

UNIVERZITA KOMENSKÉHO V BRATISLAVE  
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

PRAVDEPODOBNOSTNÉ POROVNÁVANIE  
FUNKČNÝCH SIETÍ MOZGU

Bakalárska práca

2017

Michal Puškel

UNIVERZITA KOMENSKÉHO V BRATISLAVE  
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

PRAVDEPODOBNOSTNÉ POROVNÁVANIE  
FUNKČNÝCH SIETÍ MOZGU

Bakalárska práca

Študijný program: Aplikovaná informatika  
Študijný odbor: 2511 Aplikovaná informatika  
Školiace pracovisko: Katedra aplikovej informatiky  
Školiteľ: Mgr. Andrej Jursa

Bratislava 2017

Michal Puškel



53156631

## ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Michal Puškel

**Študijný program:** aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)

**Študijný odbor:** aplikovaná informatika

**Typ záverečnej práce:** bakalárska

**Jazyk záverečnej práce:** slovenský

**Sekundárny jazyk:** anglický

**Názov:** Pravdepodobnostné porovnávanie funkčných sietí mozgu  
*Probability comparison of functional brain networks*

**Ciel:** Navrhnut' nástroj na pravdepodobnostné porovnanie grafov funkčných sietí mozgu na základe orbitových distribúcií grafletového pokrycia sietí. Práca má vychádzať z článku Biological Network Comparison Using Graphlet Degree Distribution, Nataša Pržuji, Computer Science Department, University of California, Irvine, CA 92697-3425, USA. Študent má vytvoriť nástroj na pokrytie grafu niekoľkými typmi grafletov z tejto publikácie a na základe orbitovej distribúcie porovnať podobnosť dvoch grafov.

**Literatúra:** Biological Network Comparison Using Graphlet Degree Distribution, Nataša Pržuji, Computer Science Department, University of California, Irvine, CA 92697-3425, USA

**Kľúčové slová:** grafy, siete, distribúcia stupňa vrcholov, distribúcia orbít grafletov

**Vedúci:** Mgr. Andrej Jursa

**Katedra:** FMFI.KAI - Katedra aplikovej informatiky

**Vedúci katedry:** prof. Ing. Igor Farkaš, Dr.

**Dátum zadania:** 29.09.2016

**Dátum schválenia:** 04.10.2016

doc. RNDr. Damas Gruska, PhD.

garant študijného programu

.....  
študent

.....  
vedúci práce

# Čestné prehlásenie

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....  
Michal Puškel

# **Podakovanie**

Chcem sa podakovať všetkým, ktorí mi pomáhali pri písaní tejto bakalárskej práce. Ďakujem najmä svojmu školiteľovi Mgr. Andrejovi Jursovi a ďalším; doc. RNDr. Márií Markošovej, PhD., RNDr. Tatiane Jajcayovej, PhD., RNDr. Petrovi Borovanskému, PhD., Mgr. Jánovi Klukovi, PhD., Mgr. Miroslavovi Wagnerovi, RNDr. Jaroslavovi Janáčkovi PhD., Mgr. Tomášovi Vinařovi, PhD., prof. Ing. Igorovi Farkašovi, Dr., celému zboru vrátničiek a vrátnikov FMFI UK, Bc. Mariánovi Vyslúžilovi a ďalším za cennú pomoc, rady, konzultácie a čas, ktorú mi venovali počas tvorenia tejto bakalárskej práce.

# Abstrakt

Jedným z cieľov tejto bakalárskej práce je zistenie, či sa atribút funkčnej siete mozgu „*mať Alzheimerovu chorobu*” prejavuje v štruktúre siete. Kedže porovnávať štruktúru sietí priamo pomocou izomorfizmu je ťažký problém, rozhodli sme sa využiť metódiku pravdepodobnostného porovnania navrhnutú Natašou Pržulji v práci Biological Network Comparison Using Graphlet Degree Distribution.

Ďalším cieľom tejto práce je vytvoriť sietový distribuovaný systém na výpočet grafletovej stupňovej distribúcie. Kvôli krátkosti času sa podarilo realizovať iba vytvorenie a odladenie tohto výpočtového softwaru, preto realizácia prvého ciela je v práci predostrená iba v teoretickej rovine a jej experimentálna realizácia zostáva predmetom ďalšieho výskumu.

Klúčové slová: grafy, siete, distribúcia stupňa vrcholov, distribúcia orbít grafletov

# Abstract

One of the goals of this bachelor thesis is to find out if an attribute of a functional brain network „*to suffer from the Alzheimer disease*” shows off in the structure of its graph. Because the comparison of the networks structure using isomorphism is a hard problem, we have decided to use probabilistic comparison proposed by Nataša Pržulj in her paper Biological Network Comparison Using Graphlet Degree Distribution.

Another goal of this thesis is to develop a network distributed system to compute graphlet degree distribution. Because of the shortage of time we have achieved only to develop this computational software, therefore the realisation of the first goal is introduced just in field of theory and its experimental realisation will be a subject of a further research.

Keywords: graphs, networks, degree distribution, graphlet degree distribution

# Obsah

Úvod	1
1 Ciele práce	7
2 Základy teórie grafov	8
3 Implementácia	19
4 Testovanie systému	28
5 Výsledky	45
6 Záver	46
Literatúra	47
A Prílohy	49

# Úvod

Téma grafov mi bola v programovaní vždy srdcu blízka. Toto vedel aj môj školiteľ. Teda, keď som dostal možnosť pracovať na tejto práci, po prečítaní článku Pržulj [1] som neváhal s radosťou začať pracovať na *Pravdepodobnostnom porovnávaní funkčných sietí mozgu.*

V tom čase som sa na kurze *1-AIN-430: Programovacie paradigmá* práve zoznamoval s programovacím jazykom *GO*, ako stvoreným na paraleлизáciu výpočtov a sietové programy, teda už vtedy som vedel, že chcem vytvoriť distribuovaný systém, ktorý mi problém spočíta, nakoľko sa jedná o výpočtovo náročnú úlohu.

Moju zvedavosť a túžbu po vedomostiach o grafoch som sa snažil uspokojiť v zimnom semestri na kurze *2-AIN-154/12: Komplexné siete* a v letnom na kurze *1-AIN-413/15: Grafy, grafové algoritmy a optimalizácia*. Okrem iného mi to pomohlo aj s pochopením niektorých dnes už pre mňa triviálnych pojmov z teórie grafov, ktoré sú však kľúčové pre pochopenie tejto práce. Aj preto tie podstatné z nich vysvetlím v kapitole (2) *Základy teórie grafov*.

## Porovnanie grafov distribúciou orbít grafletov

V tejto práci sa snažíme zistíť, či sa atribút funkčnej siete mozgu „*mať Alzheimerovu chorobu*“ prejavuje v štruktúre jej grafu.

Vstupnú sadu skúmaných funkčných sietí mozgu máme z práce McCarthy a kol. [2]. Sada obsahuje funkčné siete v štandardnom priestore 40 účastníkov výskumu. Participanti boli rozdelení do 3 skupín: mladí ľudia, starší ľudia s diagnostikovaným nábehom

na Alzheimerovu chorobu a starší ľudia bez takejto diagnózy. Každá funkčná siet je reprezentovaná jednoduchým neorientovaným grafom.

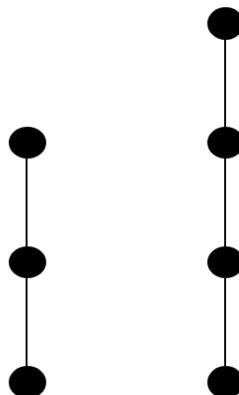
Ak sa počítačovým programom, určujúcim podobnosť dvoch grafov (na základe ich štruktúry) podarí roztriediť vstupnú sadu grafov na dve podmnožiny približne zhodné s množinou grafov predstavujúcich funkčné siete mozgu s diagnostikovanou Alzheimerovou chorobou respektíve nediagnostikovanou chorobou môže to znamenať, že táto choroba sa skutočne prejavuje v štruktúre grafov funkčných sietí mozgu.

Tieto dátá pochádzajú z pôvodného výskumu Bucknera a kol. [3], ktoré následne skúmal McCarthy a kol. [2]. Boli získané pomocou funkčnej magnetickej rezonancie (fMRI).

Sú to siete aktívnych častí mozgu, aktivovaných pri nejakej činnosti. Pri zvýšenej aktivite nejakej konkrétnej časti mozgu daná časť spotrebuje viac kyslíka, toto sa dá nasnímať pomocou fMRI skenera a z týchto obrázkov sa neskôr vytvorí 3D model mozgu a z neho jej graf.

Ako však odmerať štruktúru grafu? A čo to vlastne je štruktúra grafu?

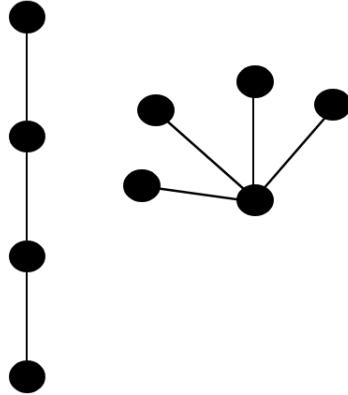
Štruktúra grafu je niečo, čo všetci intuitívne vnímame. Porovnajme napríklad dve cesty z obrázku 1:



Obr. 1: Dva rôzne grafy s podobnou štruktúrou (definícia 38).

Vidíme, že napriek rôznemu počtu vrcholov a hrán v grafoch ich štruktúra je si na vzájom podobná.

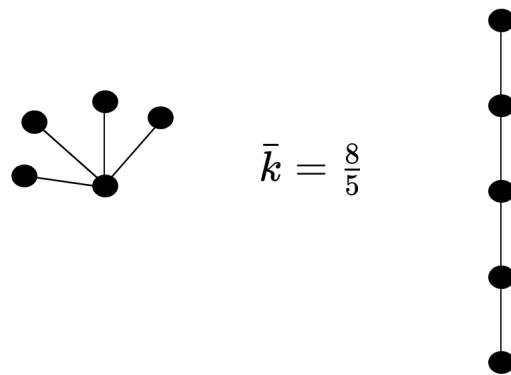
Na obrázku 2 pozorujeme značný rozdiel v štruktúre medzi cestou a hviezdou.



Obr. 2: Dva rôzne grafy s rozdielnou štruktúrou (definícia 38), (definícia 41).

Môžeme skúsiť skúmať stupeň vrchola, respektíve priemerný stupeň vrchola (definícia 14) daných grafov.

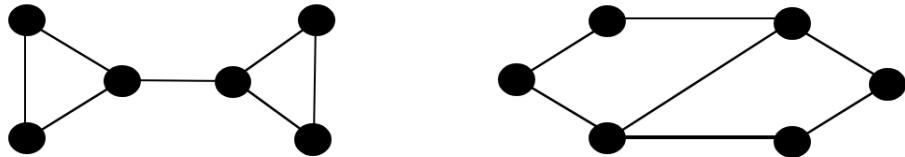
Avšak pri takomto naivnom prístupe nie je vôbec ľahké nájsť rýchlo kontrapríklad. Tak sme to urobili na obrázku 3 kde sa táto štatistika grafu zhoduje napriek rozdielnej štruktúre cesty a hviezdy.



Obr. 3: Porovnanie priemerného stupňa vrchola dvoch rôznych grafov s rozdielnou štruktúrou.

Vyskúšame teda distribúciu stupňov vrcholov (definícia 16) respektíve postupnosť stupňov vrcholov (definícia 15).

Pre grafy na obrázku 4 však táto štatistika opäť nesprávne určí podobnosť štruktúry grafu.

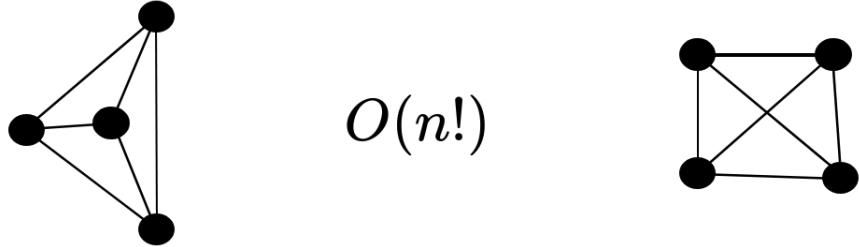


Obr. 4: Porovnanie postupnosti stupňov vrcholov dvoch rôznych grafov s rozdielnou štruktúrou.

$$\text{Postupnosť stupňov vrcholov} = (3, 3, 2, 2, 2)$$

Avšak je zrejmé, že grafy na obrázku 4 majú úplne rozdielnú štruktúru; jeden z nich napríklad obsahuje most (definícia 31) druhý zase Hamiltonovskú kružnicu (definícia 26).

Pri skúmaní štruktúry dvoch grafov je najhodnovernejším ukazateľom relácia ekvivalencie *izomorfizmus grafov* (definícia 43).



Obr. 5: Dva izomorfné grafy s úplne rovnakou štruktúrou.

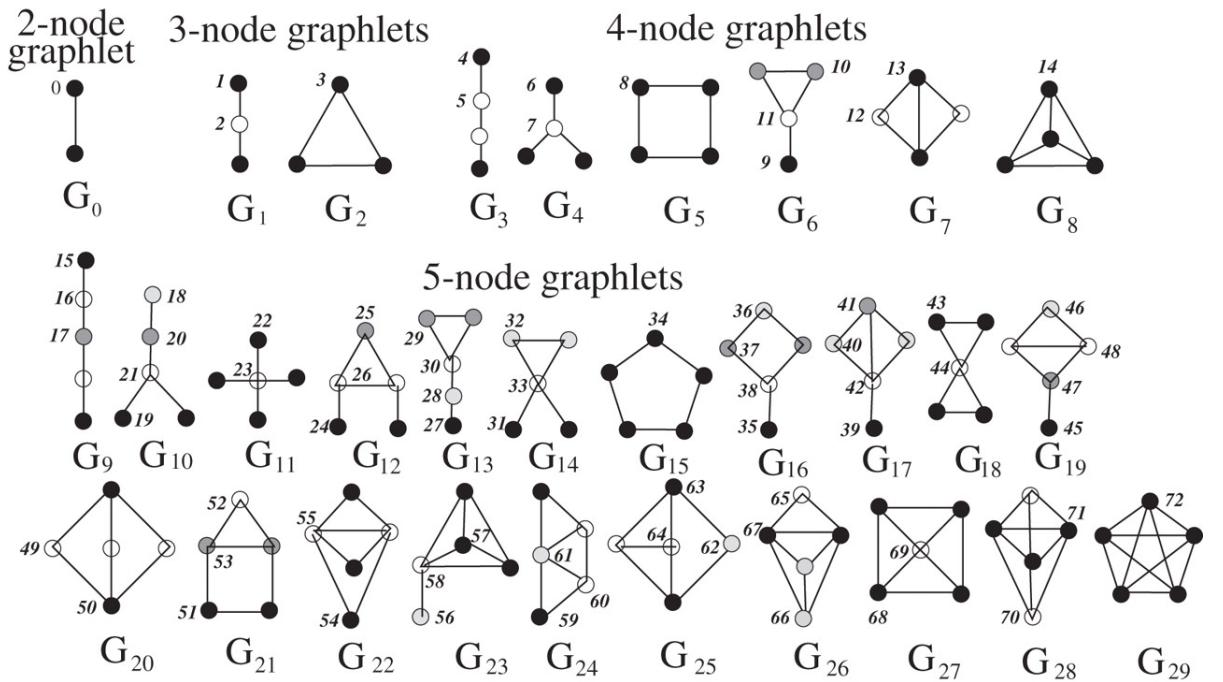
Aj keď to tak možno na prvý pohľad nevyzerá, grafy na obrázku 5 majú rovnakú štruktúru.

Riešenie takého problému má však neprijateľnú zložitosť. Najväčší graf skúmaný touto prácou obsahuje 8394 vrcholov a 450042 hrán (definícia 1).

Na zistenie izomorfizmu kladieme striktnú podmienku na rovnaký počet vrcholov a hrán. Skúmame všetkých  $n!$  možných bijekcií, kde  $n$  je počet vrcholov. Pre ilustráciu  $10! = 3628800$ .

Avšak v článku Pržulj [1] bola predstavená heuristika na určovanie podobnosti dvoch grafov skúmaním distribúcie orbít špeciálne dopredu vybratých grafletov znázornených na obrázku 6. Táto metóda je priamym zovšeobecnením distribúcie stupňov vrcholov.

Po spočítaní distribúcií jednotlivých orbít dvoch grafov získame matematickým aparátom reálne číslo z intervalu  $\langle 0, 1 \rangle$  určujúce pravdepodobnosť zhody štruktúry dvoch grafov, kde 0 znamená žiadnu zhodu a 1 veľmi vysokú pravdepodobnosť zhody.



Obr. 6: Automorfné orbity  $0, 1, 2, \dots, 72$  pre 30 2, 3, 4 a 5-vrcholových grafletov  $G_0, G_1, \dots, G_{29}$ ; v graflete  $G_i$ ,  $i \in \{0, 1, \dots, 29\}$ , vrcholy patriace rovnakému orbitu sú znázornené rovnakou farbou Pržulj [1].

# Kapitola 1

## Ciele práce

Cieľ tejto práce je zistiť, či sa atribút funkčnej siete mozgu „*mať Alzheimerovu chorobu*“ prejavuje v štruktúre jej grafu.

Vytvoriť distribuovaný systém na spočítanie distribúcie orbít grafletov rozličných grafov.

Namerané distribúcie orbít potom využijeme na overenie podobností sietí zo štatistického súboru 40 účastníkov. Budeme hľadať pravdepodobnostné rozdiely medzi sietami mozgu zdravých účastníkov a účastníkov s nabiehajúcou Alzheimerovou chorobou.

# Kapitola 2

## Základy teórie grafov

Terminológiu, základné pojmy a definície preberáme z viacerých učebníc: Gross a Yellen [4], Stanoyevitch [5], West [6], Grimaldi [7]; článkov: Pržulj [1] a Pržulj [8] a diplomovej práce Budinská [9]. V tejto kapitole sme pojmy mierne modifikovali, aby sme ich mohli následne unifikovať.

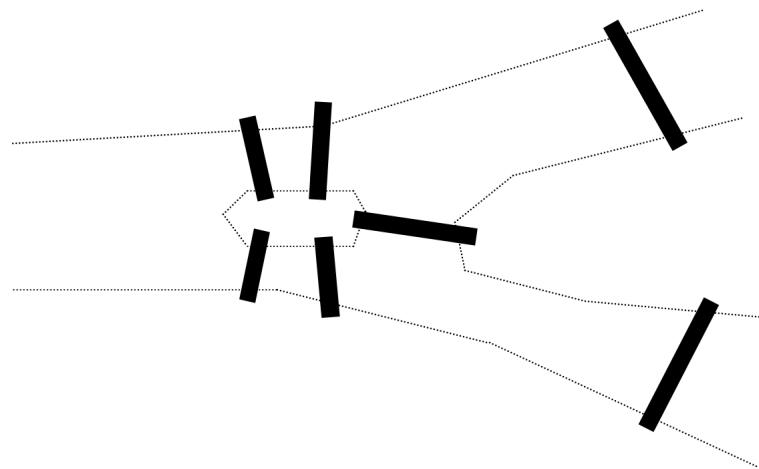
Teória grafov je jedným z mála odvetví matematiky, či informatiky, o ktorom vieme presne povedať, kedy a ako to všetko začalo.

Základy teórie grafov položil v roku 1736 Leonhard Euler. Jedným z ústredných problémov šľachty mesta Kaliningrad v 18. storočí totiž bolo: ako si spraviť peknú prechádzku po meste, ale tak aby sme každým zo 7 mostov prešli iba raz?

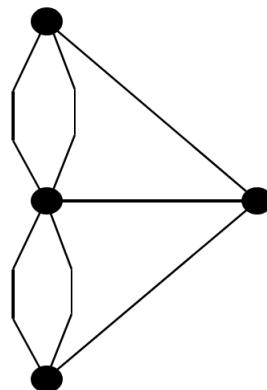
Leonhard Euler bol vedec a snažil sa teda nájsť nejaké systematické a odôvodnené riešenie problému. Začal tak, že si centrum mesta schematicky nakreslil, pokus o repliku jeho kresby vidíme na obrázku 2.1.

Uvedomil si, že pre riešenie jeho problému boli konkrétnie vzdialenosť, poloha mostov, či tvar rieky nepodstatné. Problém sa teda pokúsil zachytiať ešte viac schematicky, na obrázku 2.2 vidíme pokus o repliku schémy. Dôležité boli mosty a miesta oddelené riekou. Uvedomil si potrebu zavedenia pojmov zachytávajúcich vzťah (reláciu) medzi niektorými dvojicami z množiny miest oddelených riekou. Vzťahom bolo či miesta sú alebo nie sú spojené mostom.

Leonhard Euler si uvedomil potrebu zavedenia základných pojmov teórie grafov, ako vrchol, hrana či samotný graf.



Obr. 2.1: Schematický nákres 7 mostov mesta Kaliningrad.



Obr. 2.2: Replika prvého grafu v histórií teórie grafov; vrcholy grafu predstavujú miesta oddelené riekou; hrany grafu predstavujú mosty spájajúce brehy rieky.

**Definícia 1** (Vrchol, hrana, graf). *Graf G je matematický model, zachytávajúci vzťah (pevne určenú binárnu reláciu) prvkov množiny jeho vrcholov V. Je daný usporiadanej dvojicou množín V a E tak, že platí:*

$$G = (V, E), \quad (2.1)$$

$$V(G) = V, \quad (2.2)$$

$$E(G) = E. \quad (2.3)$$

*Existenciu relácie medzi dvoma prvkami množiny V zo vzťahu (2.1) nazývame **hrana**. Každá hrana grafu G patrí množine hrán grafu E.*

*V prípade, že je tento vzťah medzi vrcholmi jednosmerný, hovoríme o **orientovanej hrane**. Orientovanú hranu  $e_1 \in E$  teda reprezentujeme usporiadanej dvojicou vrcholov  $e_1 = (v_1, v_2)$ , pričom platí:*

$$E \subseteq V \times V, \quad (2.4)$$

*hovoríme, že hrana  $e_1$  vedie z vrcholu  $v_1$  do vrcholu  $v_2$ .*

*V prípade, že je vzťah obojsmerný, môžeme buď využiť už zadefinovanú reprezentáciu (2.4) a iba pridať ďalšiu hranu  $e_2 = (v_2, v_1)$  do množiny hrán E. Alebo zaviesť novú reprezentáciu **neorientovanej hrany**  $e_3 = \{v_1, v_2\}$  ako dvojprvkovej podmnožiny množiny V, tak, že platí:*

$$E \subseteq [V]^2. \quad (2.5)$$

*Avšak ani jedna z reprezentácií (2.4) a (2.5) nedokáže modelovať takzvané **násobné hrany** (nakolko E je jednoduchá množina). Sú to hrany incidentné s rovnakou dvojicou vrcholov.*

*Ak hrana začína a končí v tom istom vrchole, nazývame ju **slučka**.*

**Definícia 2** (Počet vrcholov). *Počet vrcholov grafu G je kardinalita (veľkosť) množiny vrcholov V. Podľa vzťahu:*

$$|V| = n. \quad (2.6)$$

Ak v texte neuvedieme ináč premenná n bude vždy označovať počet vrcholov grafu.

**Definícia 3** (Počet hrán). *Počet hrán grafu  $G$  je kardinalita (velkosť) množiny hrán  $E$ . Podľa vzťahu:*

$$|E| = m. \quad (2.7)$$

Ak v texte neuvedieme ináč premenná  $m$  bude vždy označovať počet hrán grafu.

**Definícia 4** (Orientovaný graf). *Orientovaný graf je graf, ktorého všetky hrany majú orientáciu.*

**Definícia 5** (Neorientovaný graf). *Neorientovaný graf je graf, ktorý obsahuje iba neorientované hrany.*

**Definícia 6** (Jednoduchý graf). *Jednoduchý graf je graf, ktorý neobsahuje slučky ani násobné hrany.*

Nakolko táto práca skúma jednoduché neorientované grafy, ďalšie definície uvádzame najmä pre ne.

**Definícia 7** (Multigraf). *Multigraf je graf, v ktorom sa môžu vyskytovať aj slučky alebo násobné hrany.*

**Definícia 8** (Null graf). *Null graf je graf, pre ktorý platí:*

$$V = \emptyset \wedge E = \emptyset. \quad (2.8)$$

**Definícia 9** (Triviálny graf). *Triviálny graf je graf, pre ktorý platí:*

$$|V| = 1 \wedge E = \emptyset. \quad (2.9)$$

**Definícia 10** (Incidencia vrchola a hrany). *Vrchol  $v$  je incidentný s hranou  $e$  ak existuje vrchol  $u$  taký, že hrana  $e = \{v, u\}$ .*

**Definícia 11** (Incidencia dvoch hrán). *Dve hrany  $e_1$  a  $e_2$  sú incidentné, ak existuje vrchol  $v$ , a vrcholy  $u, w$ , kde platí, že  $e_1 = \{u, v\} \wedge e_2 = \{v, w\}$ .*

**Definícia 12** (Susednosť vrcholov). *Dva vrcholy,  $u$  a  $v$  sú susedné, ak existuje hrana  $e = \{v, u\}$ .*

**Definícia 13** (Stupeň vrchola). *Stupeň vrchola* (valencia) je počet jeho susedov.

Zvyčajne označujeme funkciou pre vrchol  $v$  v  $d(v)$ , alebo premennou  $k$ :

$$d(v) = k. \quad (2.10)$$

Ak neuvedieme ináč, budeme tento zvyk dodržovať aj v texte práce.

**Definícia 14** (Priemerný stupeň vrchola). *Priemerný stupeň vrchola* v grafe je suma stupňov jednotlivých vrcholov vydelená počtom vrcholov.

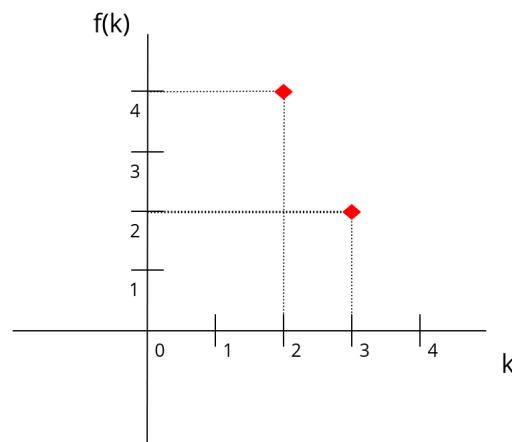
$$d(G) = \bar{k} = \frac{1}{n} \sum_{i=0}^{n-1} d(v_i). \quad (2.11)$$

**Definícia 15** (Postupnosť stupňov vrcholov). *Postupnosť stupňov vrcholov* grafu je nerastúca postupnosť stupňov jednotlivých vrcholov grafu.

**Definícia 16** (Distribúcia stupňov vrcholov). *Distribúcia stupňov vrcholov* grafu je štatistická vlastnosť grafu skúmajúca početnosť jednotlivých stupňov vrcholov vyskytujúcich sa v grafe.

To znamená, že je to zobrazenie zo stupňa vrchola na počet vrcholov s týmto stupňom.

Na obrázku 2.3 vidíme distribúciu stupňov vrcholov prislúchajúcu grafom z obrázku 4.



Obr. 2.3: Príklad distribúcie stupňov vrcholov, kde  $k$  je stupeň vrchola a  $f(k)$  je počet vrcholov v grafe so stupňom  $k$ .

**Definícia 17** (Izolovaný vrchol). *Izolovaný vrchol je vrchol, ktorý nemá žiadnych susedov.*

**Definícia 18** (Súčet stupňov vrcholov ľubovoľného grafu je vždy párný.). *Každá hrana prispieva 1 stupňom práve 2 vrcholom, preto platí:*

$$\sum_{v \in V} d(v) = 2 \cdot |E|. \quad (2.12)$$

**Definícia 19** (Sled). *Sledom nazývame striedavú postupnosť vrcholov a hrán. Dĺžka je určená počtom hrán. Pričom platí, že každá hrana sledu je incidentná s vrcholom nachádzajúcim sa bezprostredne pred a za danou hranou. Sled teda začína aj končí vrcholom.*

**Definícia 20** (Tah). *Tah je taký sled, v ktorom sa neopakujú hrany.*

*Ak tah začína aj končí tým istým vrcholom, hovoríme mu **uzavretý**.*

*Ak tah nie je uzavretý, nazývame ho **otvorený**.*

**Definícia 21** (Cesta). *Cesta je taký tah, v ktorom sa neopakujú vrcholy.*

*Ak cesta začína aj končí tým istým vrcholom, voláme ju **cyklus** (kružnica). Cykly uvažujeme iba dĺžky  $> 2$ .*

**Definícia 22** (Eulerovský tah). *Eulerovský tah je tah obsahujúci každú hranu grafu.*

*Teda v grafe sa môže nachádzať, iba ak sú v grafe nanajvýš 2 vrcholy s neparným stupňom.*

Z toho vyplýva, že graf z obrázku 2.2 neobsahuje Eulerovský tah.

**Definícia 23** (Eulerovská kružnica). *Eulerovská kružnica je uzavretý Eulerovský tah.*

*Teda v grafe sa môže nachádzať, iba ak sú v grafe všetky vrcholy s parným stupňom.*

**Definícia 24** (Eulerovský graf). *Eulerovský graf je graf obsahujúci Eulerovskú kružnicu.*

**Definícia 25** (Hamiltonovská cesta). *Hamiltonovská cesta je cesta obsahujúca každý vrchol grafu.*

**Definícia 26** (Hamiltonovská kružnica). *Hamiltonovská kružnica je Hamiltonovská cesta, ktorá je zároveň cyklom.*

**Definícia 27** (Súvislý graf). *Grafu hovoríme **súvislý** ak medzi každými dvoma vrcholmi existuje nejaká cesta, v opačnom prípade grafu hovoríme **nesúvislý**.*

**Definícia 28** (Vzdialenosť vrcholov). *Vzdialenosť vrcholov je dĺžka najkratšej cesty medzi nimi.*

**Definícia 29** (Podgraf). *O všetkých grafoch  $G'$ , pre ktoré platí:*

$$\forall G = (V, E), \forall G' = (V', E'), (V' \subseteq V) \wedge (E' \subseteq E), \quad (2.13)$$

*hovoríme, že sú **podgrafom**  $G$ .*

*Ak však platí:*

$$V' = V, \quad (2.14)$$

*podgrahu  $G'$  hovoríme **pokrývajúci**.*

*Ak  $G'$  obsahuje všetky incidentné hrany, ktoré obsahoval aj pôvodný graf s danou podmnožinou vrcholov  $V'$  (rovnako aj incidentné nehrany: neexistujúce hrany medzi danými dvoma vrcholmi) nazývame ho **indukovaný podgraf**, alebo **graflet**.*

*V opačnom prípade mu hovoríme **neindukovaný podgraf**, alebo **motív**.*

**Definícia 30** (Komponenty súvislosti). *Komponenty súvislosti grafu sú také súvislé podgrafe, že medzi ľubovoľnými dvoma vrcholmi dvoch rôznych komponentov neexistuje cesta.*

**Definícia 31** (Most). *Most je taká hrana, ktorej odobratím z grafu by sme zapríčinili zvýšenie počtu komponentov súvislosti práve o 1.*

**Definícia 32** (Artikulácia). *Artikulácia je vrchol incidentný s mostom. Avšak rovnako po odobratí takého vrchola sa musí zvýšiť počet komponentov súvislosti grafu aspoň o 1. Napríklad  $P_2$  (definícia 38) má 1 most, ale nemá žiadnu artikuláciu.*

**Definícia 33** (Acyklický graf). *Graf voláme **acyklický**, ak neobsahuje žiadny cyklus.*

**Definícia 34** (K-regulárny graf). *Graf obsahujúci všetky vrcholy rovnakého stupňa k voláme **k-regulárny graf** (regulárny graf stupňa k).*

**Definícia 35** (Komplettný graf). ***Komplettný graf** je regulárny graf stupňa n – 1.*

Niekedy mu hovoríme aj **klika**, značíme  $K_n$ , podľa počtu vrcholov n.

**Definícia 36** (Komplementárny graf). *Ak máme dané grafy  $G = (V, E)$  a  $G' = (V', E')$  a platí:*

$$(V = V') \wedge (E \cap E' = \emptyset), \quad (2.15)$$

a následným zjednotením hrán:

$$E'' = E \cup E', \quad (2.16)$$

vznikne komplettný graf  $G'' = (V, E'')$ , potom hovoríme, že grafy G a G' sú navzájom **komplementárne** (doplňkové), respektíve G a G' sú jeden druhému komplementami (doplňkami).

**Definícia 37** (Cyklický graf). ***Cyklický graf** je súvislý regulárny graf stupňa 2.*

Cyklický graf s n vrcholmi značíme  $C_n$ .

**Definícia 38** (Cesta). ***Cesta** (v zmysle označenia typu štruktúry grafu) je súvislý graf, v ktorom najviac 2 vrcholy majú stupeň 1 a ostatné sú stupňa 2.*

Cestu s n vrcholmi značíme  $P_n$ . Napríklad na obrázku 1 vidíme  $P_3$  a  $P_4$ .

**Definícia 39** (Bipartitný graf). ***Bipartitný graf** je súvislý graf, ktorého vrcholy vieme rozdeliť na dve disjunktné podmnožiny zvané **partície**, tak že medzi žiadnymi dvoma vrcholmi rovnakej parície neexistuje hrana.*

**Definícia 40** (Komplettný bipartitný graf). *Ak máme bipartitný graf, obsahujúci všetky hrany medzi dvoma partíciemi, ktoré medzi danými vrcholmi rôznych partícií mohli vzniknúť voláme ho **komplettný bipartitný graf**.*

Značíme  $K_{p,q}$ , kde p a q sú velkosti jednotlivých partícií. Samozrejme platí:

$$p + q = n. \quad (2.17)$$

**Definícia 41** (Hviezda). ***Hviezda** je komplettný bipartitný graf typu  $K_{p,1}$ .*

Značíme ju  $S_n$ .

**Definícia 42** (Strom). *Súvislý acyklický graf nazývame **strom**.*

**Definícia 43** (Izomorfizmus grafov). *Izomorfizmus grafov je relácia ekvivalencie na množine všetkých grafov, ktorá ich rozdeľuje do takzvaných izomorfných tried. Dva grafy patria do rovnakej triedy (sú izomorfné), práve vtedy keď majú rovnaký počet vrcholov a hrán a vrcholy jedného grafu vieme zobraziť na vrcholy druhého grafu tak, že každá hrana aj nehrana (neexistujúca hrana medzi dvoma vrcholmi) jedného grafu sa zobrazi na hranu (nehranu) druhého grafu.*

$$G = (V, E), G' = (V', E'), (|V| = |V'|) \wedge (|E| = |E'|), \quad (2.18)$$

$$\varphi : V \rightarrow V' \mid \forall v_1, v_2 \in V, (v_1, v_2) \in E \iff (\varphi(v_1), \varphi(v_2)) \in E', \quad (2.19)$$

$$G \cong G'. \quad (2.20)$$

**Definícia 44** (Automorfizmus grafu). *Pojmom **automorfizmus** označujeme izomorfné zobrazenie grafu na samého seba. To jest okrem (2.18), (2.19), (2.20) platí ešte aj:*

$$(V = V') \wedge (E = E'), \quad (2.21)$$

*pričom zobrazenie  $\varphi$  zobrazuje  $i$ -ty vrchol z  $V$  na  $i$ -ty vrchol z  $V'$ , teda  $\varphi$  je identita.*

**Definícia 45** (Samokomplementárny graf). *Samokomplementárny graf je graf izomorfný so svojím komplementom.*

**Definícia 46** (Automorfná grupa). *Všetky automorfizmy daného grafu tvoria **automorfnú grupu**. Pre graf  $G$  ju označujeme  $\text{Aut}(G)$ .*

**Definícia 47** (Automorfný orbit). *Skúmajme všetky automorfizmy daného grafu  $G$ . Potom rozdelme vrcholy grafu  $G$  do disjunktných podmnožín, do rovnakej podmnožiny dajme tie vrcholy, ktoré sa môžu zobraziť na seba navzájom. Tieto podmnožiny vrcholov voláme **automorfné orbity** (orbity).*

*Pre konkrétny vrchol  $v$  a jeho bijektívne zobrazenie  $g(v)$  dvoch automorfizmov grafu  $G$ , teda zistíme všetky vrcholy patriace rovnakému orbitu nasledovne:*

$$\text{Orb}(v_1) = \{v_2 \in V(G) \mid \forall g \in \text{Aut}(G), v_2 = g(v_1)\}. \quad (2.22)$$

Na obrázku 6 vidíme rozdelenie vrcholov daných grafletov do jednotlivých orbitov. Rôzne orbity grafletu sú znázornené rôznou farbou.

*Neformálne* môžeme orbit chápať podľa rekurzívnej definície orbitového stupňa vrchola. Orbitový stupeň vrchola je ako obyčajný stupeň vrchola, avšak pozeráme sa naň vzhľadom na stupne vrcholov, ktoré daný stupeň vrchola vytvorili. V rekurzii sa zastavíme práve vtedy, keď by to vyzeralo, že sa ideme zacykliť (teda keď opäť narazíme na pôvodný vrchol uváhy orbitového stupňa). Vrcholy s rovnakým orbitovým stupňom potom patria rovnakému orbitu.

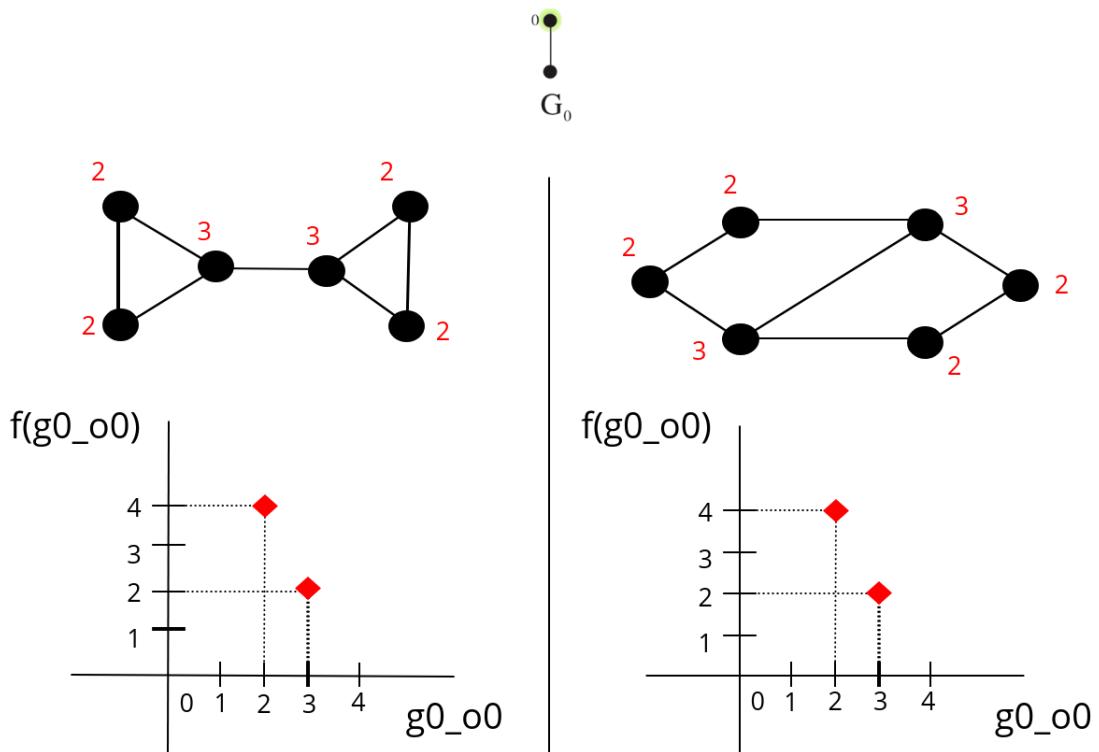
Pozorujeme, že pri distribúcií stupňov vrcholov vlastne skúmame počet dotykov každého vrchola grafu  $G$  s grafletom  $G_0$  z obrázka 6. Hovoríme, že vrchol sa grafetu dotýka, ak sa nachádza v indukovanom podgrafe grafu  $G$  izomorfom s daným grafletom. Hovoríme, že vrchol sa dotýka grafetu v danom orbite (dotýka sa orbitu), ak vrchol patrí do daného automorfného orbitu skúmaného grafetu.

Na obrázku 6 sú pridelené unikátne čísla každému orbitu každého z neizomorfných skúmaných grafletov, aby sme medzi nimi mohli rozlišovať.

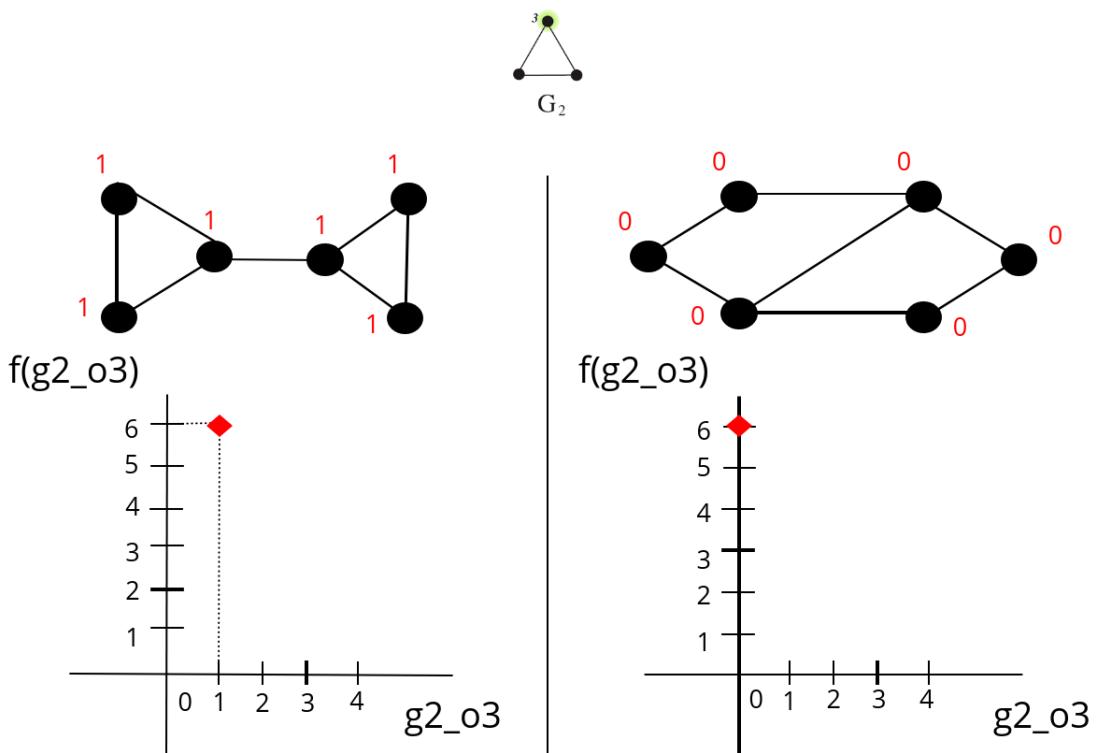
V distribúcií stupňov vrcholov, teda skúmame počet dotykov jednotlivých vrcholov s *orbitom 0* (*z grafletu 0*). Následne zostrojíme distribúciu zachytávajúcu početnosť rôznych počtov dotykov s daným orbitom vyskytujúcich sa v grafe.

Neexistuje žiadnený dôvod, prečo by sme nemali skúmať okrem distribúcie *orbitu 0*, aj distribúcie iných orbitov. Práve tieto distribúcie 73 orbitov z obrázku 6 skúmame v našej práci. Vďaka nim pravdepodobnostne neskôr určíme podobnosť štruktúry dvoch grafov.

Na obrázkoch 2.4 a 2.5 vidíme príklady distribúcie konkrétnych orbitov pre grafy z obrázku 4.



Obr. 2.4: Príklad distribúcie orbitu 0 z obrázku 6.



Obr. 2.5: Príklad distribúcie orbitu 3 z obrázku 6.

# Kapitola 3

## Implementácia

Napriek tomu, že nemusíme skúmať všetkých  $n!$  možných bijekcií vrcholov grafu ako by to bolo pri skúmaní úplného izomorfizmu dvoch grafov, riešime výpočtovo náročnú úlohu, navyše s veľmi veľkými vstupmi. Od začiatku projektu sme teda boli odhodlaní vytvoriť paraleлизovaný, distribuovaný systém.

Na implementáciu sme zvolili programovací jazyk *GO*. Podľa rýchlosťných benchmarkov [10] ho radíme približne medzi *javu* a *c++*. *GO* nám poskytuje ľahkosť a eleganciu *pythonu*, *GO* je však jazyk kompilovaný, súčasne nám teda poskytuje aj rýchlosť takmer ako *c++*.

*GO* nie je typický objektový jazyk, nepodporuje napríklad dedičnosť v klasickom zmysle, či pretažovanie funkcií. Umožňuje však definovať štruktúry **struct** a metódy pre inštancie takýchto štruktúr, môžeme ich preto považovať za objekty aké poznáme z iných jazykov.

*GO* je protagonistom procedurálnej paradigmy, tá poskytuje úplne postačujúcu funkcionality na naše účely.

Čo je však kľúčové a prečo sme si vybrali práve *GO* je ľahkosť a elegancia s akou v ňom dokážeme tvoriť sieťové programy: náš systém bude koncipovaný ako distribuovaný systém so serverom rozdeľujúcim úlohy pripojeným klientom, ktorí ich budú počítať; a špecialitu tohto jazyka takzvané *GO routiny*: niečo ako maskované thready, alebo asynchronné tasky, ktoré poznáme z iných jazykov, *GO routíny* však môžeme vytvoriť a spustiť až milióny bez toho, aby sme odrovnali počítač bežnej výkonnostnej

triedy. O všetko sa totiž stará takzvaný *GO scheduler*, bežiaci na pozadí, ktorý sám automaticky prideľuje *GO routines* skutočným systémovým threadom, tak aby využitie CPU bolo optimálne a efektívne respektíve aby počítač úplne nezamrzol pri veľkom počte threadov... To koľko skutočných jadier má náš program využívať je nastavené v premennej prostredia *GOMAXPROCS*. Toto nastavenie je možné upraviť podla účelu aplikácie avšak neodporúča sa nastavovať väčší počet jadier, než akým v skutočnosti disponuje daný počítač. Od verzie *GO 1.5* je však podľa dokumentácie [11] automaticky nastavené na skutočný počet jadier procesora, čo je najoptimálnejšie a vyhovuje aj našim účelom. Do verzie *GO 1.5* bolo nastavenie pevne dané na 1 jadro.

Teda naša aplikácia bude bežať optimálne na počítači s ľubovoľným počtom jadier procesora respektíve aj hyperthreadovaných jadier.

Ukážme si na názornom príklade, aké jednoduché je vytvoriť a naštartovať *GO routinu*: štandardnú funkciu zavolanú klúčovým slovíčkom **go**.

Bežný sekvenčný for-cyklus z obrázku 3.1 modifikujeme na paralelný for-cyklus 3.2. Dbáme pritom na to, aby telo hlavnej metódy neskončilo skôr, než sa ukončia všetky paralelné *GO routines*.

```

1 for _, n := range v.neighbours {
2     // time-consuming task
3 }
```

Obr. 3.1: Príklad bežného for-cyklu.

```

1 loop_para := new(sync.WaitGroup)
2 loop_para.Add(len(v.neighbours))
3 for _, n := range v.neighbours {
4     go func(n *Vertex) {
5         // time-consuming task
6         loop_para.Done()
7     }(n)
8 }
9 loop_para.Wait()
```

Obr. 3.2: Príklad paralelného for-cyklu.

Kód máme organizovaný do *GO modulov* (samostaných súborov), všetkých umiestnených v hlavnom packagi (scope prostredí) `package main`. Väčšinou jeden modul predstavuje jednu samostatnú triedu, avšak nie je to pravidlom. Ide skôr o to aby logické celky boli nejako oddelené, čo zabezpečuje lepšiu prehľadnosť a udržiavateľnosť zdrojového kódu.

Prvá fáza spočívala v naprogramovaní jadra nášho systému: metód počítajúcich distribúcie orbít vstupného grafu. Z obrázka 6 vyplýva, že priamym zistovaním dotykov spočítame pre každý orbit najhorším možným spôsobom jeho distribúciu s časovou zložitosťou programu  $O(n^5)$  (zatiaľ skúmame orbity iba týchto vybratých maximálne 5 vrcholových grafletov). Čo nie je súčasťou smrteľné ako je to pri backtrackujúcich programoch o zložitosti  $O(n!)$ , avšak so vstupným grafom, kde  $n = 8394$  to mohol byť hypotetický problém.

Skúšali sme sa teda zamýšľať nad nejakými kombinatorickými vylepšeniami avšak bez úspechu. Ešte v úplných (avšak už trochu rozbehnutých) začiatkoch (v zimnom semestri) sme sa dokonca od kolegu zo seminára o grafoch Mariána Vyslúžila dozvedeli o existencií konkurenčného softwaru *Orce* [12]. Vzhľadom na to sme sa snažili preštudovať aj tento software. Avšak aj po skutočne veľkom úsilí sa nám nepodarilo plne vniknúť do jeho tajov. Čo sme však zistili zo zdrojových kódov *Orcy* je, že *Orca* používa nejaký premyslený kombinatorický algoritmus, kde priamo vypočíta iba zopár vybratých dotykov orbitov a z nich následne systémom rovníc dopočíta tie zvyšné... *Orca* potrebuje dopredu poznať počet vrcholov a hrán, čo problém nie je, algoritmus má však v sebe kvôli systému rovníc isté návaznosti, je otázne do akej miery sa dá paraleлизovať... ukážkový program v *c++* nie je paraleлизovaný beží na jednom počítači a na jednom jadre. V tom čase sme mali hotové počítanie iba 4 orbít bolo, teda tažké robiť rýchlosné benchmarky dopredu, keďže náš systém sme plánovali testovať so všetkými počítačmi z I-H3 a I-H6 FMFI UK...

Aj vzhľadom na fakt, že algoritmu *Orcy* sme plne nepozozumeli a necheli sme kopírovať cudziu prácu, ktorú by sme neskôr ani nevedeli vysvetliť a obhájiť, po dôkladnej konzultácii s našim školiteľom sme sa rozhodli pokračovať v pôvodne zamýšľaných vlastných algoritmoch: *hrubou silou, ale tak dobre ako sa len dá*.

Našim cieľom bolo spočítať zadanú konkrétnu úlohu (konkrétnie grafy od školiteľa)

na konkrétnych počítačoch I-H3 a I-H6 FMFI UK, ktoré sme mali k dispozícii na vykonanie experimentu...

Postupne sme teda programovali metódy na počítanie jednotlivých orbít a časom, napriek neustálej refaktORIZÁCIÍ a rozširovaniu funkcionality, sa v kóde začali kryštaliZOVAT jednotlivé moduly:

- graph.go
- vertex.go
- vertex\_orbits.go
- gdd.go
- graph\_isomorpher.go
- a\_u\_x.go
- graph\_testing.go
- vertex\_testing.go
- vertices\_interval.go
- orbit\_distribution.go
- gdd\_test.go
- concurrent\_int.go
- server.go
- client.go
- worker.go
- task.go
- task\_blueprint.go
- message.go

- results.go
- desync\_patch.go
- config.go
- stat\_tasks.go
- stat\_results.go

Pôvodná myšlienka bola naprogramovať 73 orbitových metód. Server by potom rozdelil úlohy klientom: každý klient by rátal v jednom čase jednu úlohu: distribúciu 1 orbitu (zo 73) pre 1 súbor (zo 40) na všetkých jadrách. Každý vrchol by o sebe vedel zistiť kolkokrát sa dotýka daného orbitu. Paralelizmus by teda spočíval v tom, že by sme vytvorili *GO routinu* pre každý jeden vrchol a teda všetky vrcholy by klient spočítal akoby naraz. Z jednotlivých počtov dotykov by sme následne skompomovali distribúciu. Server by pravdaže prideľoval úlohy iba aktuálne nečinným klientom...

Ako sme postupne programovali orbitové metódy dospeli sme k názoru, že pôvodne zamýšľaný prístup neboli efektívny. Veľmi veľa grafletov z obrázku 6 je totiž indukovaným podgrafom nejakého iného grafetu z obrázku 6. Teda by bolo zbytočné napríklad postupne budovať  $P_3$  a z neho potom zistovať  $P_4$  a pri zistovaní  $P_5$  opäť duplicitne skúmať  $P_3$ . Takýchto príkladov by sme našli v práci viac.

Prvá zásadná zmena teda bola, že nemusíme počítať úplne každý orbit zvlášt, ak sa napríklad dá spočítať spoločne s nejakým iným výhodnejšie (v otázkach časovej, ale aj pamäťovej zložotosti). S každým ďalším naprogramovaným orbitom sme teda kód ustavične refaktorovali a vylepšovali. Bol to veľmi iteratívny proces, častokrát sme menili kód už úplne naprogramovaného orbitu, len aby sme mohli spočítať dva za cenu jedného. S tým súviselo jedno ďalšie zásadné vylepšenie: unit testy.

Unit testy totiž pôvodne neboli vôbec zamýšľané. Avšak s pribúdajúcim kódom bolo už veľmi tažké sledovať, či sme nejakým vylepšením nepokazili niečo už naprogramované, respektíve s pribúdajúcimi zložitejšími grafletmi častokrát už aj testovacie výpisu do konzoly nepostačovali. Bolo nepraktické zakaždým očami kontrolovať jednotlivé distribúcie... preto sme začali používať automatizované testy. Spočiatku iba na

každý orbit, či vráti očakávanú hodnotu: tú sme zistili z nakresleného grafu na paperi, ktorému sme očami spočítali distribúciu. To znamená, že aj v testoch ešte môže byť chyba, avšak je to nepravdepodobné.

Priebežným testovaním časov vykonávania orbitových metód sme napokon dokončovali k ďalšej zásadnej zmene. A sice, plne sme si uvedomili výpočtový potenciál terminálových miestností FMFI UK poskytnutých na náš experiment. Rátať celý rozsah vrcholov naraz v jednej úlohe na jednom klientovi by bolo plynváním potenciálu distribuovaného systému. Čo ak by napríklad výpočet trval aj niekoľko desiatok hodín a tesne pred koncom by sa počítač zasekol? Uvedomili sme si, že úlohy potrebujeme rozbiť na ešte menšie kúsky: v jednej úlohe chceme rátať konkrétny orbit pre konkrétny súbor avšak iba pre nejakú podmnožinu všetkých vrcholov grafu. Súbory máme zadané korektne ako zoznam hrán: jeden riadok textového súboru obsahuje dve čísla oddelené medzerou. Sú to celočíslené názvy vrcholov od 0 po  $n - 1$ . Napriek tomu pri vytváraní grafu jeho vrcholy prečíslujeme indexami od 0 po  $n - 1$ , zaujíma nás štruktúra grafu teda vytvorením izomorfného nič nepokazíme. Práve naopak umožní nám to bez obáv (náš systém rozparsuje súbory ľubovoľných nezáporných celočíselných indexov) vytvoriť podúlohy pre jednotlivé disjunktné podmnožiny všetkých vrcholov grafu a tie následne distribuovať na voľných klientov.

Mali sme snahu nepočítať nič viackrát než naozaj treba. Aj keď, žiaľ nie úplne vždy sa nám to podarilo. Napríklad *orbit 8* z obrázku 6 môžeme rátať buď hlúpo so zložitosťou  $O(n^4)$ , alebo nájst všetky grafletry, ktorých sa náš skúmaný vrchol dotýka v *orbite 1*. Keby sme teraz nejakou šikovnou dátovou štruktúrou, alebo dobrým algoritmom vedeli prefiltrovať pole týchto  $P_3$  a porovnať vrcholy *orbít 1*, v prípade zhody vieme detegovať dotyk s  $C_4$ , čo je skúmkaný *orbit 8*. Získali by sme tým teda zložitosť medzi  $O(n^3)$  a  $O(n^4)$ . A hlavne by sme štvorec nezapočítali zbytočne 2krát. Takýchto úvah sme mali v práci niekoľko, boli inšpirované najmä známymi RAGE algoritmami Marca a Shavitte [13]. Žiaľ všetky naše pokusy vždy stroskotali na rýchlosťných benchmarkoch.

V tejto práci nebudeme podrobne rozoberať veci, ktoré nefungujú avšak v prípade záujmu odporúčame podrobne preštudovať celú históriau vývoja projektu prostredníctvom elektronickej prílohy na githube v prílohe A. Pokus, ktorý sa dostal najďalej, nájdeme v commitoch *1dc5cb2f6c* až *1019568217*, zmienku nájdeme aj v issue *1\_000\_000*

*go routines... #8* (avšak upozorňujeme na neformálny štýl akým sme viedli github...).

V konečnom dôsledku bolo teda rýchlejšie spočítať *orbit* 5 hlúpo (so zložitosťou  $O(n^4)$ ) z dvoch strán a potom počet vydeliť dvoma.

Pomaly sme sa blížili k záveru fázy programovania orbít, po celý čas sme však čeliли nevysetlitelnej záhadе: „*Prečo kód stále bežal iba na jednom jadre, napriek požiadavke parallelizmu, explicitne vyjadrenej postredníctvom GO routín?*“ Vo chvíli, keď sme už boli rozhodnutí púštať 4 klientské aplikácie na 1 počítači sa nám náhodou podarilo detegovať problém: *GO routín* bolo jednoducho málo, *GO scheduler* vyhodnotil zdrojový kód tak, že nebolo výhodné ho zbiehať paralelne na viacerých jadrách. Jadrá sa sice striedali, ale vyťažené na 100% bolo vždy iba 1 jadro. Začali sme experimentovať s for-cyklami. Doteraz bolo paralelné iba 1. vnorenie, toto nebežalo na viacerých jadrách. 4 paralelné vnorenia už odrovnali aj počítač nadstandardnej výkonnostnej triedy (behom zopár nanosekúnd sme boli schopní zahľubiť 16GB RAM a počítač bol nutné tvrdo reštartovať). 2 paralelné vnorenia už bežali naozaj na všetkých jadrách a vytážovali teda procesor na 100%. 3 paralelné vnorenia už ale nedávali lepšie časy ako 2 (pravdepodobne sa viac času trávilo čakaním v kritických sekciách). Nakoniec sme teda skončili s 2 paralelnými vnoreniami nakoľko s našimi dátami to dávalo najlepšie časy. Táto zmena podnietila vylepšenie unit testov. Doteraz sme testovali iba metódy jednotlivých orbitov, či počítali očakávané distribúcie. Teraz sme však začali testovať aj staré verzie metód (s 1 paralelným vnorením) voči novým skutočne paralelným metódam (s 2 paralelnými vnoreniami). A pridali sme aj testovanie každého typu metód zvlášť, či s väčším grafom ( $n = 5676$ ,  $m = 17192$ ) tá istá metóda vráti rovnaký výsledok viackrát. Je až šokujúce, ako veľa chýb sme týmto krokom detegovali a následne opravili.

Napokon sme skončili iba s 5 rôznymi metódami, ktoré spočítajú všetkých 73 orbít.

Príklad použitia uvádzame na obrázku 3.3.

```
1 g := newGraph( "test/graph_12" )
2 v_list := newVerticesInterval(0, uint64(len(g.vertices)-1))
3 tmp := g.touching_graphlet_orbit_distribution(24, 61, v_list)
```

Obr. 3.3: Príklad práce s grafom. Premenná tmp obsahuje pole viacerých distribúcií orbít, medzi nimi aj orbit 61 grafletu 24.

Prvé dva vstupné argumenty metódy `touching_graphlet_orbit_distribution()` sú nezáporné celé čísla a vyjadrujú graflet a jeho orbit, ktorého distribúciu chceme vypočítať. Tieto argumenty sú interne namapované, aby vyvovali tú správnu metódu ktorá distribúcie spočíta. Vždy teda dostaneme distribúcie všetkých orbít grafletov, ktoré sú si navzájom indukovanými podgrafmi, alebo bolo z nejakého iného dôvodu výhodné rátať ich spolu a ušetriť tým kus pamäte alebo času procesora.

Uvádzame zoznam argumentov a časových zložitostí prislúchajúcich metód, ktoré sme použili v experimente:

- $(1, 2) = O(n^5)$
- $(1, 1) = O(n^5)$
- $(3, 4) = O(n^5)$
- $(23, 56) = O(n^5)$

Namapovaná je aj metóda  $(0, 0) = O(n)$ , avšak toto je tak triviálna úloha, že nebolo časovo výhodné ju distribuovať na klientov, sietovou komunikáciou by sa zabilo viac času než ju spustiť na jednom počítači. Teda nakoniec distribúciu stupňa vrcholov rátame zvlášť až pri spracúvaní meraných dát v module `results.go`. Dokonca nebolo výhodné používať paralelnú verziu, teda špeciálne pre tento orbit sme naprogramovali ešte jednu sekvenčnú metódu.

Strom výpočtu sme sa pokúsili zachytiť na infografike (príloha A), napriek tomu pre plné porozumenie našich algoritmov odporúčame preštudovať celý zdrojový kód (príloha A), v module `vertex_orbits.go` nájdeme orbitové metódy.

Každý klient, keď dostane úlohu s grafom, ktorý ešte nemá nacachovaný, musí si ho najskôr vytvorit. Na pridanie ďalšieho vrchola do zoznamu, musíme najskôr zistiť, či sa už v grafe taký nenachádza. Potrebujeme vhodnú dátovú štruktúru respektíve reprezentáciu grafu, aby úlohy ktoré s ním plánujeme vykonávať mali čo najlepšiu časovú zložitosť.

Nad vhodnou reprezentáciou grafu sme sa dlho zamýšľali. Matica susedností nepričádzala do úvahy. Zvolili sme zoznam vrcholov a pre každý vrchol následne zoznam ich susedov. Všetky zoznamy reprezentujeme hashtabuľkami `map`, kvôli dobrým časovým zložostiam štandardných grafových operácií (ako pridanie ďalšieho vrchola a i.).

Po dokončení tejto hlavnej fázy nás čakala sietová časť a s ňou experiment v I-H3 a I-H6 FMFI UK.

# Kapitola 4

## Testovanie systému

Sietová časť bola bezpochyby tá najťažšia fáza tohto projektu a spočiatku sme ju veľmi podcenili.

Respektíve nebolo ľažké samotné vytvorenie `tcp` spojení. O všetko sa totiž za nás postarala vstavaná knižica `net`. Na obrázkoch 4.1 a 4.2 vidíme, že práca s touto knižnicou je veľmi intuitívna. Dokonca aj posielanie ľubovoľne veľkých objemov dát už niekto vyriešil za nás: v `tcp` protokole sú nadmerne veľké dáta fragmentované na menšie paczky a vždy máme garantované, že na druhú stranu sa dostanú spoľahlivo a v správnom poradí.

Problém nastal pri riešení kritických sekcií a deadlockov. Potrebovali sme docieľiť, aby server bol schopný akceptovať ľubovoľný počet klientov žiadajúcich o pripojenie. Čo ak by sa chceli všetci klienti pripojiť naraz? Keď už boli klienti pripojení, potrebovali sme zabezpečiť komunikáciu medzi serverom a každým pripojeným klientom, aby server mohol rozdeľovať ešte nespočítané úlohy práve nečinným klientom. Opäť sme museli riešiť konkurentné problémy... Čo ak by nejaký klient spadol počas výpočtu, alebo nebodaj samotný server?

```

1 func (s *Server) start() {
2     s.server_start_time = time.Now()
3     s.load_tasks()
4
5     ln, err := net.Listen("tcp", ":"+s.port)
6     if err != nil {
7         // error notification
8         os.Exit(1)
9     }
10    log.Printf("Server is listening on port: %s", s.port)
11
12    for {
13        conn, err := ln.Accept()
14        if err != nil {
15            // error notification
16            continue
17        }
18        go s.handle_connection(conn)
19    }
20}

```

Obr. 4.1: Ukážka štartovacej metódy servera.

```

1 func (c *Client) start() {
2     conn, err := net.Dial("tcp", c.host+":"+c.port)
3     if err != nil {
4         // error notification
5         os.Exit(1)
6     }
7     log.Printf("Client dialed %v and is waiting for @ACK", c.host)
8
9     c.conn = conn
10    c.reader = bufio.NewReader(conn)
11    c.writer = bufio.NewWriter(conn)
12
13    // handshake
14    log.Printf("Successfully joined %s with ID = %d", c.host, c.id)
15
16    c.run()
17 }
```

Obr. 4.2: Ukážka štartovacej metódy klienta.

Všetky spomenuté a mnohé ďalšie problémy sme napokon vyriešili a boli sme pripravený vykonať experiment. Okrem toho nás tlačil aj čas, pretože sme dopredu nevedeli ako dlho výpočet potrvá, nechceli sme nič nechať na náhodu. Potrebovali sme mať systém prakticky úplne hotový už do Veľkej noci, nakoľko počas sviatkov sa v počítačových halách vytvorilo veľké okno v rozvrhu výučby, ktoré sme chceli využiť. Veľká noc 2017 bola teda jedinečnou príležitostou na plánovaný experiment. Tabuľky 4.1, 4.2 sumarizujú veľkosť vstupu experimentu, jednotlivých sietí mozgu 40 účastníkov.

názov súboru	V	E	maximálny $d(G)$
awith_40_std_edges.txt	5677	17200	43
awout_15_std_edges.txt	5439	20276	111
awout_24_std_edges.txt	5924	22315	65
awith_12_std_edges.txt	5335	23059	121
awith_06_std_edges.txt	5575	26204	93
awout_19_std_edges.txt	5287	30043	183
awout_07_std_edges.txt	6681	30443	121
young_25_std_edges.txt	7292	42740	176
awith_35_std_edges.txt	7347	45094	95
awith_31_std_edges.txt	6270	50549	162
awith_36_std_edges.txt	5752	55169	135
awout_11_std_edges.txt	6725	59929	242
awith_02_std_edges.txt	7529	66494	184
awith_33_std_edges.txt	7969	71869	167
awith_37_std_edges.txt	6218	73880	202
young_20_std_edges.txt	7163	83431	261
awith_30_std_edges.txt	6528	85443	193
awout_38_std_edges.txt	6847	87145	321
young_22_std_edges.txt	6482	93436	366
young_34_std_edges.txt	7131	93643	335

Tabuľka 4.1: Tabuľka vyobrazuje štatistiky vstupných datasetov. V tabuľke  $|V|$  je počet vrcholov siete,  $|E|$  je počet hrán siete, maximálny  $d(G)$  je maximálny stupeň vrchola v grafe siete.

názov súboru	V	E	maximálny $d(G)$
young_41_std_edges.txt	6941	101885	328
young_29_std_edges.txt	6977	103778	292
awith_09_std_edges.txt	7558	102091	205
young_23_std_edges.txt	6836	106575	332
awout_27_std_edges.txt	6600	110564	323
young_21_std_edges.txt	6736	112851	378
young_16_std_edges.txt	7032	131887	422
young_17_std_edges.txt	7553	134565	412
awout_13_std_edges.txt	7416	139482	416
young_18_std_edges.txt	7478	151486	433
awout_14_std_edges.txt	6849	172969	468
young_32_std_edges.txt	7539	180380	409
awout_28_std_edges.txt	8151	182021	345
awout_04_std_edges.txt	8000	183940	375
young_26_std_edges.txt	8183	211232	519
young_39_std_edges.txt	8582	307112	504
awout_05_std_edges.txt	8382	319165	591
awout_10_std_edges.txt	8131	361948	648
awout_08_std_edges.txt	8044	395861	778
awith_01_std_edges.txt	8394	450042	651

Tabuľka 4.2: Tabuľka vyobrazuje štatistiky vstupných datasetov. V tabuľke  $|V|$  je počet vrcholov siete,  $|E|$  je počet hrán siete, maximálny  $d(G)$  je maximálny stupeň vrchola v grafe siete.

Sme veľmi vďační všetkým kompetentným osobám, ktoré nám pomohli s vybavením miestnosti na experiment, ako aj jeho samotnou realizáciou.

Crossplatformovosť systému sme pôvodne plánovali demonštrovať, použitím operačného systému Linux na všetkých počítačoch, avšak v kombinácii s 1 počítačom, ktorý by bežal pod operačným systémom Windows. Avšak vzhľadom na skutočnosť, že v danom čase na Linuxoch ešte nebolo nainštalované *GO*, napokon sme sa pre jednoduchosť rozhodli využiť inštalácie Windowsu, čím sme v konečnom dôsledku, ale tiež demonštrovali crossplatformovosť, vzhľadom na to ako rýchlo a flexibilne sme sa dokázali vysporiadať s touto nepredvídateľnou situáciou, ktorá by pre menej robustný systém mohla implikovať aj fatálne následky.

Ďalšou nepredvídanicou komplikáciou bolo, že miestnosti I-H3 a I-H6 nie sú zapojené na tej istej lokálnej sieti. To sme dopredu nevedeli, no nebol pre nás problém sa rýchlo adaptovať aj na túto situáciu a iba sme upravili zoznam úloh na dve disjunktné podmnožiny. Snažili sme sa odhadnúť čas úloh a rozdeliť ich do miestností tak, aby sa čo najviac úloh vyrátkalo súčasne.

V miestnosti I-H3 sa nachádza 30 počítačov s procesorom Intel® Core™ i3-6100 CPU @ 3.70GHz, pamäťou 8.00 GB RAM a operačným systémom Windows 7 v 64-bitovom prevedení. Okrem toho je v miestnosti I-H3 ešte 1 počítač s procesorom Pentium® Dual-Core CPU E5400 @ 2.70GHz, pamäťou 2.00 GB RAM a operačným systémom Windows 7 v 32-bitovom prevedení. Miestnosť I-H6 disponuje 57 počítačmi s procesorom Intel® Core™ i3-2120 CPU @ 3.30GHz, pamäťou 2.00 GB RAM a operačným systémom Windows 7 v 32-bitovom prevedení. Všetky počítače používajú integrovanú grafickú kartu. Vo všetkých našich experimentálnych pokusoch sme používali všetky tieto počítače, výnimkou bolo iba zopár ojedinelých prípadov, keď jeden alebo maximálne dva počítače boli vyradené kvôli aktuálnemu problému s prihlásením, alebo inej poruche. Ako server sme používali výhradne učiteľský počítač.

Prípravu na experiment samotný sme nebrali vôbec na ľahkú váhu. Uvedomovali sme si riziká spojené s neoprávneným vniknutím fyzickej osoby do počítačovej miestnosti. Mali sme pripravený vlastný magnetický alarm na dvere, aby sme výstražným zvukovým znamením upozornili potenciálneho nepozorného návštěvníka na prebiehajúci experiment. Na obrázku 4.3 vidíme pripravené výstražné transparenty, ktoré sme pôvodne plánovali použiť na zabezpečenie miestnosti.

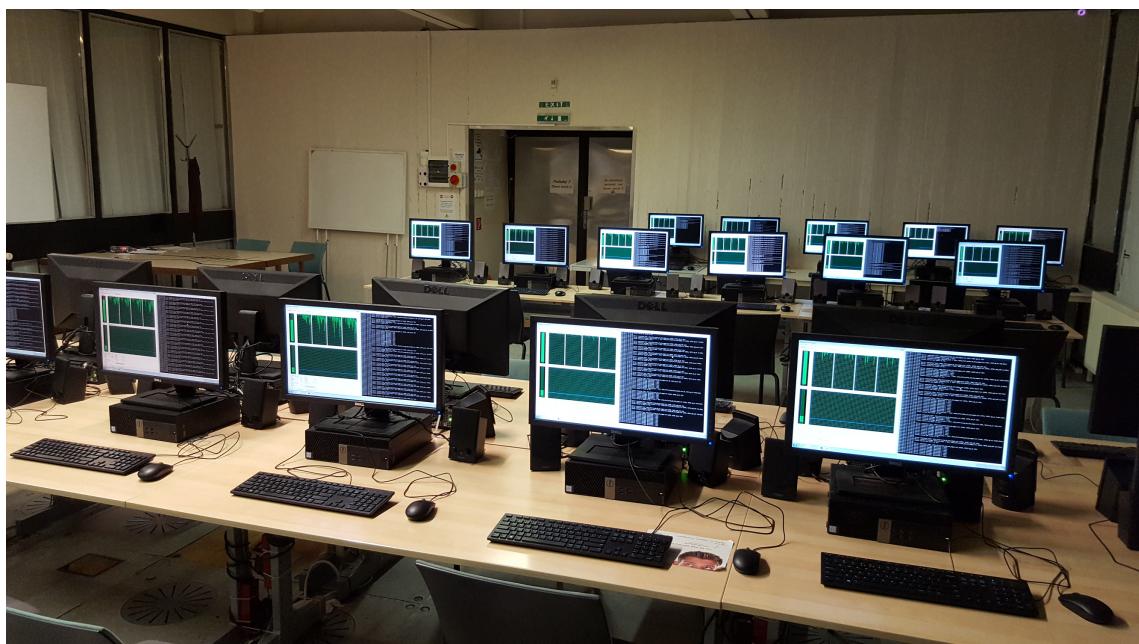


Obr. 4.3: Výstražné transparenty na zabezpečenie vchodov do I-H3 a I-H6 aj so stojanmi.

Avšak zbor vrátničiek a vrátnikov FMFI UK bol vždy veľmi ochotný nám vyjsť v ústrety, miestnosti vždy zamkol na klúč a zabezpečil školským alarmom, teda nakoniec sme naše transparenty nepoužili.

Miestnosť sme zarezervovali v čase od 12.4.2017 20:00 do 19.4.2017 07:00, konečne mohol prebehnúť experiment.

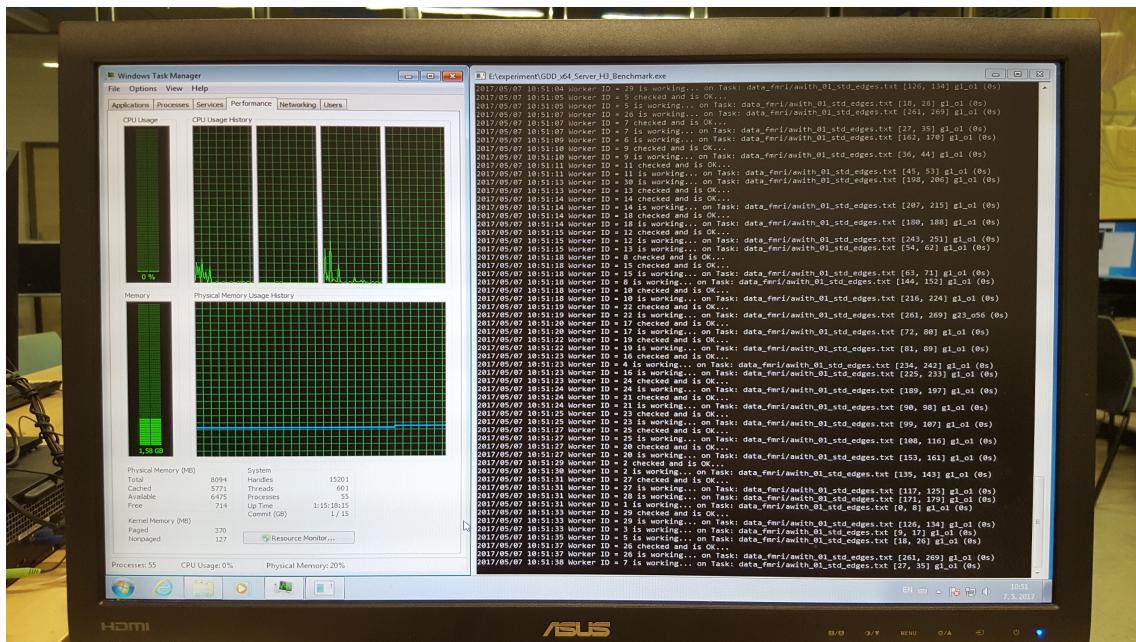
Žiaľ, posledné úpravy sietového kódu nám trvali trošku dlhšie ako pôvodne mali a prvý (nultý) experimentálny pokus sme zahájili až vo štvrtok na obed... Tento pokus skončil zlyhaním. Po pripojení asi 6 klienta sa celý nás sietový kód zosypal... sietový program nefungoval vôbec... prejavili sa chyby, ktoré v domácom prostredí bolo len veľmi ťažké nasimulovať... Bolo to obrovské sklamanie na našej strane. Následne sme celý sietový kód prepísali tak, že všetky problémy v sietovek komunikácií boli odstránené. Prvý už naozaj experimentálny pokus sme zahájili 17.4.2017 o 06:00 ráno na Veľkonočný pondelok.



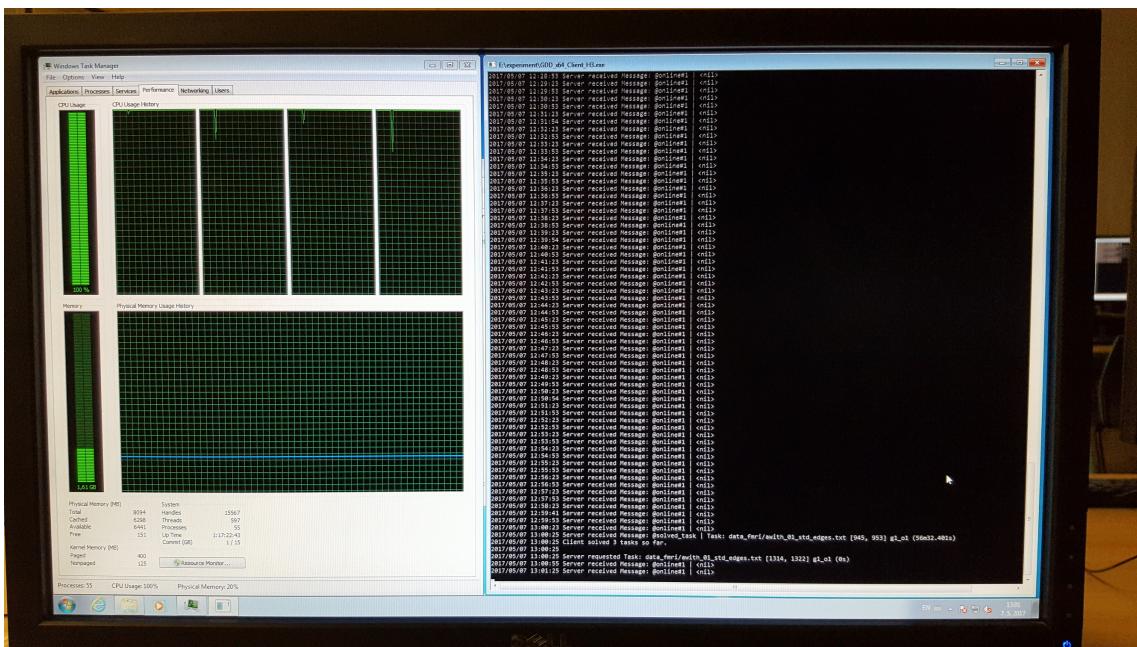
Obr. 4.4: Miestnosť I-H3 počas experimentu.

Na obrázku 4.4 vidíme úspešné zahájenie experimentu. Samozrejme, že sme už nestihli zmerať všetky úlohy, plánovali sme teda experiment dokončiť behom predĺžených víkendov od 28.4.2017 18:00 do 02.5.2017 07:00 a od 05.5.2017 18:00 do 09.

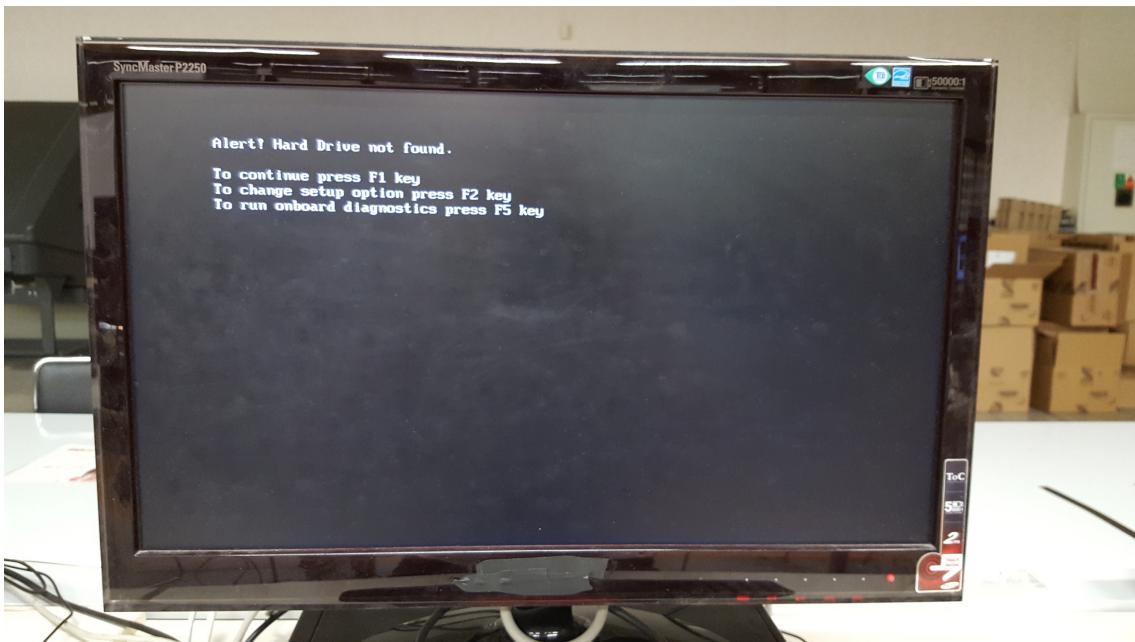
5. 2017 07:00. Počas všetkých týchto pokusov, zamíral server v I-H6. Spočiatku sme tomu neprikladali veľkú dôležitosť, pokladali sme to za náhodnú chybu počítača. Neškorším vyšetrovaním sa zistilo, že šlo o chyby v kritických sekciách a deadlockoch z našej strany. Všetky chyby sme odstránili, avšak došlo aj k chybe pri ukladaní už vy-počítaných úloh, keď sa niekedy takáto spočítana úloha uložila viackrát do súboru a vytlačila tým z poolu všetkých úloh nejakú ešte nespočítanú úlohu, teda sme úplne strátili informáciu, že danú úlohu vôbec treba počítať. Tieto skutočnosti podnietili vznik modulu `desync_patch.go`, ktorý desynchronizované záznamy z experimentu (slúžiace aj ako štartovací súbor pre ďalší experimentálny pokus, aby sme ho mohli rozdeliť na viac nespojitych pokusov v prípade potreby) skontroluje aj opraví v prípade potreby. Teda vďaka tomuto modulu boli časové straty, ktorým sme čelili minimálne, takmer všetko čo sme mali spočítané sa dalo použiť na výskum. Až posledný víkend sme počí-tali s vyladeným programom, ktorý sa už nezasekával. Obrázky 4.5, 4.6, 4.7, 4.5 a 4.9 nás vnesú do atmosféry experimentu.



Obr. 4.5: Obrazovka servera počas počítania úloh. Vidíme, že server iba rozdeľuje úlohy, nič nepočíta, CPU je vyťažené len minimálne.



Obr. 4.6: Obrazovka klienta počas počítania úloh. Vidíme, že klient efektívne využíva zdroje, vyťažuje CPU na všetkých jadrach na 100%. Treba si uvedomiť, že dosiahnut podobnú funkcionality v iných jazykoch (*c++, java...*) by bolo bez hlbokých vedomostí zo systémového programovania len veľmi ťažké. Klient sa v pravidelných 30 sekundových intervaloch ohlasuje serveru. Ak by tak neurobil server ho vyradí z poolu workerov. Ak však klient úspešne dopočíta zadanú úlohu, okamžite ju pošle serveru.



Obr. 4.7: Experiment si vyžiadal aj ujmu na harddiskoch počítačov. Vďaka robustnosti nášho systému sa aj v takejto nepredvídateľnej situácii úlohy vždy dopočítali korektne.

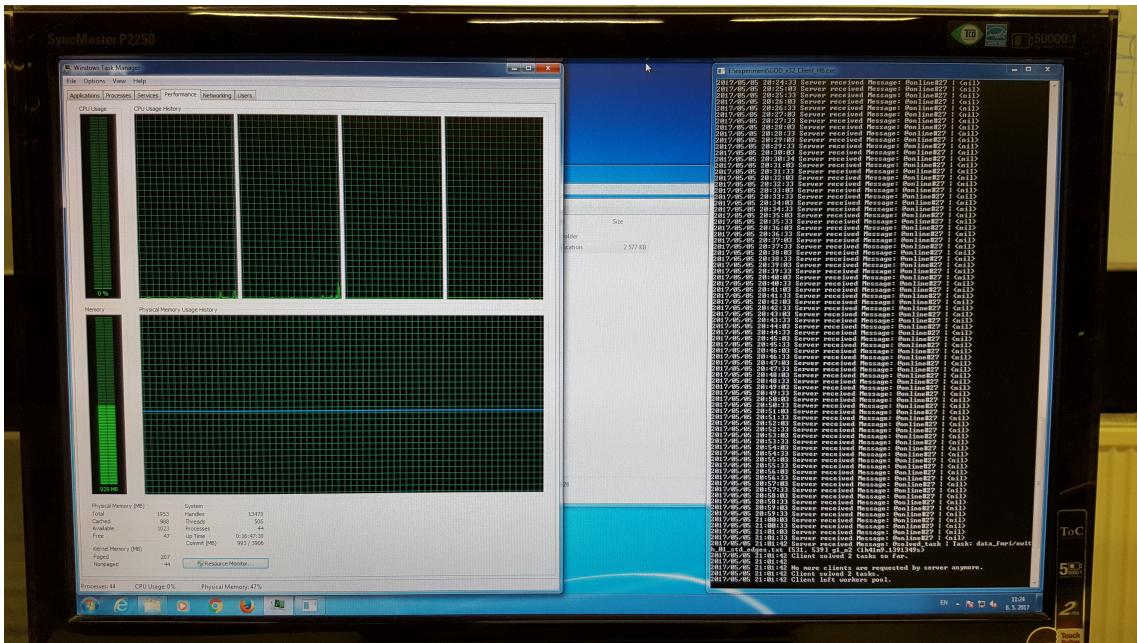
Konečne sme dopočítali všetky úlohy, pripravovali sme sa na analýzu nameraných hodnôt. Avšak teraz, s už úplne hotovým systémom, sme sa rozhodli ešte vykonať rýchlosťny benchmark na porovnanie konkurenčného softwaru *Orcy* a nášho systému. V letnom semestri sa nám podarilo objaviť aj článok Hočevara a Demšara [14] vysvetľujúci algoritmy *Orcy*, avšak pre pokročilý čas a navyše aj stav implementácie nášho systému bolo už nemysliteľné ho znova prerábať.

Testovali sme ten najväčší súbor awith\_01\_std\_edges.txt s parametrami:

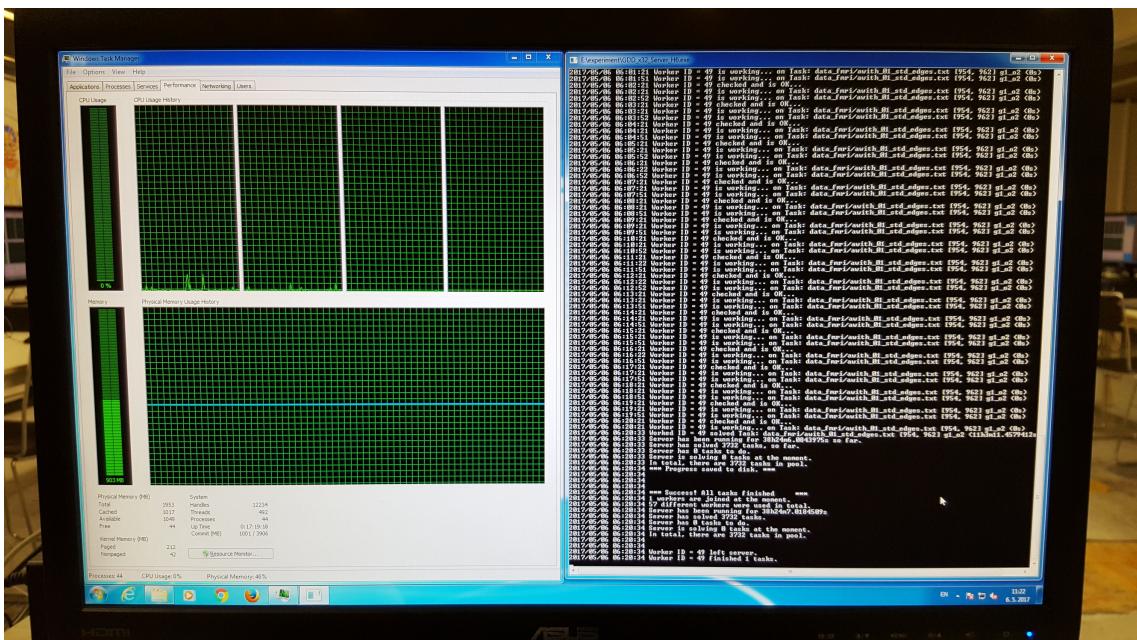
$$|V| = 8394, |E| = 450042 \text{ a } max\_degree = 651.$$

Náš systém spočítal úlohu za **20h56m50.7101602s**. Úlohu sme rozdelili medzi I-H3 a I-H6, aby sme dosiahli, čo najlepší čas. Na obrázkoch 4.10 a 4.11 vidíme záznamy z úspešne dokončeného benchmarku.

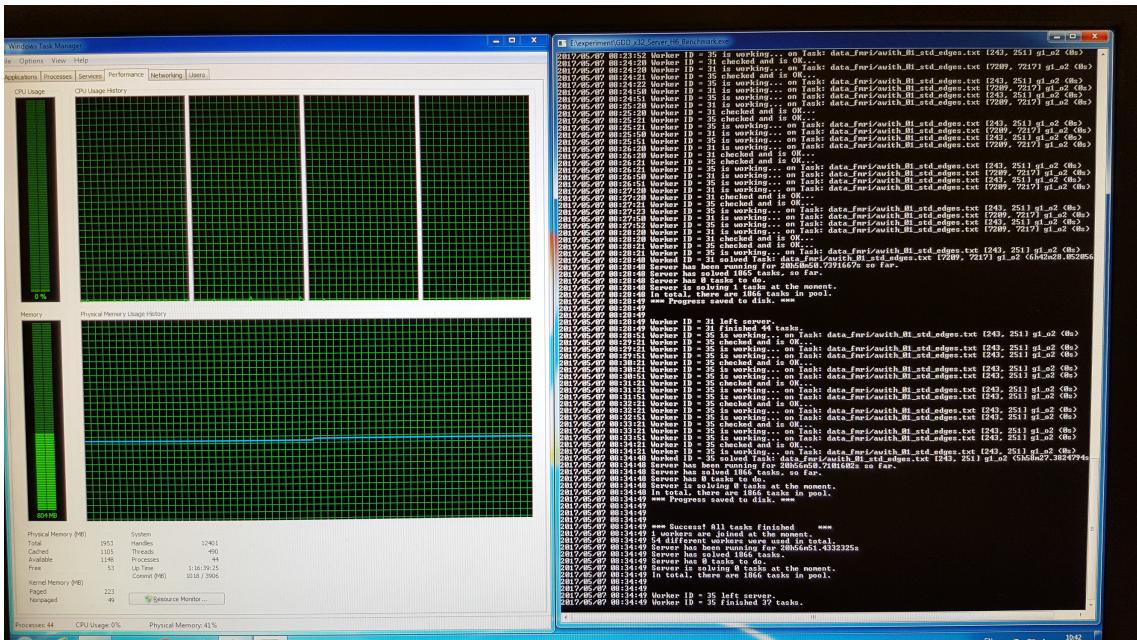
Žiaľ, *Orce* sa nepodarilo úlohu dopočítať, program sa zasekol. Toto sme nepovažovali za relevantné porovnanie, teda sme chceli benchmark opakovať už posledným pokusom od 13.5.2017 07:00 do 14.5.2017 19:00.



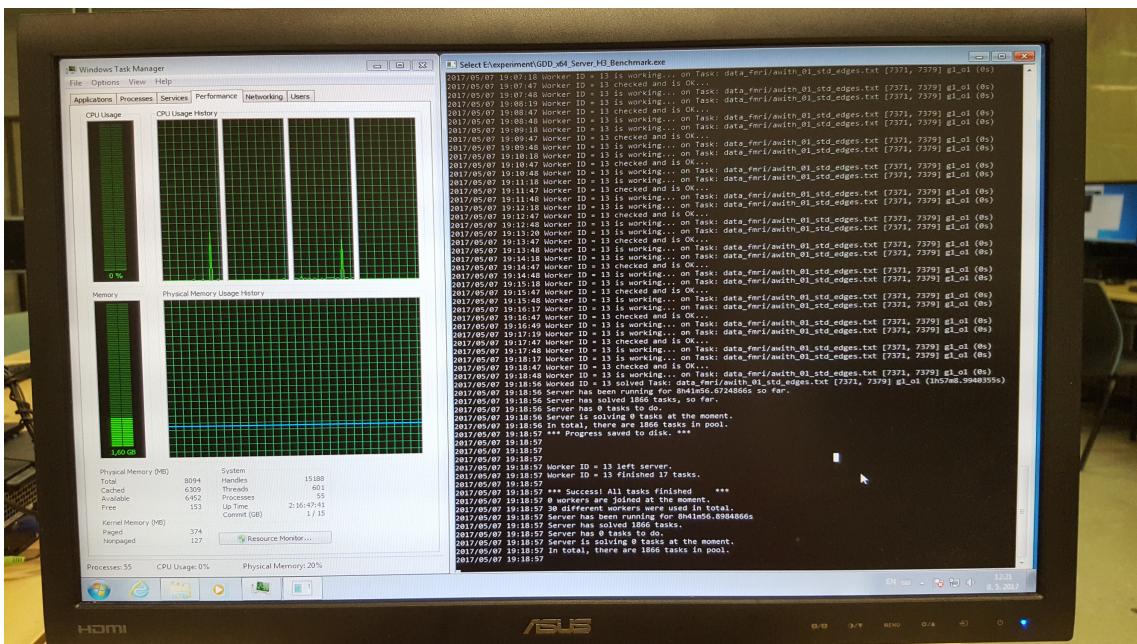
Obr. 4.8: Obrazovka klienta po spočítaní všetkých úloh.



Obr. 4.9: Obrazovka servera po spočítaní všetkých úloh.



Obr. 4.10: Obrazovka servera v I-H6 po dokončení benchmarku.

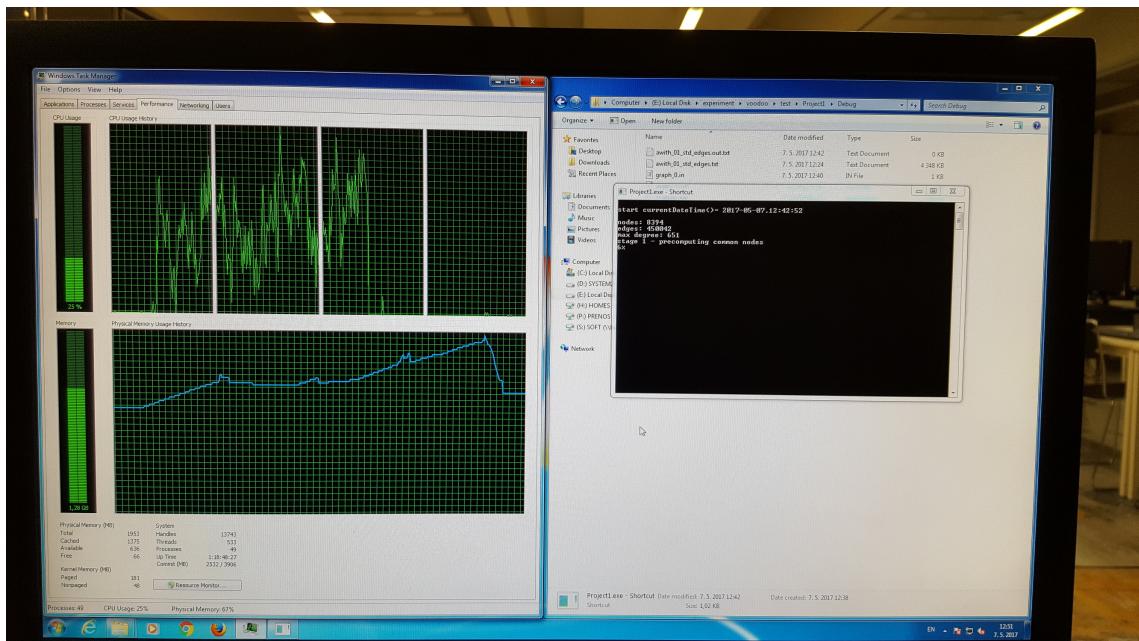


Obr. 4.11: Obrazovka servera v I-H3 po dokončení benchmarku.

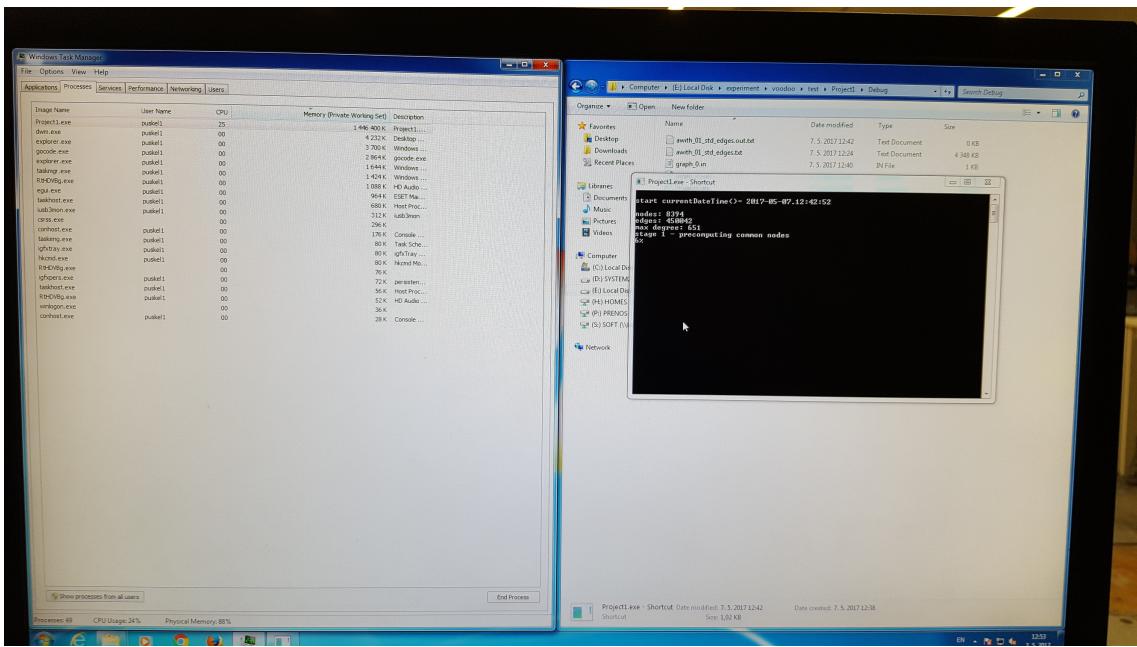
Medzitým, sme sa so zvedavosťou pustili do analýzy nameraných hodnôt spoločne so školiteľom. Pri analyzovaní distribúcií orbít sme však v MATLABe zistili veľký problém: záporné hodnoty. V programe sme používali znamienkový `integer`, dokonca 32-bitový. Ten pretiekol, čo znamenalo, že ani kladné hodnoty neboli garantované nakoľko sme nevedeli kolkokrát počítadlá mohli pretieť... výsledky merania sa nedali použiť. To ako je nám to úprimne lúto nedokážu vyjadriť ani nasledujúce čísla: v I-H3 sme zmerali **39 mozgov**, úlohy sme rozdelili na **23480 podúloh** trvalo to **155h46m30.6447931s**; v I-H6 sme zmerali **1 (najväčší) mozog**, úlohy sme rozdelili na **3732 podúloh** trvalo to **38h24m6.0843975s**.

Okrem plánovaného benchmarku *Orcy*, zdokumentovaného na obrázkoch 4.12, 4.13, 4.14, 4.15 a 4.16, sme teda posledný víkend zmerali ešte raz ten najväčší mozog (súbor awith\_01\_std\_edges.txt) s už opraveným programom používajúcim neznamienkový 64-bitový `integer`. V rámci predvádzania prototypu sme ešte raz zmerali aj ten najmenší mozog (súbor awith\_40\_std\_edges.txt).

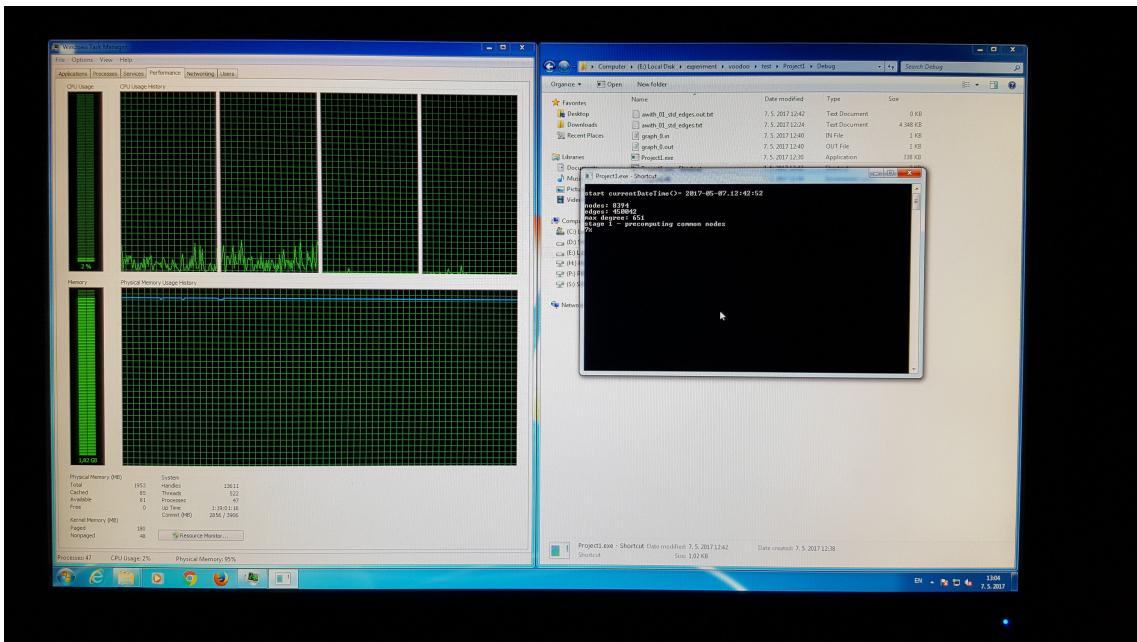
Avšak pre nedostatok času sme už ostatné merania neopakovali, pravdepodobnostné porovnanie sme napokon nevykonali nakoľko len s dvoma mozgami to bolo nezaujímavé. V prípade, že nám to bude umožnené vo výskume budeme pokračovať v rámci diplomovej práce.



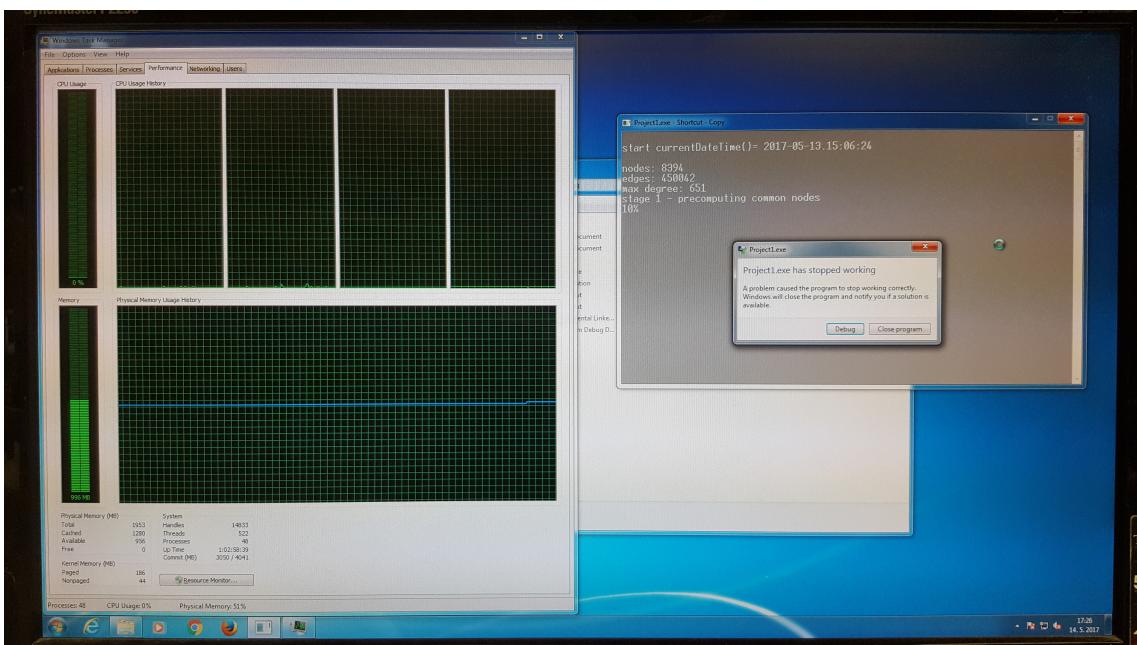
Obr. 4.12: 32-bitová *Orca* v I-H6.



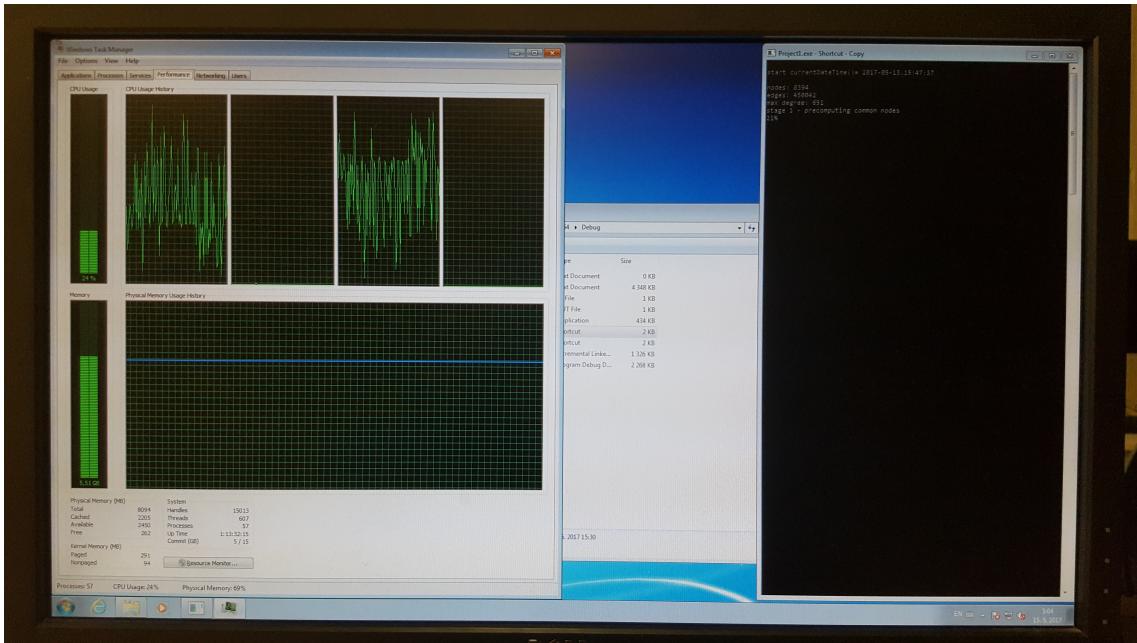
Obr. 4.13: Vidíme, že *Orca* (program v *c++*) velmi neefektívne využíva zdroje. Nedokáže plne zaťažiť procesor, no pamäťová náročnosť je nadštandardne vysoká.



Obr. 4.14: *Orce* nepotrva dlho, kým plne zahltí celú pamäť počítača v I-H6. Zato procesor takmer nie je vyťažený.



Obr. 4.15: V I-H6 sme skúšali otestovať *Orcu* 3krát. Každý pokus skončil rovnako, padnutím programu. Domnievame sa, že konkrétny testovaný súbor je tak veľký, že na konkrétnej zostave aká je v I-H6, nedokáže *Orca* **nikdy** vypočítať dané distribúcie, kvôli nedosatku pamäte RAM a nadmernej pamäťovej náročnosti.



Obr. 4.16: *Orca* sme testovali aj v jej 64-bitovom prevedení v I-H3. Táto verzia vykazovala väčšiu spoloahlivosť, avšak po **37h16m23s** sa dostala iba na **21%** 1. fázy (zo svojich 3 fáz) výpočtu. Žiaľ testovanie sme už potom museli prerušiť kvôli nadchádzajúcej výučbe. *Orca* má bezpochyby lepší (kombinatorický) algoritmus, avšak pamäťová zložitosť je značne vysoká. Náš systém spotrebuje viac elektriny, vyžaduje veľa počítačov, avšak vďaka paralelizmu a distribúcií problému na viacero počítačov sme zadanú konkrétnu úlohu na konkrétnych strojoch dokázali vypočítať rýchlejšie a spoľahlivejšie.

# Kapitola 5

## Výsledky

Vytvorili sme distribuovaný systém na spočítanie orbitovej distribúcie akéhokoľvek jednoduchého neorientovaného grafu. Tento sme následne komplexne otestovali v I-H3 a I-H6 FMFI UK. Opravili sme všetky zistené chyby. Najzávažnejšími boli problémy synchronizácie v kritických sekciách, deadlocky a pretekajúci znamienkový `integer`.

Skúsení programátori vedia, že každá posledná chyba je v skutočnosti tou predposlednou, no my veríme, že náš systém je teraz už naozaj komplexne otestovaný a odladený. Je pripravený na reálny experiment.

Samotné experimentálne zistenie výsledkov orbitovej distribúcie a tiež určenie podobností funkčných sietí mozgu 40 účastníkov Bucknerovho [3] výskumu zostáva predmetom nášho ďalšieho výskumu.

# Kapitola 6

## Záver

Dosiahli sme úspešné vytvorenie distribuovaného systému aj s jeho komplexným otestovaním a testovaním použiteľnosti. Systém je schopný riešiť problém orbitovej distribúcie grafu pre akékolvek jednoduché neorientované grafy aj s počítačmi štandardnej výkonnostnej triedy.

V prípade, že nám bude umožnené vo výskume a vývoji softwaru pokračovať, plánujeme rôzne vylepšenia napríklad:

- refaktorovať celý systém tak, aby orbitové metódy nevracali pole distribúcií, ale hashtabuľku distribúcií `map`,
- vylepšiť štatistiky reportované systémom, konkrétnie modul `stat_results.go`,
- doplniť modul na určenie pravdepodobnostnej zhody štruktúry dvoch grafov na základe ich orbitovej distribúcie,
- umožniť väčšiu variabilitu pri zadávaní úloh, umožniť spočítať iba vopred zadané jednotlivé orbity (aktuálna verzia vyberá z 5 podmnožín všetkých orbitov, ak si vyberieme 1 konkrétnu metódu počítajú sa vždy všetky orbity v podmnožine, aj keď nás zaujíma iba 1 konkrétny, časová zložitosť je vždy rovná zložitosti výpočtu celej skupiny).

V ďalšom výskume by sme radi naplnili aj prvý cieľ, zistenie, či sa Alzheimerova choroba prejavuje v štruktúre funkčných sietí mozgu.

# Literatúra

- [1] Nataša Pržulj, *Biological Network Comparison Using Graphlet Degree Distribution*, Bioinformatics, vol. 23 (2), p. e177, 2007
- [2] McCarthy P, Benuskova L and Franz EA, *The age-related posterior-anterior shift as revealed by voxelwise analysis of functional brain networks*, Frontiers in Aging Neuroscience, vol. 6 (301), pp. 1-14, 2014
- [3] RL Buckner and Snyder, A.Z and Sanders, A.L and Raichle, M.E and JC Morris, *Functional Brain Imaging of Young, Nondemented and Demented Older Adults*, Journal of Cognitive Neuroscience, vol. 12 (2), pp. 24-34, 2000
- [4] Jonathan L. Gross, Jay Yellen, *Graph Theory and Its Applications, Second Edition (Discrete Mathematics and Its Applications)*, Chapman & Hall/CRC, 2005, ISBN 158488505X
- [5] Alexander Stanoyevitch, *Discrete Structures with Contemporary Applications*, Chapman and Hall/CRC, 2011, ISBN 9781439817681
- [6] Douglas B. West, *Introduction to Graph Theory (3rd Edition)*, Prentice Hall, 2007, ISBN 0131437372
- [7] Ralph P. Grimaldi, *Discrete and Combinatorial Mathematics: An Applied Introduction, Fifth Edition*, Pearson, 2003, ISBN 0201726343
- [8] Nataša Pržulj, *Biological Network Comparison Using Graphlet Degree Distribution*, Bioinformatics, vol. 26 (6), p. 853, 2010
- [9] Lucia Budinská, *Hľadanie grafelov vo funkčných sietach mozgu (Diplomová práca)*, Bratislava: Univerzita Komenského, 2016

- [10] *The Computer Language Benchmarks Game*, [Online]. Dostupné na: <http://benchmarksgame.alioth.debian.org/> [Cit. 29. Máj 2017]
- [11] *The Go Programming Language*, [Online]. Dostupné na: <https://golang.org/> [Cit. 29. Máj 2017]
- [12] Tomaž Hočev, Janez Demšar, *Orca, A combinatorial approach to graphlet counting*, [Online]. Dostupné na: <http://www.biolab.si/supp/orca/> [Cit. 21. Február 2017]
- [13] Dror Marcus, Yuval Shavitt, *RAGE – A rapid graphlet enumerator for large networks*, Computer Networks, vol. 56 (2), pp. 810–819, 2012
- [14] Tomaž Hočev, Janez Demšar, *A combinatorial approach to graphlet counting*, Bioinformatics, vol. 30 (4), pp. 559–565, 2014

# Dodatok A

## Prílohy

Príloha je v podobe verejného repozitára <https://github.com/michalpuskel/gdd> avšak jeho obsah prikladáme aj na CD [Cit. 2. Jún 2017] a obsahuje:

- zdrojové kódy aplikácie a skripty použité pri výskume,
- infografiku stromu výpočtu algoritmov „*hrubou silou, ale tak dobre ako sa len dá*“ na počítanie orbitových distribúcií,
- spustiteľné aplikácie (klient, server) na spočítanie distribúcie výskytu jednotlivých orbít grafletov (Windows),
- jednoduchý návod ako aplikácie použiť,
- sadu unit testov,
- súbory obsahujúce namerané hodnoty ako aj záznamy z konkrétnych experimentálnych pokusov,
- celú históriu postupného vývoja tohto projektu, vrátane význačných milestones a postrechov vo forme githubových issues a commentov (CD neobsahuje).