaných bundles, které umožní například generovat dokumentaci k REST API, logování a podobné funkcionality.

Kromě Symfony existuje ještě třeba Laravel [18]. Pro použití Symfony rozhodlo především to, že už s ním mám zkušenosti a nemusím se učit nový framework.

2.10 Databáze

Původní aplikace používala jako databázi SQLite [19], což je malá a rychlá SQL databáze napsaná v C. V nové aplikaci bude použita Mysql [20] nebo její fork MariaDB [21]. Hlavní důvod pro změnu databáze je ten, že MariaDB běží na serveru, kde pak bude aplikace nasazena. MySQL je relační SQL databáze. Je poměrně jednoduché se ji naučit ovládat oproti ostatním robustnějším databázím jako Oracle Database nebo Microsoft SQL Server, ale základ je stále stejný - SQL. Jediné, v čem se databáze liší, je, že každá databáze má svůj dialekt SQL. Například MySQL používá pro grupování řetězců funkci `group_concat()` a Microsoft používá funkci `string_agg()`. [22]

2.11 CycloBranch

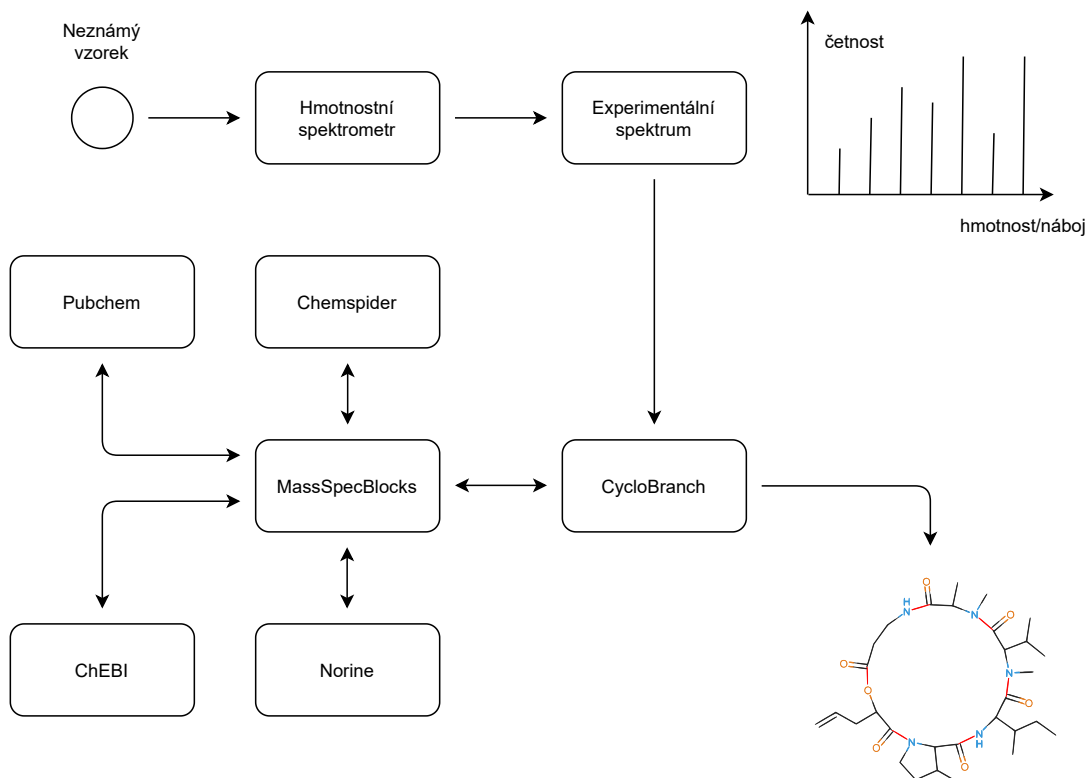
CycloBranch je open-source program napsaný v C++ Jiřím Novákem. Slouží pro datovou analýzu hmotnostní spektrometrie. Hmotnostní spektrometrie je metoda pro identifikaci molekul z neznámých látek. Taková molekula je v MassSpecBlocks nazývána sekvencí, jsou jí například peptidy. Ty jsou tvořeny řetězcem aminokyselin. Při analýze jsou převedeny na ionty, rozbity na fragmenty a je zaznamenáno jejich hmotnostní spektrum, což je graf závislosti intenzity iontů na poměru jejich hmotnosti a náboje m/z [23]. Identifikaci peptidů lze provést pomocí porovnání s teoretickou databází spekter generovaných ze sekvencí. Zde nastupuje MassSpecBlocks, který slouží k uchování takovéto databáze teoretických struktur. Pro bližší pochopení, jak celý proces funguje jsem připravil obrázek 2.1. MassSpecBlocks umožňuje vyhledat sekvence v sobě i dalších chemických databázích (Pubchem [24], ChemSpider [25], Norine [26] [27], ChEBI [28], PDB [29]), nebo lze také zadat sekvence ručně. MassSpecBlocks slouží ke správě databáze těchto struktur. Na opačné straně je CycloBranch, do kterého vstupuje experimentální spektrum změřené hmotnostním spektrometrem na neznámém vzorku a také do něj vstupuje databáze teoretických spekter na základě MassSpecBlocks. Výstupem je pak anotace spektra peptidu. Data z MassSpecBlocks se do CycloBranch předají txt souborem, který lze v aplikaci vyexportovat. Samotný CycloBranch je dostupný na <https://ms.biomed.cas.cz/cyclobranch/>. [30], [31]

2.12 Podobné projekty

Existuje několik obdobných projektů s různorodou podobností. Samotné chemické databáze jako Norine, ChemSpider, ChEBI, PDB, Pubchem ukládají struktury jako MassSpecBlocks. Celá databáze je ale veřejná. Například u Norine může kdokoli přidat novou strukturu, ale tento požadavek musí někdo schválit, což může trvat nějaký čas. MassSpecBlocks umožňuje správu struktur v uzavřeném okruhu uživatelů, jak veřejně, tak soukromě a není třeba schvalovat struktury. Hlavní výhodou MassSpecBlocks oproti chemickým databázím, je sdružení vyhledaných informací do jedné společné uživatelské databáze.

Dalším podobným projektem z hlediska rozpadu sekvence na stavební bloky jsou NRPro, rBAN a s2m. NRPro [32] uživatelům pomůže při dvou úkonech - dereplikace a anotace. Dereplikace funguje tím způsobem, že uživatel nahraje MS/MS spektrum a NRPro jej vyhodnotí a identifikuje peptidy. Tato část s MassSpecBlocks nesouvisí přímo, ale tuto funkci má související program CycloBranch, pro který existuje v MassSpecBlocks export struktur. Anotace je funkčnost, která je totožná s rozpadem sekvence na stavební bloky. Rozdíl NRPro oproti MassSpecBlocks spočívá v tom, že NRPro neumožňuje ukládat struktury do databáze a vyhledávat struktury v chemických databázích třetích stran a vstupem pro NRPro je MS/MS spektrum.

Obrázek 2.1: Jak to celé dohromady funguje



S2m [33] je zkratka ze Smiles2Monomers. Jako vstup slouží SMILES. Nejdříve algoritmus mapuje bloky na sekvenci na základě izomorfismu podgrafů a zároveň musí zajistit, aby celá struktura byla pokryta bloky. Jedná se o tutéž funkčnost jako má rozpad sekvence na bloky, ale dosaženou jiným způsobem.

rBAN [34] je utilita vyvinutá v Javě pomocí knihovny CDK (Chemistry Development Kit). Vstupem pro rBAN jsou SMILES. rBAN funguje velmi podobně jako rozpad bloků v MassSpecBlocks. Mapuje cílové vazby ve struktuře a snaží se identifikovat stavební bloky. Utilita je poměrně nová. Vyšla ve stejném roce, jako jsem vytvořil původní aplikaci Bbdgnc, která už rozpad na bloky používala.

Některé části má MassSpecBlocks podobné s výše uvedenými projekty. V širším pohledu má MassSpecBlocks navíc další funkce a řekl bych, že spojuje funkce a data všech těchto projektů dohromady.

Návrh a realizace

3.1 Uživatelé a jejich databáze

Jak plyne ze zadání práce, největší přínosem je přidání uživatelů a jejich osobních databází. Každý uživatel si tak může vytvořit svůj vlastní kontejner, který reprezentuje osobní databázi, se kterou si může uživatel nakládat dle libosti. Existují 3 úrovně práv přístupu ke kontejneru, které jsou inkluzivní: **R**, **RW** a **RWM**.

R = pouze čtení

RW = **R** + vytváření, editace a mazání struktur

RWM = **RW** + správa uživatelů v kontejneru

Při vytvoření kontejneru se automaticky uživateli přiřadí nejvyšší oprávnění a to role **RWM**. Uživatel tak může do databáze přidat i jiné uživatele a nastavit jim výše zmíněná práva přístupu. Při vytváření nového kontejneru je ještě třeba uvést, zda se bude jednat o veřejný nebo privátní kontejner. Pokud by se jednalo o veřejný kontejner, mohou číst data i uživatelé ke kontejneru nepřičtení (všichni registrovaní i neregistrovaní uživatelé). Pro privátní kontejner platí přístupová práva bez výjimek.

K samotné registraci uživatelů stačí unikátní jméno (nick) a heslo. Volitelný je email, který slouží pouze pro obnovení přístupu k účtu po ztrátě hesla. Heslo je do databáze ukládáno jako hash. Dále je zde ověření proti robotům. Nechtěl jsem použít captchu nebo podobný nástroj, kde by bylo třeba používat další API třetí strany. Funguje to tím způsobem, že server vybere nějakou z 20 základních aminokyselin, pošle ji na frontend a informaci si uloží do session. Uživatel pak napíše třípísmennou zkratku aminokyseliny a na backendu se ověří, zda je správná. U registrace je také checkbox pro přijetí podmínek používání aplikace. Zmiňují použití cookies, použití zadaných údajů při registraci a použité technologiích. Podmínky může administrátor aplikace kdykoliv

změnit. Pokud se tak stane, při přihlášení bude uživatel v popupu vyzván k potvrzení nových podmínek. Tímto bude nejen informován o změně, ale bude se také moci rozhodnout, zda s ní souhlasí.

Pro přihlášení stačí pouze jméno a heslo. Samotné ověřování uživatele funguje na principu tokenu. Pro první přihlášení použije uživatel jméno a heslo. Po úspěšném přihlášení se na backendu vygeneruje token, který se předá zpět na frontend. Token se poté používá místo hesla pro ověření pro další requesty na backend. Pokud není uživatel aktivní po určitou dobu, token se vymaže a uživatel je donucen se znovu přihlásit.

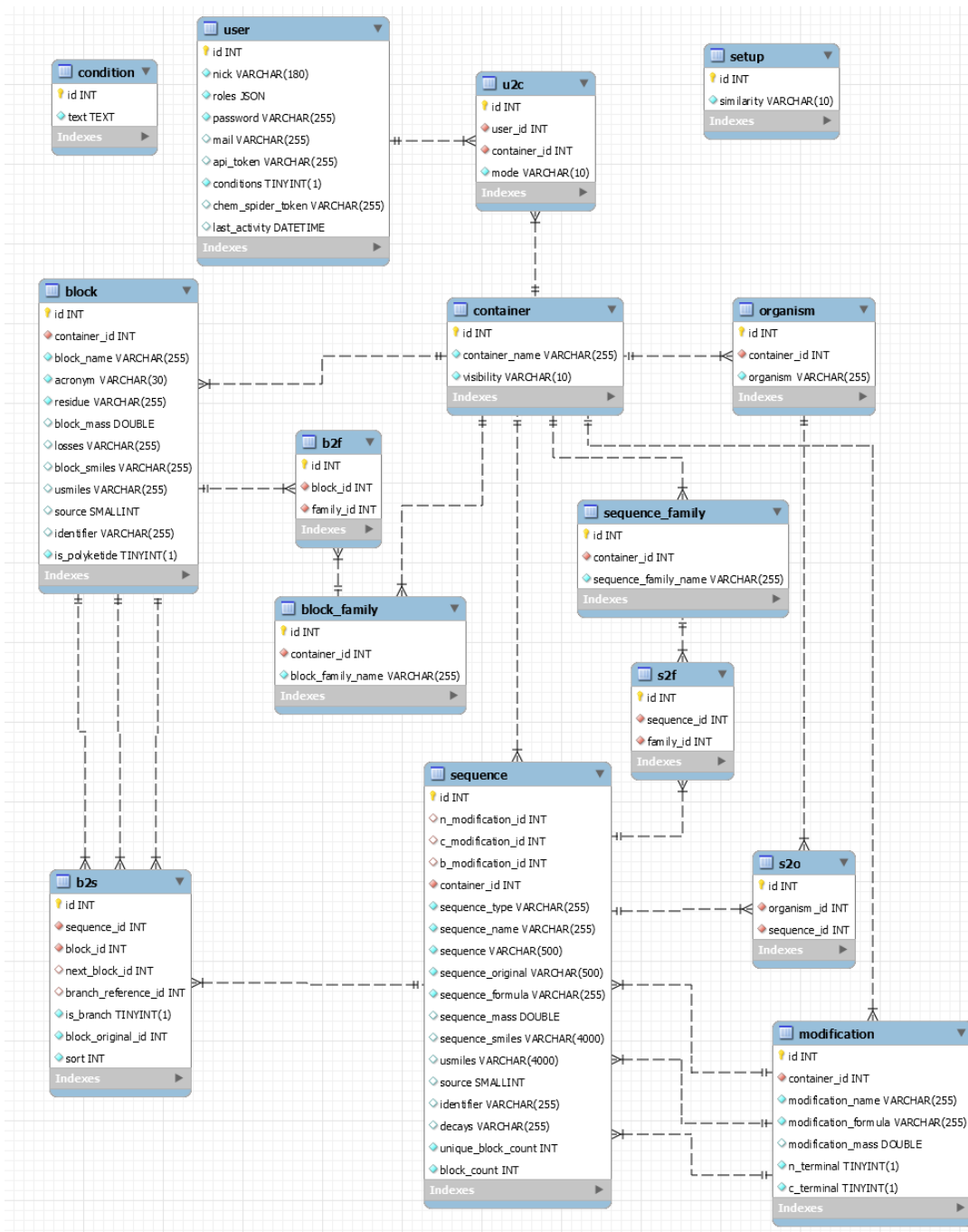
Uživatel se může rozhodnout svůj účet smazat. V takovém případě je nutné rozhodnout, co udělat s kontejnery, kde je daný uživatel uveden jako jediný s rolí RWM. Aby kontejner zůstal zachován pro ostatní uživatele, zprovoznil jsem následující chování: pokud má ke kontejneru přístup alespoň jeden uživatel s rolí RW, povýším všechny takové uživatele na roli RWM. Pokud neplatí předchozí a ke kontejneru má přístup alespoň jeden uživatel s rolí R, přidám přímý přístup do kontejneru administrátorovi s rolí RWM. A poslední situace, pokud má přístup ke kontejneru pouze uživatel k odstranění, kontejner také odstraním.

3.2 Model aplikace

Nejprve bych probral stavební kamen aplikace, což je databáze. Databázový model vychází z původního datového modelu, který staví na třech hlavních entitách: sekvence, bloky a modifikace. Mění se vazební tabulka b2s tak, aby pojala i informace pro složení zápisu sekvence. Přibývá tabulka s definicí kontejneru a tabulka s definicí uživatele. Dále přibývají tabulky pro rodiny bloků, sekvencí i tabulka pro organismy spolu s vazebními tabulkami (M:N). Na obrázku 3.1 je vidět hlavní část databázového modelu.

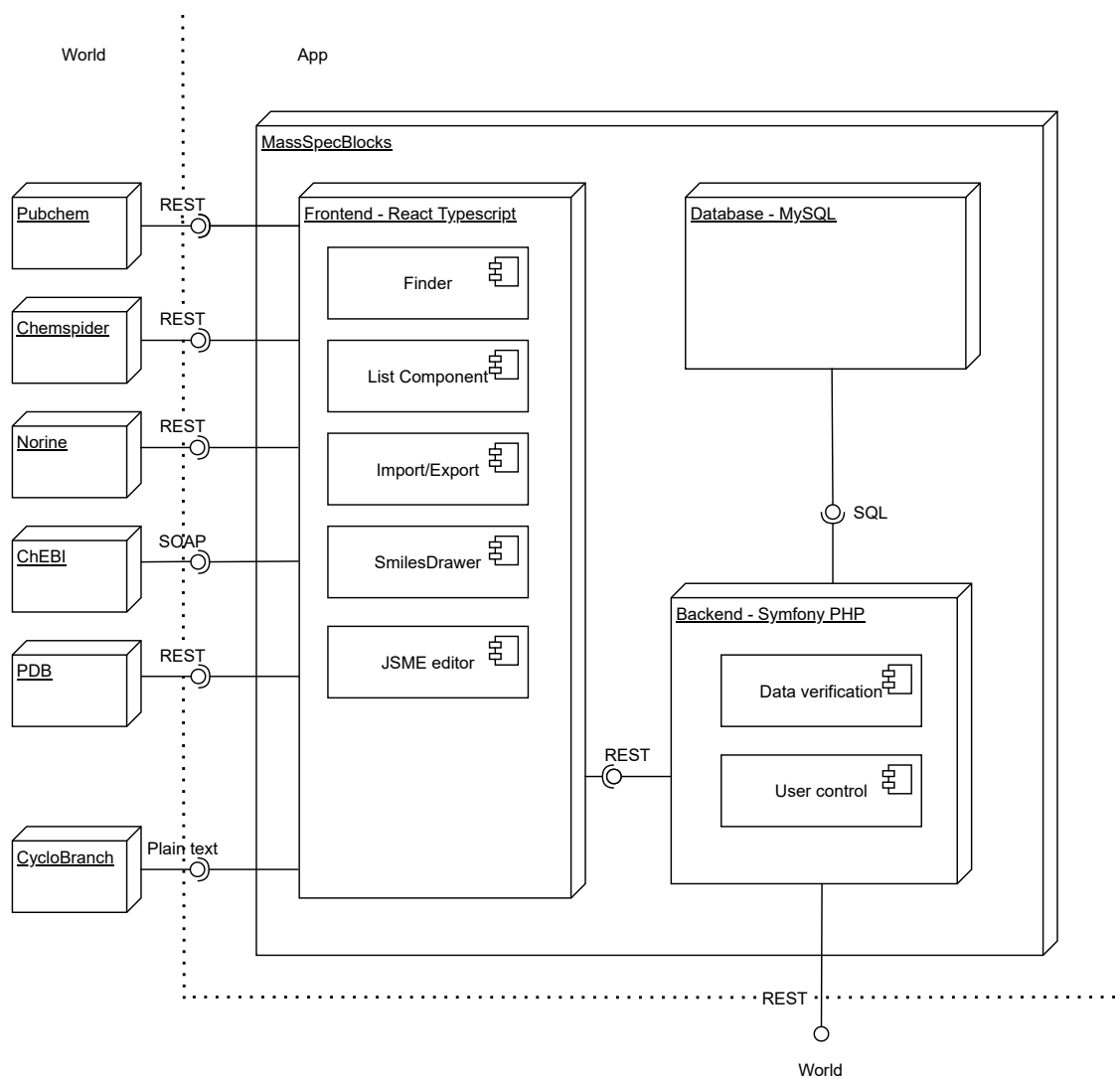
Tabulka user uchovává informace o uživateli jako je přihlašovací jméno (unikátní), email, role uživatele, zahashované heslo, token pro verifikaci, čas poslední aktivity a zda byly odsouhlaseny podmínky užívání aplikace. Tabulka kontejner obsahuje informace o kontejneru, čímž je jméno a visibility. Visibility nabývá pouze dvou hodnot, a to PUBLIC nebo PRIVATE. Protože uživatel může mít více kontejnerů a ke kontejneru může přistupovat více uživatelů, je nutná vazební tabulka u2c M:N, do které jsem přidal ještě vlastnost mode, který nabývá hodnot R, RW a RWM. Do kontejneru spadají sekvence, bloky, modifikace a rodiny bloků a modifikací. Jméno sekvence, bloku (acronym) a modifikace musí být v rámci kontejneru unikátní, proto je na daných tabulkách vytvořen unique index (name, containerId). Bloky, sekvence a modifikace mají několik společných sloupců. Jsou to jméno, formule (u bloku residue, př.: CH_1NO_2) a monoisotopická hmotnost. U sekvencí i bloků jsou ještě uloženy SMILES pro vykreslování struktur a také je zde identifikátor (identifier) struktury v externí databázi (source). K sekvencím

Obrázek 3.1: Databázový model



3. NÁVRH A REALIZACE

Obrázek 3.2: MassSpecBlocks detail



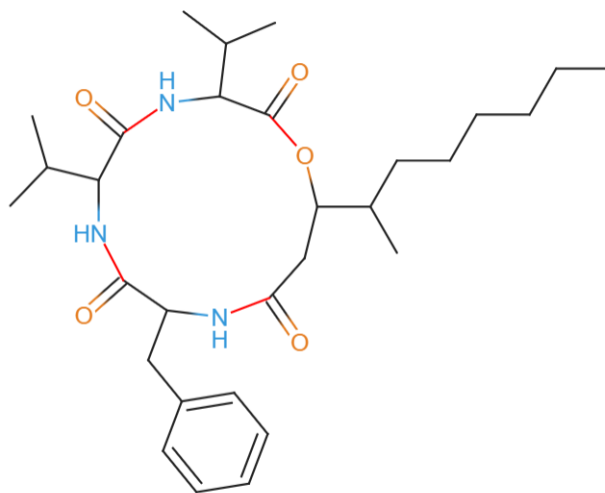
jsem připravil ještě sloupce s předpočítanými hodnotami `block_unique_count` a `block_count`, které se hodí pro algoritmus podobnosti, kterému se věnuji v následné sekci. Vztah mezi sekvencemi, bloky a modifikacemi je následující. Sekvence se skládá z bloků a může obsahovat až 3 koncové modifikace dle typů N, C nebo Branch, které závisí na typu sekvence. Mezi sekvencí a bloky je nutná opět vazební tabulka M:N, neboť blok se může vyskytovat ve více sekvencích. Navíc jsou v tabulce `b2s` uloženy informace k získání zápisu sekvence. Je nutné mít zde aktuální blok, odkaz na blok, který v sekvenci následuje, a v případě, že se sekvence větví, mít ještě odkaz na větvící se blok. Pravidla zápisu sekvence jsou převzaty z programu `CycloBranch`. Sestavený zápis je i uložený v sekvenci, pro rychlý přístup. Této problematice se zápisem sekvence se budu věnovat později samostatně v sekci 3.7. Pro bloky a sekvence se nově mohou přidávat rodiny. Jsou to v podstatě totožné tabulky `block family` a `sequence family`, ale každá je navázaná na svoji strukturu. Toto řešení jsem zvolil, oproti variantě s jednou tabulkou, vzhledem ke své jednoduchosti. Kromě `family` existuje ještě velmi podobná struktura a to `organismus`, mají ho pouze sekvence a také ve vztahu M:N.

Pohled, z jakých komponent se aplikace skládá, je názorně vidět na obrázku 3.2. Jak je na obrázku vidět, aplikace se skládá z frontendu a backendu, pod kterým běží MySQL. Každá z těchto tří částí může být teoreticky nasazena na jiném serveru. Backend vystavuje REST API pro frontend a ostatní svět. Převážně se stará o kontrolu uživatelských vstupů, jejich transformaci a správu uživatelského přístupu. Frontend se stará o komunikaci s databázemi třetích stran (Pubchem, Chempid, Norine, ChEBI, PDB) a poskytuje import/export pro `CycloBranch`. Další významnou částí je `ListComponent`, což je abstraktní komponenta, která komunikuje s backendem a zajišťuje výstup dat na obrazovku. Zajišťuje především volání CRUD nad zdroji. A také samozřejmě frontend vykresluje struktury pomocí `SmilesDraweru` a umožňuje jejich editaci díky JSME editoru.

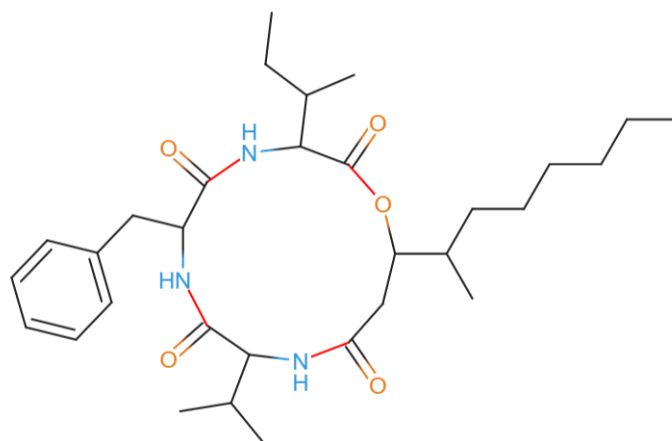
3.3 Rodiny bloků a sekvencí

Nad sekvencemi a bloky lze nově definovat jejich rodiny, jde v podstatě o určitou segmentaci struktur. Například u bloků jsem základních 20 aminokyselin označil jako jednu rodinu. U sekvencí jde o seskupování podobných struktur, například `beauverolide A` a `beauverolide B`. Obě sekvence se rozpadají na čtyři bloky, kde tři z nich jsou stejné. Konkrétně jde o Valin, Phenylalanin a `C10:0-Me(4)-OH(3)`. V `beauverolide A` je pak ještě jednou Valin a v `beauverolide B` Isoleucin. Je promícháno i propojení bloků. To že patří do stejné rodiny napovídá už jejich jméno. Obě sekvence jsem pro porovnání přidal na obrázcích 3.3 a 3.4.

Obrázek 3.3: Beauverolide A



Obrázek 3.4: Beauverolide B



Pro automatický návrh rodiny pro sekvenci je třeba umět sekvence porovnávat. Prvním řešením, které se nabízí, je porovnávání grafů. Vždy je potřeba porovnat jeden graf se všemi grafy v databázi. Samotný problém izomorfismu grafů spadá do třídy NP, což není správný směr, kterým bych se rád vydal. Pro porovnávání molekul je vhodnější volbou použitím tanimoto koeficientu [35].

$$T = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (3.1)$$

Tanimoto koeficient je definován v rovnici 3.1. Nabývá hodnot od 0 do 1. Porovnávání pomocí tanimoto koeficientu se spojuje s otisky (binárními vektory) jako reprezentací, kde 1/0 indikuje, zda sekvence obsahuje určitý fragment nebo neobsahuje. Je potřeba mít zdefinovanou sadu fragmentů, nad kterou se otisk provádí. V případě přidání nového bloku, by se musel otisk pro každou sekvenci v databázi přepočítat. Granularita může být různá, ale logicky se nabízí použít jako fragmenty stavební bloky.

Vzhledem k uložení stavebních bloků k jednotlivým sekvencím v databázi, není nutné dělat otisky. Pro výpočet průniku použijí SQL dotaz nad databází s výpočtem tanimoto koeficientu, a následně vyberu nejlepší možný výsledek. Na následujícím příkladu ukazují, jak se vypočte tanimoto koeficient pro beauverolide A a B. Obě sekvence jsou cyklické. Beauverolide A se rozpadá na [Val]-[Val]-[Phe]-[C10:0-Me(4)-OH(3)]. Beauverolide B se rozpadá na [Val]-[Phe]-[Ile]-[C10:0-Me(4)-OH(3)]. Z čehož plyne, že

$$|A| = 3 \quad (3.2)$$

$$|B| = 4 \quad (3.3)$$

$$|A \cap B| = 3 \quad (3.4)$$

$$T = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} = \frac{3}{3 + 4 - 3} = \frac{3}{4} = 0.75 \quad (3.5)$$

Bohužel samotný koeficient tanimoto v některých případech nestačí. Jde o případy, kde se více struktur skládá například ze dvou typů bloků, ale jejich počet je různý. Takový případ nastává například u Desferri-ferrioxamine E a bisucaberinu. Oba mají jako základ bloky Hpd a Suc. Jejich celkový počet se ale liší. Pro Desferri-ferrioxamine E je celkový počet bloků šest a pro bisucaberin 4. Kdybych vyhledával jeden z nich, tanimoto vyjde pro oba případy stejně a následně je třeba dle celkového počtu bloků rozhodnout, který z výsledků je lepší. Toto rozhodnutí lze provést na základě porovnání celkového počtu bloků. Databázový dotaz pro získání podobnosti zde podrobněji rozeberu, je

nepatrně složitější, ale poměrně elegantní, celý je vidět na algoritmu 1. K dotazu potřebuji tabulky `sequence_family`, `sequence`, `s2f` a `b2s`. Dotaz se skládá ze dvou poddotazů.

Nejprve začnu s nejnvnitřnějším dotazem. Dotaz má za úkol vybrat všechny sekvence, které mají definovanou rodinu. Tím lze množinu sekvencí omezit a ušetřit počítání podobnosti nad molekulami, které ve výsledku nebudou zohledňovat, neboť nemají definovanou rodinu. Vyberu si tedy všechny identifikátory sekvencí z tabulky `sequence`. Pomocí `inner join` na tabulku `s2f` odstráním z množiny ty sekvence, které nemají definovanou rodinu. Dále musím ještě grupovat dle id sekvence, protože tabulka `s2f` mi roznásobí řádky (Pro jednu sekvenci je v ní $0 \dots * \text{id rodin}$).

Nejnvnitřnější dotaz použiji v nadřazeném poddotazu díky `inner join` pouze pro omezení se na správnou množinu. V tomto dotaze se již počítá `tanimoto` koeficient. Vycházím z tabulky sekvencí, ke které si připojím vazební tabulku na tabulku s bloky. Celý výpočet je skryt ve sloupci nazvaném `RN`. Výraz `count(distinct b2s.block_id)` spočte průnik sekvence v databázi a sekvence zadané uživatelem. Sekvence zadaná uživatelem se v dotazu projeví jako podmínka `b2s.block_id in (25, 12, 13, 24)`, což jsou id bloků, které jsou použity v zadané sekvenci. Tímto filtrem elegantně vypočtu hodnotu průniku, protože jsem omezen pouze na bloky v zadané sekvenci a pokud některá z nich není v porovnávané sekvenci, není v tabulce `b2s` k dané sekvenci, a do výběru vůbec nedostane. Jako vstup do dotazu se ještě používá číslo 4, což je počet bloků zadané sekvence ($|\{25, 12, 13, 24\}| = 4$). Počet bloků porovnávané sekvence je již v databázi předpočítaný ve sloupci `seq.unique_block_count` a `seq.block_count`. Tato čísla se pouze dosadí do vzorce pro výpočet `tanimoto` koeficientu a porovnání celkového počtu bloků pomocí `abs()` funkce. Celý tento výpočet je ještě obalen funkcí `row_number()`, která jen očísluje `tanimoto` koeficienty v sestupném pořadí. Za zmínku ještě stojí klauzule `having`, podle které se z výsledku odstraní sekvence s nižší hodnotou `tanimoto` koeficientu, než je určitá mez.

Nejvyšší úroveň dotazu slouží k omezení se na jeden řádek s nejlepším výsledkem `src.RN = 1`, ke kterému se dohledají názvy rodin a jejich id. Možná by se v tuto chvíli použití klauzule `having` zdálo zbytečné, když je výstup omezen na jeden záznam. Není tomu tak. V případě, že klauzule `having` omezí již výsledek poddotazu na nulový počet řádek, potom nenajdu žádnou shodu. Toto by samotné omezení na konci dotazu neumožnilo, i kdyby `tanimoto` koeficient byl sebestorší. Přejmenování sloupců na úplném konci slouží pouze pro transformaci názvů pro frontend. V dotazu se ještě vyskytuje omezení na kontejner `seq.container_id = 1`. To je pouze proto, že kontejner 1 je nejobsáhlejší veřejný kontejner, nad kterým dává největší smysl se dotazovat. Pro ukázkou přikládám obrázek 3.5, který ukazuje, jak by vypadal výstup s nejbližšími hodnotami, když jsem jako vstup zadal `cyclosporin`, pokud bych neomezil výsledky na nejlepší a neodstraňoval výsledky pod určitou hranicí.

Kdyby dotaz byl nad velkou databází již moc pomalý, připravil jsem ještě poměrně jednoduchou variantu pro nalezení nejlepší rodiny pomocí porovnávání názvu sekvence a názvu rodiny. Pro mnoho případů toto jednoduché porovnání, které je na úrovni SQL dotazu implementováno jen jako klauzule `like`, bude fungovat, ale existují i případy, kdy nebude. Například roseotoxin A, má rodinu destruxin, což zmíněné porovnání vyhodnotí špatně. Uvažoval jsem ještě o využití editační vzdálenosti na porovnání jmen, ale došel jsem k názoru, že by se jednalo o pomalejší a ne o moc efektivnější řešení než jednoduché `like`. Přestože by porovnání jmen vyřešilo překlepy v názvech, výše uvedený problém s roseotoxinem by stále přetrvával. Hlavním účelem tohoto porovnání má být rychlost, proto jsem zvolil obyčejný `like`. Zda aplikace použije verzi s `like` nebo `tanimoto` koeficientem, může administrátor přepínat za běhu aplikace. Tato informace se ukládá do databáze do tabulky `setup`.

3. NÁVRH A REALIZACE

```
select fam.id as value, fam.sequence_family_name as label
from (

select
  seq.id as sequence_id,
  row_number() over (order by
    count(distinct b2s.block_id) /
    :počet unikátních bloků: + seq.unique_block_count
    - count(distinct b2s.block_id)) desc,
    abs(:počet bloků: - seq.block_count) asc) as RN
from sequence seq
left join msb.b2s b2s on b2s.sequence_id = seq.id
and b2s.block_id in :Id unikátních bloků:
join (

select seq.id
from msb.sequence seq
  join msb.s2f on s2f.sequence_id = seq.id
  and s2f.family_id is not null
group by seq.id

) fam on fam.id = seq.id
where seq.container_id = 1
group by seq.id, seq.unique_block_count
having count(distinct b2s.block_id) /
  (:počet unikátních bloků: seq.unique_block_count
  - count(distinct b2s.block_id))
  >= 0.4

) src
join msb.s2f on s2f.sequence_id = src.sequence_id
join msb.sequence_family fam on fam.id = s2f.family_id
and fam.container_id = 1
where src.RN = 1
```

Algoritmus 1: SQL dotaz s výpočtem tanimoto koeficientu

Obrázek 3.5: Tanimoto - SQL ukázka

sequence_name	SEQUENCE_ID	BLOCK_INTERSECT	SEQUENCE_BLOCK_COUNT	BLOCK_COUNT_INPUT	TANIMOTO	FAMILIES
cydospirin I	23	7	7	7	1.0000	cydospirin
cydospirin B	16	6	6	7	0.8571	cydospirin
cydospirin D	18	6	6	7	0.8571	cydospirin
cydospirin G	21	6	7	7	0.7500	cydospirin
cydospirin H	22	6	7	7	0.7500	cydospirin
cydospirin A	2	6	7	7	0.7500	cydospirin
cydospirin C	17	6	7	7	0.7500	cydospirin
cydospirin F	20	5	6	7	0.6250	cydospirin
cydospirin E	19	5	6	7	0.6250	cydospirin
cydospirin L	24	5	7	7	0.5556	cydospirin
gramicidin A	33	3	5	7	0.3333	gramicidin
gramicidin C	3	3	6	7	0.3000	gramicidin
beauverolide H	31	2	4	7	0.2222	beauverolide
beauverolide I	1	2	4	7	0.2222	beauverolide
gramicidin S	34	2	5	7	0.2000	gramicidin
beauverolide A	25	1	3	7	0.1111	beauverolide
beauverolide D	28	1	3	7	0.1111	beauverolide
beauverolide La	32	1	4	7	0.1000	beauverolide
beauverolide B	26	1	4	7	0.1000	beauverolide
beauverolide E	29	1	4	7	0.1000	beauverolide
pseudocyclin B	10	1	6	7	0.0833	pseudocyclin

3.4 Zjednodušení rozhraní

Jak už jsem zmiňoval, rozhraní aplikace je zachovááno tak, aby bylo podobné rozhraní z Bbdgnc. Tedy horní menu s patičkou a uprostřed měnící se stránky. Vrchní část hlavní stránky s vyhledáváním nedoznala žádné změny. Co jsem změnil, jsou obecně tabulky nad nějakým zdrojem (sekvence, blok, modifikace a další). Tabulka obsahuje výpis dat a v posledním sloupci možné akce. Po kliknutí na konkrétní řádek lze téměř ve všech případech data uvnitř editovat a následně uložit. Výjimka je u editace sekvence, kdy se zobrazí formulář na hlavní stránce. Nahoře nad tabulkou se typicky vyskytuje formulář pro přidání nové struktury. Některé tabulky umožňují filtrování a řazení. Řazení je možné po kliknutí na záhlaví sloupce. Pro filtrování přibývá v tabulce pod hlavičkou řádek s textovými poli. Dále stojí za zmínku flash messages, což jsou hlášení po provedené akci. Dělí se hlavně na errors a ok zprávy, dále třeba indikaci načítání nebo warningny. Tato hlášení se typicky zobrazují nad tabulkou, ale někdy tomu tak být nemusí. Další komponentou je popup, vyskakovací okno, které překryje hlavní obrazovku a váže se k němu nějaká akce. Typicky se popup použije po kliknutí na delete, kde v popupu je nutné smazání potvrdit. Popup jsem také využil na zjednodušení rozhraní v případech, kdy je třeba editovat SMILES nebo je zobrazit. Například na hlavní stránce po rozpadu sekvence na bloky lze díky popupu editovat konkrétní blok na stejné stránce a není třeba otevírat stránku novou, jako tomu bylo u předchozí aplikace.

3.5 Naplnění daty

Vytvořil jsem veřejný kontejner s názvem Nonribosomal Peptides and Siderophores, do kterého jsem nahrával všechna data. Začal jsem 20 základními aminokyselinami, kterým jsem přiřadil rodinu Proteinogenic Amino Acids a se čtyřmi koncovými modifikacemi: Acetyl, Amidated, Ethanolamine a Formyl. Poté jsem vycházel z databáze programu CycloBranch. Nejdříve jsem začal s Nonribosomal Peptides, ke kterým jsem postupně dohledával další sekvence ze stejných rodin. S přidáváním sekvencí jsem nutně musel přidávat i bloky, ze kterých se skládají, a které nejsou obsaženy v základních 20 aminokyselinách. Mezi nahranými sekvencemi jsou například pseudocycliny, cyclosporiny, beauverolidy a mnohé další. Jako rozpadové vazby mají neribozomální peptidy vazbu -CO-NH-.

Další větší skupinou, kterou jsem nahrál jsou siderophory. Společným znakem pro siderophory je obsah železa. Jako sekvence se do databáze ukládají bez železa, neboť pro následující použití v hmotnostní analýze se s tímto železem pracuje odlišně. Narozdíl od neribozomálních peptidů mohou obsahovat k peptidovým vazbám -CO-NH- také esterové vazby -CO-O-. Mezi siderophory patří například ornibactiny, ferrioxaminy, fusarininy a další. K nim jsem také doplnil organismy dle stránky bertrandsamuel.free.fr/siderophore_base [36], kterou vytvořil Samuel Bertrand.

Poslední skupinou, kterou jsem nahrál, je skupina siderophorů a sekundárních metabolitů. Tato skupina má svůj vlastní kontejner Siderophores and Secondary Metabolites. Skupina je zvláštní tím, že neobsahuje SMILES. U těchto sekvencí není nutné dělat rozpad na bloky, protože v CycloBranchi byly tyto sekvence zkoumány pouze z hlediska celku. Vyšel jsem z databáze sekvencí právě pro CycloBranch. [37]

3.6 Identifikace polyketidů

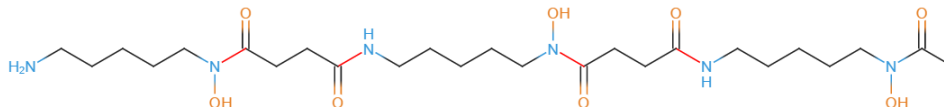
Polyketidy jsou struktury, které se vyskytují například v rodině siderophores. Polyketid se pozná tak, že obsahuje alespoň jeden blok, který nemá na obou koncích bodů rozpadu uhlík. Je to například Desferri-ferrioxamine B na obrázku 3.6. Skládá se opakovaně z bloků Hpd a Suc. Navíc na konci je připojena koncová modifikace Acetyl. Hpd je právě ten blok, díky kterému poznám, že jde o polyketid, ten, který má dusík na obou koncích. Tyto bloky, díky kterým lze identifikovat polyketid, mají ještě jednu zvláštnost. U standardního bloku se od formule odečte H₂O a získá se residuum (jak u formule, tak u hmotnosti). Zde se odčítá pouze 2H. Pro názornost se do jmen bloků přidává na začátek prefix (-2H). Tato skutečnost se promítá i do databáze.

Identifikace typu sekvence probíhá již ve SmilesDraweru při rozpadu sekvence na bloky. Rozšířil jsem tuto identifikaci o polyketidy tím, že k typům struktur jako je linear nebo cyclic přidám postfix -polyketide. Algoritmus pro detekci je následující:

1. Klasicky se detekuje, zda je struktura linear, cyclic apod.
2. Pomocí flagu se bude zjišťovat, zda se k bloku přidalo OH
3. Zajistí se, aby se k bloku přidalo OH maximálně 1x
4. Pokud se najde blok, ke kterému se nepřidalo žádné OH, označí se struktura jako polyketide

Algoritmus 2: Detekce polyketidů

Obrázek 3.6: Desferri-ferrioxamine B



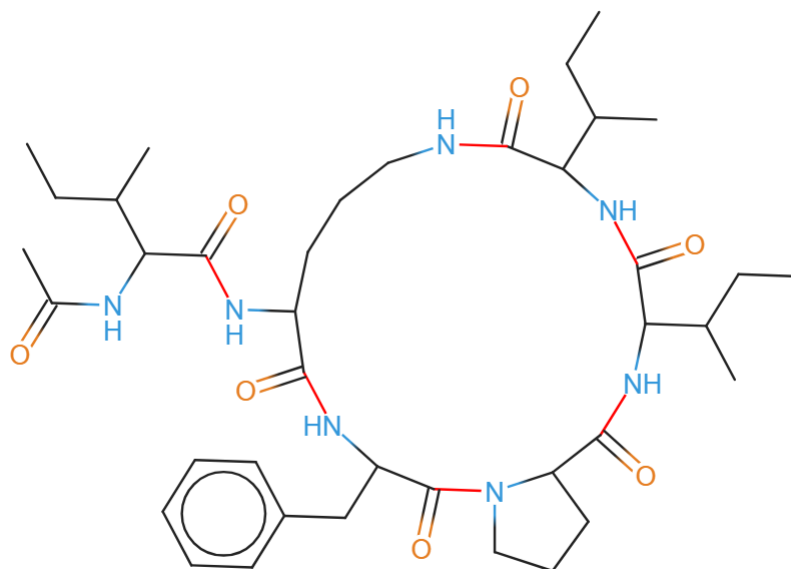
Tento jednoduchý algoritmus nefunguje pro všechny případy. Například linearizovaný pseudocyclin A označí jako siderophor, kvůli koncovému bloku Phenylalaninu, na který je přilepen Acetyl. Bod rozpadu mezi Phenylalaninem a Acylem není defaultně označen, protože je blízko konci sekvence. Aby algoritmus fungoval správně, je potřeba počítat i s těmito body rozpadu. Naopak nastává také situace, kdy je ještě koncová modifikace označena a díky ní se sekvence označí jako siderophor. Takovým příkladem je třeba Ethanolamine připojený na Tryptophan v gramicidinu A. Nejlepší způsob, který mě napadl jak toto řešit, je ignorovat koncové bloky, které jsou velikostně menší než čtyři dle počtu atomů (kromě vodíků).

3.7 Zápis sekvence

Pod zápisem sekvence se zde myslí zápis bloků použitých v sekvenci do textového řetězce. Pravidla pro strukturování tohoto textu jsou přebrána z programu CycloBranch [30]. Pro přehled zde uvádím, jak vypadá zápis pro pseudocyclin A, ten lze vidět na obrázku 3.7. Jeho zápisu sekvence odpovídá: [Ile]-[Pro]-[Phe]\([Orn]-[NAc-Ile]\)[Ile]. Každý blok je uzavírán do hranatých závorek a kulaté závorky spolu s lomítkem označují místo, kde se sekvence větví. První blok v závorce je místo rozvětvení, další bloky v závorce jsou už na samostatné větvi. Zápis nepočítá se sekvencemi, které mají více rozvětvení. Takové situace nastávají minimálně a CycloBranch je řeší jiným způsobem. Struktura, kdy tato definice zápisu nestačí, je například Desferri-ferrichrome A.

Do aplikace lze importovat sekvence z programu CycloBranch. Zde ale nejsou definované SMILES. Bloky se tak dají získat pouze ze zápisu sekvence, ale nikoli už grafická podoba sekvence, neboť nevím, ve kterých částech jsou bloky propojené. Bloky ukládám do databáze tak, abych neztratil informaci o zapojení v sekvenci. Ke každému bloku si tak ukládám 3 položky navíc - následující blok v pořadí, zda je blok na větvi a zda má následující blok na větvi. Tyto informace získám přímo ze zápisu a typu sekvence. Se zápisem sekvence pracuji podobně jako s textovým řetězcem, jen si představím, že nejmenší jednotka není znak, ale blok. Zápis pak procházím cyklem po blocích a tyto informace získávám. Struktura v tabulce b2s pro sekvenci pseudocyclin A je vidět v tabulce 3.1, kde jsem místo identifikátorů použil pro přehlednost zkratky bloků.

Obrázek 3.7: Pseudacyclin A



Tabulka 3.1: Struktura B2s pro pseudacyclin A

Blok	Následující	Připojená větev	Větev	Pořadí
Ile	Pro		0	0
Pro	Phe		0	1
Phe	Orn		0	2
Orn	Ile	NAc-Ile	0	3
NAc-Ile			1	4
Ile	Ile		0	5

Jedním z důvodů, kdy podobně procházím sekvenci, je potřeba smazat blok po rozpadu sekvence na bloky ze zápisu sekvence. Pokud se v sekvenci vyskytuje vícekrát stejný blok, nelze určit, který ze zápisu smazat, pouze na základě zápisu sekvence. Z tohoto důvodu do databáze ukládám ještě jeden zápis sekvence, ve kterém jsou zkratky bloků nahrazeny čísly (Pro stejnou zkratku jsou jiná čísla). Nejsou to id bloků, ta by byla totožná pro stejné bloky, ale jen očíslování bloků. Toto očíslování si také musím uložit do databáze. Na základě těchto informací jsem schopen smazat ze sekvence správný blok. Funguje to přibližně tak, že získám očíslování bloku, který se má smazat. Poté procházím zápis sekvence s očíslováním a získám pozici bloku v zápisu sekvence. Tuto pozici pak vymažu v originálním zápisu sekvence i zápisu s očíslováním.

Pokud nastane situace, že je třeba změnit zkratku bloku, který se vyskytuje v sekvencích, je nutné přepočítat zápis sekvence u všech těchto sekvencí. K přepočítání slouží právě výše zmíněná struktura. K samotnému vygenerování sekvence jsem použil SQL dotaz 3. Vychází z tabulky `b2s`, kde jsou uloženy potřebné informace. Jako základ jsem použil funkci `group_concat()` a jako separátor jsem zadal pomlčku. Pozor je třeba si dát na správné uzavorkování, pokud jde o sekvenci s větví. K tomu slouží obě funkce `row_number()` v poddotazu. Na konci je ještě třeba odstranit pomlčky kolem větví, k čemuž dopomůže funkce `replace()`.

Zápis sekvence dostal ještě jedno vylepšení při generování na úrovni SmilesDraweru. Ve staré verzi zápis nebral ohled na N-C směr sekvence. Zápis byl vygenerován směrem od konce sekvence, na který narazil algoritmus jako první. V nové verzi je definován začátek na N konec s pokračováním směrem ke konci C. Tyto konce se poznají dle zapojení bloků a rozpadových vazeb. Například lineární sekvence `Acv` na obrázku 3.8 se skládá ze tří bloků v N-C směru: 2-Aminohehexanedioic acid (`Amh`), Cysteine (`Cys`) a Valin (`Val`). N-C směr je dán tím, že Valin na rozpadové vazbě `-CO-NH-` obsahuje N a Cysteine C, kde N je ve "středu" sekvence a samotný Valin pak obsahuje `COOH` mimo peptidovou vazbu. Toto zapojení je pak opakováno i na dalším bodu rozpadu mezi `Val` a `Amh`. Výjimku z tohoto pravidla mají bloky polyketidů (například Putrescine), u kterých je na obě strany vazba N-C. V implementaci si při průchodu grafem při generování SMILES bloků označím směry a v následném průchodu nad malým grafem (blok je brán jako uzel grafu), který slouží pro vygenerování zápisu sekvence, identifikuji nově správný blok, kterým bude průchod začínat. Tento blok určím tak, že vezmu jen bloky, které mají jednoho souseda a dále se rozhodují dle směru na této jediné rozpadové vazbě. Pokud je C součástí bloku jako startovní uzel, vezmu tento blok. Pokud je N součástí, tento blok ignoruji. Blokem mohu začínat také v případě, že se jedná o polyketid (Na obou stranách je dusík). Zápis sekvence se už pak generuje dále, stejně jako tomu bylo v původní verzi (DFS). Vygenerovaný zápis sekvence je pro `Acv` `[Amh]-[Cys]-[Val]`.

3.8 Import a export

Import a export uměla již aplikace `Bbdgnc`. Soubory pro vstup a výstup do `CycloBranch` se nemění. Co se mění, je zpracování v aplikaci. Export se téměř nezměnil, až na věci spojené s výměnou frameworku. Jediné, co přibývá, je funkcionality stáhnutí všech exportovaných typů najednou v archivu. Zato u importu díky rozdělení na frontend a backend jsem změnil myšlenku nahrávání souboru. Importovaný soubor se předzpracuje na frontendu a na backend se pošle pomocí `HTTP POST`. Samotný vstupní soubor se tak vůbec nedostane na backend. Na frontendu probíhá základní kontrola, která ověřuje validitu daného řádku. Na backendu již probíhají kontroly, které na frontendu


```

select src.id,
  replace(
    replace(
      group_concat(
        concat(
          case
            when BRANCH_START = 1 then '\\('
            else ''
          end,
          '[', acronym, ']',
          case
            when BRANCH_END = 1 then '\\)'
            else ''
          end)
        order by sort asc separator '-')
      , '\\(', '\\(')
    , '\\)-', '\\)') as SEQUENCE
from (

  select seq.id, blc.acronym, b2s.sort,
  case
    when seq.sequence_type not in ('branched', 'branch-cyclic')
    then 0
    else row_number() over (order by
      case
        when branch_reference_id is not null
        then 0
        else 1
      end asc, is_branch asc, sort asc)
  end as BRANCH_START,
  case
    when seq.sequence_type not in ('branched', 'branch-cyclic')
    then 0
    else row_number() over (order by is_branch desc, sort desc)
  end as BRANCH_END
  from msb.sequence seq
  join msb.b2s b2s on b2s.sequence_id = seq.id
  join msb.block blc on blc.id = b2s.block_id
  where seq.id = 2

) src

```

Algoritmus 3: SQL dotaz s generováním zápisu sekvence

Tabulka 3.2: Možnosti vyhledávání na API třetích stran

API	jméno	SMILES	formule	hmotnost	id	přístup
Pubchem	ano	ano	ano	ne	ano	REST
ChemSpider	ano	ano	ano	ano	ano	REST
Norine	ano	ne	pomalé	ne	ano	REST
PDB	ne	ne	ne	ne	ano	REST
ChEBI	ano	ano	ano	ano	ano	SOAP

Tabulka 3.3: API MassSpecBlocks

Parametr	URI
jméno	/rest/container/{containerId}/name
podobnost (ze SMILES)	/rest/container/{containerId}/formula
formule	/rest/container/{containerId}/similarity
identifikátor	/rest/container/{containerId}/identifier

Ještě bych zde porovnal jednotlivé databáze obsahově. Všechny údaje jsou k datu 21.03.2021. Největší z databází jsou Pubchem a Chemspider. Pubchem obsahuje 110 miliónů záznamů o chemických strukturách, Chemspider 103 miliónů. ChEBI je databáze zaměřená spíše na menší struktury, jako jsou bloky, ale obsahuje i větší struktury. ChEBI obsahuje 58 829 struktur, které jsou kompletní, a dalších 75 452 ne zcela o anotovaných. Celkem tedy 133 981 záznamů. Norine je menší databáze, obsahuje 1740 struktur s 544 stavebními bloky. PDB obsahuje sekvence i bloky, ale pro sekvence neposkytuje SMILES. Dá se tak použít v MassSpecBlocks pouze pro hledání malých struktur. PDB obsahuje 175 759 struktur.

API poskytuje i MassSpecBlocks. Aplikaci lze samostatně používat bez frontendu a dotazovat se na backend aplikace. Pro vyhledávání poskytuje backend následující API 3.3. V tabulce jsou vidět URI pro vyhledávání. Všechny requesty jsou POST a obsahují JSON tělo s údaji pro vyhledávání. Zde se ještě zmíním o dokumentaci REST API. Samotný kód lze pomocí anotací dokumentovat a lze z něj vygenerovat stránku s dokumentací, kde si lze i jednotlivé dotazy zkusit. Tato stránka se nachází na adrese `:endpoint:/api/doc`. Výše zmíněné rozhraní pro hledání je tam popsáno pod názvem Finder a také tam jsou všechna ostatní rozhraní.

3.10 Frontend a Backend

Když už jsem se věnoval REST API na backendu v minulé sekci, shrnul bych zde navazující věci na komunikaci mezi frontendem a backendem. Backend vystavuje REST API, ke kterému přistupuji na frontendu pomocí metody `fetch()`. Typicky potřebuji CRUD dotazy nad zdrojem. Proto jsem v Reactu vytvořil abstraktní komponentu, kterou jsem nazval `ListComponent`. Ta má předpřipravené dotazy na zdroj, který specifikuji v podtřídě. Komponenta připravuje rozhraní pro zobrazení dat na obrazovku v tabulce. Dále komponenta spravuje flash messages, které zobrazují status odpovědi z backendu. Pokud je třeba udělat nad CRUD metodou jinou než defaultní akci, není problém metodu přetížít. Pokud je vhodné udělat jinou reakci, než zobrazit flash, pak také lze jen podstrčit metodě jiný callback, který se má provést. Dále komponenta pomáhá s filtrováním a řazením, kde se jen přidávají parametry do URL dotazu na backend.

Na straně backendu je třeba ošetřit všechny špatné vstupy. Z tohoto důvodu jsem navrhl strukturu, která zpracuje vstup (JSON), ověří správnost parametrů a případně je transformuje do použitelné formy. Připravil jsem třídu `AbstractStructure`, která předepisuje implementovat metody `checkInput()` a `transform()`. Nejprve se automaticky zpracuje JSON a do podtřídy `AbstractStructure` se vloží naparsovaná data, se kterými lze dále pracovat. První metoda zkontroluje vstupy a vrátí `Message`, což je objekt, který indikuje, zda jsou všechny vstupy správné. A pokud ne, obsahuje chybovou zprávu a také v závislosti na úspěšnosti operace HTTP status code. Pokud je vstup špatný, rovnou se vrátí JSON s obsahem v `Message`. Pokud je vstup správný, spustí se transformace vstupu pomocí metody `transform()`. Kromě transformace lze také z některých atributů dopočítat jiné hodnoty. Například u zadání nového bloku lze dopočítat ze SMILES chemickou formuli i monoisotopickou hmotnost. Z transformace je vrácen objekt, se kterým už lze dále pracovat. Pokud by někdo chtěl používat aplikaci a zároveň použít jiný nástroj než `CycloBranch` pro hmotnostní analýzu, může využít právě REST API na backendu, kde získá potřebná data, a následně si je převede na formát, který potřebuje pro použitou aplikaci.

Testování

4.1 Paralelizace

V bakalářské práci jsem zjistil, že je vhodné paralelizovat dotazy na API třetích stran, která se volají při rozpadu sekvence na bloky. Tato informace vycházela z měření rychlosti od doby stisknutí tlačítka „Building Blocks“ po vykreslení všech stavebních bloků na obrazovku v Bbdgnc. Dotazy jsem zkoušel paralelizovat na různých počtech vláken. V tabulce 4.1 uvádím měření na různých strukturách s různým počtem vláken, včetně měření z původní aplikace pro porovnání. Při měření se všechny bloky dohledávaly na Pubchemu. Pokud bych využil lokální databáze, což patří k běžnému postupu, ušetřilo by se síťové spojení a výsledek by byl zobrazen podstatně rychleji. To ovšem lze pouze pokud naleznou bloky v databázi MassSpecBlocks. Měření jsem provedl na stejném zařízení jako měření u tanimoto podobnosti na PC s Windows 10, vybaveném procesorem Intel Core i7 o frekvenci 1,8 GHz, 16 GB RAM, 500 GB SSD a 1 TB HDD. Připojení k Internetu bylo realizováno pomocí VDSL o maximální rychlosti 20Mb/s. Pro samotné měření času jsem využil vývojářskou konzoli v prohlížeči. Pro každou strukturu jsem měřil desetkrát a poté jsem vypočetl aritmetický průměr. Měření na Bbdgnc jsem změřil znovu, aby bylo porovnatelné kvůli výměně stroje. Oproti měření před 2 lety je zde nepatrné zrychlení, na které má vliv výměna stroje a možné změny na API Pubchemu.

4. TESTOVÁNÍ

Tabulka 4.1: Měření vyhledávání po rozpadu bloků

Sekvence	# bloků	# vláken	Aplikace	Čas [s]
Acv	3	1	Bbdgnc	10.431
Acv	3	1	MSB	7.273
Acv	3	2	MSB	6.410
Acv	3	4	MSB	3.876
Acv	3	6	MSB	3.912
Roseotoxin A	6	1	Bbdgnc	20.762
Roseotoxin A	6	1	MSB	13.720
Roseotoxin A	6	2	MSB	9.814
Roseotoxin A	6	4	MSB	5.831
Roseotoxin A	6	6	MSB	3.458
Vibriobactin	4	1	Bbdgnc	13.082
Vibriobactin	4	1	MSB	8.098
Vibriobactin	4	2	MSB	6.275
Vibriobactin	4	4	MSB	4.094
Vibriobactin	4	6	MSB	3.983
Pseudacyclin A	6	1	Bbdgnc	20.098
Pseudacyclin A	6	1	MSB	11.431
Pseudacyclin A	6	2	MSB	7.134
Pseudacyclin A	6	4	MSB	4.892
Pseudacyclin A	6	6	MSB	3.463
Cyclosporin A	11	1	Bbdgnc	24.241
Cyclosporin A	11	1	MSB	14.929
Cyclosporin A	11	2	MSB	10.081
Cyclosporin A	11	4	MSB	9.033
Cyclosporin A	11	6	MSB	5.794
Valinomycin	12	1	Bbdgnc	13.965
Valinomycin	12	1	MSB	7.020
Valinomycin	12	2	MSB	5.453
Valinomycin	12	4	MSB	3.686
Valinomycin	12	6	MSB	3.144

Tabulka 4.2: Měření rychlosti podobnosti

Sekvence	Čas [s]
Roseotoxin A	0.303
Vibriobactin	0.273
Pseudacyclin A	0.377
Cyclosporine A	0.522
Valinomycin	0.294

Z měření lze vypořádat, že oproti Bbdgnc je nárůst v rychlosti MassSpec-Blocks i na jednom vláknu. Předpokládám, že je to použitím jiných technologií. Když porovnávám měření pouze v MassSpecsBlocks, zrychlení na 2 vláknech je na různých strukturách v rozpětí cca 2-4 vteřiny. Každá vteřinka dobrá. Zrychlení na 4 vláknech je na tom obdobně. Při použití 6 vláken je na menších strukturách vidět, že se již nezrychluje, neboť všechna vlákna nejsou využita. U cyclosporinu A je vidět, že 6 vláken dosahuje nejlepšího času. Nicméně i tak nelze 6 vláken použít, protože v mnoha případech již Pubchem začal kvůli vytížení, dotazy odmítat. Podobná situace nastala i u 4 vláken, ale v méně případech. Z tohoto důvodu jsem nastavil paralelizaci pouze na dvě vlákna.

4.2 Měření podobnosti

Pro zajímavost jsem změřil, jak dlouho trvá dotaz pro výpočet podobnosti. Měřil jsem pouze rychlost vyhodnocování SQL dotazu nikoli již přenesení informací z backendu na frontend. Naplnil jsem databázi 20 000 sekvencemi pomocí funkce klonování sekvence. Tabulka b2s tak čítala 161 559 záznamů. Rychlost dotazu pro vyhledávání nejbližší rodiny bych zhodnotil jako dostačující. V tabulce 4.2 jsou vidět výsledky měření při vyhledávání různých sekvencí. Měření jsem provedl na PC s Windows 10, vybaveném procesorem Intel Core i7 o frekvenci 1,8 GHz, 16 GB RAM, 500 GB SSD a 1 TB HDD. Připojení k Internetu bylo realizováno pomocí VDSL o maximální rychlosti 20 Mb/s. Pro měření jsem použil nástroj MySQL WorkBench. Pro každou strukturu bylo měření provedeno desetkrát a poté byl vypočten aritmetický průměr.

4.3 Testy

V této sekci se budu věnovat testování aplikace. Aplikaci jsem testoval jak jednotkovými testy na backendu, tak jsem použil nástroj Sonar Cube [38] pro odhalování code smells. Také jsem využil Github Actions pro spouštění testů, vytvoření a naplnění databáze, spuštění frontendu, což je samo o sobě také test. Dále jsem změřil rychlost aplikace při dohledávání stavebních bloků na Pubchemu a rychlost výpočtu podobnosti sekvencí, čemuž jsem se věnoval v dřívějších sekcích.

Začnu od testů ve SmilesDraweru. Zde jsem testy pouze rozšiřoval o testy na detekci polyketidů a detekci N-C směru a upravoval původní testy. SmilesDrawer používá testovací framework Jasmine [39]. Pro spouštění testů po commitu na git se automaticky testy spouští pomocí Travis CI [40]. Travis je podobný nástroj Github Actions [41]. Travis je u SmilesDraweru použit, protože v době vytvoření forku SmilesDraweru, ještě Github Actions neexistovaly. Github Actions jsem u MassSpecBlocks použil z toho důvodu, že jsou přímo na GitHubu, kde je zdrojový kód a není třeba dávat přístupová práva do repositáře další třetí straně. Na backendu jsem přebral některé testy z Bbdgnc pro parsování SMILES a sestavení grafu struktury. Tyto testy jsem rozšířil na testy nad databází a REST API. Pro testování na backendu používám PHPUnit [42]. Pro testování na frontendu stačí spustit `npm test`.

Samotný zdrojový kód je uchováván na Githubu v několika repositářích. Jako rozcestník slouží hlavní repositář github.com/privrja/MassSpecBlocks, zde lze najít kromě odkazů na frontend a backend repositáře také informace o datovém modelu, nasazení, tutoriál a další.

Deployment

Deployment je v případě MassSpecBlocks poměrně komplikovaná operace. Je potřeba nasadit frontend i backend a mít zprovozněnou databázi. Všechny tři části běží na stránkách <https://ms.biomed.cas.cz>. K tomu jsem ještě nasadil frontend na službu firebase, který používá backend na biomedu. Pro vlastní potřeby si může kdokoli nasadit aplikaci kam bude potřebovat, veškeré informace o aplikaci lze nalézt na <https://github.com/privrja/MassSpecBlocks>. Adresy k nasazeným endpointům:

Backend <https://ms.biomed.cas.cz/msb-backend/public/index.php/rest>

Frontend Biomed <https://ms.biomed.cas.cz/msb>

Frontend Firebase <https://my-app-9b30f.web.app>

5.1 Deploy backend

Pro nasazení backendu je nejprve třeba stáhnout zdrojové kódy z GitHubu.

```
git clone https://github.com/privrja/thesis.git
```

Záleží pro jaké prostředí (prod/dev) nasazují, dále budu popisovat postup nasazení pro produkční prostředí. Většina nastavení lze nakonfigurovat v souboru `.env`. Pro produkční nastavení doporučuji následující hodnoty:

```
APP_ENV = prod
```

```
APP_DEBUG = 0
```

```
APP_SECRET = d399e0294744d42591df25fd7e155283
```

```
SHELL_VERBOSITY = -1
```

```
DATABASE_URL = mysql://usr:pass@address:port/database?serverVersion=8.0
```

Proměnná `APP_ENV` nastavuje prostředí `prod/dev`. `APP_DEBUG` přepíná debugovací mód `0/1`. `APP_SECRET` by měl být náhodný hexadecimální řetězec dlouhý 32 znaků. Proměnná `SHELL_VERBOSITY` slouží pro logování. Její přípustné hodnoty jsou `-1` (Error), `1` (Notice), `2` (Info), `3` (Debug). Log se ukládá do adresáře `/var/log/`. Pro připojení databáze slouží `DATABASE_URL`. Výše je popsána struktura pro připojení v `MySQL`, pro připojení k `MariaDB` se liší syntaxe na konci s verzí databáze. Například `?serverVersion=mariadb-10.4.17`. Doporučuji spustit příkaz, pro vytvoření souboru `.env.local.php`. [43]

```
composer dump-env prod
```

V souboru `src/Constant/Constants.php` je také třeba nastavit adresu endpointu, kde se backend nachází a také je možné nastavit čas, po kterém bude uživatel automaticky odhlášen. Nutné je též nainstalovat závislosti. Pro instalaci závislosti používám `composer` [44].

```
composer install --no-dev --optimize-autoloader.
```

Variantu `--no-dev` lze samozřejmě vynechat pro testovací účely. Dalším krokem je vyčistit cache.

```
php bin/console cache:clear
```

Pak už jen nahrát kód na server. Aby byla v aplikaci nějaká data, je potřeba je nahrát do databáze. K tomuto účelu jsem vytvořil `create-insert SQL` skript, který je uložený na `GitHubu`. Skript stačí spustit nad databází, ten vytvoří všechny potřebné tabulky a nahraje do nich data. Poté už jen doporučuji změnit hesla uživatelů, buď přímo v `create-insert` skriptu nebo později pomocí nasazeného frontendu.

5.2 Deploy frontend

Opět je potřeba naklonovat zdrojový kód z `GitHubu`.

```
git clone https://github.com/privrja/thesis-frontend-react.git
```

Pro instalaci závislostí, `build`, testování na frontendu používám `npm` [45]. K instalaci produkčních závislostí slouží příkaz:

```
npm install --production
```

V souboru `src/constant/Constants.ts` lze nastavit proměnné `ENDPOINT` a `URL_PREFIX`. `ENDPOINT` je pro nastavení adresy backendu a `URL_PREFIX` pro nastavení prefixu adresy frontendu, pokud není frontend na hlavní stránce.

Pro hledání na ChEBI a Norine jsem musel nastavit adresu proxy serveru. Využil jsem projektu `cors-anywhere` [46], který jsem si naklonoval a proxy nasadil na Heroku [47]. Pokud budete nasazovat vlastní server, doporučuji vytvořit si svoji vlastní proxy. Nastavení proxy lze nalézt ve zdrojovém kódu v souborech `src/finder/NorineFinder.ts` a `src/finder/ChebiFinder.ts` na proměnné `ENDPOINT_URI`. Následně stačí spustit následující:

```
npm run build
```

Toto vytvoří složku `build`, ve které je vše potřebné pro nahraní na server. Ještě bych se zmínil o způsobu routování. Konkrétně mám na mysli `HashRouter` a `BrowserRouter` v Reactu. `BrowserRouter` je preferovaná varianta, ale například na serveru `ms.biomed.cas.cz` není povoleno přepisování URL, což tento router využívá, a proto nemůže být použit. Kromě této těžkosti může také nastat problém pokud uživatel napíše do internetového prohlížeče URL adresu „natvrdo“. Na biomedu je vrácena 404 stránka, protože se uživatel snaží přistoupit na adresu, kde nic není. `HashRouter` nepotřebuje přepisování URL, na druhou stranu ji má „ošklivější“. Díky přidání `#` do URL však výše popsané problémy nenastanou.

Co tedy aplikace umí?

6.1 Popis funkcionalit

Jednou z mnoha funkcionalit je vyhledávání struktur v externích chemických databázích. Vyhledávat je možné přímo na hlavní stránce aplikace i bez přihlášení či registrace. Stačí vybrat zdrojovou databázi a poté zadat, dle kterého parametru chci vyhledávat. Po kliknutí na tlačítko Find mohou nastat tři možnosti: byla nalezena pouze jedna sekvence a ta se zobrazí, bylo nalezeno více výsledků a zobrazí se seznam, ve kterém je nutné jeden vybrat. Třetí možností je, že nebude nic nalezeno.

Nad již vyhledaným blokem mohu provádět dvě operace se SMILES. Pokud se jedná o isomeric SMILES, mohu zmáčknutím tlačítka Generic SMILES odstranit isomerní prvky SMILES (jednosměrná operace). Pokud stisknu tlačítko Unique SMILES, tak se SMILES přepíše na Unique SMILES použitá pro uložení do databáze pro porovnávání struktur.

Další funkcí přímo na hlavní stránce je rozpad sekvence na stavební bloky. Pro vstup je potřeba vyplnit SMILES a označené bloky rozpadu na vykresleném obrázku. Body rozpadu jsou označeny červeně a kliknutím na hranu lze daný bod odebrat či přidat. Po stisknutí tlačítka Build Blocks se naleznou jednotlivé bloky a zobrazí se na stránce spolu s typem sekvence a dalšími informacemi. Zde už závisí na tom, zda je uživatel přihlášen, či není. V prvním případě, použiji aktuálně jím vybraný kontejner (výběr kontejneru se dá změnit na stránce Containers). Pokud kontejner neobsahuje bloky z rozpadu, zkusí se vyhledat na Pubchemu. Samotné dohledání bloků může chvíli trvat. Nejdříve se zobrazí SMILES s obrázkem a teprve poté se na výstupu objeví ostatní atributy. Po kliknutí na náhledový obrázek se obrázek zvětší na celou obrazovku. Po kliknutí na tlačítko Edit se přes stránku otevře JSME editor a blok v něm lze editovat. Při kliknutí do řádku tabulky se změny popisky na textboxy blok lze editovat. Blok z databáze edituji na všech sekvencích, naproti tomu pokud bych chtěl editovat konkrétní blok pouze v této sekvenci, musím změnit acronym. Všechny stejné bloky v rámci sekvence se defaultně

editují najednou při editaci jednoho z nich. Tuto možnost lze online během používání zrušit zaškrtnutím volby nad tabulkou. Pro zrušení této možnosti je zobrazen nad tabulkou bloků checkbox, který lze online přepínat. K sekvenci lze přiřadit jednu nebo více rodin. Rodiny se do vstupního pole předvyplní dle výsledku algoritmu pro výpočet podobnosti sekvencí. Kromě rodiny lze také přiřadit k sekvenci jeden a více organismů. Dále lze přiřadit koncové modifikace v závislosti na typu sekvence. Po konečných úpravách lze sekvenci pomocí tlačítka Save uložit do vybraného kontejneru.

Stránky Sequences, Blocks a Modifications jsou seznamy daných struktur v rámci vybraného kontejneru. Zde lze filtrovat, řadit, přidat struktury nové, editovat současné a také struktury smazat. Navíc dle zobrazených struktur mohou být na výběr specializované funkce. Například u bloků je po stisknutí tlačítka Usage získán seznam sekvencí použitých v daném bloku včetně informace, kolikrát je tato sekvence v bloku zastoupena. Nebo při použití tlačítka show zobrazím náhled bloku. Sekvenci lze například klonovat. Aplikace vytvoří novou sekvenci a uloží ji do kontejneru. Její nepatrnou editací mohou vytvořit novou sekvenci ze stejné rodiny. Tato funkce je užitečná, protože některé sekvence se liší například jen v jednom bloku, který lze vyměnit za jiný.

Na stránce Container jsou zobrazeny dva seznamy kontejnerů. První obsahuje pouze kontejnery s přímým oprávněním ke vstupu, druhý obsahuje kontejnery veřejné. Na této stránce se lze přepínat mezi kontejnery tlačítkem Select. Dále zde lze vyexportovat daný kontejner do textového souboru pro program CycloBranch. Zvolit lze hned několik možností exportu dle modifikací, bloků, mergovaných bloků, sekvencí nebo exportovat vše. Mergované bloky je speciální zápis v textovém souboru, kdy jsou bloky se stejnou formulí shlukovány na stejný řádek. U kontejnerů, ke kterým je uživatel přiřazen, lze rozkliknout detail kontejneru. Na detailu kontejneru je zobrazen seznam uživatelů s přístupovými právy, rodiny bloků a sekvencí. Pokud mám dostatečná práva mohu i vytvářet/editovat/mazat rodiny a přiřazovat/měnit/odebírat uživatele a jejich práva v kontejneru.

Do aplikace lze také naopak data importovat. Očekává se formát programu CycloBranch. Mohu importovat opět dle modifikací, sekvencí, bloků a merge bloků. Data se naimportují do vybraného kontejneru. Pokud se při importu nějaká struktura nepovede naimportovat, zobrazí se v textboxu s chybou, proč se import nepovedl. Uživatel tak může struktury opravit a nahrát znovu jen ty, které se nahrát nepovedly.

6.2 Ukázka

Jako názornou ukázkou pro demonstraci aplikace bych zde prošel přidání několika sekvencí. Začnu tím, že se přihlásím a půjdu na stránku s kontejnery, kde naklonuji kontejner Proteinogenic Amino Acids a přejmenuji si ho například na My container. Tento krok je vidět na obrázku 6.1. Dále se podívám na stránku s bloky 6.2. Zde je základních 20 aminokyselin. V tabulce lze řadit a filtrovat dle libosti. Řekněme, že budu chtít přidat do nově vytvořeného kontejneru lineární sekvenci Acv. Přejdu na hlavní stránku a zkusím ji vyhledat. Vyhledání dle jména na Pubchemu našlo mnohé výsledky 6.3, ovšem správná sekvence je pouze ta první. Pokud kliknu na malý náhled sekvence, zobrazí se velký náhled sekvence přes celou obrazovku. Pokud kliknu na identifikátor, otevře se nová stránka se sekvencí nad vyhledávanou databází. Pokud najedu myší nad konkrétní zobrazenou strukturu, zobrazí se mi její jméno. Jako poslední možnost lze sekvenci vybrat, po kliknutí na Select, což také provedu. Následně se sekvence vykreslí v levé části hlavní stránky a na pravé části se zobrazí nalezené informace o sekvenci. V náhledu sekvence se již vykreslily body rozpadu (červeně). Je tedy vidět dopředu, že sekvence se skládá ze tří bloků. V tuto chvíli má uživatel možnost změnit toto označení bodů rozpadu pomocí kliknutí myší na konkrétní hranu v náhledu. U Acv tomu není třeba. Následně kliknu na tlačítko Build Blocks. Pod aktuálním náhledem sekvence se zobrazí další informace 6.4. Jde konkrétně o počet bloků, typ struktury, její zápis. Dále políčka pro rodiny, organismy a modifikace. Pod těmito informacemi je tabulka s konkrétními bloky. Aplikace automaticky rozpoznala 2 základní bloky, kterými jsou Valine a Cysteine. Poslední blok nebyl nalezen v lokální databázi. Byl ale nalezen na Pubchemu, odkud se i převzala informace o jméně. Zkratka bloku se pokusila vygenerovat ze jména bloku. V tomto případě se jmenuje 2-a, což není úplně šťastné pojmenování. Proto si zkratku přejmenuji například na Amh. Pro editaci stačí poklepat myší na položku, kterou chci editovat, na daném řádku tabulky se zobrazí formulář pro editaci. Po změně bloku kliknu na tlačítko Update a tím potvrdím nově zadané informace. Zkratka bloku se automaticky změnila i v zápisu sekvence nad tabulkou s bloky. První sloupec v tabulce zůstává pro Amh stále prázdný. To je správné chování. Tento sloupec pouze indikuje, zda je daný blok uložen v databázi nebo se uloží jako nový. Pokud je sloupec prázdný, bude se ukládat nový blok. V opačném případě obsahuje zkratku bloku v databázi. Může se stát, že algoritmus blok špatně rozpozná a přesto v databázi daný blok existuje. V tomto případě lze ručně otevřít editaci bloku v tabulce a vybrat danou zkratku bloku v prvním sloupečku. V popisovaném případě to není třeba. Jediné co zbývá, je uložení sekvence do databáze pomocí tlačítka Save.

Dalším zvoleným příkladem vložení do kontejneru může být fusarinin. Tentokrát nebudu sekvenci ukládat, ale pro demonstraci dalších funkcí se přepnu do základního kontejneru Nonribosomal Peptides and Siderophores. Vyhledám fusarinin na Pubchemu dle jména. Na výběr mám čtyři struktury 6.5. Hledání

v prvním případě zobrazuje blok, ten není v tuto chvíli důležitý. Jako další zobrazuje dvě cyklické struktury. Jedná se konkrétně o fusarinin C, kde první varianta je bez železa a druhá se železem. Jako poslední byl nalezen lineární fusarinin B. Vyberu si cyklický fusarinin C bez železa. Doporučuji ukládat struktury vždy bez železa, neboť do programu CycloBranch se následně toto železo zadává jiným způsobem. Vyberu tedy druhou strukturu a provedu rozpad na bloky. Rozpad nám ukazuje, že struktura obsahuje šest bloků. Některé bloky se v sekvenci opakují. Defaultně pokud edituji nějaký blok, editují se v tabulce všechny stejné bloky naráz. Toto lze změnit pomocí volby nad tabulkou s popisem Edit same blocks together. Nad tabulkou je také vidět, že se díky algoritmu podobnosti předvyplnilo políčko pro zadávání rodiny. Správně totiž algoritmus poznal, že podobná struktura je v základním kontejneru uložena a do pole Family, proto předvyplnil hodnoty: siderophores a fusarinins.

Jako další strukturu bych vyzkoušel některý polyketid, ukáži třeba putre-bactin. Opět strukturu vyhledám, zkusme například tentokrát hledat na ChemSpideru dle jména. Pokud nemám vytvořený účet na ChemSpideru nebo nemám zadaný apikey v MassSpecBlocks, nemohu na ChemSpideru vyhledávat. Na tento fakt mě upozorní warning na hlavní stránce. Pokud účet na ChemSpideru mám, nebo si jej založím, apikey lze nastavit v záložce Settings do MassSpecBlocks. Po vyhledání sekvence provedu rozpad na bloky 6.6. Tentokrát se identifikoval typ sekvence správně jako cyclic-polyketide. Sekvence je polyketid kvůli bloku Putrescine, který má na obou koncích dusík. Ten je v sekvenci obsažen dvakrát střídavě s blokem Succinic semialdehyde.

Jako poslední bych chtěl demonstrovat sekvenci s koncovými modifikacemi, například gramicidin C. Vyhledám ho na Norine. Před provedením rozpadu na bloky si přeo značím body rozpadu, kvůli koncovým modifikacím 6.7. Na N konci je vidět Ethanolamine, který je označen, a na C konci je Formyl, který není defaultně označen. Tento bod rozpadu označím ručně pomocí myši a poté provedu rozpad 6.8 (gramicidin obsahuje mnoho bloků, na obrázku jsou vidět pouze první tři). Koncové modifikace se zobrazí v tabulce s bloky. Z této tabulky je jednoduše odstraním pomocí tlačítka Remove a následně kliknu nad tabulkou na plus u N a C modifikace, kde vyplním koncové modifikace tak, že je vyberu z databáze. Políčka se pak automaticky vyplní a zablokují k úpravě. Také si lze všimnout, že Tryptophan se nepředvyplnil do tabulky z databáze. Tento konkrétní blok v sekvenci má nepatrně odlišný zápis v cyklické části bloku u dvojných vazeb. Tryptophan z databáze lze vybrat pomocí editace prvního sloupce u jakéhokoliv Tryptophanu v tabulce, díky společnému editování se změní všechny výskyty.

Obrázek 6.1: Kontejnery ukázka

Create new container

Container name: [Create new container](#)

Your containers - 1 rows

Container name	Visibility	Mode	Is selected	Actions				
My container	PRIVATE	RWM	Yes	Select	Details	Clone	Export	Delete

Public containers - 3 rows

Container Name	Is selected	Actions		
Nonribosomal Peptides and Siderophores	No	Select	Clone	Export
Proteinogenic Amino Acids	No	Select	Clone	Export
Siderophores and Secondary Metabolites (MS)	No	Select	Clone	Export

Obrázek 6.2: Bloky ukázka

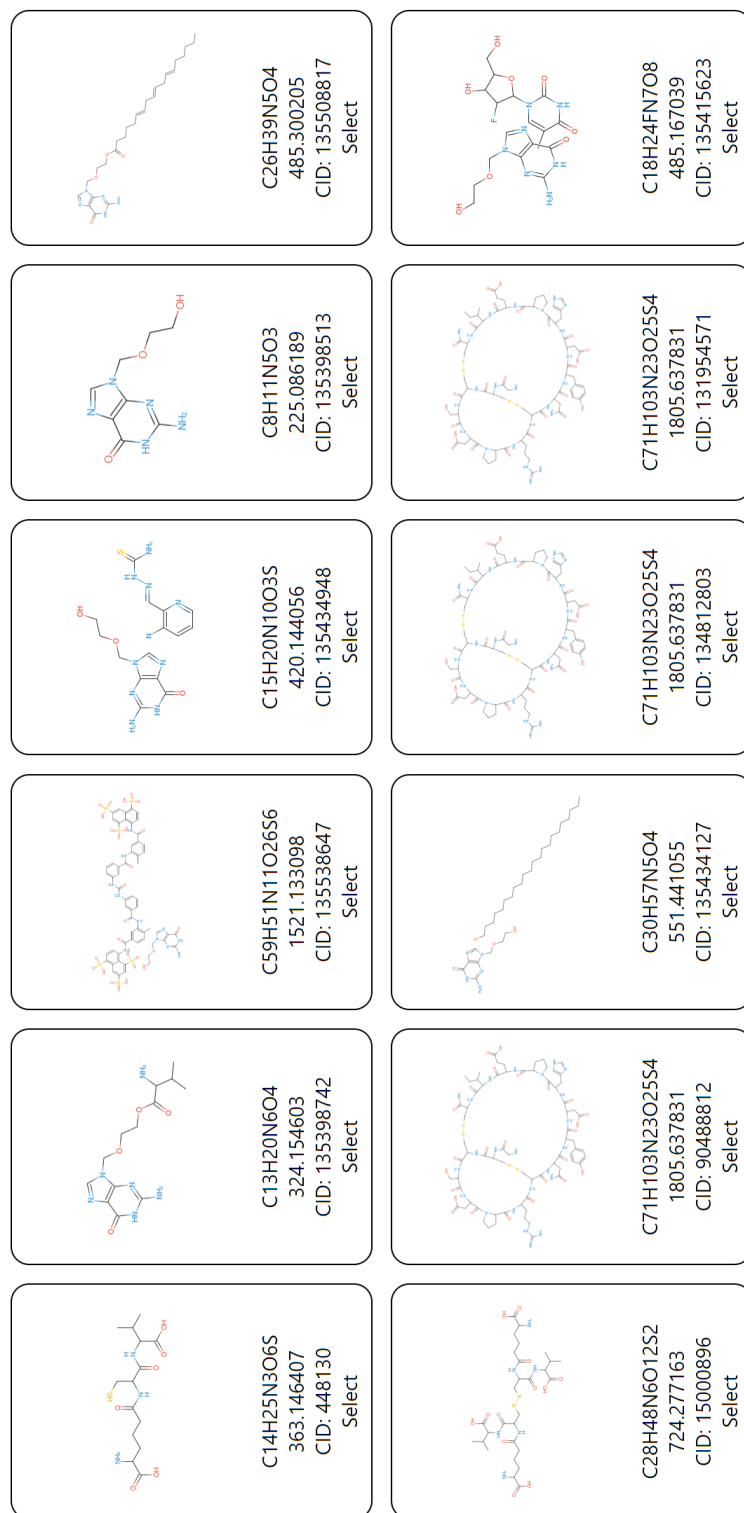
Create new block

Name: Acronym: Formula: Residue: SMILES: Family:

List of blocks - Proteinogetic Amino Acids - 20 rows

Name	Acronym	Residue	Mass	Losses	Family	SMILES	Identifier	Actions
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="button" value="Filter"/> <input type="button" value="Clone"/>
Alanine	Ala	C3H5NO	71.037114		proteinogetic amino acids	CC(N)(O)=O	CID: 5950	<input type="button" value="Editor"/> <input type="button" value="Show Usage"/> <input type="button" value="Clone"/> <input type="button" value="Delete"/>
Arginine	Arg	C6H12N4O	156.101111		proteinogetic amino acids	NC(CCCN=C(N)N)(O)=O	CID: 6322	<input type="button" value="Editor"/> <input type="button" value="Show Usage"/> <input type="button" value="Clone"/> <input type="button" value="Delete"/>
Asparagine	Asn	C4H6N2O2	114.042927		proteinogetic amino acids	NC(CCN=O)(O)=O	CID: 6267	<input type="button" value="Editor"/> <input type="button" value="Show Usage"/> <input type="button" value="Clone"/> <input type="button" value="Delete"/>
Aspartic acid	Asp	C4H5NO3	115.026943		proteinogetic amino acids	NC(C(=O)O)(O)=O	CID: 5960	<input type="button" value="Editor"/> <input type="button" value="Show Usage"/> <input type="button" value="Clone"/> <input type="button" value="Delete"/>
Cysteine	Cys	C3H5NOS	103.009184		proteinogetic amino acids	NC(CS)(O)=O	CID: 5862	<input type="button" value="Editor"/> <input type="button" value="Show Usage"/> <input type="button" value="Clone"/> <input type="button" value="Delete"/>
Glutamine	Gln	C5H8N2O2	128.058578		proteinogetic amino acids	NC(CCN=O)(O)=O	CID: 5961	<input type="button" value="Editor"/> <input type="button" value="Show Usage"/> <input type="button" value="Clone"/> <input type="button" value="Delete"/>

Obrázek 6.3: Acv výsledky hledání

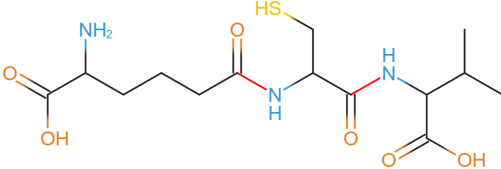


6. CO TEDY APLIKACE UMÍ?

Obrázek 6.4: Acv ukázka

MassSpecBlocks
Containers
Sequences
Blocks
Modifications
Import
Settings
Logout

Success! Done
Notice - Create new sequence



Proteinogenic Amino Acids

Database:

Search by:

Name:

acv

SMILES:

Molecular Formula:

Monoisotopic Mass:

Identifier:

Find
Edit

Generic SMILES
Unwrap SMILES

Build Blocks
Save

Sequence - 3 blocks

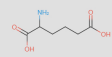
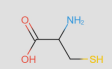
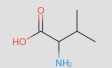
Type: Sequence:

Family: Organism:

Edit same blocks together

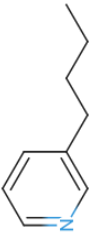
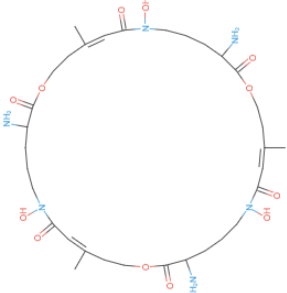
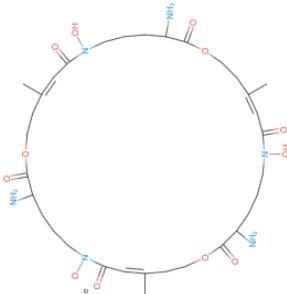
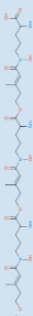
N-terminal modification +

C-terminal modification +

MSB acronym	Preview	Acronym	SMILES	Name	Formula	Mass	Losses	Identifier	Actions
		2-a	<chem>NC(CCC(=O)O)C(=O)O</chem>	2-Aminohexanedioic acid	C6H9NO3	143.058243		CID: 469	Edit Delete
Cys		Cys	<chem>NC(CS)C(=O)O</chem>	Cysteine	C3H5NO2S	103.009184		CID: 5862	Edit Delete
Val		Val	<chem>CC(C)C(N)C(=O)O</chem>	Valine	C5H9NO2	99.068414		CID: 6287	Edit Delete

2020 - 2021 | Jan Pivratsky | [References](#) | [Terms and conditions](#) | Logged as privra

Obrázek 6.5: Fusarinin výsledky hledání

	<p>C9H13N 135.104799 CID: 10874 Select</p>
	<p>C33H54N6O12 726.379971 CID: 102576136 Select</p>
	<p>C33H54FeN6O12 782.314907 CID: 134745197 Select</p>
	<p>C33H56FeN6O13 800.325472 CID: 134717664 Select</p>

6. CO TEDY APLIKACE UMÍ?

Obrázek 6.6: Putrebactin ukázka

MassSpecBlocks
Containers
Sequences
Blocks
Modifications
Import
Settings
Logout



Success! Done

Notice - Create new sequence

Nonribosomal Peptides and Siderophores

Database:

Search by:

Name:

SMILES:

Molecular Formula:

Monoisotopic Mass:

Identifier:

Find	Edit
Generate SMILES	Unwrap SMILES
Build Blocks	Save

Sequence - 4 blocks

Type: Sequence:

Family: Organism:

Edit same blocks together

MSB acronym	Preview	Acronym	SMILES	Name	Formula	Mass	Losses	Identifier	Actions
Hbd		Hbd	NCCCCNO	(-2H) N-hydroxy-1,4-butanediamine	C4H10N2O	102.079313		CID: 24883439	Edit Remove
Suc		Suc	OC(=O)CCC=O	Succinic semialdehyde	C4H4O2	84.021129		CID: 1112	Edit Remove
Hbd		Hbd	NCCCCNO	(-2H) N-hydroxy-1,4-butanediamine	C4H10N2O	102.079313		CID: 24883439	Edit Remove
Suc		Suc	OC(=O)CCC=O	Succinic semialdehyde	C4H4O2	84.021129		CID: 1112	Edit Remove

2020 - 2021 Jan Privratsky | [References](#) | [Terms and conditions](#) | Logged as privra

Obrázek 6.8: Gramicidin C ukázka - bloky

Sequence - 16 blocks

Type: linear Sequence: [Val][Gly][Ala][Leu][Ala][Val][Val][Val][Trp][Leu][Leu][Trp][Leu][Trp][I]

Family: gramicidins x | v Organism: Select...

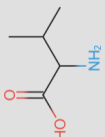
Edit same blocks together

N-terminal modification

Select Modification: Formyl Name: Formyl Formula: CO N-terminal: C-terminal:

C-terminal modification

Select Modification: Ethanolamine Name: Ethanolamine Formula: H5C2N N-terminal: C-terminal:

MSB acronym	Preview	Acronym	SMILES	Name	Formula	Mass	Losses	Identifier	Actions
Val		Val	CC(C)C(N)C(=O)O	Valine	C5H9NO	99.068414		CID: 6287	Editor Remove

6.3 Nápady do budoucna

Aplikace umí poměrně dost věcí, ale i tak se ukazují další možnosti vylepšení. Jedno z vylepšení by se mohlo týkat zvláštních sekvencí, které momentálně aplikace ne zcela podporuje. Jedná se například o sekvence, jako jsou ferri-chromy 6.9, které mají tři větve, nebo také coprogeny 6.10, ty většinou obsahují dva bloky tvořící cyklus a ostatní bloky pokračují z těchto bloků na strany. S těmito bloky se i nestandardním způsobem vypořádává CycloBranch. Zde by bylo třeba více rozmyslet detaily provedení.

Bylo by také zajímavé zamyslet se nad možností přidat alternativní odkazy na další zdroje struktur. Například mít jako hlavní zdroj ten současný a uchovávat alternativní odkazy a DOI. DOI lze ke struktuře přidat i aktuálně, ale ztratím tím informaci o původním zdroji.

Poměrně složitějším vylepšením by mohla být funkce s opačnou funkcí k rozpadu na bloky. Tedy, že by bylo možné vybrat jednotlivé bloky a určit jim grafovou strukturu, například pomocí zápisu sekvence, a program by se pokusil poskládat bloky do sekvence. Tuto operaci ztěžuje neznámé místo napojení bloků. Jeden blok obsahuje typicky 1-3 "rozpuštěné" peptidové/esterové vazby a je tak hned několik možností, jak sekvenci poskládat v závislosti na počtu bloků a možnostech jejich napojení. Na spojování také bude mít vliv N-C směr sekvence, který by měl omezit počet možností propojení, pokud se ho budu chtít držet. K zamýšlení je, jak by šlo tohoto poskládání dosáhnout efektivně a dosáhnout ho s co nejmenší chybovostí.

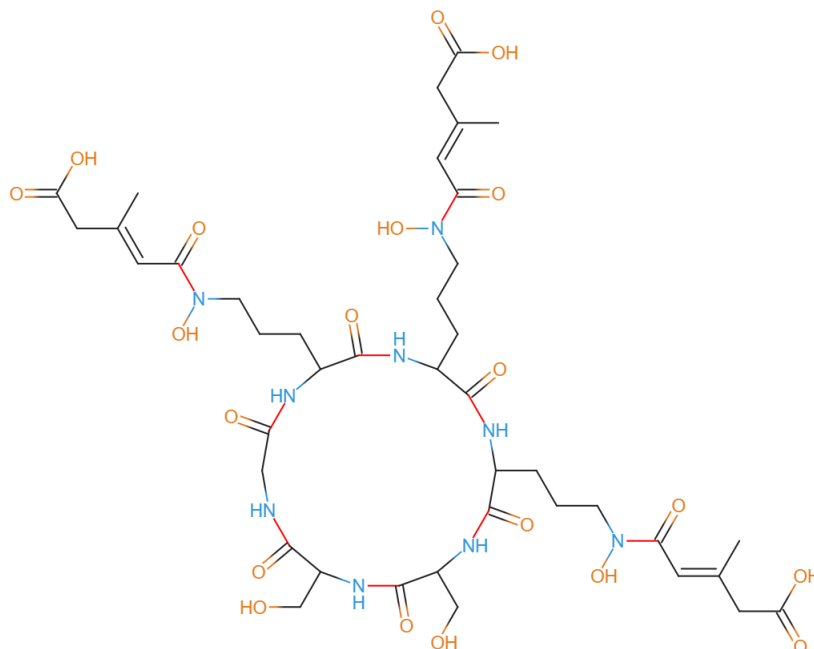
Momentálně lze klonovat celý kontejner, ale mohla by v budoucnu nastat situace, že bych chtěl z kontejneru naklonovat pouze určitou část. Například pouze bloky nebo organismy. Další zajímavou možností je kopírování dat mezi kontejnery. Zde by ovšem záleželo na tom, zda by pro někoho byla taková funkce užitečná.

Zajímavou možností by bylo také přebírat určitou funkcionalitu programu CycloBranch a vizualizovat spektrum sekvence. Také by se šlo věnovat lepšímu generování zkratk bloků ze jmen. Někdy není vygenerovaná zkratka úplně ideální. Tady by musel nastoupit již poměrně komplexní parser chemických názvů.

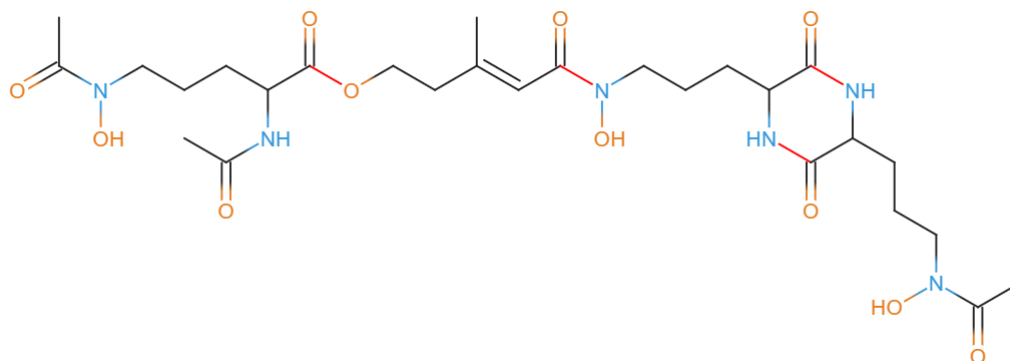
Také mě napadlo, že by nemuselo být špatné mít možnost porovnávat sekvence nebo bloky. Podobně jako na eshopu, když si vybírám například nové pc a porovnávám různé varianty mezi sebou, mít podobné porovnání sekvencí. Užitečná by také mohla být funkce pro smazání bloku přímo ze sekvence, po rozpadu na bloky. Momentálně když rozpadnu sekvenci na bloky a nějaký blok odstraním, mění se pouze zápis sekvence, ale samotné SMILES sekvence se nemění, někdy to může být žádoucí, ale bylo by zajímavé mít možnost daný blok odstranit přímo ze SMILES sekvence.

Jako menší vylepšení by mohlo přibýt přidání klávesových zkratk pro rychlejší a pohodlnější ovládání aplikace nebo také u bloků zobrazovat informace jako je celkový počet atomů, vazeb, vazeb konkrétních typů a podobně.

Obrázek 6.9: Ferrichrome



Obrázek 6.10: Neocoprogen



Ke konci práce jsem ještě do aplikace naprogramoval možnost přidávat reference na Lipid Maps [48]. Tato databáze poskytuje REST API, které by bylo také výhodné implementovat do hledání.

Závěr

Reimplementoval jsem aplikaci Bbdgnc pod novým názvem MassSpecBlocks a také jsem do ní přidal nové funkce. Aplikaci jsem rozdělil na frontend a backend. Primárně jsem přidal možnost registrace uživatelů a vytváření osobních databází s pomocí vyhledávání struktur v databázích třetích stran, které jsem rozšířil o vyhledávání na ChemSpideru. Podařilo se mi paralelizovat dotazy na API třetích stran, kde jsem dosáhl zrychlení oproti původní Bbdgnc. Nutno dodat, že rychlost ovlivnilo i přepsání aplikace do jiných technologií, neboť se projevilo nemalé zrychlení i na jednom vlákně. Dále jsem rozšířil funkce rozpadu sekvence na bloky o detekci polyketidů a detekci N-C směru sekvence. Také jsem naimplementoval algoritmus pro porovnávání struktur na základě podobnosti s využitím tanimoto koeficientu, který umožňuje vyhledávat struktury dle podobnosti a automaticky přiřazovat sekvencím rodiny. Pro budoucí vývoj aplikace jsem zmínil i několik tipů na rozšíření.

Bibliografie

1. PŘÍVRATSKÝ, Jan. *Generátor databáze stavebních bloků přírodních látek*. 2019. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
2. FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Dis. University of California, Irvine.
3. *Getting Started - React* [online]. [N.d.]. Dostupné také z: <https://reactjs.org/docs/getting-started.html>. cit. 2021-02-12.
4. RASCIA, Tania. *React Tutorial: An Overview and Walkthrough* [online]. [N.d.]. Dostupné také z: <https://www.taniarascia.com/getting-started-with-react/>. cit. 2021-02-12.
5. JELISEJEVS, Pavels. *React vs Angular: An In-depth Comparison* [online]. [N.d.]. Dostupné také z: <https://www.sitepoint.com/react-vs-angular/>. cit. 2021-02-12.
6. *Introduction to the Angular Docs* [online]. [N.d.]. Dostupné také z: <https://angular.io/docs>. cit. 2021-02-12.
7. *Typescript* [online]. [N.d.]. Dostupné také z: <https://www.typescriptlang.org>. cit. 2021-02-20.
8. *Fetch API* [online]. [N.d.]. Dostupné také z: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API. cit. 2021-02-20.
9. PROBST, Daniel; REYMOND, Jean-Louis. SmilesDrawer: Parsing and Drawing SMILES-Encoded Molecular Structures Using Client-Side JavaScript. *Journal of Chemical Information and Modeling*. 2018, roč. 58, č. 1, s. 1–7. Dostupné z DOI: 10.1021/acs.jcim.7b00425.
10. WEININGER, David. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*. 1988, roč. 28, č. 1, s. 31–36. Dostupné z DOI: 10.1021/ci00057a005.

11. WEININGER, David; WEININGER, Arthur; WEININGER, Joseph L. SMILES. 2. Algorithm for generation of unique SMILES notation. *Journal of Chemical Information and Computer Sciences*. 1989, roč. 29, č. 2, s. 97–101. Dostupné z DOI: 10.1021/ci00062a008.
12. BIENFAIT, Bruno; ERTL, Peter. JSME: a free molecule editor in JavaScript. *Journal of Cheminformatics*. 2013, roč. 5, č. 1, s. 24. ISSN 1758-2946. Dostupné z DOI: 10.1186/1758-2946-5-24.
13. BACHLER, Daniel. *jsme-react* [online]. [N.d.]. Dostupné také z: <https://github.com/douglasconnect/jsme-react>. cit. 2021-02-12.
14. HELLER, S. R.; MCNAUGHT, A.; PLETNEV, I. InChI, the IUPAC International Chemical Identifier. *J Cheminform*. 2015. Dostupné z DOI: 10.1186/s13321-015-0068-4.
15. *What is Symfony* [online]. [N.d.]. Dostupné také z: <https://symfony.com/what-is-symfony>. cit. 2021-02-20.
16. *Getting Started* [online]. [N.d.]. Dostupné také z: <https://getcomposer.org/doc/00-intro.md>. cit. 2021-02-20.
17. *Getting Started with Doctrine* [online]. [N.d.]. Dostupné také z: <https://www.doctrine-project.org/projects/doctrine-orm/en/current/tutorials/getting-started.html>. cit. 2021-02-20.
18. *Installation* [online]. [N.d.]. Dostupné také z: <https://laravel.com/docs/8.x>. cit. 2021-02-20.
19. *SQLite* [online]. [N.d.]. Dostupné také z: <https://sqlite.org/index.html>. cit. 2021-02-20.
20. *MySQL* [online]. [N.d.]. Dostupné také z: <https://www.mysql.com>. cit. 2021-02-20.
21. *MariaDB* [online]. [N.d.]. Dostupné také z: <https://mariadb.org>. cit. 2021-02-20.
22. *What Is MySQL* [online]. [N.d.]. Dostupné také z: <https://www.mysqltutorial.org/what-is-mysql/>. cit. 2021-02-20.
23. HOLČAPEK, Michal. *Hmotnostní spektrometrie* [online]. [N.d.]. Dostupné také z: http://holcapek.upce.cz/teaching/Mol_spek/Mol_spek_prednaska6_MS.pdf. [cit. 2019-04-08].
24. GINDULYTE, Asta; SHOEMAKER, Benjamin A; YU, Bo; HE, Jia; ZHANG, Jian; CHEN, Jie; ZASLAVSKY, Leonid; THIESSEN, Paul A; LI, Qingliang; HE, Siqian; KIM, Sunghwan; CHENG, Tiejun; BOLTON, Evan E. PubChem 2019 update: improved access to chemical data. *Nucleic Acids Research*. 2018, roč. 47, č. D1, s. D1102–D1109. ISSN 0305-1048. Dostupné z DOI: 10.1093/nar/gky1033.
25. CHEMSPIDER. *What is ChemSpider* [online]. [N.d.]. Dostupné také z: <http://www.chemspider.com/AboutUs.aspx>. [cit. 2021-03-21].

26. FONTAINE, Arnaud; KUCHEROV, Gregory; PUPIN, Maude; JACQUES, Philippe; LECLÈRE, Valérie; CABOCHE, Ségolène. NORINE: a database of nonribosomal peptides. *Nucleic Acids Research*. 2007, roč. 36, s. D326–D331. ISSN 0305-1048. Dostupné z DOI: 10.1093/nar/gkm792.
27. NOÉ, Laurent; TONON, Laurie; JANOT, Stéphane; DUFRESNE, Yann; FLISSI, Areski; PUPIN, Maude; LECLÈRE, Valérie; MICHALIK, Juraj; JACQUES, Philippe. Norine, the knowledgebase dedicated to non-ribosomal peptides, is now open to crowdsourcing. *Nucleic Acids Research*. 2015, roč. 44, č. D1, s. D1113–D1118. ISSN 0305-1048. Dostupné z DOI: 10.1093/nar/gkv1143.
28. HASTINGS, Janna; OWEN, Gareth; DEKKER, Adriano; ENNIS, Marcus; KALE, Namrata; MUTHUKRISHNAN, Venkatesh; TURNER, Steve; SWAINSTON, Neil; MENDES, Pedro; STEINBECK, Christoph. ChEBI in 2016: Improved services and an expanding collection of metabolites. *Nucleic acids research*. 2016, roč. 44, č. D1, s. D1214–9. ISSN 0305-1048. Dostupné z DOI: 10.1093/nar/gkv1031.
29. GILLILAND, Gary; BERMAN, Helen M.; WEISSIG, Helge; SHINDYALOV, Ilya N.; WESTBROOK, John; BOURNE, Philip E.; BHAT, T. N.; FENG, Zukang. The Protein Data Bank. *Nucleic Acids Research*. 2000, roč. 28, č. 1, s. 235–242. ISSN 0305-1048. Dostupné z DOI: 10.1093/nar/28.1.235.
30. NOVÁK, Jiří; LEMR, Karel; SCHUG, Kevin A.; HAVLÍČEK, Vladimír. CycloBranch: De Novo Sequencing of Nonribosomal Peptides from Accurate Product Ion Mass Spectra. *Journal of The American Society for Mass Spectrometry*. 2015, roč. 26, č. 10, s. 1780–1786. ISSN 1879-1123. Dostupné z DOI: 10.1007/s13361-015-1211-1.
31. NOVÁK, Jiří; SOKOLOVÁ, Lucie; LEMR, Karel; PLUHÁČEK, Tomáš; PALYZOVÁ, Andrea; HAVLÍČEK, Vladimír. Batch-processing of imaging or liquid-chromatography mass spectrometry datasets and De Novo sequencing of polyketide siderophores. *Biochimica et Biophysica Acta (BBA) - Proteins and Proteomics*. 2017, roč. 1865, č. 7, s. 768–775. ISSN 1570-9639. Dostupné z DOI: <https://doi.org/10.1016/j.bbapap.2016.12.003>. MALDI Imaging.
32. *NRPro User manual* [online]. [N.d.]. Ver. 1.0. Dostupné také z: <https://bioinfo.lifl.fr/nrpro/documents/manualNRPro.pdf>. cit. 2021-02-26.
33. DUFRESNE, Y.; NOÉ, L.; LECLÈRE, V.; PUPIN, M. Smiles2Monomers: a link between chemical and biological structures for polymers. *Journal of Cheminformatics*. 2015. Dostupné z DOI: 10.1186/s13321-015-0111-5.

34. RICART, E.; LECLÈRE, V.; FLISSI, A. et al. rBAN: retro-biosynthetic analysis of nonribosomal peptides. *Journal of Cheminformatics*. 2019. Dostupné z DOI: 10.1186/s13321-019-0335-x.
35. BAJUSZ, D.; RÁCZ, A; HÉBERGER, K. Why is Tanimoto index an appropriate choice for fingerprint-based similarity calculations? *Journal of Cheminformatics*. 2015. Dostupné z DOI: 10.1186/s13321-015-0069-3.
36. BERTRAND, Samuel. *Siderophore Base* [online]. [N.d.]. Dostupné také z: http://bertrandsamuel.free.fr/siderophore_base. cit. 2021-03-19.
37. PLUHÁČEK, Tomáš; LEMR, Karel; GHOSH, Dipankar; MILDE, David; NOVÁK, Jiří; HAVLÍČEK, Vladimír. Characterization of microbial siderophores by mass spectrometry. *Mass Spectrometry Reviews*. 2016, roč. 35, č. 1, s. 35–47. Dostupné z DOI: <https://doi.org/10.1002/mas.21461>.
38. SONARSOURCE. *About SonarQube* [online]. [N.d.]. Ver. 7.7. Dostupné také z: <https://www.sonarqube.org/about>. [cit. 2019-03-30].
39. JASMINE. *Your first suite* [online]. [N.d.]. Ver. 3.0. Dostupné také z: https://jasmine.github.io/tutorials/your_first_suite. [cit. 2019-03-30].
40. TRAVIS. *Core Concepts for Beginners* [online]. [N.d.]. Dostupné také z: <https://docs.travis-ci.com/user/for-beginners>. [cit. 2019-03-30].
41. GITHUB. *GitHub Actions* [online]. [N.d.]. Dostupné také z: <https://docs.github.com/en/actions>. [cit. 2021-03-21].
42. BERGMANN, Sebastian. *Getting Started with PHPUnit 8* [online]. [N.d.]. Ver. 8. Dostupné také z: <https://phpunit.de/getting-started/phpunit-8.html>. [cit. 2019-03-30].
43. SYMFONY. *How to Deploy a Symfony Application* [online]. [N.d.]. Dostupné také z: <https://symfony.com/doc/current/deployment.html>. [cit. 2021-04-22].
44. COMPOSER. *Composer - Getting Started* [online]. [N.d.]. Dostupné také z: <https://getcomposer.org/doc/00-intro.md>. [cit. 2021-04-22].
45. NPM. *About npm* [online]. [N.d.]. Dostupné také z: <https://docs.npmjs.com/about-npm>. [cit. 2021-04-22].
46. *CORS Anywhere* [online]. [N.d.]. Dostupné také z: <https://github.com/Rob--W/cors-anywhere>. [cit. 2021-04-22].
47. HEROKU. *What is Heroku?* [Online]. [N.d.]. Dostupné také z: <https://www.heroku.com/about>. [cit. 2021-04-22].

48. SUD, Manish; FAHY, Eoin; COTTER, Dawn; BROWN, Alex; DENNIS, Edward A.; GLASS, Christopher K.; MERRILL Alfred H., Jr; MURPHY, Robert C.; RAETZ, Christian R. H.; RUSSELL, David W.; SUBRAMANIAM, Shankar. LMSD: LIPID MAPS structure database. *Nucleic Acids Research*. 2006, roč. 35, s. D527–D532. ISSN 0305-1048. Dostupné z DOI: 10.1093/nar/gkl838.

Seznam použitých zkratk

- API** Application Programming Interface
- Bbdgnc** Building Blocks Database Generator of Natural Compound
- CDK** Chemistry Development Kit
- CORS** Cross-origin resource sharing
- CRUD** Create, Read, Update, Delete
- CSS** Cascading Style Sheets
- DFS** Depth-First Search
- DI** Dependency Injection
- HTML** Hypertext Markup Language
- HTTP** Hypertext Transfer Protocol
- ChEBI** Chemical Entities of Biological Interest
- IDE** Integrated Development Environment
- IUPAC** International Union of Pure and Applied Chemistry
- InChI** IUPAC International Chemical Identifier
- JSON** JavaScript Object Notation
- JSX** JavaScript XML
- MSB** MassSpecBlocks
- MS/MS** Tandem Mass Spectrometry
- MVC** Model-View-Controller

A. SEZNAM POUŽITÝCH ZKRATEK

Norine Database of Nonribosomal Peptides

ORM Object Relational Mapping

PDB Protein Data Bank

PHP PHP: Hypertext Preprocessor (dříve Personal Home Page)

rBAN retro-Biosynthetic Analysis of Nonribosomal peptides

REST Representational State Transfer

S2M Smiles2Monomers

SMILES Simplified Molecular Input Line Entry System

SOAP Simple Object Access Protocol

SQL Structured Query Language

URI Uniform Resource Identifier

XSS Cross-site Scripting

Obsah přiloženého CD

readme.txt	stručný popis obsahu CD
src	
├─ massSpecBlocks	hlavní repo - rozcestník, dokumentace
├─ backend	zdrojové kódy implementace backend
├─ frontend	zdrojové kódy implementace frontend
├─ smilesDrawer	zdrojové kódy forku SmilesDraweru
├─ thesis	zdrojová forma práce ve formátu $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$
text	text práce
├─ thesis.pdf	text práce ve formátu PDF