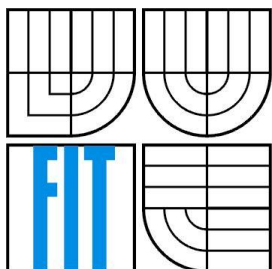


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF COMPUTER SYSTEMS

# DNA VÝPOČTY A JEJICH APLIKACE

DNA COMPUTING AND APPLICATIONS

DIPLOMOVÁ PRÁCE  
MASTER'S THESIS

AUTOR PRÁCE  
AUTHOR

Bc. JAN FIALA

VEDOUCÍ PRÁCE  
SUPERVISOR

Ing. MICHAL BIDLO, Ph.D.

BRNO 2014

## **Abstrakt**

Tato práce se zabývá návrhem a implementací aplikace pro simulaci DNA výpočtů pro řešení vybraných problémů. DNA výpočty patří mezi nekonvenční výpočetní paradigmatata, které se zásadně liší od konceptu elektronických počítačů. Hlavní myšlenkou DNA počítání je požití DNA jako média, kde může poměrně efektivně probíhat výpočet. I přesto, že DNA reakce jsou mnohem pomalejší než výpočetní taktiky dnešních křemíkových počítačů, má DNA počítání velmi slibnou budoucnost. DNA operace jsou založeny na dvou důležitých aspektech: masivní paralelismus DNA operací a princip komplementarity bází. Existuje mnoho důležitých problémů, pro které na konvenčních počítačích neexistuje algoritmus řešení v polynomiálním čase. Takto obtížné problémy se musí řešit prohledáváním celého stavového prostoru všech jejich řešení. Zde se ukazuje být masivní paralelismus DNA operací velmi důležitý, aby se snížila složitost hledání řešení.

## **Abstract**

This thesis focuses on the design and implementation of an application involving the principles of DNA computing simulation for solving some selected problems. DNA computing represents an unconventional computing paradigm that is totally different from the concept of electronic computers. The main idea of DNA computing is to interpret the DNA as a medium for performing computation. Despite the fact, that DNA reactions are slower than operations performed on computers, they may provide some promising features in the future. The DNA operations are based on two important aspects: massive parallelism and principle of complementarity. There are many important problems for which there is no algorithm that would be able to solve the problem in a polynomial time using conventional computers. Therefore, the solutions of such problems are searched by exploring the entire state space. In this case the massive parallelism of the DNA operations becomes very important in order to reduce the complexity of finding a solution.

## **Klíčová slova**

DNA počítání, problém nalezení hamiltonovské cesty, Adlemanův experiment, SAT problém, DNA operace.

## **Keywords**

DNA computing, Hamiltonian path problem, Adleman's experiment, SAT problem, DNA operations.

## **Citace**

Fiala Jan: DNA výpočty a jejich aplikace, diplomová práce, Brno, FIT VUT v Brně, 2014

# DNA výpočty a jejich aplikace

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Michala Bidla, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....  
Bc. Jan Fiala  
27. května 2014

## Poděkování

Chtěl bych tímto poděkovat vedoucímu práce Ing. Michalovi Bidlovi, Ph.D. za vedení a poskytnutí cenných rad a nápadů, které mi pomohly tuto práci vytvořit.

© Jan Fiala, 2014

*Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.*

# Obsah

Obsah.....	1
1 Úvod.....	3
2 Molekula DNA a její zpracování .....	5
2.1 Struktura DNA.....	5
2.2 Měření délky DNA molekul .....	7
2.3 Denaturace a renaturace DNA .....	8
2.4 Doplnění a zkracování DNA.....	8
2.5 Rozdělování a spojování DNA .....	9
2.6 Hledání určité DNA.....	10
2.7 Množení DNA .....	10
2.8 Čtení sekvence DNA .....	11
3 DNA počítání .....	13
3.1 Adlemanův experiment.....	14
3.1.1 Zakódování grafu.....	15
3.1.2 Hlavní fáze.....	16
3.1.3 Formalismus .....	17
3.2 SAT-problém .....	19
3.2.1 Převod na grafový problém.....	19
3.2.2 Formalismus .....	20
3.3 Hledání maximální kliky grafu.....	22
3.3.1 Zakódování do DNA.....	23
3.3.2 Hlavní fáze.....	25
4 Návrh aplikace .....	27
4.1 Datová struktura DNA sekvence .....	28
4.2 Parametry simulátoru.....	29
4.3 Generování kandidátních řešení.....	30
4.4 Paralelizace výpočtu pomocí OpenMP.....	32
4.4.1 OpenMP API .....	33
5 Dosažené výsledky.....	36
5.1 Správnost simulace .....	36
5.2 Složitost instancí problémů.....	38
5.3 Vliv parametrů simulátoru .....	42
5.4 Časová náročnost řešení problémů .....	47
5.5 Využití paměti v průběhu výpočtu.....	49

6	Závěr .....	51
6.1	Zhodnocení výsledků.....	51
6.2	Přínosy práce .....	52
6.3	Možnosti dalšího vývoje.....	52
	Literatura .....	53
	Seznam příloh .....	55
	Příloha A.....	56
	Příloha B .....	57
	Příloha C .....	59

# 1 Úvod

V dnešní době si lidé pod pojmem počítač jistě představí řadu elektronických komponent, které tvoří jeden fungující celek. Počítače jsou obecně definovány jako stroje, které přijímají, modifikují a ukládají vstup a produkují nějaký výstup. Počítání je zde založeno na křemíkových čípech. Dnes již však tento výpočetní koncept pomalu naráží na své hranice. Hlavním problémem je to, že prvky na čípech nelze zmenšovat do nekonečna. I při aktuálních velikostech se naráží na řadu problémů, které se zatím daří obcházet. Pokud bude ale zmenšování prvků na čípech dále pokračovat, brzy se narazí na fyzikální limity. A ty už tak lehce obejít nepůjde. Nabízí se zde hned několik otázek. Jaká bude budoucnost výpočetních systémů? Je možné překročit hranice silikonových součástek a najít jiné médium pro výpočet?

Nastal čas se odprostit od klasického výpočetního konceptu a začít zkoumat nějaké úplně nové, které nejsou tak limitované. Při hledání nového materiálu k postavení nové generace mikroprocesorů přišli vědci s velmi překvapivým objevem. Zjistili, že v každém živém organismu včetně našich buněk se nachází milióny přírodních superpočítačů. Nový koncept nazvali DNA počítání. Hlavní myšlenkou DNA počítání je požití molekul DNA jako média, kde může poměrně efektivně probíhat výpočet. Opírá se totiž o dva velmi významné rysy DNA molekul. Jsou jimi masivní paralelismus DNA operací a princip komplementarity bází. DNA molekuly mají potenciál provádět výpočty několikrát rychleji než ty nejvýkonnější konvenční superpočítače. Tento potenciál zatím nebyl plně využit, ale technologický pokrok časem umožní ze stavebních částí našeho genomu vytvořit paměti a procesory, které katapultují hodnoty rychlostí a kapacit na hodnoty, které si z pohledu dnešního měřítka neumíme představit.

Cílem této práce je představit koncept DNA výpočtu, který je aplikovatelný v oblasti informačních technologií, i se všemi principy molekulární biologie, které jsou pro pochopení principu DNA počítání klíčové. Dále implementovat aplikaci simulující řešení vybraných problémů pomocí DNA počítání a zvolit vhodné techniky za účelem zvýšení efektivity simulátoru. Dále provést sadu experimentů řešících vybrané problémy s ohledem na složitost instancí, parametry simulátoru a časové nároky. Práce na závěr shrne všechny experimenty a uvede výhody i nevýhody navrženého řešení a další možnosti činnosti v této oblasti.

Práce je členěna do několika kapitol. Ve druhé kapitole práce je popsána základní struktura molekuly DNA a popis některých DNA operací, které jsou pro pochopení principu DNA počítání stěžejní. Ve třetí kapitole je popsán vlastní princip DNA výpočtu na třech problémech z oblasti informačních technologií. Jsou jimi: hledání hamiltonovské cesty grafu, řešení SAT problému a hledání maximální kliky grafu. Čtvrtá kapitola se zabývá návrhem aplikace, která bude simulovat řešení daných problémů pomocí DNA výpočtu, a popisem technologie OpenMP, pomocí níž lze zvýšit efektivity simulátoru. V páté kapitole je popsáno několik experimentů řešících vybrané

problémy pomocí DNA počítání s ohledem na složitost instancí, parametry simulátoru a časové nároky. V závěrečné kapitole č. 9 jsou shrnuty všechny poznatky z této práce, její přednosti a nedostatky a návrhy na další rozšíření.

## 2 Molekula DNA a její zpracování

V této kapitole si uvedeme základní strukturu molekuly DNA a některé metody, jak s ní lze manipulovat. Uvedeny budou pouze metody týkající se DNA počítání, které se provádí *in vitro* (vně živých buněk). [1] [2]

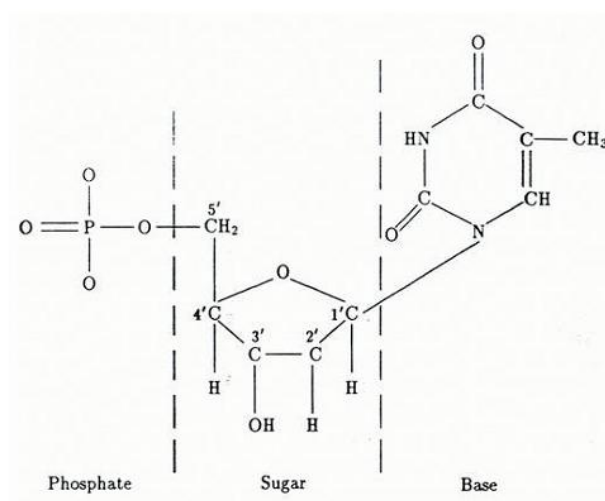
### 2.1 Struktura DNA

Molekula DNA hraje zásadní roli ve všech živých buňkách, nicméně v této práci se jí budeme zabývat hlavně z hlediska DNA počítání. Její struktura je pro princip DNA počítání velmi důležitá.

Molekula DNA (*DeoxyriboNucleic Acid*) je polymer sestavený z monomerů zvaných deoxyribonukleotidy. Tyto monomery se skládají ze tří částí: cukru, fosfátové skupiny a dusíkaté báze. Cukr se přesněji jmenuje deoxyribóza, to vysvětluje prefix „deoxyribo“ používaný výše. Abychom si zjednodušili terminologii, budeme dále používat pojem „nukleotid“ místo „deoxyribonukleotid“.

Cukr obsahuje mimo jiné 5 uhlíků (značíme je 1' až 5'). Fosfátová skupina je připojena na 5' uhlík a dusíkatá báze na 1' uhlík. Dále je pak na 3' uhlík připojena hydroxylová skupina (OH).

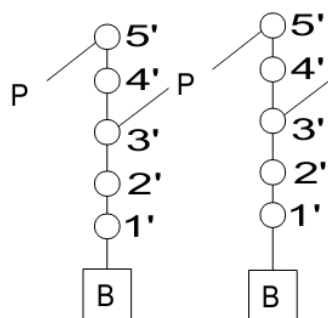
Existuje několik druhů nukleotidů. Liší se pouze svými bázemi. Báze dělíme do dvou skupin: puriny a pyrimidiny. Mezi puriny řadíme adenin (A) a guanin (G), mezi pyrimidiny cytosin (C) a thymin (T). Chemickou strukturu thyminu můžete vidět na *obrázku č. 1*. Jelikož se nukleotidy liší pouze bázemi, jsou zjednodušeně označovány jako nukleotidy A, T, C a G.



Obrázek č. 1: Struktura nukleotidu (thyminu) [2]

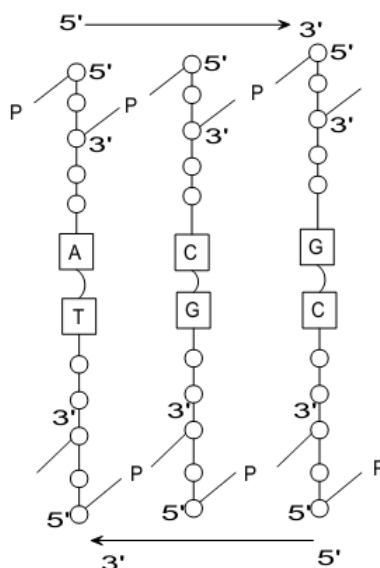
Nukleotidy na sebe v rámci DNA vzájemně působí dvěma vazbami:

1. Fosfátová skupina jednoho nukleotidu je vázána na hydroxylovou skupinu nukleotidu druhého. Tato vazba je silná (kovalentní) a nazývá se *fosfordiesterová*. Její strukturu lze vidět na *obrázku č. 2*.



*Obrázek č. 2: Fosfordiesterová vazba. P značí fosfátovou skupinu, B značí jednu ze čtyř dusíkatých bází a kruhy značí kostru cukru. [2]*

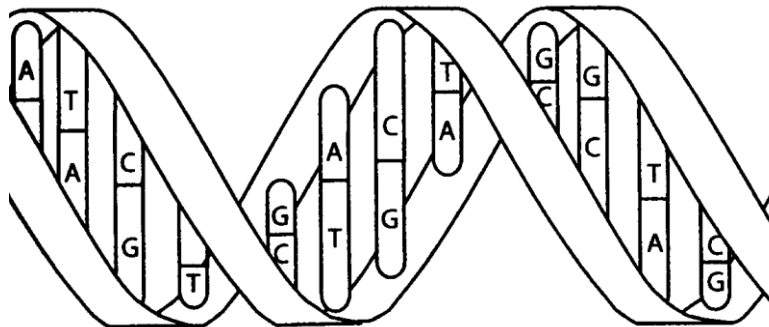
2. Mezi bázemi dvou nukleotidů vzniká vodíková vazba. Vazba ale vzniká jen v určité kombinaci bází, podle tzv. komplementarity bází. Báze A je komplementární k T a báze C je komplementární ke G. Žádné jiné vazby v normálním stavu nevznikají. Jedná se o velmi slabou vazbu, která může zaniknout pouhým zahřátím roztoku s molekulami DNA.



*Obrázek č. 3: Vodíková vazba (komplementarita bází) [2]*

Působením těchto dvou vazeb vzniká výsledná „dvoušroubovicová“ struktura molekuly DNA. Dvoušroubovici tvoří dvě jednořetězcová vlákna spojená pomocí vodíkových vazeb na základě

komplementarity bází. Obě vlákna mají navzájem opačný směr: nukleotid na 5' konci jednoho vlákna je vázán na nukleotid na 3' konci druhého vlákna. Navíc se obě vlákna obtáčí kolem společné osy a tím vytváří slavný model pravotočivé dvoušroubovice. V buňce je situace ještě složitější, protože se velmi „velká“ molekula DNA musí vejít do malého prostoru. Dosáhne se toho díky různým typům sbalení DNA molekuly.



Obrázek č. 4: Struktura rozvinuté DNA v podobě pravotočivé dvoušroubovice.

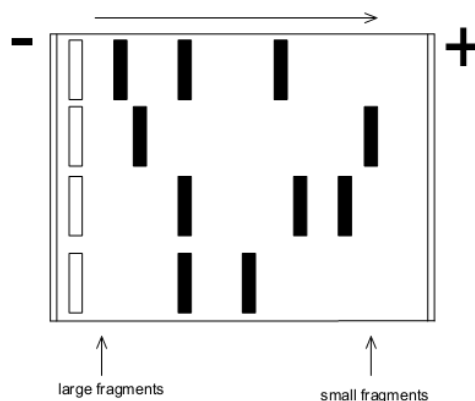
## 2.2 Měření délky DNA molekul

Měření délky DNA molekul je pro DNA počítání velmi významné. Často se k problému hledá řešení o předem známé délce. Abychom mohli z velkého množství řešení vyfiltrovat pouze ty správně dlouhé a usnadnit si tak prohledávání, musíme umět délku molekul DNA změřit.

Délkou jednoho vlákna DNA rozumíme počet nukleotidů, ze kterých se toto vlákno skládá. Jednotkou této hodnoty bývá *mer* (vlákno s 12 nukleotidy je dlouhé 12 *mer*). Délkou dvoušroubovice DNA se pak rozumí počet párů nukleotidů. Jednotkou je zde *bp* (base pairs).

K měření délky molekul DNA se používá metoda zvaná gelová elektroforéza. Tato metoda využívá toho, že molekula DNA je záporně nabitá. Pokud se tedy umístí do elektrického pole, začne se pohybovat směrem ke kladné elektrodě. Náboj DNA molekul je závislý na jejich délce, proto by se ve výsledku pohybovaly všechny stejnou rychlostí. Aby se dosáhlo jejich rozdělení podle délky, musí se pohybovat v gelu, který jim vytváří odpor. Delší molekuly budou zpomalovány více než ty kratší. Po určité době je elektrické pole vypnuto a molekuly se zastaví. Na základě vzdálenosti, kterou jednotlivé molekuly urazily, se potom může určit jejich délka.

Jelikož jsou molekuly DNA v normálním stavu bezbarvé a nebyly by v gelu pozorovatelné, označí se před provedením měření fosforeskujícími nebo radioaktivními látkami.



Obrázek č. 5: Výstup gelové elektroforézy [2]

## 2.3 Denaturace a renaturace DNA

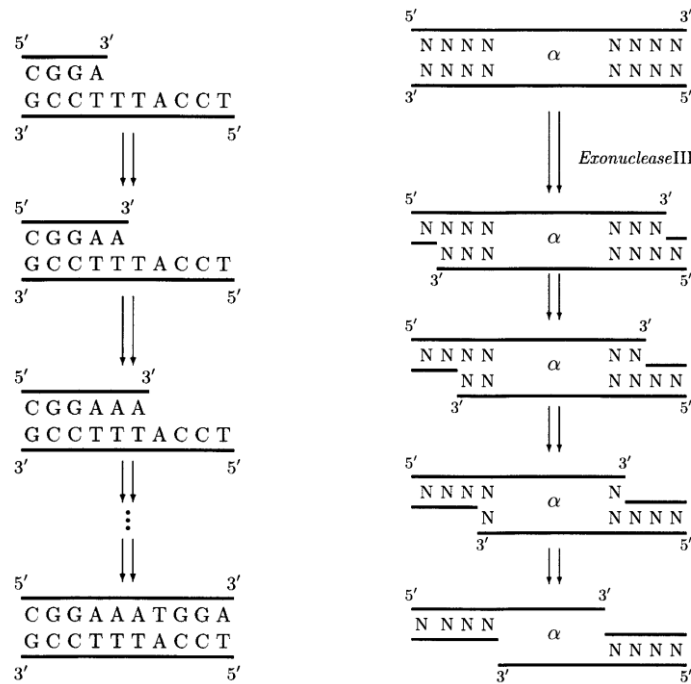
Jak již bylo řečeno, dvojitá šroubovice DNA je formována pomocí dvou druhů vazeb. Vodíková vazba mezi komplementárními bázemi vláken je mnohem slabší než vazba fosfodiesterová mezi nukleotidy.

Proces rozbití dvoušroubovice DNA na dvě samostatná vlákna se nazývá denaturace DNA. Docílí se toho zahřátím roztoku s DNA na teplotu 85°C až 95°C, při které se přeruší slabé vodíkové vazby. Navíc může být denaturace ulehčena pomocí různých katalyzátorů. Opačný proces se nazývá renaturace DNA a probíhá při pomalém snižování teploty roztoku s DNA.

## 2.4 Doplnování a zkracování DNA

Třída enzymů zvaných DNA polymerázy je schopna doplňovat nukleotidy na již existující DNA molekuly. Ke své činnosti potřebuje vlákno DNA, které poslouží jako šablona. Toto vlákno předepisuje, které nukleotidy na něj podle komplementarity polymeráza naváže. DNA polymeráza doplňuje nukleotidy vždy na 3' konec vlákna (doplňuje šablonu ve směru 5' - 3').

Zkracování DNA zajišťují enzymy zvané exonukleázy. Zkracují DNA odebráním nukleotidů z konců DNA molekuly. Jsou mnohem flexibilnější než polymerázy. Je jich více druhů a jsou více specifické. Některé exonukleázy odebírají nukleotidy z 3' konce, jiné zase z 5' konce DNA molekul. Některé mohou být specifické pro jednovláknové molekuly, jiné zase pro dvouvláknové.

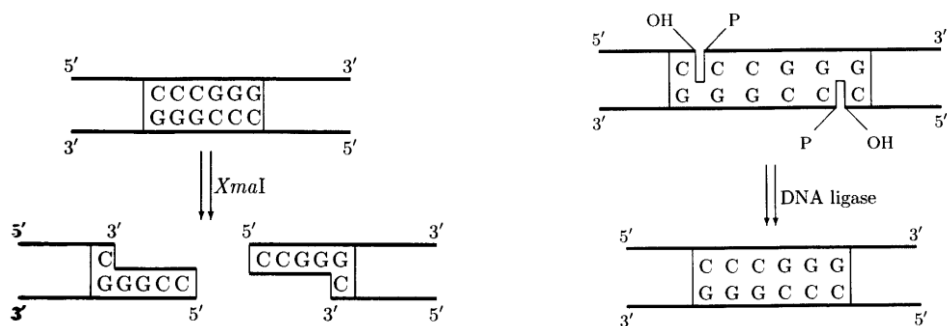


Obrázek č. 6: Činnost DNA polymerázy (vlevo) a činnost exonukleázy (vpravo) [2]

## 2.5 Rozdělování a spojování DNA

K rozdělení molekuly DNA je třeba přerušit fosfodiesterové vazby mezi nukleotidy. Dokážou to enzymy zvané endonukleázy. Je jich opět mnoho specializovaných druhů. Speciálním druhem jsou restriční endonukleázy, které rozdělují pouze dvojité šroubovice a to v pouze určitých charakteristických místech. Existují dva druhy rozdělení: přímé a střídavé. Přímé rozdělení dělí dvoušroubovici v místě jednoho páru bází, střídavé nechá přesahovat část vlákna na obou stranách.

Spojování rozdělených molekul DNA umožňují enzymy zvané ligázy. Při rozdělení molekul došlo k přerušení fosfodiesterových vazeb, slabé vodíkové vazby však stále působí. U střídavého rozdělení se proto rozdělené části drží stále u sebe. Ligáza dokáže pouze znovu obnovit fosfodiesterové vazby obou částí. U přímého rozdělení spojuje ligáza obě části nezávisle na koncových nukleotidech.



Obrázek č. 7: Střídavé rozdělení endonukleázou (vlevo) a spojení ligázou (vpravo) [2]

## 2.6 Hledání určité DNA

Komplementarita vláken molekuly DNA může být velmi výhodná v situacích, kdy chceme najít v roztoku určitou molekulu DNA se známou sekvencí. Hledaná molekula se nazývá cílová.

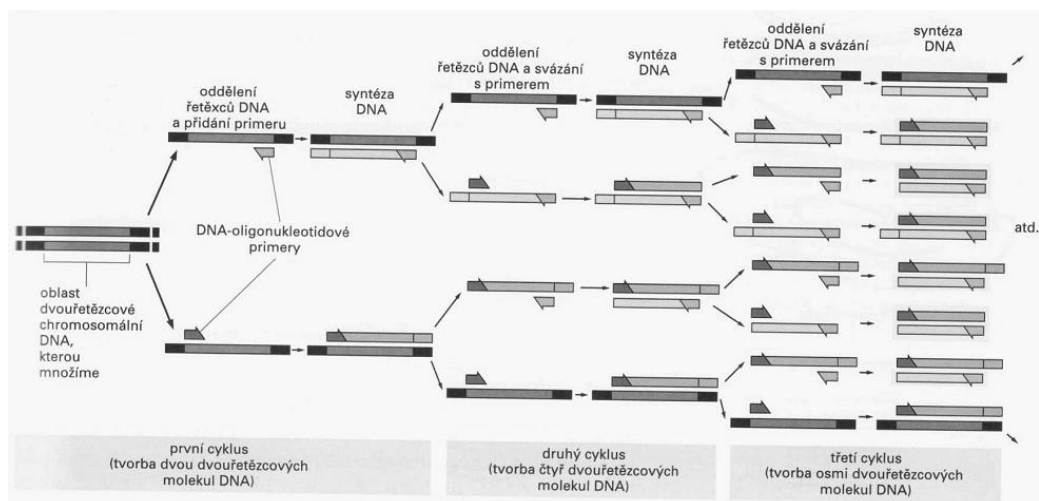
Pokud jsou před vyhledáváním v roztoku molekuly DNA ve dvouvláknové podobě, musí být provedena jejich denaturace. Předpokládejme, že se pokoušíme najít jednovláknovou molekulu DNA se sekvencí  $\alpha$  v roztoku, který obsahuje řadu dalších jednovláknových molekul. Vytvoří se jednovláknové molekuly DNA se sekvencí  $\bar{\alpha}$  a připojí k nějakému pevnému základu (vlákno  $\bar{\alpha}$  je komplementární k  $\alpha$ ). Tím vznikne požadovaný filtr. Následně roztok s cílovou molekulou přefiltrujeme přes tento filtr. Vlákna  $\alpha$  se navážou na filtr, roztok po přefiltrování je již neobsahuje. K získání cílových molekul  $\alpha$  se provede denaturace dvouvláknových molekul na filtru a následně je pevný základ odstraněn.

## 2.7 Množení DNA

Namnožení dostatečně velkého množství fragmentů DNA je jeden z klíčových problémů genetického inženýrství. Nejnáročnější je namnožení malého množství fragmentů, které jsou v roztoku spolu s mnoha dalšími.

Technika množení DNA se nazývá PCR (polymerázová řetězová reakce). Jedná se o velmi jednoduchou a elegantní metodu, která dokáže během krátké chvíle vyprodukovat miliony zdrojové DNA molekuly. Aby se mohla namnožit molekula  $\alpha$ , musí být známy její okrajové sekvence  $\beta$  a  $\gamma$ . Namnožení molekuly  $\alpha$  se dosáhne opakováním třech kroků: denaturace, přidání primerů a rozšiřování. Počáteční roztok musí obsahovat: molekulu  $\alpha$ , primery (řetězce DNA komplementární k  $\beta$  a  $\gamma$ ), polymerázy a nukleotidy.

Ve fázi denaturace je zdrojová molekula  $\alpha$  za vysoké teploty rozdělena na dvě vlákna. Nyní se roztok opět zchladí a primery se navážou na své komplementární okrajové sekvence. Teplota se opět zvýší a polymeráza začne postupně připojovat chybějící komplementární nukleotidy, a tím rozšiřovat chybějící druhé vlákno. Děje se tak vždy ve směru 5' - 3'. Opakují-li se tyto kroky  $n$ -krát, výsledkem bude  $2^n$  kopií molekuly  $\alpha$ . Schéma průběhu techniky PCR je uvedeno na *obrázku č. 8*.



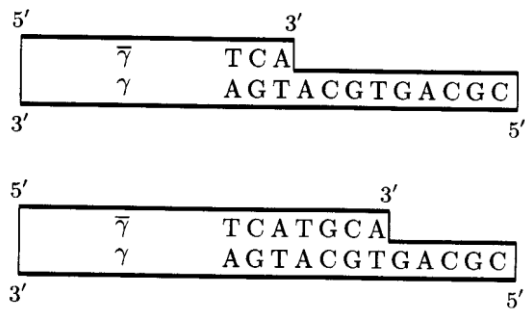
Obrázek č. 8: Schéma průběhu techniky PCR [5]

## 2.8 Čtení sekvence DNA

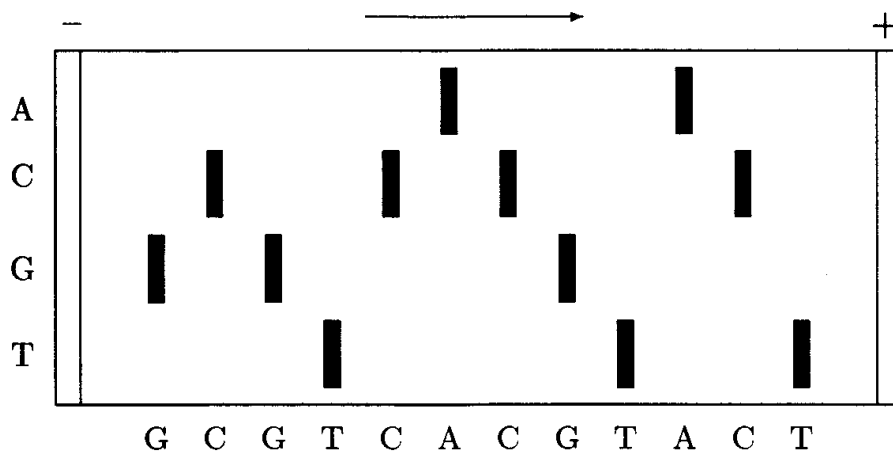
Nejpopulárnější metody čtení sekvence DNA jsou založeny na enzymu polymerázy, který doplňuje chybějící nukleotidy k samostatnému vláknu. Princip metod je velmi podobný technice PCR. Na rozdíl od ní se však používají speciální modifikované nukleotidy. Nejčastěji se jedná o úpravu 3'-hydroxylové skupiny na 3'-vodík a nukleotidy se potom značí ddA, ddT, ddC a ddG. Taková metoda se pak nazývá Sangerova. [12]

Ke zdrojovému vláknu  $\alpha$  se nejprve připojí krátká sekvence  $\gamma$ , aby na ni mohl být navázán primer. Primer je obvykle označen (fosforečně nebo radioaktivně), aby bylo později možné zjistit, kde je začátek čteného vlákna. Připraví se čtyři zkumavky, z nichž každá obsahuje nukleotidy všech typů spolu s modifikovanými nukleotidy téhož typu a polymerázy. Připravená vlákna se namnoží a přidají do zkumavek. Ve zkumavkách začne polymeráza postupně doplňovat nukleotidy na čtené vlákno. Po doplnění několika obyčejných nukleotidů najednou polymeráza připojí modifikovaný nukleotid, díky čemuž už není možno vázat další nukleotidy a reakce pro molekulu končí.

Jakmile reakce ve všech zkumavkách skončí, obsah zkumavek bude následující. Každá zkumavka bude obsahovat molekuly DNA doplněné na různou velikost (obrázek č. 9). Je jisté, že každá molekula ve zkumavce končí modifikovaným nukleotidem určitého typu (např. ddA, ddT, ...). Následuje změření délky všech molekul pomocí gelové elektroforézy. Ze zjištěných délek lze odvodit, na kterých pozicích se vyskytují modifikované nukleotidy (obrázek č. 10). Z tohoto poznatku je už snadné díky komplementaritě vyčíst jednotlivé nukleotidy na původní čtené sekvenci (na pozici modifikovaného nukleotidu ddA je v sekvenci nukleotid T atd.).



Obrázek č. 9: Molekuly ukončené nukleotidem dda [2]



Obrázek č. 10: Odvození sekvence (dole) podle délek doplněných molekul [2]

### 3 DNA počítání

Hlavní myšlenkou DNA počítání je požití DNA jako média, kde může poměrně efektivně probíhat výpočet. V roce 1994 ukázal Leonard Adleman, že to skutečně lze. Uvědomil si, že operace molekulární genetiky nad DNA lze chápat jako operace nad řetězci. Vhodnou posloupností těchto operací lze pak získat výsledek výpočtu. Zatím je tento obor velmi mladý, proto nejsou ještě používané biochemické techniky přesné a sofistikované tak, jak bychom potřebovali. Nicméně i zde probíhá vývoj. [5] [7]

I přesto, že DNA reakce jsou mnohem pomalejší než výpočetní takty dnešních křemíkových počítačů, má DNA počítání velmi slibnou budoucnost. Opírá se o dva velmi významné rysy DNA molekul:

1. masivní paralelizmus DNA operací
2. komplementaritu bází (objevili ji James D. Watson a Francis Crick) [6]

Existuje mnoho důležitých problémů, pro které neexistuje algoritmus řešení v polynomiálním čase. Takto obtížné problémy se musí řešit prohledáváním celého stavového prostoru všech jejich řešení. Nicméně velikost takových stavových prostorů je tak velká, že při úrovni dnešních výpočetních technologií nelze řešení najít v rozumném čase. Zde se ukazuje být masivní paralelizmus DNA operací velkou výhodou. Navíc hustota informace, která je uložena v molekule DNA, je velmi vysoká. Když se tedy obtížný problém vhodně zakóduje do DNA řetězce a využije se výše zmíněných výhod DNA počítání, lze vypočítat řešení problému v rozumném čase. Typickým příkladem je dešifrování zašifrovaného textu – všechny možné klíče mohou být prověřeny paralelně. Šifrování v kombinaci s DNA počítáním je také velký příslib do budoucna. [16]

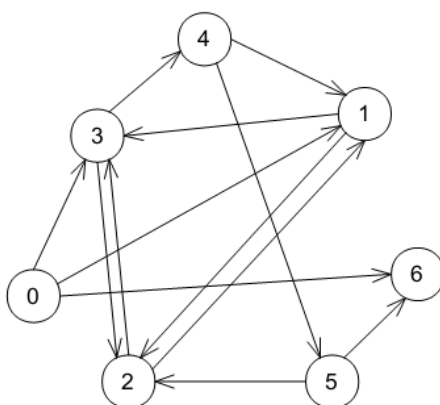
Kromě řešení výpočetně obtížných problémů pomocí zakódování do DNA řetězců existuje ještě řada dalších důvodů, proč zkoumat právě DNA počítání. Je velmi důležité snažit se pochopit, jak „počítá“ příroda. Pouze za pomoci DNA a manipulací s ní je schopna řídit něco tak robustního a složitého, jako je život.

Princip DNA počítání vede na úplně nové výpočetní paradigma, které je velmi odlišné od toho klasického: nové datové struktury, nové typy operací s těmito strukturami, nové typy operací s klasickými strukturami (řetězce, ...), nový výpočetní model. I kdyby se tedy nakonec ukázalo, že vytvořit DNA počítač není reálné (např. kvůli vysoké chybovosti), může být paradigma DNA počítání implementováno do výkonné klasické architektury. [15]

### 3.1 Adlemanův experiment

Adlemanův experiment řeší problém nalezení hamiltonovské cesty (HPP) v daném orientovaném grafu. Definice problému je následující: necht'  $G$  je orientovaný graf s označeným vstupním uzlem  $v_{in}$  a výstupním uzlem  $v_{out}$ . Cesta z uzlu  $v_{in}$  do  $v_{out}$  se nazývá hamiltonovská, pokud obsahuje každý uzel grafu právě jednou. Vyplývá z toho, že  $v_{in}$  se nesmí rovnat  $v_{out}$ , protože by byl tento uzel v cestě dvakrát. [3] [14]

Na *obrázku č. 11* je uveden graf použitý Adlemanem v jeho experimentu. Vstupním uzlem je zde 0 a výstupním 6. Hamiltonovská cesta pro takový graf se skládá z orientovaných hran:  $0 \rightarrow 1$ ,  $1 \rightarrow 2$ ,  $2 \rightarrow 3$ ,  $3 \rightarrow 4$ ,  $4 \rightarrow 5$ ,  $5 \rightarrow 6$ .



*Obrázek č. 11: Graf v Adlemanově experimentu [3]*

Číslování uzlů bylo vybráno tak, že hrany hamiltonovské cesty jsou číslovány vzestupně. Ne vždy se to ale podaří. Samozřejmě se číslování uzlů může poupravit až po nalezení hamiltonovské cesty tak, aby se v něm lépe orientovalo, jako je tomu v předchozím grafu. V tomto případě se ukázalo, že zmíněná hamiltonovská cesta je jediná, kterou graf obsahuje. Jelikož se jedná o velmi jednoduchý graf, není problém projít všechny možné cesty. U složitějších grafů je ale prostor možných cest obrovský a nelze ho celý projít v rozumném čase.

Obecně je problém nalezení hamiltonovské cesty chápán jako rozhodnutí, zda daný graf hamiltonovskou cestu obsahuje či nikoliv. Jak již bylo řečeno, HPP může být řešen prohledáváním prostoru všech možných cest. Vymyšleno bylo již mnoho algoritmů pro řešení tohoto problému. Každý z nich je sice úspěšný u speciálních tříd grafů, ale u obecných orientovaných grafů se u nich projeví exponenciální složitost a selžou.

Ve skutečnosti se ukázalo, že HPP je NP-úplný problém. Znamená to, že pravděpodobně neexistuje efektivní algoritmus, který by jej dokázal vyřešit v polynomiálním čase. Všechny jeho obecná řešení tedy nutně vedou na masivní prohledávání stavového prostoru všech možných řešení.

Navzdory tomu se Adlemanovi podařilo vyřešit HPP uvedeného grafu pomocí principu DNA počítání. Nicméně z principu je možné takové řešení aplikovat i na mnohem větší grafy. Základními stavebními kameny řešení HPP pomocí DNA počítání je masivní paralelismus a komplementarita bází. Jeho nedeterministický algoritmus je následující.

---

*Vstup:* Orientovaný graf  $G$  s  $n$  uzly. Vstupní uzel je označen  $v_{in}$  a výstupní je označen  $v_{out}$ .

*Krok 1:* Náhodně se generují cesty v grafu  $G$  a to ve velkém množství.

*Krok 2:* Odstraní se všechny cesty, které nezačínají v uzlu  $v_{in}$  a nekončí v  $v_{out}$ .

*Krok 3:* Odstraní se všechny cesty, které neprocházejí právě  $n$  uzly.

*Krok 4:* Odstraní se všechny cesty, které neobsahují některý uzel grafu  $G$ .

*Výstup:* „ANO“ – pokud je seznam cest neprázdný

„NE“ – pokud je seznam cest prázdný

---

*Algoritmus č. 1: Algoritmus Adlemanova experimentu [3]*

I zde dochází k prohledávání stavového prostoru všech cest. Masivní paralelismus DNA vláken však dokáže vyřešit nedeterminismus NP-problému v polynomiálním čase. Komplementarita bází zaručí, že vytvořená sekvence řešení je určitě cestou v grafu  $G$ . V následujících podkapitolách bude popsán Adlemanův experiment ještě podrobněji.

### 3.1.1 Zakódování grafu

Každý uzel grafu je označen náhodným unikátním 20-mer DNA vláknem  $s_i$ , kde pro daný graf  $0 \leq i \leq 6$ . Adleman například použil pro uzly 2 – 4 následující vlákna (orientace vláken je ve směru 5' - 3'):

$s_2 = \text{TATCGGATCGGTATATCCGA},$

$s_3 = \text{GCTATTTCGAGCTTAAAGCTA},$

$s_4 = \text{GGCTAGGTACCAGCATGCTT}.$

Pro lepší názornost principu si zavedeme pomocnou funkci  $h()$ , která mapuje každou bázi nukleotidu na svůj komplement. Funkce  $h()$  vrací tedy pro jednotlivé nukleotidy tyto hodnoty:

$h(A) = T,$

$h(C) = G,$

$$h(G) = C,$$

$$h(T) = A.$$

Dále budeme předpokládat rozšíření funkce  $h()$  takové, že ji lze aplikovat i na sekvence nukleotidů:

$$h(\text{ACTGCATGA}) = \text{TGACGTA}CT,$$

$$h(s_2) = \text{ATAGCCTAGCCATATAGGCT}.$$

Vzhledem k tomu, že výstupem funkce  $h()$  je vlákno, které je komplementární ke vstupnímu vláknu, lze vyvodit, že pokud je vstupní vlákno orientováno ve směru  $5' - 3'$ , bude mít výstup funkce orientaci ve směru  $3' - 5'$ . Funkce  $h()$  je tedy morfismem.

Kód každého uzlu je rozdělen na dvě stejně dlouhé části (v tomto případě délky 10):

$$s_i = s_i' s_i'', \text{ pro } 0 \leq i \leq 6.$$

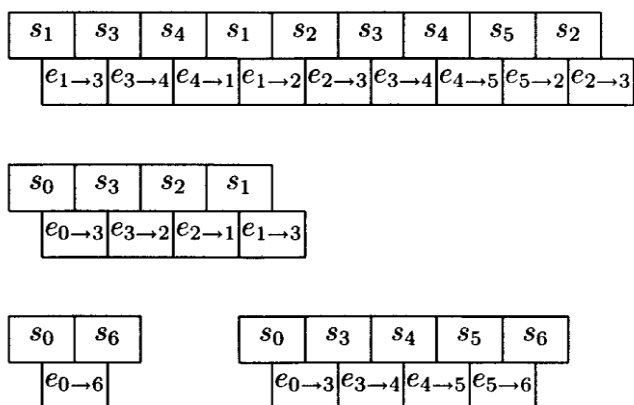
Orientovaná hrana grafu  $G$  z uzlu  $i$  do uzlu  $j$  je zakódována jako  $h(s_i'' s_j')$ . Z toho vyplývá, že délka kódů hran grafu je stejná jako délka kódů uzlů. První polovina kódu hrany je komplementární k druhé polovině kódu uzlu, ze kterého hrana vychází. Druhá polovina kódu hrany je komplementární k první polovině kódu uzlu, do kterého směřuje. Kódování hran zajišťuje rozlišení jejich orientace (hrany  $2 \rightarrow 3$  a  $3 \rightarrow 2$  mají odlišný kód):

$$e_{2 \rightarrow 3} = \text{CATATAGGCTCGATAAGCTC},$$

$$e_{3 \rightarrow 2} = \text{GAATTCGATATAGCCTAGC}.$$

### 3.1.2 Hlavní fáze

Nyní je již vše připraveno k popisu hlavní fáze Adlemanova experimentu. Pro každý uzel a pro každou hranu orientovaného grafu bylo připraveno velké množství oligonukleotidů, které nesly jejich kód. Následně byly umístěny do roztoku spolu s enzymem ligázy. Ligáza začne z oligonukleotidů vytvářet různě dlouhé molekuly DNA. Vhodné zakódování uzlů a hran grafu zajistí, že vytvořené DNA molekuly mohou být čteny jako náhodné cesty v grafu. Uzly se mohou totiž v DNA molekule vyskytovat vedle sebe, pouze pokud existuje hrana, která je spojuje a je ve správné orientaci.



Obrázek č. 12: Příklady DNA molekul v Adlemanově experimentu [2]

V roztoku se nyní nachází obrovské množství zakódovaných náhodných cest v grafu. Adleman ve svém experimentu použil cca  $10^{13}$  kopií oligonukleotidů reprezentujících jednu hranu v grafu, což je pro graf se 7 uzly opravdu mnoho. Je velmi pravděpodobné, že z tohoto počtu vznikne mnoho DNA molekul kódujících hamiltonovskou cestu v grafu. Jako důkaz existence hamiltonovské cesty ale stačí pouze jedna taková DNA molekula. Jinými slovy, pravděpodobně by stačilo použít mnohem menší množství oligonukleotidů než Adleman k nalezení hamiltonovské cesty v daném grafu nebo by při stejném množství mohl být prohledáván větší graf.

Další kroky experimentu už jsou založeny na čtení kódu DNA a filtrování (viz *algoritmus č. 1*). Jedná se o pokročilé techniky genetického inženýrství, které již byly zjednodušeně popsány v kapitole 2. Pro potřeby této práce stačí vědět, že tyto metody jsou možné a v praxi se používají.

Obecně lze říci, že určení optimálního počtu použitých oligonukleotidů, je velmi těžké odhadnout. Jedná se o velmi obtížný problém z teorie grafů. Intuitivně musí být jejich množství takové, aby s vysokou pravděpodobností vznikla alespoň jedna DNA molekula kódující hamiltonovskou cestu v grafu. Množství použitých oligonukleotidů musí tedy exponenciálně růst s počtem uzlů grafu. Délka oligonukleotidů je další významná proměnná. Adleman použil délku 20-mer, což mu dávalo celkový počet  $4^{20}$  možných oligonukleotidů. Bylo velmi nepravděpodobné, že by měli dva uzly grafu podobný nebo dokonce stejný kód. Také to zaručovalo, že vytvořené DNA molekuly byly stabilní i při pokojové teplotě. Na mnohem větší grafy by bylo vhodné délku oligonukleotidů zvětšit.

Celý experiment rozhodně nebyl jednoduchý. Adleman strávil 7 dní náročnou laboratorní prací, která vedla k nalezení hamiltonovské cesty v grafu. Nejvíce času mu zabrala závěrečná fáze experimentu (odstraňování DNA molekul, které neobsahují některý uzel grafu).

### 3.1.3 Formalismus

Pro lepší pochopení celého Adlemanova experimentu a použitých technik se zavádí jistý formalismus. Metody genetického inženýrství budou chápány jako součást „programovacího jazyka“.

**Zkumavka** je definována jako multimnožina konečných řetězců nad abecedou  $\{A, C, G, T\}$ . Zkumavku lze chápat jako kolekci jednoduchých vláken DNA, která jsou v ní různě namnožena. Nad zkumavkami jsou definovány tyto operace:

**Merge**            *Vstup:* Zkumavky  $N_1$  a  $N_2$ .

*Výstup:* Sjednocení obsahů zkumavek  $N_1 \cup N_2$  (chápano jako multimnožina).

- Amplify**      *Vstup:* Zkumavka  $N$ .  
*Výstup:* Dvě kopie zkumavky  $N$ .
- Detect**      *Vstup:* Zkumavka  $N$ .  
*Výstup:* „*true*“ - pokud zkumavka  $N$  obsahuje alespoň jedno DNA vlákno  
„*false*“ – pokud je zkumavka  $N$  prázdná
- Separate**      *Vstup:* Zkumavka  $N$  a řetězec  $w$  nad abecedou  $\{A, C, G, T\}$ .  
*Výstup:* Dvě zkumavky  $+(N, w)$  a  $-(N, w)$ .  $+(N, w)$  obsahuje všechny řetězce ze zkumavky  $N$ , které obsahují podřetězec  $w$ .  $-(N, w)$  obsahuje všechny řetězce ze zkumavky  $N$ , které podřetězec  $w$  neobsahují.
- Length-separate**      *Vstup:* Zkumavka  $N$  a celé kladné číslo  $n$ .  
*Výstup:* Zkumavka  $(N, \leq n)$  obsahující všechny řetězce z  $N$  s délkou menší nebo rovnu  $n$ .
- Position-separate**      *Vstup:* Zkumavka  $N$  a řetězec  $w$ .  
*Výstup:* Zkumavka  $B(N, w)$  obsahující všechny řetězce z  $N$  začínající podřetězcem  $w$ . Zkumavka  $E(N, w)$  obsahující všechny řetězce z  $N$  končící podřetězcem  $w$ .

Nyní je již možné pomocí těchto operací popsat formálně filtrační fáze Adlemanova experimentu. Počáteční obsah zkumavky  $N$  bude velké množství vzniklých DNA molekul (možných cest v grafu). Pro získání jednoduchých vláken DNA se zkumavka zahřeje. Oligonukleotidy hran se ignorují. Zkumavka tedy obsahuje pouze jednoduchá vlákna složená z 20-mer oligonukleotidů  $s_i$ , kde  $0 \leq i \leq 6$ , protože graf v Adlemanově experimentu obsahoval 7 uzlů.

- 
1. *input*( $N$ )
  2.  $N \leftarrow B(N, s_0)$
  3.  $N \leftarrow E(N, s_6)$
  4.  $N \leftarrow (N, \leq 140)$
  5. **for**  $i = 1$  **to**  $5$  **do begin**  $N \leftarrow +(N, s_i)$  **end**
  6. *detect*( $N$ )
- 

*Algoritmus č. 2: Formální zápis Adlemanova experimentu*

## 3.2 SAT-problém

Dalším problémem, na jehož vyřešení lze aplikovat techniku DNA počítání, je SAT-problém (satisfiability problem). Jedná se o určení splnitelnosti množiny výrokových formulí. Formule výrokové logiky  $\alpha$  je sestavena z proměnných  $x_1, x_2, x_3 \dots$  a logických spojek  $\sim, \wedge, \vee$  (negace, konjunkce, disjunkce) (*tabulka č. 1*). Je většinou zapsaná v konjunktivní normální formě a pokud ne, lze ji na ní převést. Příklad takové formule  $\alpha$  je

$$\alpha = (x_1 \vee \sim x_2 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\sim x_1 \vee x_3) \wedge x_3.$$

$\vee$	<b>0</b>	<b>1</b>
<b>0</b>	0	1
<b>1</b>	1	1

$\wedge$	<b>0</b>	<b>1</b>
<b>0</b>	0	0
<b>1</b>	0	1

$\mathbf{x}$	<b>0</b>	<b>1</b>
$\sim\mathbf{x}$	1	0

*Tabulka č. 1: Hodnoty logických operací*

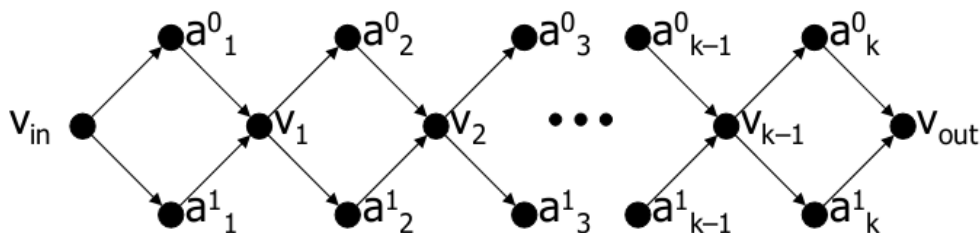
Proměnné ve formuli mohou nabývat hodnoty z množiny  $\{0, 1\}$ . Zde je hodnotou 0 myšlena pravdivostní hodnota „false“, hodnota 1 znamená „true“. SAT-problém je definován jako rozhodnutí, zda existuje takové přiřazení pravdivostních hodnot proměnným, že formule  $\alpha$  je pravdivá. Z toho plyne, že formule  $\alpha$  není splnitelná v případě, že její negace  $\sim\alpha$  je tautologie.

SAT-problém je opět ukázkou NP-úplného problému (stejně jako HPP) [13]. Není tedy možné najít jeho řešení pomocí efektivního algoritmu v polynomiálním čase a nezbyvá nic jiného, než masivní prohledávání stavového prostoru kombinací mapování všech proměnných ve formuli  $\alpha$ .

Řešení SAT-problému pomocí techniky DNA počítání navrhl Richard Lipton [4]. Jeho návrh je založen na převedení SAT-problému na grafový problém a použití podobného postupu jako Adleman. Díky masivnímu paralelismu DNA operací se stavový prostor kombinací mapování všech proměnných prohledává obrovskou rychlostí.

### 3.2.1 Převod na grafový problém

Je dána formule  $\alpha$ , která obsahuje  $k$  proměnných  $x_1, x_2, x_3, \dots, x_k$ . Formuli  $\alpha$  lze vyjádřit orientovaným grafem znázorněným na *obrázku č. 13*.



Obrázek č. 13: Orientovaný graf vyjadřující formuli  $\alpha$  [2]

Z grafu lze jasně vyvodit, že existuje  $2^k$  různých cest z uzlu  $v_{in}$  do  $v_{out}$ , přičemž žádná z nich není hamiltonovská. Je zde zohledněno to, že každá proměnná může nabývat jedné ze dvou hodnot. V orientovaném grafu má každý uzel  $v_{in}$ ,  $v_1$ , ...,  $v_{k-1}$  výběr ze dvou možností pokračování cesty, přičemž výběr každého uzlu je nezávislý na ostatních. Například cesta  $v_{in}a_1^0v_1a_2^0 \dots v_{k-1}a_k^0v_{out}$  odpovídá přiřazení hodnoty 0 všem proměnným formule  $\alpha$ . Obecně lze napsat, že cesta orientovaným grafem  $v_{in}a_1^i v_1 a_2^i v_2 \dots v_{k-1} a_k^i v_{out}$  odpovídá přiřazení hodnoty  $i_j$  proměnné  $x_j$ , kde  $j = 1, \dots, k$ .

Po sestrojení orientovaného grafu odpovídajícímu formuli  $\alpha$ , pro kterou se řeší SAT-problém, se pokračuje obdobným způsobem jako v Adlemanově experimentu (kapitola 3.1). Kódování uzlů a hran grafu je stejné. Pomocí ligázy se nechá vytvořit obrovské množství cest v grafu. Poslední krok Adlemanova experimentu již se ale neprovádí, jelikož v tomto případě se nehledá hamiltonovská cesta (graf ji ani neobsahuje).

Zajímavým rozdílem mezi Adlemanovým experimentem a Liptonovým návrhem řešení SAT-problému je to, že pro různé formule  $\alpha_1, \alpha_2, \dots, \alpha_n$  o stejném počtu proměnných  $k$  je orientovaný graf stále stejný. Skládá se ze stejných uzlů i hran ve stejném pořadí, proto se nemění ani jejich zakódování do oligonukleotidů. Počáteční obsah zkumavky je tedy pro každou formuli  $\alpha_i$  stejný. V Adlemanově experimentu není počáteční obsah zkumavky stejný pro žádné dva rozdílné orientované grafy, protože graf je v tomto případě hlavním vstupem problému.

### 3.2.2 Formalismus

K formálnímu řešení SAT-problému budou zapotřebí operace *separate*, *merge* a *detect* definované v podkapitole 3.1.3. Operace *separate* bude pro zpřehlednění zápisu ještě rozšířena.

- $S(N, i, j) = +(N, a^i_j)$  Vybere množinu řetězců z  $N$ , které mají na  $i$ -tém bitu hodnotu  $j$ .
- $S(N, i, j) = -(N, a^i_j)$  Vybere množinu řetězců z  $N$ , které mají na  $i$ -tém bitu hodnotu komplementu  $j$ .

Je dána formule  $\beta = (x_1 \vee x_2) \wedge (\sim x_1 \vee \sim x_2)$ . Formální zápis algoritmu řešení SAT-problému je následující.

1.  $input(N_0)$
2.  $N_1 = S(N_0, 1, 1)$
3.  $N'_1 = S(N_0, 1, 1)$
4.  $N_2 = S(N'_1, 2, 1)$
5.  $merge(N_1, N_2) = N_3$
6.  $N_4 = S(N_3, 1, 0)$
7.  $N'_4 = S(N_3, 1, 0)$
8.  $N_5 = S(N'_4, 2, 0)$
9.  $merge(N_4, N_5) = N_6$
10.  $detect(N_6)$

---

*Algoritmus č. 3: Formální zápis řešení SAT-problému formule  $\beta$*

---

Z algoritmu lze vyčíst postup při řešení SAT-problému. V krocích 2, 4, 6, 8 se používá funkce *seperate* ve smyslu  $+(N, w)$ . V krocích 3 a 7 ve smyslu  $-(N, w)$ . Pro lepší představu o průběhu filtrace si po každém kroku uvedeme obsah zkumavky.

Krok	1	2	3	4	5	6	7	8	9
Obsah zkumavky	00,01,10,11	10,11	00,01	01	10,11,01	01	10,11	10	01,10

*Tabulka č. 2: Obsah zkumavky v průběhu filtrace při řešení SAT problému*

Na konci algoritmu je neprázdná zkumavka, výstupem je tedy „true“. Na začátku obsahuje zkumavka množinu všech možných přiřazení proměnných. V kroku 5 již vzniká zkumavka, která obsahuje přiřazení splňující první klauzuli formule  $\beta$ . Tato množina je nadále filtrována a v kroku 9 již splňuje i druhou klauzuli formule.

Výše uvedený postup lze zobecnit pro libovolnou formuli  $\alpha$  a vede na lineární časovou složitost, což je při DNA počítání přijatelné. Platí to ale pouze za předpokladu, že jsou používané operace bezchybné, což nelze z biologického pohledu zaručit. I kdyby vykazovaly určitou chybovost, v počáteční zkumavce mohou být prvky v mnoha kopiích. Potom se sice některé správné řešení může ztratit při filtraci, ale s vysokou pravděpodobností je stále konečné řešení nalezeno.

### 3.3 Hledání maximální kliky grafu

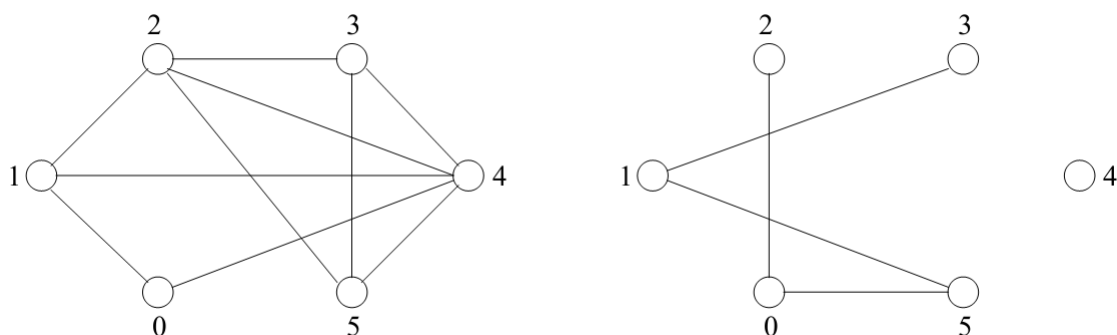
Po objevu Adlemana, který dokázal, že pomocí DNA počítání lze řešit NP-úplné problémy (v jeho případě se jednalo o problém nalezení hamiltonovské cesty grafu) v lineárním čase, se začali zkoumat i další takové problémy a jejich vhodné zakódování pro techniku DNA počítání. V roce 1997 dokázal Ouyang a kol. [8] vyřešit pomocí DNA počítání problém nalezení maximální kliky grafu.

Klika grafu je matematicky definována jako množina uzlů grafu, ve které je každý uzel spojen hranou se všemi ostatními uzly v této množině. Jedná se tedy o největší úplný podgraf grafu. Řešením problému maximální kliky grafu o  $N$  uzlech a  $M$  hranách je počet uzlů v největší klice grafu. Na *obrázku č. 14* je znázorněn graf se 6 uzly a 11 hranami, pro který je definován problém maximální kliky grafu. V tomto grafu tvoří maximální kliku uzly  $\{5, 4, 3, 2\}$ , řešením problému je tedy 4.

Při řešení tohoto problému se dále využije pojem komplementární graf. Mějme graf  $G'$ , který obsahuje stejné uzly jako původní graf  $G$  a všechny hrany, které v původním grafu chyběly. Takový graf  $G'$  je komplementem grafu  $G$  a naopak.

$$G = (V, E), \quad G' = (V', E'), \text{ kde } V, V' \text{ jsou množiny uzlů a } E, E' \text{ jsou množiny hran}$$
$$V' = V$$
$$E' = K/E, \text{ kde } K \text{ obsahuje všechny dvouprvkové podmnožiny množiny } V$$

Ukázka komplementárního grafu je uvedena na *obrázku č. 14*. Každé dva uzly, které jsou v komplementárním grafu spojeny hranou, nejsou v původním grafu propojeny, tudíž nemůžou být oba součástí stejné kliky původního grafu. Pokud vytvoříme ze dvou navzájem komplementárních grafů  $G$  a  $G'$  nový graf  $U = (V, E \cup E')$ , jedná se logicky o úplný graf.



Obrázek č. 14: Graf se 6 uzly (nalevo) a k němu komplementární graf (napravo) [8]

Zakódování kliky grafu je velmi intuitivní. Pro graf obsahující  $N$  uzlů je každá jeho klika reprezentována  $N$ -místným binárním číslem. Bit nastaven na  $1$  značí, že jemu odpovídající uzel se nachází v dané klice grafu. Bit nastaven na  $0$  naopak značí, že jemu odpovídající uzel se v dané klice nenachází. V grafu na *obrázku č. 14* je například jeho maximální klika  $(5, 4, 3, 2)$  reprezentována číslem  $111100$ . K vyřešení problému nalezení maximální kliky grafu byl navržen následující algoritmus.

- 
- 1. Podle stylu zakódování uvedeného výše se převede soubor všech možných klik grafu o  $N$  uzlech na soubor  $N$ -místných binárních čísel. Takovému souboru se bude říkat kompletní datový soubor.*
  - 2. Vytvoří se komplementární graf. Tím se naleznou uzly v původním grafu, které nejsou spojeny hranou. Každé dva uzly, které jsou v komplementárním grafu spojeny hranou, nejsou v původním grafu propojeny, tudíž nemůžou být oba součástí stejné kliky původního grafu. To znamená, že jim odpovídající bity nesmějí být oba nastaveny na  $1$ .*
  - 3. Z kompletního datového souboru se odstraní všechna binární čísla, která obsahují spojení v komplementárním grafu. Zbývá binární čísla reprezentují všechny kliky původního grafu.*
  - 4. Zbývající část kompletního datového souboru se seřadí podle počtu bitů nastavených na  $1$ , resp. podle počtu uzlů v klice grafu. Klika s největším počtem bitů nastavených na  $1$  tedy udává velikost maximální kliky grafu.*

---

*Algoritmus č. 4: Algoritmus pro řešení problému nalezení maximální kliky grafu [8]*

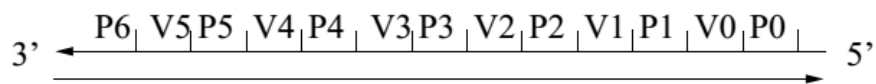
Nejdůležitější částí algoritmu je krok č. 3, který odfiltruje z kompletního datového souboru všechny podgrafy, které nejsou klikami grafu. Pro graf na *obrázku č. 14* se z datového souboru konkrétně odstraní čísla  $***1*1$ ,  $1****1$ ,  $1***1*$  a  $**1*1*$  (\* může nabývat hodnot  $1$  nebo  $0$ ). U všech těchto čísel platí, že nerepresentují kliku daného grafu.

### **3.3.1 Zakódování do DNA**

V předcházející podkapitole bylo vysvětleno zakódování grafových struktur do podoby binárních čísel tak, aby s nimi mohl algoritmus efektivně pracovat. Pokud se pro problém nalezení maximální kliky grafu využije takové zakódování, není problém ho vyřešit za pomoci *algoritmu č. 4* na

konvenčním počítači. Tato práce se ale zabývá řešením problémů DNA výpočtem za využití masivního paralelismu. Nastává tedy další problém: Jak zakódovat binární čísla reprezentující podgrafy na DNA sekvence na úrovni nukleotidů, aby byl tento způsob vhodný pro řešení problému DNA výpočtem?

Každý podgraf grafu (binární číslo) je zakódován jako dvouřetězcová DNA molekula. Každý bit binárního čísla je reprezentován dvěma DNA sekcemi. Jedna sekce koresponduje s hodnotou bitu ( $V_i$ ) a druhá s jeho pozicí ( $P_i$ ). Podgraf grafu o 6 uzlech (6-místné binární číslo) je tedy reprezentován 6 hodnotovými DNA sekcemi ( $V_0 - V_5$ ), které jsou prokládané 7 pozičními DNA sekcemi ( $P_0 - P_6$ ). Poslední poziční DNA sekce  $P_6$  je důležitá zejména pro namnožení DNA pomocí metody PCR. Ouyang vyzkoumal, že ideální délka poziční sekce ( $P_i$ ) je 20 párů bází. Hodnotová sekce ( $V_i$ ) má nulovou délku, pokud je odpovídající bit nastaven na 1, nebo délku 10 párů bází, pokud je odpovídající bit nastaven na 0. Nejdelší DNA pro podgraf grafu o 6 uzlech má tedy délku 200 párů bází, což odpovídá číslu 000000, a nejkratší DNA má délku 140 párů bází odpovídající číslu 111111. Příklad zakódování podgrafu grafu o 6 uzlech je uveden na *obrázku č. 15* a *obrázku č. 16*.



Obrázek č. 15: Zakódování podgrafu do DNA [8]

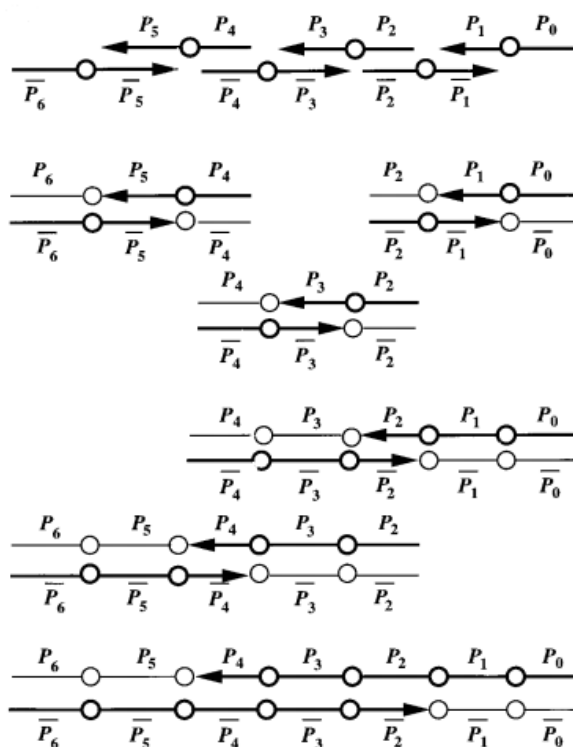
Index	Position Sequence	Value Sequence
0	CGTAGAATTCTGCGAACCTT	GACGCGGCAG
1	AAGAGTCACCTATCAGTAAG	TTCATAACCG
2	CTTCCTGAAGAGAGATTACT	CCGGTCACTT
3	AGTGCTAACAAGCCTGCGCA	CCTGACGATT
4	TGCAGTCTTTGAGATAAAGG	AAAAACCCAC
5	CCTACGTGTGAAATAGACTC	CCACCGATCT
6	GAGAGGGGCGGGATCCAGGG	

Obrázek č. 16: DNA sekvence jednotlivých sekcí [8]

Aby bylo možné pomocí DNA počítání tvořit různé podgrafy, je nutné začínat s jednotlivými oligonukleotidy, ze kterých se různými kombinacemi vytvoří výsledná DNA sekvence. Každý oligonukleotid obsahuje jednu hodnotovou sekci ohraničenou dvěma pozičními sekcemi.  $P_iV_iP_{i+1}$  pro sudá  $i$  a komplementární sekvence  $P_{i+1}V_iP_i$  pro lichá  $i$ . DNA sekvence jednotlivých sekcí byly nejprve náhodně generovány, ovšem v některých případech začal výpočet selhávat. Proto se kvůli spolehlivému párování komplementárních sekcí začal klást důraz na to, aby se v nich nevyskytovaly náhodné homologie o delší než 4 páry bází.

### 3.3.2 Hlavní fáze

Všechny druhy oligonukleotidů se na začátku smíchají dohromady a komplementární poziční sekce se začnou spojovat. Toto spojování probíhá v tzv. teplotních cyklech, kdy v každém cyklu dochází k postupnému prodlužování dvouřetězcové DNA. Po několika teplotních cyklech je vytvořen kompletní datový soubor všech kombinací  $V_0 V_1 V_2 V_3 V_4 \dots V_i$ . Průběh tohoto procesu je uveden na obrázku č. 17. Následuje namnožení DNA molekul pomocí metody PCR, kde hrají okrajové poziční sekce  $P_0$  a komplementární  $P_{i+1}$  roli primerů. To zajistí, že se namnoží pouze molekuly, které obsahují na svých koncích sekce  $P_0$  a  $P_{i+1}$ , tudíž molekuly obsahují všechny hodnotové sekce.



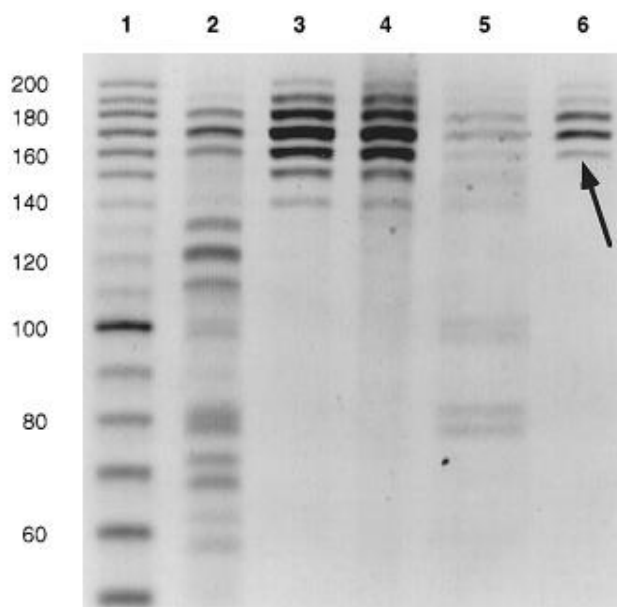
Obrázek č. 17: Stavění výsledné DNA molekuli z jednotlivých oligonukleotidů [8]

Následuje filtrace DNA molekul, které nerepresentují kliku grafu. K tomuto úkolu se používají speciální restrikční enzymy. Restrikční enzymy mají schopnost rozbít DNA molekulu v přesně určených restrikčních zónách, které byly na začátku výpočtu vloženy do sekvence každé hodnotové sekce  $V_i$  reprezentující hodnotu  $I$ . Rozbité molekuly již neobsahují všechny hodnotové sekce, a tudíž nebudou metodou PCR namnoženy.

K filtraci DNA molekul se použije komplementární graf. Například na obrázku č. 14 obsahuje komplementární graf hranu mezi uzly 0 a 2. Tyto dva uzly tedy nemohou být oba současně v klice grafu. Kompletní datový soubor DNA molekul se proto rozdělí do dvou zkumavek. V první zkumavce se pomocí restrikčních enzymů rozbijí všechny molekuly, které obsahují hodnotovou sekci

$V_0$  reprezentující hodnotu 1, a ve druhé zkumavce všechny ty, které obsahují hodnotovou sekci  $V_2$  reprezentující hodnotu 1. Následně se obsah obou zkumavek smíchá a neporušené zůstanou pouze molekuly, které neobsahují vzor \*\*\*\*1\*1. Stejným způsobem se odfiltrují všechny ostatní DNA molekuly nereprezentující kliku grafu. Na závěr se kompletní obsah namnoží metodou PCR, která ignoruje všechny rozbité molekuly, jelikož neobsahují všechny hodnotové sekce (výsledkem je sloupec č. 6 na obrázku č. 18).

Určení maximální kliky grafu je díky zakódování DNA molekuly velmi snadné. Maximální klika obsahuje nejvíce hodnotových sekcí reprezentujících hodnotu 1, tudíž musí mít její molekula nejkratší DNA sekvenci (označeno šipkou na obrázku č. 18). Délka DNA molekul se určí gelovou elektroforézou. Z nejkratší délky se z velikostí jednotlivých sekcí odvodí, kolik uzlů obsahuje maximální klika grafu. Například délka DNA molekuly 160 párů bází reprezentuje kliku grafu o 4 uzlech.

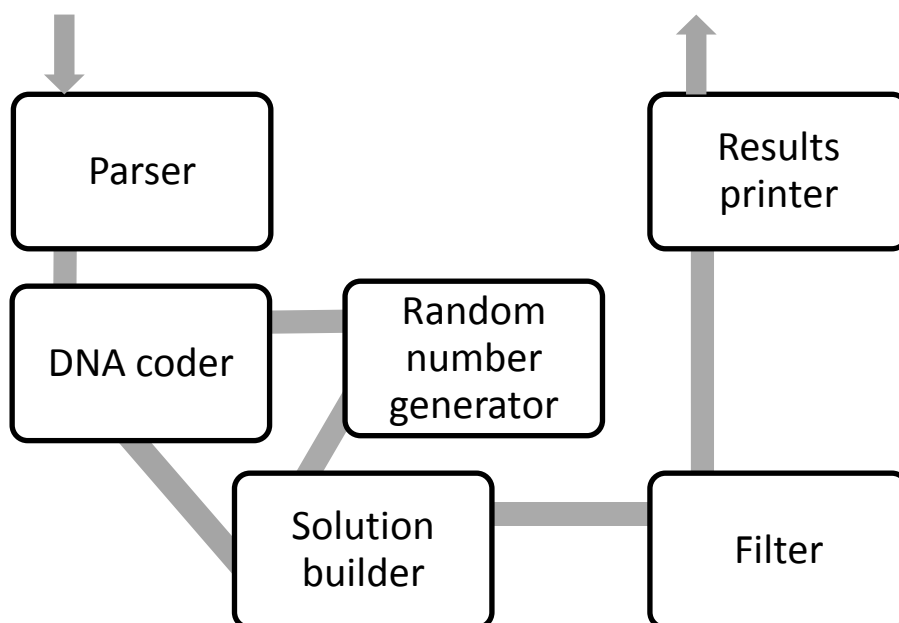


Obrázek č. 18: Výsledky gelové elektroforézy v průběhu hledání maximální kliky grafu pomocí DNA počítání [8]

## 4 Návrh aplikace

Tato práce se zabývá návrhem a implementací aplikace pro simulaci DNA výpočtů pro řešení problémů uvedených v předcházející kapitole. Důraz bude následně kladen hlavně na zjištění, zda opravdu takové problémy lze pomocí těchto technik řešit, na měření výkonnosti simulátoru a jeho zefektivnění. Hlavním cílem rozhodně není usnadnit detailní pochopení vyskytujících se DNA operací z chemicko-biologického hlediska. Není tedy zapotřebí navrhovat aplikaci, která bude veškeré chemické procesy vizualizovat a která by tím pádem musela logicky obsahovat komplexní grafické uživatelské rozhraní. Pro implementaci simulátoru DNA počítání byl použit jazyk C v prostředí operačního systému Linux. Jedná se o nízkoúrovňový a relativně minimalistický programovací jazyk, který je zcela dostačující pro efektivní implementaci simulátoru. Výsledným produktem bude tedy jednoduchá konzolová aplikace.

Simulátor se bude skládat z několika komponent, z nichž každá bude mít svou logickou funkci. Diagram jednotlivých komponent je uveden na *obrázku č. 19*. Po zvolení typu řešeného problému otevře aplikace zadaný vstupní soubor, ve kterém bude vždy uveden přesný popis struktury, pro kterou se bude problém řešit. U problému hledání hamiltonovské cesty jí bude orientovaný graf, u SAT problému půjde o množinu výrokových formulí a u problému nalezení maximální kliky grafu o neorientovaný graf. Každá z těchto struktur má svůj definovaný formát vstupního souboru. O správné načtení struktury pro řešení daného problému se stará komponenta *Parser*.



Obrázek č. 19: Diagram jednotlivých komponent aplikace

U SAT problému provede ještě komponenta *Parser* převedení z formy množiny výrokových formulí na grafový problém. Nyní již může aplikace pracovat s grafovými strukturami u všech typů řešeným problémů.

Komponenta *DNA coder* zakóduje jednotlivé části, ze kterých se bude skládat výsledné DNA řešení. Z hlediska DNA výpočtů velmi důležitou komponentou je *Solution builder*, která pomocí komponenty *Random generator* vybírá náhodně zakódované části a staví z nich výsledné DNA řešení. Stavění probíhá na základě náhody a jsou při něm dodržena omezení vyplývající ze zakódování daného problému. Komponenta *Solution builder* sestaví daný počet kandidátních DNA řešení a odešle je do nejdůležitější komponenty simulátoru, a tím je *Filter*.

V něm se odehrává nejobtížnější část DNA počítání. U každého kandidátního DNA řešení musí být zkontrolováno, jestli struktura, kterou reprezentuje, vyhovuje zadanému problému. Například jestli se opravdu jedná o hamiltonovskou cestu grafem nebo jestli se jedná o kliku grafu. Toto zjištění by nebylo u klasické reprezentace struktur příliš obtížné, u struktur zakódovaných do DNA je to ale naopak.

Poslední komponentou je *Results printer*, která slouží hlavně pro účely měření. Ze souboru přefiltrovaných kandidátních DNA řešení vypočítá jednoduché statistiky, popř. vytiskne řešení problému v pseudo-DNA formě. Detailněji je tato struktura popsána v následující podkapitole.

## 4.1 Datová struktura DNA sekvence

Cílem aplikace má být ověření schopnosti DNA počítání najít řešení pro dané problémy a změřit jeho výkonnost. Pro efektivní a přehlednou implementaci byla proto zvolena datová reprezentace DNA sekvence na vyšší úrovni. Jednotlivé oligonukleotidy, ze kterých se postupně skládá výsledné kandidátní DNA řešení, nejsou reprezentovány na úrovni nukleotidů, ale každý má své jedinečné ID. Ve skutečnosti se také výsledné DNA řešení objevuje ve formě dvouřetězcové DNA, přičemž v aplikaci se počítá pouze s jednořetězcovou DNA.

Obě tyto datové abstrakce přísně dodržují principy DNA počítání a byly již použity k simulaci DNA počítání v [9]. Je při nich dodržen princip komplementarity bází a unikátnost sekvence oligonukleotidů. Dvouřetězcová forma DNA je ve skutečnosti velmi důležitá pro správné skládání výsledného DNA řešení a díky ní je možno při počítání využít komplementarity bází, což je pro DNA výpočty klíčové. V aplikaci je sice pro přehlednost DNA řešení reprezentováno jednovláknově, ale při připojování dalšího oligonukleotidu se vždy imaginární komplementární vlákno bere v potaz tak, aby byl dodržen princip komplementarity bází.

Každé výsledné kandidátní DNA řešení je implementováno strukturou, jejíž hlavním prvkem je pole celých čísel, z nichž každé značí ID oligonukleotidu nebo neobsazené místo (hodnota 0). Struktura obsahuje i ukazatele na první a poslední oligonukleotid v poli. Pole má vždy omezenou

velikost, která plyne ze zadání řešeného problému. Simulace hybridizace oligonukleotidů probíhá od středu pole, následně se podle pravidla komplementarity bázi DNA kandidátní řešení prodlužuje na obou svých koncích. Proto mohou mít i stejná kandidátní DNA řešení jiný obsah svých datových polí v důsledku toho, že jejich hybridizace začala od jiného počátečního oligonukleotidu ve středu pole.

...AAA	CCCAAA	CCCAAA	GGGTTT	CCCGGG	AAAGGG
TTTGGG	TTTGGG	TTTCCC	AAATTT	CCCTTT	CCC...

a)

Oligonukleotid	ID
CCCAAA	1
GGGTTT	2
CCCGGG	3
AAAGGG	4
...	...

b)

0	0	0	0	1	1	2	3	4	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1	1	2	3	4	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0	0	1	1	2	3	4	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

c)

Obrázek č.20: Na obrázku a) je znázorněna skutečná struktura DNA řešení. Obrázek b) ukazuje zakódování jednotlivých oligonukleotidů do datové abstraktní formy. Obrázek c) ukazuje několik datových struktur DNA řešení odpovídající a).

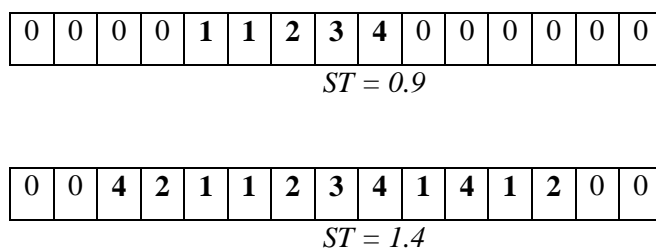
## 4.2 Parametry simulátoru

Aby bylo možné pomocí aplikace provádět nejrůznější experimenty a zkoumat výkonnost DNA počítání, obsahuje simulátor několik důležitých parametrů. Parametr *IT* udává, kolik se bude simulátor snažit vytvořit celkem kandidátních DNA řešení. Slovo „snažit“ je zde na místě a jeho další význam bude dále osvětlen. Ne vždy se totiž simulátoru podaří vytvořit daný počet kandidátních DNA řešení. Tento parametr je důležitý zejména pro experimenty s na poměry DNA menším počtem kandidátních řešení (řádově  $10^6$ ), kdy bude hrát důležitou roli jejich přesný počet. U složitějších instancí problémů bude vhodné tento parametr nastavit na 0, což zaručí, že se budou vytvářet kandidátní DNA řešení tak dlouho, dokud nebude nalezeno řešení. Zde je nutné zmínit, že nastavení parametru *IT* na 0 má význam pouze pro problém nalezení hamiltonovské cesty grafu a SAT problém. U obou těchto problémů totiž simulátor bezpečně pozná, zda se jedná o řešení problému.

Pokud tedy ve filtrační fázi zjistí, že dané kandidátní řešení je hamiltonovská cesta grafu nebo že splňuje všechny klauzule SAT problému, vytiskne řešení a končí výpočet. U problému nalezení maximální kliky grafu však simulátor nezná opravdovou maximální kliku grafu a nevěděl by, kdy výpočet ukončit.

Dalším parametrem je  $AM$ . Na něm velmi závisí celkový počet vygenerovaných kandidátních DNA řešení. Tento parametr totiž udává množství každého typu oligonukleotidů, ze kterých se staví kandidátní řešení. Pokud bude toto množství malé, může se stát, že se nestačí vygenerovat zadaný počet kandidátních řešení  $IT$ , jak již bylo zmíněno v předcházejícím odstavci.

Posledním velmi důležitým parametrem je  $ST$ . Simuluje dobu, po kterou se nechají oligonukleotidy reagovat, a intenzitu jejich promíchání. Při malém nastavení tohoto parametru vznikají kratší kandidátní DNA řešení, jelikož šanci na prodloužení DNA molekuly na obou stranách je poměrně málo. Naopak při vysokém  $ST$  a dostatečném počtu oligonukleotidů  $AM$ , by měla vznikat delší kandidátní DNA řešení, bude jich ale méně (obrázek č. 21). Různé nastavení těchto parametrů a jejich různé kombinace budou zkoumány v kapitole č. 5.



Obrázek č.21: Vliv parametru  $ST$  na délku kandidátního DNA řešení při dostatečném počtu oligonukleotidů.

### 4.3 Generování kandidátních řešení

Generování kandidátních řešení má v aplikaci simulovat proces hybridizace jednotlivých stavebních oligonukleotidů. Tento proces probíhá sice striktně na základě pravidla komplementarity bází, ale obsahuje v sobě výrazný podíl náhody. Určení prvního oligonukleotidu, který se naváže na druhý, je nedeterministické. Stejně tak prodlužování kandidátního řešení na obou koncích, pokud vyhovuje více oligonukleotidů, je náhodné. A to nejdůležitější, délka generovaných kandidátních DNA řešení je ve skutečnosti také nedeterministická. Všechny tyto principy, které se uplatňují u reálného DNA výpočtu, musí simulátor vhodně implementovat. Navíc musí vhodně implementovat omezenou kapacitou stavebních oligonukleotidů a ošetřit omezení paměti pro datové pole DNA řešení. Přehledně zapsaný implementovaný algoritmus využívající parametry  $IT$ ,  $AM$  a  $ST$  lze nalézt v *algoritmu č. 5*.

---

```

setOligosCount(AM);
for(int i = 0; i < IT; i++) {
    if(oligosEmpty) {
        break;
    }

    random = rand() % edge_count;           // add first oligo
    placeOligo(oligos[random]);
    oligos_am[random]--;

    for(int y = 0; y < node_count*ST; y++) {
        random = rand() % edge_count;

        // add to front
        if(possibilityPlaceOligoFront(oligos[random])) {
            if(!outOfArray) {
                placeOligoFront(oligos[random]);
                edges_am[random]--;
                if(oligosEmpty) {
                    break;
                }
            }
        }
        // add to back
        else if(possibilityPlaceOligoBack(oligos[random])) {
            if(path(!outOfArray) {
                placeOligoBack(oligos[random]);
                edges_am[random]--;
                if(oligosEmpty) {
                    break;
                }
            }
        }
    }
}

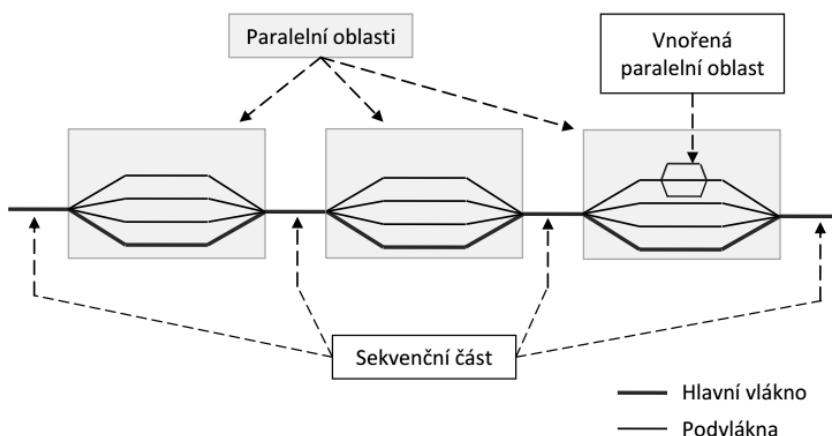
```

## 4.4 Paralelizace výpočtu pomocí OpenMP

Při návrhu urychlení simulátoru byl kladen důraz hlavně na zachování principů DNA počítání. Hlavní výhodou DNA výpočtů je jejich masivní paralelismus. Při sekvenční simulaci na počítači se tedy hlavní výhoda vytrácí a výpočet trvá dlouho. Proto byl do aplikace zaveden alespoň částečný paralelismus. Je jasné, že na dnešních počítačích se nikdy nedosáhne paralelismu tak masivního, jako je tomu v případě DNA operací.

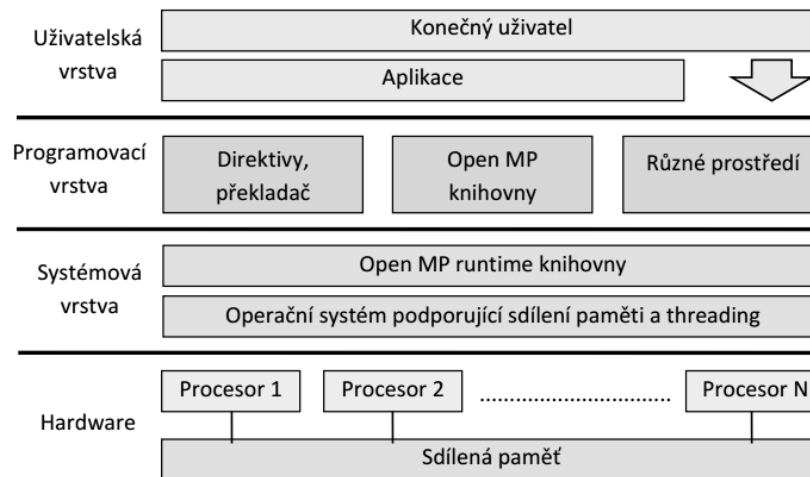
OpenMP (Open Multi-Processing) je speciální API, které umožňuje paralelní programování v jazycích C, C++ a Fortran na většině operačních systémů a procesorových architekturách. Jedná se o standard pro programování počítačů se sdílenou pamětí. Obsahuje soustavu direktiv pro překladač, knihovných procedur a proměnných prostředí, které ovlivňují způsob běhu aplikace.

OpenMP implementuje paralelizaci pomocí více vláken. Hlavní vlákno (master thread) vytváří podle potřeby skupinu podvláken (subthread), mezi které se daná úloha rozdělí. Vlákná pak běží současně, díky alokaci na různé procesory. Schéma tohoto procesu lze vidět na *obrázku č. 22*. Každé vlákno má své unikátní ID, které lze získat pomocí funkce `omp_get_thread_num()`. ID vlákna je datové typu integer a hlavní vlákno má vždy ID 0.



Obrázek č. 22: Znárodnění hlavního vlákna a jeho podvláken [11]

OpenMP se skládá z několika vrstev (*obrázek č. 23*). Uživatel přistupuje k OpenMP prostřednictvím aplikace. Aplikaci naprogramoval programátor, který má možnost použít direktivy překladače, knihovni procedury nebo proměnné prostředí z programovací vrstvy. Všechny tyto prvky ovlivňují runtime knihovny v systémové vrstvě, které jsou již implementovány na konkrétní druh operačního systému. Programovací vrstva je tedy nezávislá na operačním systému, kdežto u systémové vrstvy je tomu naopak. Poslední vrstvou, která ale nejvíce ovlivňuje rychlost paralelizace pomocí počtu a taktu jader, je hardware.



Obrázek č. 23: Schéma jednotlivých vrstev OpenMP [11]

### 4.4.1 OpenMP API

Direktivy OpenMP pro jazyky C a C++ jsou specifikovány direktivou preprocesoru `pragma`. Zápis direktiv je case-sensitive. Každá OpenMP direktiva tedy začíná zápisem `#pragma omp`. Platnost každé OpenMP direktivy je vztažena na následující strukturovaný blok instrukcí. Každou direktivu lze modifikovat pomocí několika klauzulí.

```
#pragma omp název_direktivy [ klauzule ]
```

Většina proměnných je v OpenMP sdílená mezi všemi vlákny. Někdy je ale zapotřebí zavést soukromé proměnné vlákna, aby se zabránilo datovým konfliktům. Navíc je zapotřebí ošetřit předávání hodnot mezi sekvenční a paralelní částí programu. Toto a další problémy lze řešit pomocí následujících klauzulí (uvedeny budou pouze ty nejdůležitější a jejich přesnou syntaxi lze nalézt v [10]).

Sdílení dat:

- *shared*: proměnné jsou v paralelní části sdíleny mezi vlákny. Ve výchozím nastavení jsou sdíleny všechny proměnné v paralelní části s výjimkou iteračních čítačů smyček.
- *private*: každé vlákno má svou lokální kopii proměnné a používá ji jako dočasnou proměnnou. Soukromá proměnná není inicializována.
- *default*: výchozí nastavení sdílení pro všechny proměnné.
- *firstprivate*: stejné jako *private*, ale každá proměnná se inicializuje na hodnotu, kterou měla před paralelní částí.
- *lastprivate*: stejné jako *private*, ale každá proměnná předá svou hodnotu své originální kopii vně paralelní části.

- *reduction*: každé vlákno má svou soukromou proměnnou a na konci paralelní části se provede jejich redukce do globální sdílené proměnné pomocí některé asociativní operace (+, \*, &, &&, |, ||).

#### Synchronizace:

- *critical*: následující strukturovaný blok bude v určitém čase počítán vždy pouze jedním vláknem. Používá se nejčastěji k ochraně sdílených dat před konflikty v kritické sekci.
- *atomic*: operace s pamětí (čtení, úprava, zápis) v následující instrukci bude provedena atomicky. Vztahuje se to pouze na operaci s pamětí, nikoliv celý příkaz. Kompilátor u této klauzule může využít speciální hardwarové instrukce k dosažení lepší výkonnosti než u *critical*.
- *ordered*: následující strukturovaný blok bude spuštěn ve stejném pořadí, v jakém by tomu tak bylo při sekvenčním výpočtu.
- *barrier*: každé vlákno nejprve čeká, než všechny ostatní příbuzná vlákna nedosáhnou tohoto bodu. Ve výchozím nastavení je při paralelním výpočtu nastaven barrier bod na konec výpočtu.
- *nowait*: vlákno nemusí čekat na ostatní příbuzná vlákna.

#### Plánování:

- *schedule*: tato klauzule je vhodná zejména pro smyčky do a for. Jednotlivé iterace smyček jsou rozděleny mezi vlákna podle jednoho ze tří druhů plánování.
  - *static*: iterace jsou mezi jednotlivá vlákna rozdělena před samotným výpočtem. Ve výchozím nastavení je rozdělení rovnoměrné, lze ho však pomocí parametrů přesně specifikovat.
  - *dynamic*: sady blízkých iterací se přiřadí ke vláknům. Jakmile vlákno iterace dokončí, přiřadí se mu další sada iterací, která ještě nebyla alokována.
  - *guided*: alokace iterací probíhá stejně jako u klauzule *dynamic*. Na začátku se přiřazují poměrně velké sady blízkých iterací a postupem času se alokované sady zmenšují.

#### Další:

- *if*: následující strukturovaný blok se bude počítat paralelně, pouze pokud je splněna podmínka. Jinak bude počítán sekvenčně.
- *num\_threads*: specifikuje počet vláken pro výpočet následujícího strukturovaného bloku.
- *flush*: hodnota dané proměnné je převedena z registru do paměti, aby ji bylo možné použít vně paralelní části.
- *master*: následující strukturovaný blok bude počítán pouze hlavní vlákno (master thread).

Nezákladnější OpenMP direktivou je **parallel**. Způsobí, že následující strukturovaný blok instrukcí bude zpracován více vlákny. Další velmi důležitou direktivou je **for**, která se používá pro paralelizaci for cyklů. Všechna vlákna tedy vykonávají stejnou úlohu (zpracovávají stejné sekce kódu) – dělí se o cyklus for. Pro případ, kdy má každé vlákno provádět jinou úlohu (vlákna zpracovávají různé sekce kódu) se používá direktiva **sections**.

OpenMP také využívá některé pomocné funkce z hlavičkového souboru `<omp.h>` a systémové proměnné. Nejdůležitější jsou:

- `void omp_set_num_threads(int num_threads);`  
- nastaví počet vláken.
- `int omp_get_num_threads();`  
- vrací počet vláken.
- `int omp_get_thread_num();`  
- vrací číslo aktuálního vlákna.
- `int omp_get_num_procs();`  
- vrátí počet procesorů, které jsou k dispozici, když je zavolána funkce.
- `OMP_NUM_THREADS`  
- proměnná obsahuje nastavený počet vláken.

## 5 Dosažené výsledky

Při provádění experimentů bude velmi často používán pojem výkonnost simulátoru. Výkonností simulátoru je myšleno nejen to, jestli dokáže nalézt řešení problému, ale hlavně kolik jich z celkového počtu vygenerovaných kandidátních řešení bylo. Pro zkoumání závislosti složitosti instancí na výkonnosti simulátoru je tato úprava velmi vhodná. Bez ní by nešel odvodit charakter závislosti a byla by nalezena pouze hranice složitosti instance, kterou simulátor při daných parametrech nedokázal vyřešit.

Na začátku vyhodnocování výkonnosti simulátoru je nejprve nutné otestovat, zda opravdu funguje správně a podle principu DNA počítání. Ke každému řešenému problému bylo sestaveno několik různých referenčních instancí. Pro každou takovou instanci je nutné otestovat, jestli simulátor je schopen problém vyřešit a naopak jestli u neřešitelných problémů nedává simulátor falešné výsledky. Pokud například graf neobsahuje žádnou hamiltonovskou cestu, nesmí aplikace najít žádné řešení ani po nastavení kapacity oligonukleotidů a počtu kandidátních DNA řešení na velmi vysoké hodnoty.

Protože testovaných instancí problémů bude mnoho druhů, je velmi vhodné zavést vhodnou terminologie názvů instancí a souborů, které obsahují jejich detailní popis, aby se v nich lépe orientovalo a byly už z názvu jasné hlavní parametry instance. Nicméně je jasné, že dvě instance o stejných parametrech (např. orientované grafy o 7 uzlech a 13 hranách) nemusí dávat stejné výsledky, jelikož struktura grafu může být jiná. Popis terminologie názvů je uveden v *tabulce č. 3*.

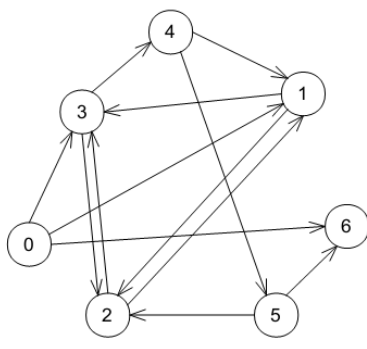
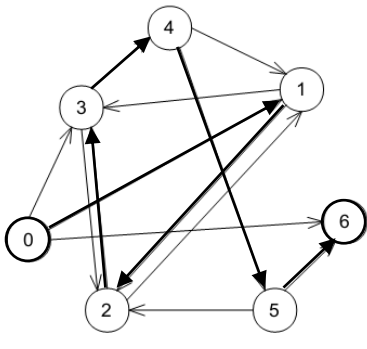
<i>hppX<sub>Y</sub></i>	<i>satX<sub>Y</sub></i>	<i>cliX</i>
<b>Problém nalezení hamiltonovské cesty</b>	<b>SAT problém</b>	<b>Problém nalezení maximální kliky grafu</b>
X – počet uzlů grafu Y – počet orientovaných hran	X – počet proměnných Y – počet klauzulí	X – počet uzlů grafu

*Tabulka č.3: Terminologie názvů instancí problémů*

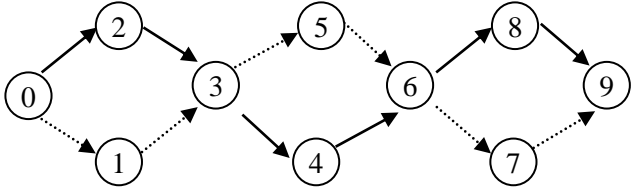
### 5.1 Správnost simulace

V této části experimentů se ověří, jestli simulátor funguje správně a podle principu DNA počítání. K ověření validity výsledků simulace byly použity jednodušší referenční instance problémů, aby se daly snadno dodatečně vizualizovat a správnost potvrdit. U složitějších instancí by se správnost výsledků musela ověřovat pomocí speciálních nástrojů určených k řešení daných problémů. Už

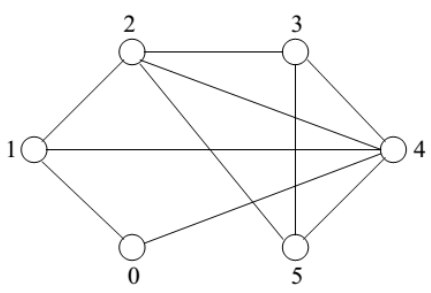
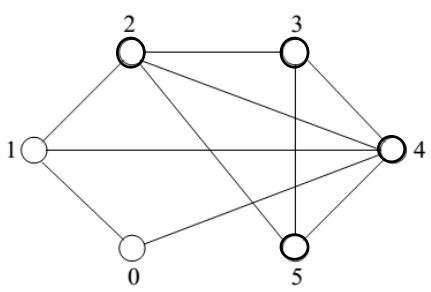
z principu DNA počítání lze však vyvodit, že pokud funguje simulátor pro řadu jednodušších instancí korektně, bude fungovat i pro složitější instance ovšem za výrazného zvýšení časových a paměťových nároků. V následujících tabulkách budou uvedeny výsledky těchto experimentů spolu s dodatečnými vizualizacemi.

Vizualizace instance	
Datová struktura DNA řešení	<b>0 0 &lt;&lt; 0 1 2 3 4 5 6 &gt;&gt; 0 0 0 0 0</b>
Vizualizace DNA řešení	

Tabulka č. 4: Výsledek simulátoru pro instanci hpp7\_13

Zápis instance	$(x \vee \sim y \vee z) \wedge (y \vee z) \wedge (\sim x \vee z) \wedge z$
Datová struktura DNA řešení	<b>0 0 0 0 0 0 &lt;&lt; 0 2 3 4 6 8 9 &gt;&gt; 0 0 0 0 0 0 0</b>
Vizualizace a zápis DNA řešení	 <p style="text-align: center;"><math>(x \sim y z)</math></p>

Tabulka č. 5: Výsledek simulátoru pro instanci sat3\_4

Vizualizace instance	
Datová struktura DNA řešení	<b>0 0 &lt;&lt; 0 <u>0</u> <u>1</u> <u>0</u> <u>2</u> <u>1</u> <u>3</u> <u>1</u> <u>4</u> <u>1</u> <u>5</u> <u>1</u> <u>6</u> &gt;&gt; 0 0 0 0 0 0 0 0 0 0</b>
Vizualizace DNA řešení	

Tabulka č. 6: Výsledek simulátoru pro instanci cli6. Podtržené hodnoty v datové struktuře jsou hodnotové sekce DNA řešení, nepodtržené hodnoty uvnitř řešení značí poziční sekce.

Z výsledků je zřejmé, že zvolené kódování oligonukleotidů a DNA řešení spolu se simulací DNA operací je funkční a pro dané instance problémů dává aplikace správné výsledky. U takto jednoduchých instancí počítá simulátor výsledky v řádu sekund. Nicméně cílem tohoto experimentu bylo ověřit pouze správnost kódování a funkčnost simulátoru. Časové nároky řešení různých instancí problémů budou zkoumány v jedné z příštích podkapitol.

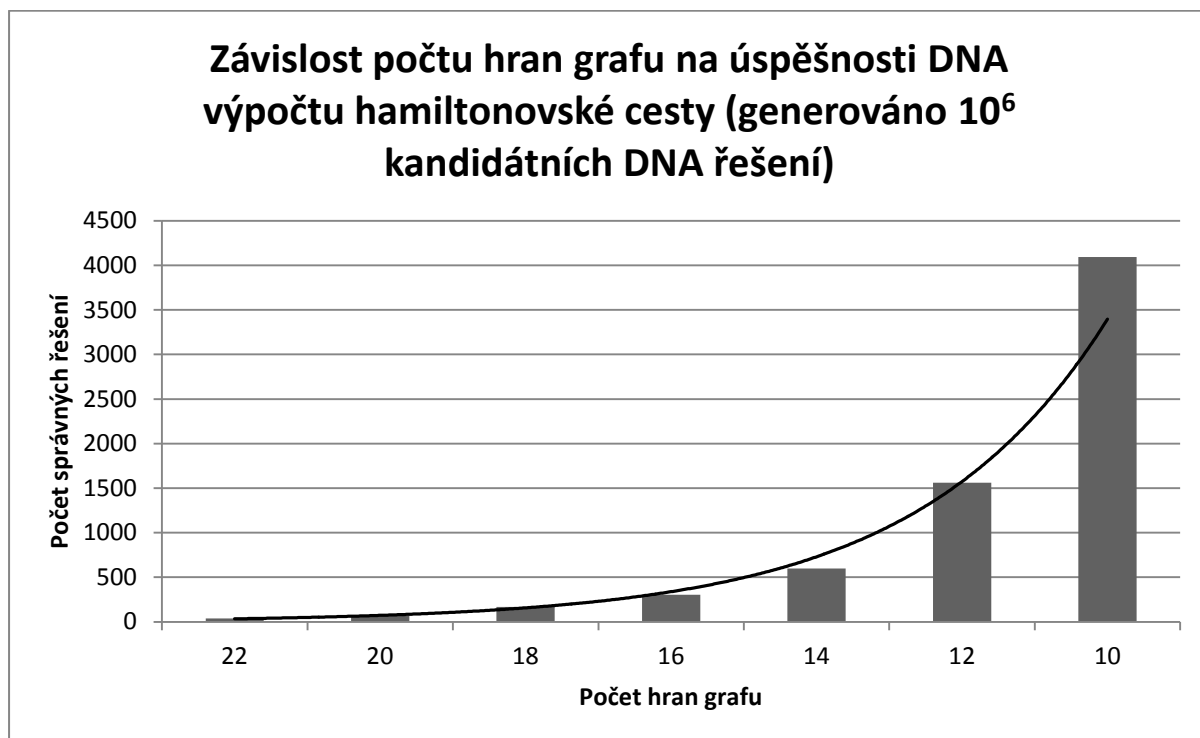
## 5.2 Složitost instancí problémů

Další sada experimentů bude zkoumat výkonnost navrženého simulátoru v závislosti na složitosti instancí problémů. Problémy, které se v této práci řeší pomocí DNA počítání, mají svá specifická řešení. U problému nalezení hamiltonovské cesty grafu a SAT problému stačí při jejich obecné definici nalézt pouze jedno DNA řešení, které problém řeší. Při nalezení takového řešení lze problém

okamžitě vyřešit a ukončit výpočet. U problému nalezení maximální kliky grafu se nejprve vygeneruje množina DNA kandidátních řešení reprezentujících kliky grafu. Následně se z této množiny nalezne maximální klika grafu. Zde nastává drobná komplikace. Na rozdíl od dvou předchozích problémů není jisté, že zvolená maximální klika je opravdu maximální klika grafu. V malé množině klik grafu se totiž ta opravdu maximální nemusí vyskytovat.

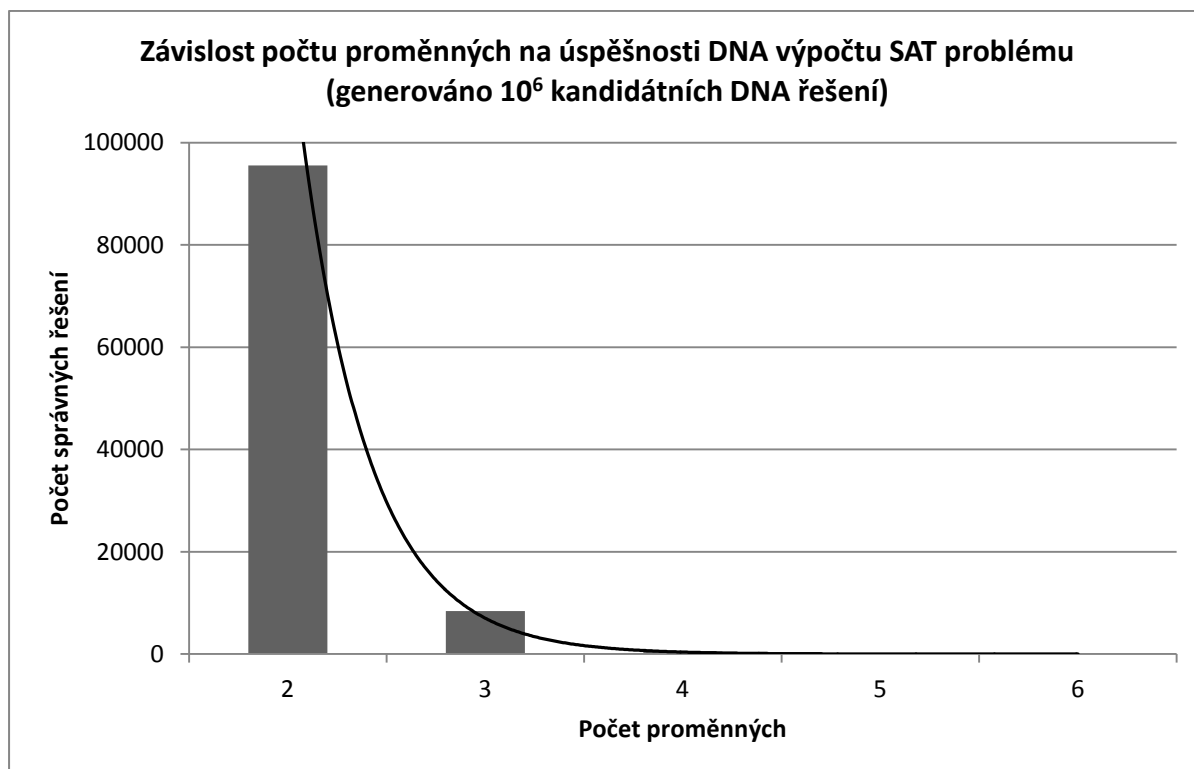
Proto byly obecné definice řešených problémů pro tuto sadu experimentů ignorovány a bylo zkoumáno, kolik správných DNA řešení se podařilo nalézt.

Při zkoumání závislosti složitosti orientovaného grafu na výkonnosti simulátoru při řešení problému nalezení hamiltonovské cesty se narazilo na další komplikace. Jednak bylo zjištěno, že srovnávat dva různé grafy není z pohledu výkonnosti simulátoru vůbec relevantní. Každý obsahuje totiž jiné hrany a tudíž jinou hamiltonovskou cestu. Při zkoumání závislosti počtu hran grafu na výkonnosti simulátoru se ukázalo, že velmi záleží na přesném umístění přidávané/odebírané hrany. Při odebrání hrany, která je součástí hamiltonovské cesty, logicky dochází k výraznému poklesu počtu řešení. Naopak při přidání hrany, jejíž pomocí vznikne nová hamiltonovská cesta, dochází k výraznému zvýšení počtu nalezených řešení. Při experimentu byl tudíž kladen důraz na to, aby se při měnícím počtu hran zachovával počet druhů hamiltonovských cest. Výsledky ukazuje *graf č. 1*. Z grafu je zřejmá exponenciální závislost.



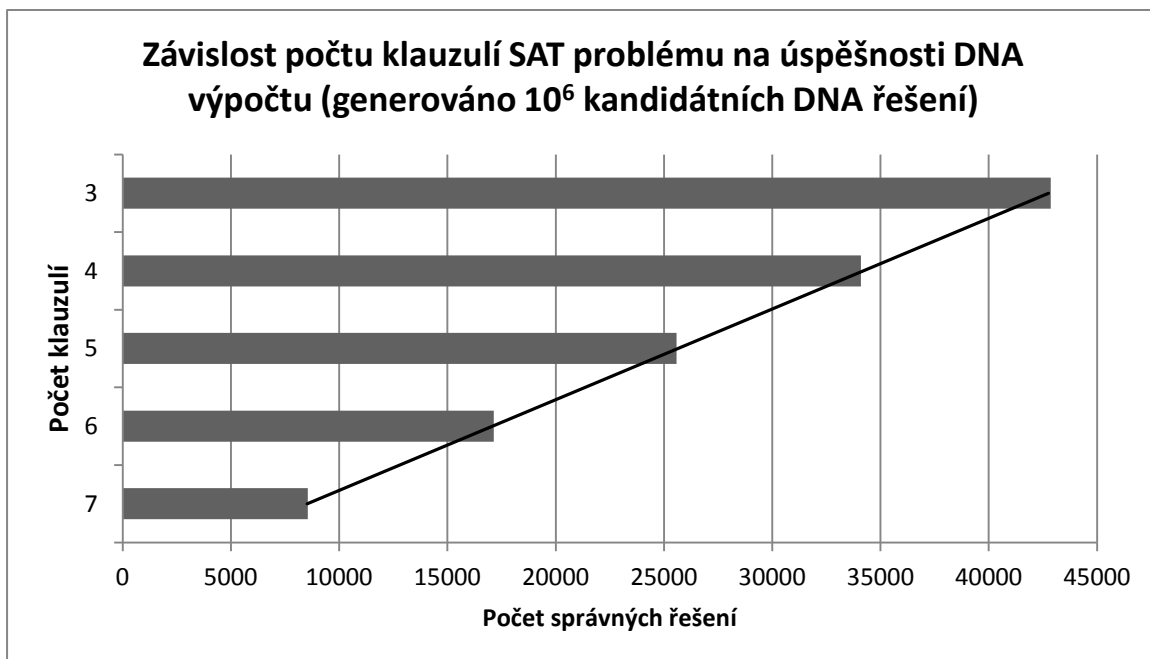
*Graf č. 1: Závislost počtu hran grafu na úspěšnosti DNA výpočtu hamiltonovské cesty*

Dále byla zkoumána závislost složitosti SAT problému na výkonnosti simulátoru. Platí zde stejné podmínky jako u problému nalezení hamiltonovské cesty. I u SAT problému může existovat více typů ohodnocení proměnných, pro které je výroková formule splnitelná. U prvního experimentu zabývajícím se změnou počtu proměnných (*graf č. 2*) byla proto vždy upravena množina klauzulí tak, aby existoval pouze jeden typ správného ohodnocení proměnných. Z grafu je opět patrná exponenciální závislost.



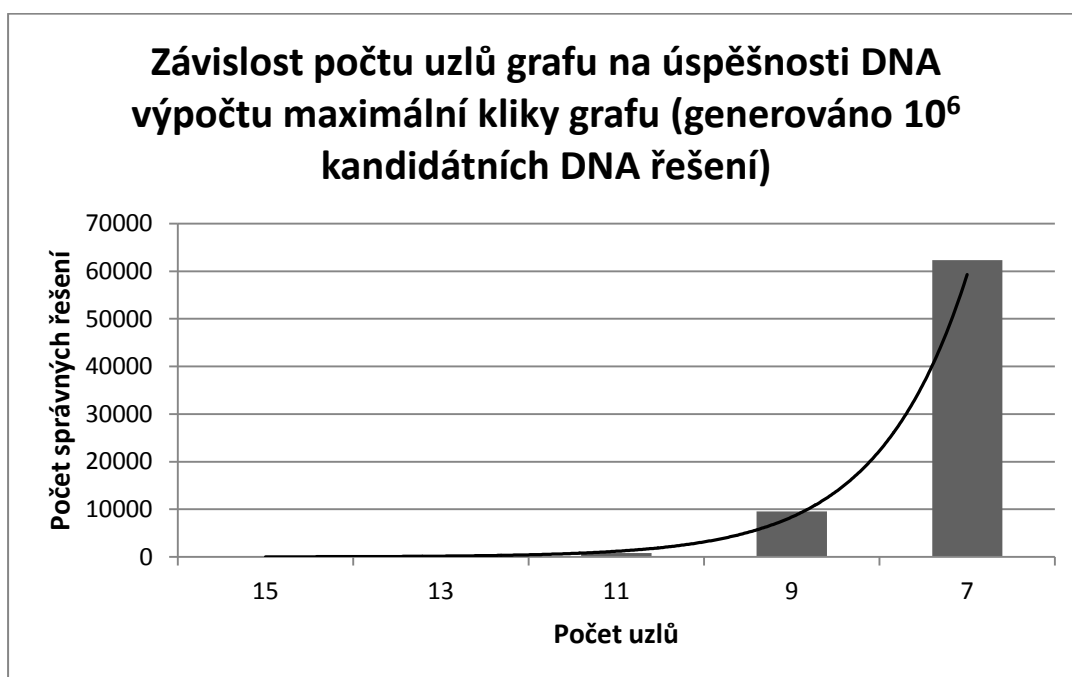
*Graf č. 2: Závislost počtu proměnných na úspěšnosti DNA výpočtu SAT problému*

Dále byl zkoumán vliv počtu klauzulí na výkonnost simulátoru (*graf č. 3*). Při zachování počtu proměnných a změně počtu klauzulí je vidět, že počet klauzulí nemá tak zásadní vliv na výkonnost simulátoru jako počet proměnných. U změny počtu proměnných se totiž mění celý graf, ve kterém se hledá správná cesta. Naopak při změně počtu klauzulí zůstává prohledávaný graf pořád stejný, změní se pouze počet odfiltrovaných kandidátních DNA řešení.

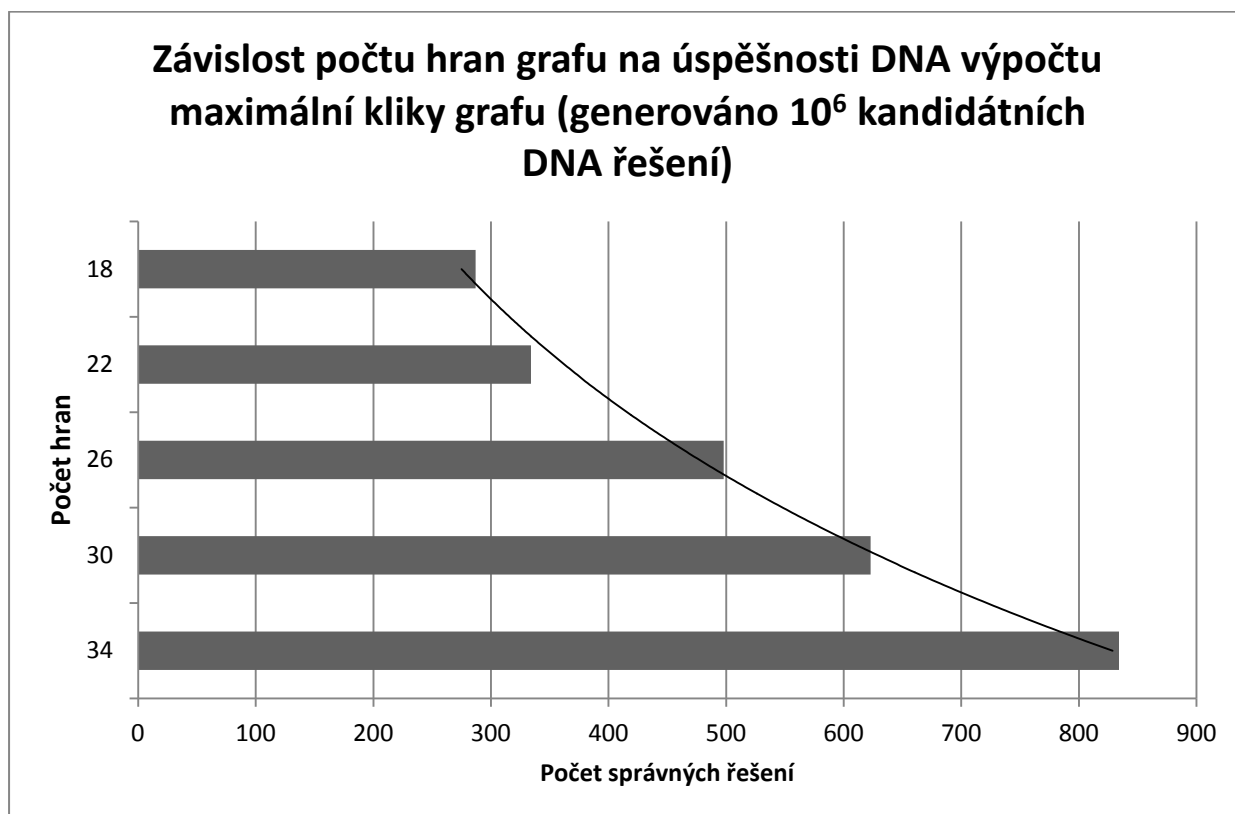


Graf č. 3: Závislost počtu klauzulí na úspěšnosti DNA výpočtu SAT problému

U problému hledání maximální kliky grafu je výkonnost simulátoru klíčová. Jak již bylo zmíněno na začátku této podkapitoly, na rozdíl od dvou předchozích problémů není jisté, že zvolená maximální klika je opravdu maximální klika grafu. V malé množině klik grafu se totiž ta opravdu maximální nemusí vyskytovat. Aby simulátor našel skutečnou maximální kliku grafu, je nutné, aby měl velkou výkonnost a nageneroval co největší množinu klik grafu. Výsledky experimentů znázorňují graf č. 4 a graf č. 5.



Graf č. 4: Závislost počtu uzlů grafu na úspěšnosti DNA výpočtu maximální kliky grafu



*Graf č. 5: Závislost počtu hran grafu na úspěšnosti DNA výpočtu maximální kliky grafu*

Z grafů vyplývá, že změna počtu uzlů grafu při hledání maximální kliky mění výkonnost simulátoru exponenciálně. Logicky totiž dochází ke zvětšení množiny stavebních oligonukleotidů, ze kterých se generují kandidátní DNA řešení. Zatímco zvýšení počtu hran v grafu ovlivňuje pouze filtrační fázi DNA výpočtu, což se na výkonnosti simulátoru neprojeví tak zásadně jako zvýšení počtu uzlů grafu.

### 5.3 Vliv parametrů simulátoru

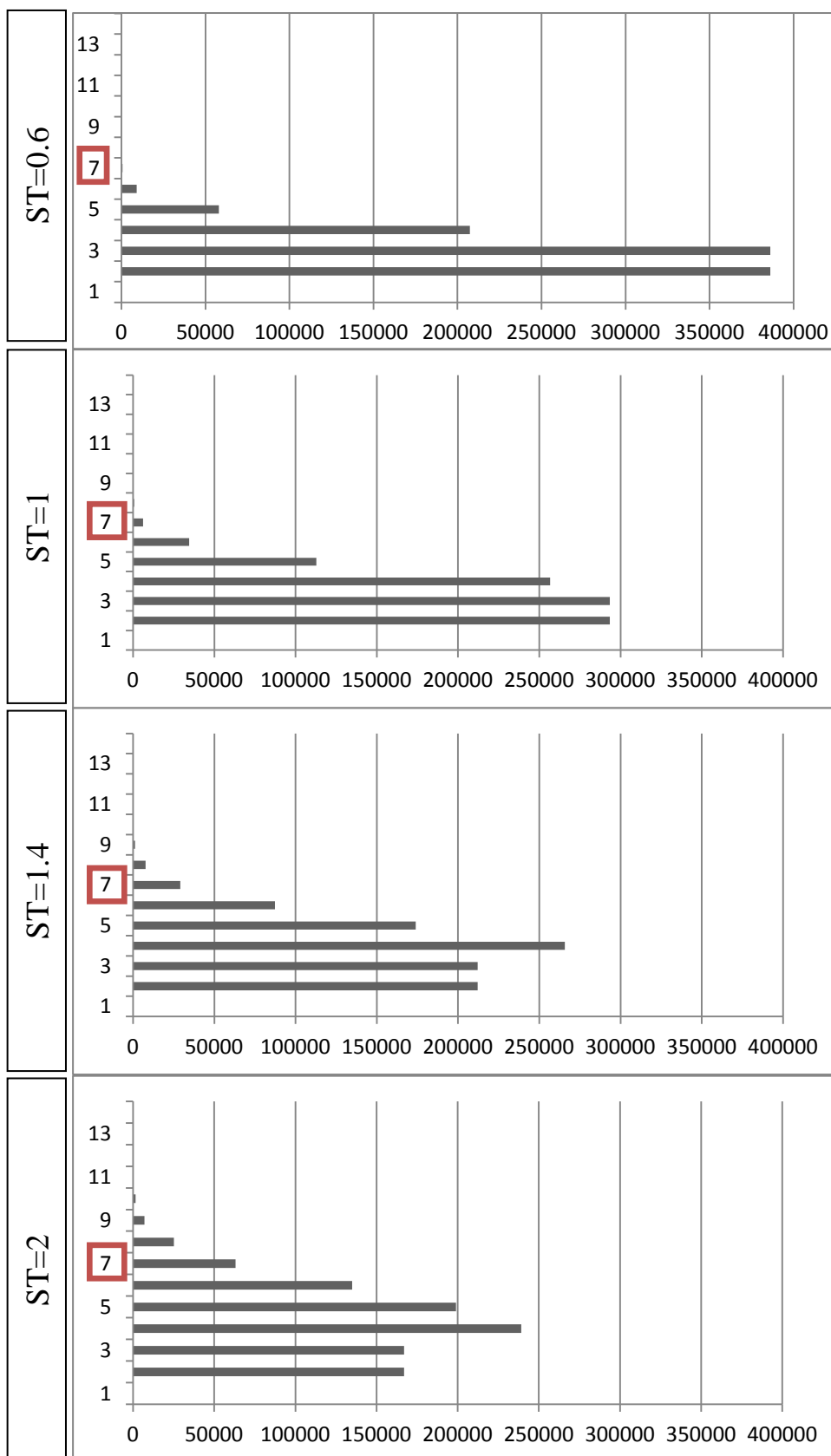
Pro předchozí experimenty bylo pro správné vyhodnocení závislostí důležité konstantní nastavení parametrů simulátoru. Tato podkapitola se bude věnovat výhradně vlivu parametrů simulace na její průběh. Hlavní parametry simulace jsou tři a každý výrazně ovlivňuje průběh simulace DNA výpočtu (*podkapitola 4.2*). Vliv prvního parametru IT, který udává celkový počet vygenerovaných kandidátních DNA řešení, je zcela zřejmý a proto nemá cenu jej samostatně zkoumat. Čím více se nageeneruje kandidátních DNA řešení, tím větší šance, že se v této množině bude vyskytovat správné řešení problému.

Druhým parametrem je ST. Simuluje dobu, po kterou se nechají oligonukleotidy reagovat, a intenzitu jejich promíchání. Při malém nastavení tohoto parametru vznikají kratší kandidátní DNA řešení a naopak. Experiment byl proveden na instanci *hpp7\_13* (jedná se o graf, který použil Adleman ve svém experimentu) a bylo zkoumáno měnící se spektrum délek generovaných kandidátních DNA řešení. Již z názvu instance je zřejmé, že graf obsahuje 7 uzlů, tudíž délka správného DNA řešení bude 7 oligonukleotidů. Je důležité zmínit, že pro tento experiment bylo nastaveno neomezené množství stavebních oligonukleotidů. Vliv nastavení množství stavebních oligonukleotidů bude zkoumán v dalším experimentu.

ST	DÉLKA KANDIDÁTNÍCH DNA ŘEŠENÍ [oligonukleotidů]						
	1	2	3	4	5	6	7
0,6	0	386102	386103	207366	57938	9106	568
0,8	0	334282	334283	237961	85408	20282	2366
1	0	293474	293475	256741	112801	34505	6185
1,2	0	234457	234458	268670	156604	69449	19644
1,4	0	212004	212005	265772	173898	87355	29021
1,6	0	179902	179903	249990	193211	120505	51656
1,8	0	166976	166977	239193	198936	135020	63054
2	0	156154	156155	226676	201342	146219	75934

ST	DÉLKA KANDIDÁTNÍCH DNA ŘEŠENÍ [oligonukleotidů]						
	8	9	10	11	12	13	14
0,6	0	0	0	0	0	0	0
0,8	128	0	0	0	0	0	0
1	655	20	0	0	0	0	0
1,2	4068	521	46	1	0	0	0
1,4	7728	1245	124	7	0	0	0
1,6	18148	4319	823	75	8	0	0
1,8	25149	7067	1499	220	13	4	0
2	33225	10352	2622	429	66	7	1

Tabulka č. 7: Histogramy délek generovaných kandidátních DNA řešení pro instanci *hpp7\_13* v závislosti na parametru ST.



Graf č. 6: Histogramy délek generovaných kandidátních DNA řešení pro instanci hpp7\_13. Červeně jsou označeny kandidátní řešení, mezi kterými se vyskytuje správné řešení instance problému.

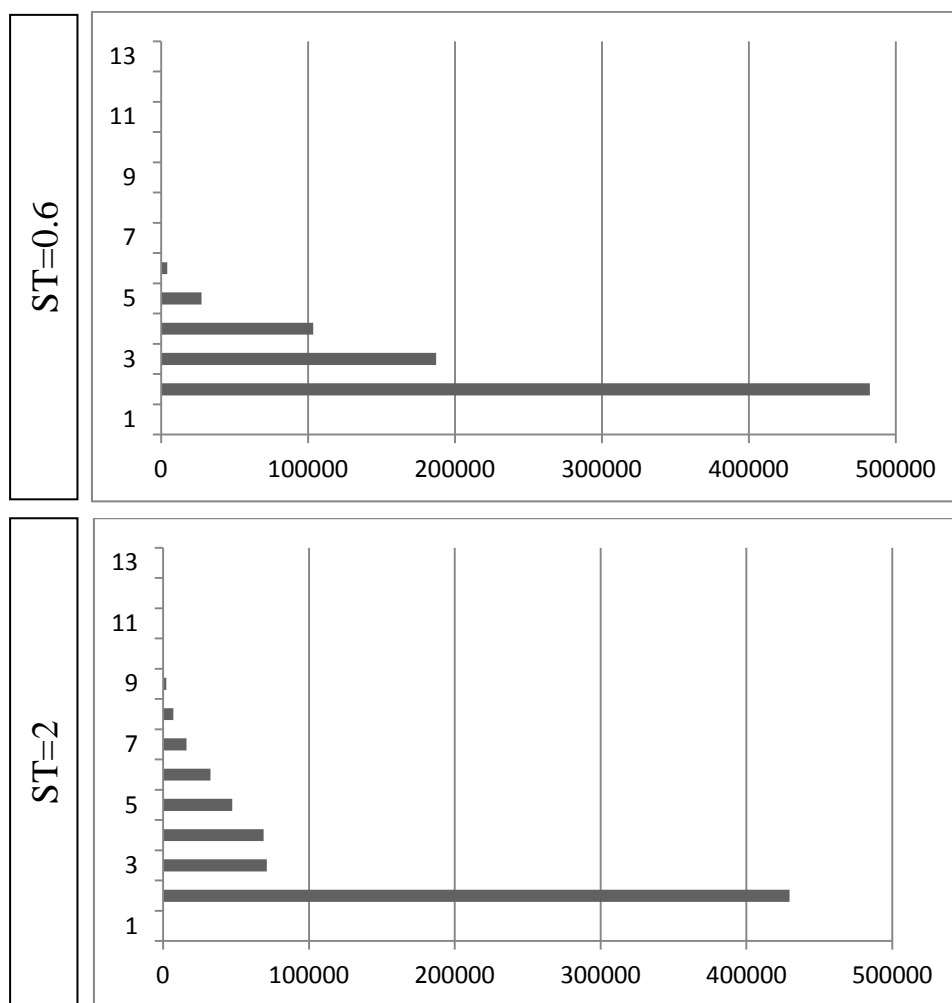
Vliv parametru ST na průměrnou délku generovaných kandidátních DNA řešení se potvrdil. V histogramu je velmi výrazným prvkem stálý nulový počet kandidátních řešení o délce 1 oligonukleotidu (jednoho uzlu grafu). Toto je způsobeno způsobem generování kandidátních řešení, kdy simulátor napojuje oligonukleotidy reprezentující hrany grafu. Výpočet tedy začíná s jednou hranou, tedy dvěma uzly.

Při nastavení parametru ST na velmi nízkou hodnotu se pro instanci *hpp7\_13* generovala většina kandidátních DNA řešení kratších než 7 oligonukleotidů. Přitom hamiltonovská cesta musí obsahovat 7 uzlů. Při tomto nastavení parametru se tedy může stát, že simulátor existující hamiltonovskou cestu nenajde a DNA výpočet selže. Naopak velmi vysoká hodnota parametru ST může způsobit generování příliš dlouhých kandidátních DNA řešení, mezi kterými se hamiltonovská cesta také nemusí vyskytovat a výpočet bude trvat velmi dlouho. Ukázalo se, že ideální nastavení tohoto parametru se pro každou instanci problému liší.

Až doposud se při všech experimentech předpokládalo, že počet stavebních oligonukleotidů, z nichž se tvoří kandidátní DNA řešení, je neomezený. Ve skutečnosti tomu však tak není. I ve světě DNA molekul a operací existuje určité omezení množství DNA. Proto je tento fakt promítnut do dalšího parametru simulace AM, který udává počet stavebních oligonukleotidů každého typu.

ST	AM	Počet KŘ	Histogram délek KŘ
0.6	1000	8053	[0, 481452, 188671, 103238, 27113, 4273, 279, 0, 0, 0, 0, 0, 0, 0]
0.6	10000	80517	[0, 48249, 18877, 10370, 2624, 436, 35, 0, 0, 0, 0, 0, 0, 0]
0.6	100000	805150	[0, 482494, 187179, 103472, 27543, 4213, 249, 0, 0, 0, 0, 0, 0, 0]
1.0	1000	7523	[0, 4439, 1425, 1046, 448, 118, 27, 4, 0, 0, 0, 0, 0, 0]
1.0	10000	75183	[0, 44482, 14281, 10627, 4181, 1346, 242, 24, 0, 0, 0, 0, 0, 0]
1.0	100000	752750	[0, 445518, 143456, 105500, 42999, 12814, 2250, 201, 12, 0, 0, 0, 0, 0]
1.5	1000	7031	[0, 4291, 969, 881, 507, 243, 106, 26, 7, 0, 1, 0, 0, 0]
1.5	10000	69876	[0, 42919, 9004, 8549, 5172, 2820, 962, 315, 71, 8, 2, 0, 0, 0]
1.5	100000	69864	[0, 429000, 90740, 84656, 51335, 28100, 10444, 3011, 632, 86, 14, 0, 0, 0]
2.0	1000	6734	[0, 4229, 753, 691, 491, 334, 154, 64, 22, 1, 0, 0, 0, 0]
2.0	10000	67533	[0, 43079, 7009, 6745, 4723, 3365, 1657, 699, 195, 57, 2, 2, 0, 0]
2.0	100000	675432	[0, 429590, 71196, 68816, 47345, 32594, 16110, 6994, 2207, 540, 92, 15, 0]

Tabulka č. 8: Histogramy délek generovaných kandidátních DNA řešení (KŘ) v závislosti na parametrech ST a AM.



Graf č. 7: Histogramy délek generovaných kandidátních DNA řešení

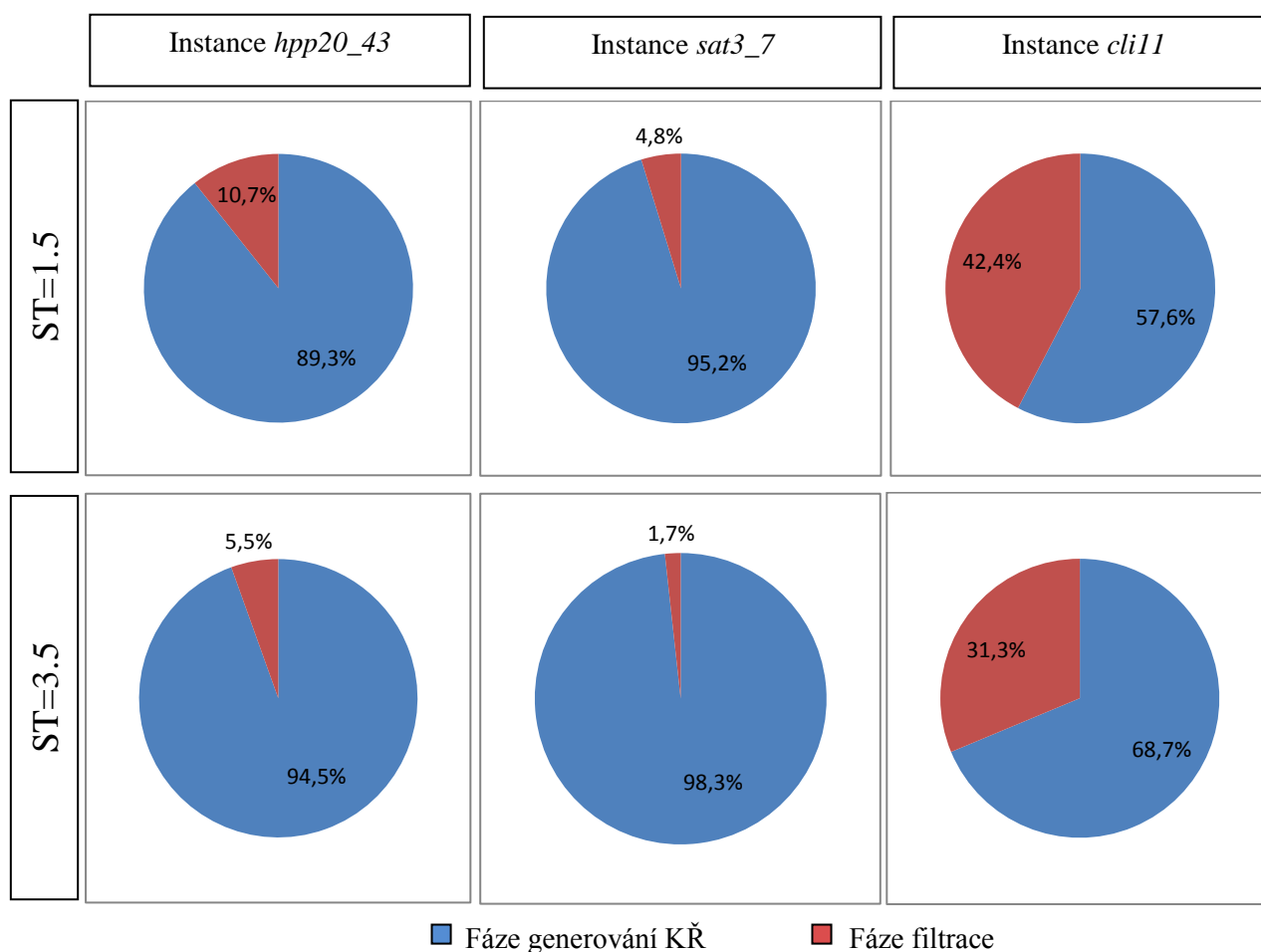
Z tabulky č. X a grafu č. X je patrné, že množství stavebních oligonukleotidů spolu s parametrem ST ovlivňuje nejenom rozložení histogramu délek generovaných kandidátních DNA řešení, ale i vlastní počet vygenerovaných kandidátních řešení. Stejně jako v případě neomezeného počtu stavebních oligonukleotidů zvyšuje parametr ST průměrnou délku kandidátních DNA řešení.

V tomto případě však dochází k výraznému navýšení počtu kandidátních řešení, které mají délku jednoho hranového oligonukleotidu (resp. dvou uzlových oligonukleotidů) oproti zbytku. Je to pravděpodobně způsobeno tím, že některé oligonukleotidy reprezentující významné prvky grafu se po nějaké době vypotřebují na stavbu delších DNA řešení. Z neúplné sady stavebních oligonukleotidů už dále simulátor není schopen sestavit další užitečná DNA řešení.

## 5.4 Časová náročnost řešení problémů

V předchozích experimentech byly zkoumány závislosti, které nebyly přímo zaměřeny na časovou náročnost simulátoru. Při znázornění předchozích závislostí byla důležitá zejména přehlednost výsledků, proto byly dosud experimenty prováděny se vzorovými instancemi problémů (např. graf, který použil Adleman ve svém experimentu), jejichž vyřešení nebyl pro simulátor žádný problém. Tato podkapitola se ale zaměří pouze na řešení různých instancí problému z pohledu časové složitosti. Budou zde použity mnohem složitější instance řešených problémů a simulátor bude tlačěn na hranu své výkonnosti. Také zde bude uvedeno porovnání neoptimalizované a optimalizované verze simulátoru.

U neoptimalizované verze simulátoru bylo nejprve nutné zjistit, jaký podíl na celkovém času výpočtu má fáze generování kandidátních řešení a fáze filtrace. Tento poznatek by mohl být velmi užitečný při optimalizace simulátoru. Měření bylo provedeno na vzorku  $10^6$  kandidátních řešení v závislosti na parametru ST. Výsledky měření jsou uvedeny v grafu č. 8.



Graf č. 8: Časová náročnost fází DNA výpočtu

Z výsledků experimentu vyplývá, že fáze generování kandidátních řešení je časově mnohem náročnější než fáze filtrace. Navíc se ukázalo, že časová náročnost fáze generování KŘ silně závisí na parametru ST. U fáze filtrace tomu tak není a její časová náročnost závisí pouze na druhu řešeného problému. Nejméně zabere filtrace kandidátních řešení u řešení SAT problému, naopak nejvíce u hledání maximální kliky grafu, kde je součástí filtrace i tvorba komplementárního grafu. Nicméně do optimalizované verze simulátoru byl zaveden paralelní výpočet do obou fází. Tím se simulátor alespoň částečně přiblížil skutečnému masivnímu paralelismu DNA výpočtu.

Experimenty s řešením instancí problémů pomocí principů DNA počítání vzhledem k časovým nárokům se provádí velmi obtížně. Hlavním faktorem DNA výpočtu je prvek náhody. Stavební oligonukleotidy se sice skládají do kandidátního DNA řešení podle pravidla komplementarity, ale náhoda zde hraje významnou roli. Proto se klidně může stát, že simulátor v jednom běhu nalezne řešení za krátkou dobu, ve druhém běhu ale bude hledat řešení problému velmi dlouho nebo ho dokonce za daný časový limit ani nenalezne.

Druhým prvkem, který zásadně ovlivňuje dobu hledání řešení u problému nalezení hamiltonovské cesty je struktura prohledávaného grafu. Zde nejvíce záleží na tom, kolik cyklů orientovaný graf obsahuje. Při cyklení grafem totiž cesta porušuje podmínky hamiltonovské cesty a DNA výpočet generuje mnoho neplatných kandidátních řešení. Navíc se při omezeném množství oligonukleotidů může snadno stát, že se při tvorbě neplatných kandidátních řešení vypotřebuje některý významný oligonukleotid, který reprezentuje některou hranu hamiltonovské cesty.

Aby se ukázal hlavní přínos optimalizace simulátoru z hlediska časových nároků na výpočet, byl měřen čas generování  $10^6$  kandidátních řešení při stálém nastavení parametru ST na hodnotu 3.

Experiment byl proveden na školním serveru Merlin, který má následující specifikace:

- SuperMicro 2021M-T2R+B, MB H8DME-2, 2x AMD Opteron 2387 (2.8GHz, 4core), 16 GB RAM.

	<i>sekvenční</i>	<i>2 vlákna</i>	<i>3 vlákna</i>	<i>4 vlákna</i>	<i>5 vláken</i>	<i>6 vláken</i>
<b>hpp7_13</b>	1,242	0,808	0,711	0,609	0,566	0,505
<b>hpp10_24</b>	1,725	1,118	0,923	0,765	0,697	0,632
<b>hpp15_30</b>	2,543	1,323	1,025	0,887	0,781	0,705
<b>hpp20_43</b>	3,103	1,664	1,239	1,101	0,957	0,858
<b>hpp40_88</b>	5,762	2,929	2,132	1,886	1,628	1,513

*Tabulka č. 9: Srovnání doby výpočtu neoptimalizované a optimalizované aplikace na řešení různých instancí problému hledání hamiltonovské cesty (čas v sekundách)*

	sekvenční	2 vlákna	3 vlákna	4 vlákna	5 vláken	6 vláken
sat3_4	1,865	1,187	1,02	0,903	0,842	0,793
sat10_11	4,542	2,256	1,674	1,478	1,298	1,187
sat15_15	6,648	3,329	2,457	2,157	1,921	1,756
sat20_15	8,534	4,108	3,102	2,459	2,203	2,021

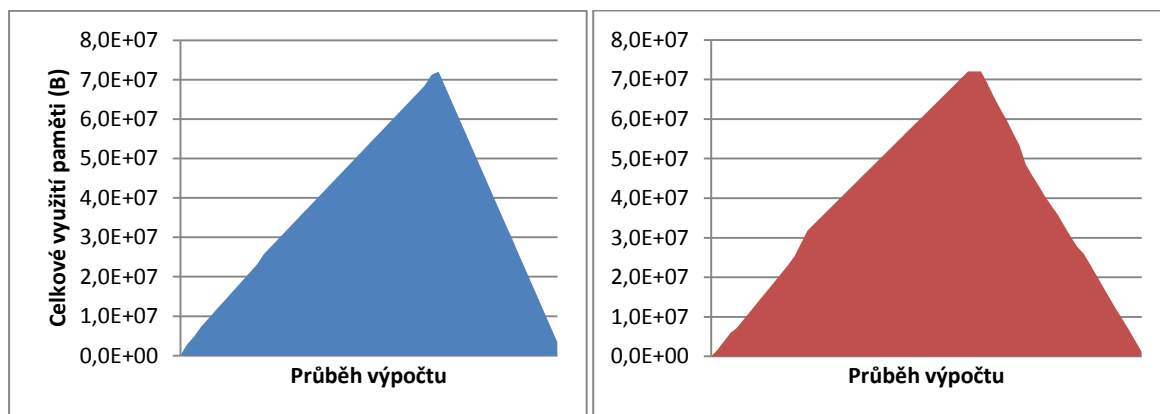
Tabulka č. 10: Srovnání doby výpočtu neoptimalizované a optimalizované aplikace na řešení různých instancí SAT problému (čas v sekundách)

	sekvenční	2 vlákna	3 vlákna	4 vlákna	5 vláken	6 vláken
cli6	2,458	1,862	1,625	1,397	1,286	1,127
cli11	5,327	3,726	3,111	2,983	2,532	2,012
cli15	5,689	3,875	2,798	2,324	2,076	1,854
cli20	8,547	4,982	3,967	3,303	2,974	2,615

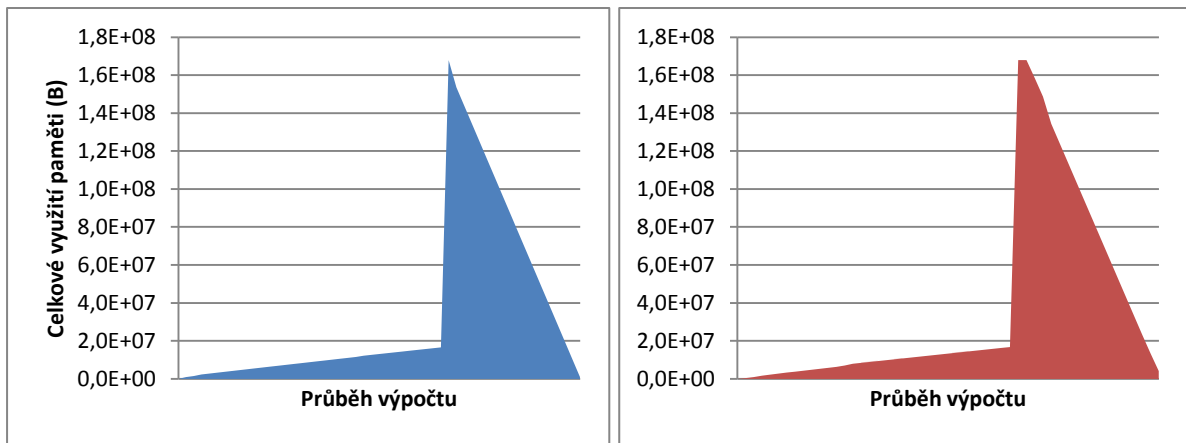
Tabulka č. 11: Srovnání doby výpočtu neoptimalizované a optimalizované aplikace na řešení různých instancí problému hledání maximální kliky grafu (čas v sekundách)

## 5.5 Využití paměti v průběhu výpočtu

Po změření časových nároků řešení problémů je vhodné provést experimenty také z hlediska paměťových nároků. K měření využití paměti v průběhu výpočtu byla použita profilovací funkce paměťového nástroje *valgrind*. Měření bylo prováděno při generování  $10^6$  kandidátních DNA řešení na dvou různých instancích problému hledání hamiltonovské cesty. Výsledky jsou znázorněny v následujících grafech.



Graf č. 9: Celkové využití paměti v průběhu řešení instance *hpp7\_13* pomocí DNA výpočtu. Neoptimalizovaná verze simulátoru (vlevo) a optimalizovaná verze (vpravo).



Graf č. 10: Celkové využití paměti v průběhu řešení instance *hpp20\_43* pomocí DNA výpočtu. Neoptimalizovaná verze simulátoru (vlevo) a optimalizovaná verze (vpravo).

## 6 Závěr

V této práci byla navržena a implementována aplikace pro simulaci DNA výpočtů pro řešení vybraných problémů. Konkrétně se práce zaměřila na problém nalezení hamiltonovské cesty v grafu, řešení SAT problému a hledání maximální kliky grafu. K vypracování práce vedlo hned několik důvodů. Existuje jen velmi málo publikací, které se DNA počítáním zabývají z praktického hlediska v kombinaci s dnešními konvenčními výpočetními systémy. Do DNA počítání se vkládá velký příslib pro budoucí využití, naneštěstí se však musí vyřešit ještě mnoho překážek. Dalším důvodem ke vzniku této práce bylo zoptimalizování simulátoru tak, aby se ověřil přínos paralelních výpočtů (s ohledem na možnosti zvolené platformy), který vyplývá z principu DNA počítání. Výsledky získané řešením různých problémů pomocí simulátoru ukázaly, že principy DNA výpočtu jsou navrženy a implementovány správně a že se pomocí nich skutečně dají tyto problémy řešit. Na druhé straně experimenty ukázaly, že na dnešních výpočetních architekturách se dá DNA počítání u složitějších instancí problémů z důvodů velké časové náročnosti pouze simulovat.

V následující podkapitole bude uvedeno zhodnocení dosažených výsledků. V další podkapitole budou uvedeny přínosy této práce a poslední podkapitola obsahuje návrhy na další rozšíření této práce.

### 6.1 Zhodnocení výsledků

Při návrhu simulátoru byl kladen důraz především na přehledné zakódování řešených problémů a na zachycení důležitých prvků DNA výpočtu v podobě parametrů simulátoru. Také musel být při návrhu brán zřetel na budoucí optimalizaci v podobě zavedení paralelních výpočtů.

Po provedení několika experimentů na vzorových instancích problémů se ukázalo, že bylo zvolené zakódování problémů s ohledem na principy DNA počítání správné a že se pomocí nich dá opravdu vyřešit dané problémy.

Výsledky zkoumání vlivu složitosti instance řešeného problému na úspěšnosti simulátoru dokazují, že řešení složitých instancí problémů pomocí DNA počítání na dnešní architektuře výpočetního systému není možné. Ukázalo se, že počet prvků instancí, které ovlivňují fázi generování kandidátních DNA řešení (počet uzlů grafu, počet proměnných), závisí na úspěšnosti výpočtu exponenciálně. Naopak prvky, které mají význam pouze u fáze filtrace (počet klauzulí), ovlivňují úspěšnost simulátoru lineárně.

Parametry simulátoru mají také z hlediska DNA počítání velký význam. Výsledky prokázaly důležitost dostatečného počtu stavebních oligonukleotidů pro úspěšné nalezení hledaného DNA řešení. Po vypotřebování některého stavebního oligonukleotidu, který je nutnou součástí správného

řešení, na stavbu neplatných kandidátních DNA řešení byl v grafu patrný velký nárůst krátkých řešení a výpočet pak probíhal zbytečně.

Při měření časových nároků řešení různých instancí problémů se ukázalo, že použití paralelních výpočtů (technologie OpenMP) je při simulaci DNA výpočtů velmi efektivní a bylo zvoleno velmi vhodně. Charakter skutečných DNA výpočtů k tomuto přístupu logicky vybízel. Paralelní výpočty byly použity jak ve fázi generování kandidátních řešení, tak ve fázi filtrace.

Samozřejmě se ukázalo, že řešení vybraných problémů pomocí principů DNA počítání na dnešních architekturách je velmi neefektivní v porovnání s heuristickými metodami navrženými přímo pro řešení daného problému (výkonnost simulátoru se např. nemůže rovnat moderním SAT solverům). Je to vcelku logický jev, protože DNA výpočet prohledává stavový prostor řešení na principu náhodnosti a spoléhá na obrovskou rychlost a masivní paralelismus DNA reakcí.

## 6.2 Přínosy práce

Je jasné, že praktické využití simulátoru pro určení řešení složitých problémů v krátkém čase nebude příliš efektivní. Výsledek této práce by měl spíše umožnit ověřit principy DNA počítání na řešení reálných problémů a zkoumat různé závislosti, které se v průběhu výpočtu mohou vyskytovat. Dále lze pomocí optimalizované aplikace ověřit přínos paralelních výpočtů (s ohledem na možnosti zvolené platformy), který vylívá z principů DNA počítání. Implementovaná aplikace může také sloužit jako základ pro další výzkum v této oblasti, která má velmi slibnou budoucnost.

## 6.3 Možnosti dalšího vývoje

Cílem práce bylo ověření schopnosti DNA počítání najít řešení pro dané problémy a změřit jeho výkonnost. Pro efektivní a přehlednou implementaci byla proto zvolena datová reprezentace DNA sekvence na vyšší úrovni a textový výpis výsledků. Pokud by byla do aplikace doimplementována reprezentace na úrovni jednotlivých nukleotidů a přehledná vizualizace výsledků, hodil by se simulátor pro pochopení principů DNA počítání na řešení daných problémů.

Další možností vývoje aplikace je doimplementování dalších problémů, které jsou vhodné k řešení pomocí DNA výpočtů. Je třeba zde zmínit, že s tímto rozšiřováním simulátoru bylo počítáno již při návrhu a lze jej tedy provést velmi snadno.

Další možností optimalizace DNA výpočtů by bylo portovat implementovaný simulátor na GPU, kde je možno spouštět stovky až tisíce vláken najednou. Druhou variantou další optimalizace je vytvořit pro simulátor specifický HW akcelerátor, pomocí kterého by bylo také možné dosáhnout výrazného zrychlení.

# Literatura

- [1] Alberts, B. a kol.: *Základy buněčné biologie*. Ústí nad Labem, Espero Publishing, 1997, ISBN 80-902-9060-4
- [2] Paun, G., Rozenberg, G., Salomaa, A.: *DNA Computing, new computing paradigms*. Springer, Verlag, 1998, ISBN 35-406-4196-3
- [3] Adleman, L. M.: *Molecular computation of solutions to combinatorial problems*. Science, 266, November 11, 1994, s. 1021-1024
- [4] Lipton, R. J.: *Using DNA to solve NP-complete problems*. Science, 268, April 1995, s. 542-545
- [5] Sekanina, L.: *Biologii inspirované počítače - 11. DNA počítače*. Brno, Presentace, FIT VUT v Brně, 2013
- [6] Watson, J. D., Crick, F. H. C.: *A structure for deoxyribose nucleic acid*. Nature, 171, 1953, s. 737-738
- [7] Ignatova, Z., Martínez-Pérez, I., Zimmermann, K.H.: *DNA Computing Models*, Springer, 2008, ISBN 978-0-387-73635-8
- [8] Qi, Ouyang, Peter, D. Kaplan, Shumao, Liu, Albert, Libchaber: *DNA Solution of the Maximal Clique Problem*. Science, 278, October 17, 1997, s. 446-448
- [9] Sanjeev, Baskiyar: *Simulating DNA computing*. Lecture-Notes in Computer Science, v. 2552, Springer, 2002, s. 411-419
- [10] Quinn, Michael J.: *Parallel programming in C with MPI nad OpenMP*. Boston, McGraw-Hill Science, 2004, ISBN 00-728-2256-2
- [11] Studnička, Vladimír: *Genetické algoritmy – multi-core CPU implementace*. Brno, Diplomová práce, FSI VUT v Brně, 2010

- [12] Sanger, F., Nicklen, S., Coulson, A. R.: *DNA sequencing with chain-terminating inhibitors*. USA, Proc Natl Acad Sci, 74(12), 1977, s. 5463–5467
- [13] Cook, Stephen A.: *The complexity of theorem-proving procedures*. USA, Shaker Heights, 1971, s. 151-158
- [14] Tomica, Petr: *Výpočty na bázi DNA jako nové výpočetní schéma*. Brno, Semestrální project, FIT VUT v Brně, 2002
- [15] Watada, Junzo: *DNA Computing and its Application*. Computational Intelligence: A Compendium, 115, Springer, 2008, s. 1065-1089
- [16] Xing, Wang, Qiang, Zhang: *DNA computing-based cryptography*. Bio-Inspired Computing, 2009, s. 1-3

# Seznam příloh

Příloha A. Obsah CD

Příloha B. Formáty souborů instancí

Příloha C. Uživatelský manuál

# Příloha A

## Obsah CD

- `bin/` - adresář se spustitelnými soubory aplikací.
  - `dna` - spustitelný soubor neoptimalizované aplikace.
  - `dnaopt` - spustitelný soubor optimalizované aplikace.
- `src/` - adresář se zdrojovými soubory aplikací.
  - `dna.c` - zdrojový soubor neoptimalizované aplikace.
  - `dnaopt.c` - zdrojový soubor optimalizované aplikace.
  - `Makefile` - skript pro překlad obou aplikací.
- `instances/` - adresář s instancemi řešených problémů.
- `doc/` - adresář se zdrojovým textem této práce.
- `doc.pdf` - text této práce v elektronické podobě.
- `README` - uživatelská dokumentace.

# Příloha B

## Formáty souborů instancí

### *Problém nalezení hamiltonovské cesty (soubor hppX\_Y)*

```
[počet uzlů]      [počet hran]
[počáteční uzel]  [koncový uzel]

[z uzlu]  [do uzlu]
[z uzlu]  [do uzlu]
[z uzlu]  [do uzlu]
.
.
.
```

} množina hran

### *SAT problém (soubor satX\_Y)*

```
[počet proměnných]  [počet klauzulí]

[proměnná č.1]  [proměnná č.2]  [proměnná č.3]  ...
[proměnná č.1]  [proměnná č.2]  [proměnná č.3]  ...
[proměnná č.1]  [proměnná č.2]  [proměnná č.3]  ...
.
.
.
```

} množina klauzulí

Hodnoty proměnné:

- -1 – negace proměnné
- 0 – proměnná se nevyskytuje
- 1 – proměnná se vyskytuje

*Problém nalezení maximální kliky grafu (soubor cliX)*

[počet uzlů]

[hrana 1->1]	[hrana 1->2]	[hrana 1->3]...
[hrana 2->1]	[hrana 2->2]	[hrana 2->3]...
[hrana 3->1]	[hrana 3->2]	[hrana 3->3]...
.		
.		
.		

} *incidenční matice*

# Příloha C

## Uživatelský manuál

### *Požadavky:*

K překladu optimalizované verze simulátoru je zapotřebí mít nainstalovanou technologii OpenMP.

### *Příklad:*

Příkazem `make` dojde automatickému přeložení obou verzí simulátoru.

### *Ukázkové instance problémů:*

Soubory s ukázkovými instancemi problémů se nachází ve složce `/instances`.

### *Parametry při spuštění:*

```
dna [problem] [soubor]
  -h    Problém hledání hamiltonovské cesty
  -s    SAT problém
  -c    Problém hledání maximální kliky grafu
```