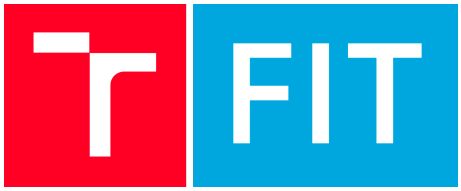
Největší klika v neorientovaném grafu

Lukáš Lev

Semestrální práce
IAL



UIFS
VUT-FIT
28. listopadu 2024

1 Zadání

Náhradní projekt je určen pouze pro studenty, kteří v předmětu IFJ neřeší souběžný projekt (např. studenti FEKT nebo studenti opakující předmět). Tento projekt je týmový a řeší jej trojice nebo čtveřice studentů.

Zadání varianty
Klika grafu je podgraf, který je úplným grafem (=kterýkoliv vrchol kliky je tedy spojen hranou se všemi ostatními vrcholy kliky).
Vytvořte program pro hledání největší kliky v neorientovaném grafu. Pokud existuje více řešení, nalezněte všechna. Výsledky prezentujte vhodným způsobem. Součástí projektu bude načítání grafů ze souboru a vhodné testovací grafy. V dokumentaci uveďte teoretickou složitost úlohy a porovnejte ji s experimentálními výsledky.

Všeobecné informace a pokyny k náhradním projektům
Řešení bude vypracováno v jazyce C a bude přeložitelné (pomocí příkazu make) na serveru eva.fit.vutbr.cz. Všechny zdrojové kódy, hlavičkové soubory, testovací data aj. budou logicky separovány a uloženy v příhodně pojmenovaných podadresářích. Použití nestandardních knihoven není dovoleno. Všechny části zadání varianty jsou nutnou součástí řešení.

Celkové hodnocení projektu sestává z následujících kategorií:

- funkčnost implementace (až 6 bodů),
- projektová dokumentace (až 4 body),
- obhajoba (až 5 bodů).

Řešení zabalené v jediném ZIP archivu je odevzdáváno pouze vedoucím týmu prostřednictvím STUDISu. Závazné pokyny pro vypracování projektové dokumentace a doporučení pro závěrečné obhajoby naleznete v Moodlu v sekci Projekty.

2 Abstrakt

Tento projekt byl proveden podle zadání z kapitoly 1, jež bylo poskytnuto vyučujícím.

Předmětem tohoto projektu je hledání **největší kliky v neorientovaném grafu**, což je jeden z typických problémů v teorii grafů [2]. Tato problematika je krátce popsána v kapitole 1.

Pro hledání největší kliky v neorientovaném grafu byly navrženy dva algoritmy, a to sice

- algoritmus metodou hrubé síly (anglicky *brute force*),
- algoritmus zpětného vyhledávání (anglicky *backtracking*).

Pro každý z těchto algoritmů byla stanovena časová komplexita nejprve teoreticky a následně také experimentálně pomocí jednoduchého programu v jazyce C.

3 Teorie

3.1 Zkoumané objekty

Graf je základní objekt teorie grafů. Skládá se z uzlů (vrcholů) a hran.[3]

Neorientovaný graf je graf, jehož všechny uzly jsou symetricky spojeny neorientovanou hranou.[3]

Úplný graf je neorientovaný graf, pro jehož každou dvojici vrcholů existuje právě jedna neorientovaná hrana.[3]

Klika (anglicky *clique*) je podmnožina vrcholů neorientovaného grafu. Tato podmnožina tvoří úplný graf.[1]

3.2 Teorie použitých algoritmů

Tato kapitola se věnuje teoretickým vlastnostem použitých algoritmů metody hrubou silou a metody zpětného vyhledávání. Především pak jejich časovým a prostorovým složitostem. Implementace obou těchto algoritmů je kritická pro jejich složitost. Touto problematikou se zabývá kapitola 4.

Kruciální pro tuto kapitolu je reprezentace grafu, kterou implementace používá. Jelikož graf je reprezentován pomocí matice sousednosti (kapitola 4.1) je velice těžké vyhnout se při řešení problému hledání největší kliky exponenciální složitosti.

3.2.1 Algoritmus metodou hrubé síly

Časová složitost: Algoritmus iteruje každým možným podgrafem původního grafu o velikosti (počtu uzlů) n a u každého z těchto podgrafů provádí zkoušku, zda jsou klikami, a to tak, že kontroluje každý uzel tohoto podgrafu. Oba tyto cykly mají vliv na časovou složitost algoritmu.

Protože pro konstrukci každého podgrafu musíme určit stav každého uzlu jako 1 (náleží podgrafu), nebo 0 (nenáleží podgrafu) - tedy vybrat ze dvou stavů, určíme počet všech možných podgrafů jako 2^n. Dále stanovíme počet iterací vycházející z kódu (implementace je rozebrána v kapitole 4). Protože kód zabraňuje kontrole duplikátních dvojic uzlů v podgrafu, bude nejhorším možným počtem iterací kontrola pro podgraf o velikosti n. Z něj se vybere právě $\binom{n}{2} = \frac{n(n-1)}{2}$.

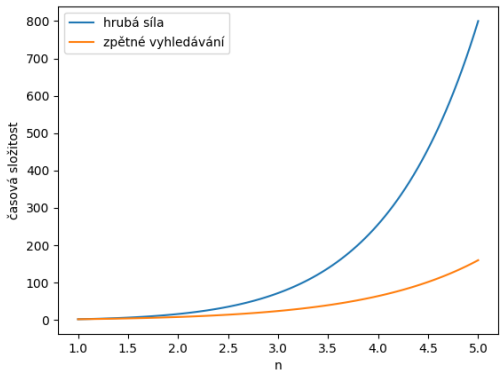
Časovou složitost tedy určíme zanedbáním konstant pro tyto vnořené cykly jako $\mathbf{O(2^n \cdot n^2)}$.

Prostorová složitost: podobně jako při určování časové složitosti vycházíme z toho, že pro každý možný podgraf ukládá implementace až n možných klik (například pokud není mezi uzly žádná hrana). V nejhorším případě je tedy uloženo pro 2^n cyklů n prvků. Prostorová složitost je tedy $\mathbf{O(2^n \cdot n)}$.

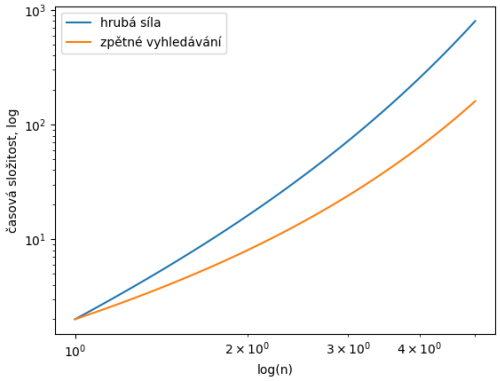
3.2.2 Algoritmus metodou zpětného vyhledávání

Časová složitost: postup pro určení časové složitosti je obdobný jako pro algoritmus metodou hrubé síly. Největší možný počet vytvořených podgrafů je 2^n, pokud by bylo pomocí zpětného vyhledávání natrefeno na výslednou kliku až jako poslední možný podgraf. Tentokrát však vnořený cyklus pro kontrolu zachování kliky iteruje nejhůře n-krát. Proto je časová složitost určena jako $\mathbf{O(2^n \cdot n)}$.

Prostorová složitost: pokud dojde k maximální rekurzi, vnoření proběhne právě n-krát. Pro ukládání právě analyzované kliky je zapotřebí pole o velikosti n. Obě tyto veličiny jsou však nezávislé na ukládání největších klik. V nejhorším případě je v grafu 2^n podgrafů o velikosti až n. Proto uvažujeme, že prostorová složitost je $\mathbf{O(2^n \cdot n)}$.



Obrázek 1: Grafické vyjádření horní hranice časových složitostí obou algoritmů.



Obrázek 2: Grafické vyjádření horní hranice časových složitostí obou algoritmů s logaritmickým škálováním na osách.

4 Implementace v jazyce C

Podle zadání (kapitola 1), byly oba algoritmy vypracovány v jazyce C. Další rysy implementace, které vycházejí ze zadání, se týkají zavedených datových struktur (kapitola 4.1). Implementace obou algoritmů vykazují požadované chování a v případě výskytu několika klik o maximální velikosti nacházejí všechny tyto výsledky. Zároveň, nenachází-li se v grafu žádná hrana, jsou nalezeny všechny vrcholy jako největší kliky.

4.1 Reprezentace neorientovaného grafu

Podle zadání (kapitola 1) je třeba vytvořit takovou datovou strukturu neorientovaného grafu, kterou lze snadno reprezentovat záznamem do jednoduchého souboru.

Jednou z možných variant je **matice sousednosti**. Tato matice je čtvercová a, jelikož v rámci implementace neuvažujeme hrany uzlů vedoucí na sebe sama, hlavní diagonála této matice je nulová a matice je podle této diagonály symetrická. Pro ostatní prvky platí, že vyjadřují přítomnost hrany mezi uzly s indexem shodným s řádkem či sloupcem matice.

Následující rovnice uvádí názorný příklad matice sousednosti (grafická reprezentace této matice je na obr. 3). Prvky této matice jsou hrany mezi vrcholy označenými indexem. Tedy například prvek $h_{04} = 1$ tvrdí, že mezi uzly 0 a 1 se nachází hrana, zatímco mezi uzly 0 a 1 nikoliv ($h_{01} = 0$).

$$\mathrm{M_s} = \begin{pmatrix} h_{00} & h_{01} & h_{02} & h_{03} & h_{04} \\ h_{10} & h_{11} & h_{12} & h_{13} & h_{14} \\ h_{20} & h_{21} & h_{22} & h_{23} & h_{24} \\ h_{30} & h_{31} & h_{32} & h_{33} & h_{34} \\ h_{40} & h_{41} & h_{42} & h_{43} & h_{44} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \tag{1}$$

Zdrojový kód pro implementaci této matice se nachází v souboru `graph.c`. V něm je definována struktura `graph` popsaná níže.

```
typedef struct {
    int size;  // velikost matice (odpovida poctu uzlu)
    int** matrix;  // 2D pole pro ulozeni prvku matice
} graph;
```

Listing 1: Definice struktury neorientovaného grafu.

Soubor `graf.c` také obsahuje řadu funkcí vztahujících se k těmto grafům. Těmito funkcemi jsou:

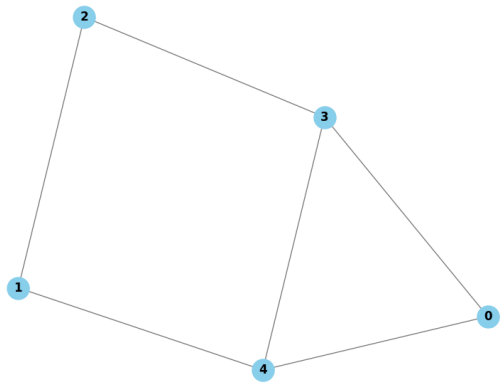
- `graph* graph_init(int size)` pro inicializaci prázdného grafu s vhodnou velikostí,
- `void graph_delete(graph* g)` pro smazání grafu a uvolnění jeho paměti,
- `int graph_read_size(const char* filename)` pro přečtení velikosti grafu (počtu uzlů) v matici sousednosti ze souboru,
- `int graph_read(graph* g, const char* filename)` pro přečtení matice sousednosti ze souboru, vlastnosti grafu jsou uloženy do vstupní proměnné `g` (nevyužívá funkci `graph_read_size`),
- `int graph_write(graph* g, const char* filename)` pro zápis matice sousednosti do souboru,
- `void graph_print(graph* g)` pro vytištění grafu do terminálu

Soubory, které slouží pro ukládání grafu, jsou označeny příponou `.gh` a matici sousednosti uchovávají v následujícím formátu, kde první řádek reprezentuje velikost matice a ostatní řádky informace o jejich prvcích:

```
5
0 0 0 1 1
0 0 1 0 1
0 1 0 1 0
1 0 1 0 1
1 1 0 1 0
```

Listing 2: Zápis matice v `.gh` souboru.

Tato matice je reprezentována také graficky na obrázku 3.



Obrázek 3: Grafická reprezentace neorientovaného grafu z kapitoly 4.1.

Bližší popis implementovaného kódu je součástí komentářů v souboru `graph.c`.

4.2 Implementace algoritmu metody hrubou silou

Zdrojový kód pro algoritmus hledání největší kliky v neorientovaném grafu metodou hrubé síly (anglicky *brute force*) je obsahem souboru `algorithms/bruteforce.c`. Kód operuje se strukturou definovanou v kapitole 4.1. Samotný soubor obsahuje podrobné komentáře, a tak je shrnutí implementace v této kapitole pouze zběžné.

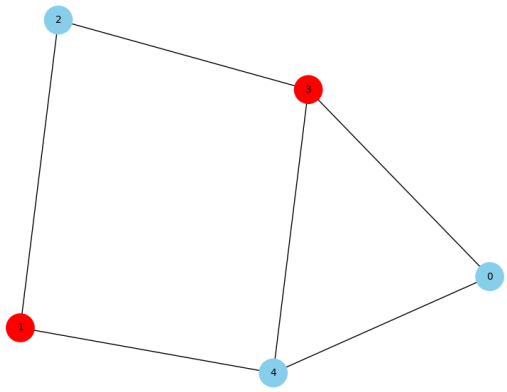
Protože matice sousednosti obsahuje pouze prvky s hodnotou 1 nebo 0, byla zvolena reprezentace vybrané podmnožiny vrcholů pomocí binární masky. Na tu je referováno celým číslem `int subset`,

které po převodu do binární soustavy určuje, které uzly jsou vybrány. Indexace se shoduje s indexací uzlů v matici.

Například pro matici definovanou v souboru v ukázce kódu 2 použijeme masku `subset` určenou číslem 10. Pro masku `subset` tak platí:

$$10_{(10)} = 01010_{(2)} \implies u_0u_1u_2u_3u_4 \tag{2}$$

Proto jsou maskou vybrány vrcholy na indexech 1 a 3 graficky reprezentovány na obrázku 4. Pro vytvoření všech podgrafů vstupního grafu definovaného maticí sousednosti stačí iterativně inkrementovat hodnotu masky od nuly až do hodnoty $\mathbf{2^n}$, kde n je počet uzlů v grafu. Při iteraci těmito podgrafy stačí sledovat, zda jsou klikami, a pokud ano, pak také jejich velikost. Největší kliky jsou uchovávány v proměnné `int** largest_cliques` ve funkci `void bruteforce(graph* g)`.



Obrázek 4: Označení uzlů neorientovaného grafu maskou o hodnotě 10.

Soubor `algorithms/bruteforce.c` obsahuje následující funkce:

- `int is_clique_bruteforce(graph* g, int subset)` pro kontrolu, zda podgraf označený maskou `subset` je klikou (zda je matice, krom diagonály, naplněna hodnotami 1),
- `void bruteforce(graph* g)` pro iterativní hledání největší kliky. Tato funkce vytiskne všechny největší nalezené kliky do terminálu.

4.3 Implementace algoritmu metody zpětného vyhledávání

Zdrojový kód pro hledání největší kliky v neorientovaném grafu pomocí metody zpětného vyhledávání je obsažen v souboru `algorithms/backtracking.c`. V něm je pro tuto metodu použito následujících funkcí:

- `int is_clique_backtracking(graph* g, int* clique, int clique_size, int vertex)` pro kontrolu, zda přidání uzlu (neboli vrcholu, tedy `vertex`) zachová vlastnost kliky podgrafu `g`,
- `void find_clique_backtracking(graph* g, int* current_clique, int clique_size, int*** largest_cliques , int* max_size, int* clique_count, int start)` pro rekurzivní hledání největší kliky,
- `void backtracking(graph* g)` pro správu proměnných pro obě výše zmíněné funkce a jejich tisk do konzole.

Bližší informace o fungování kódu jsou uvedeny v komentářích souboru `algorithms/backtracking.c`. Následující popis je pouze stručné shrnutí.

Funkce `void backtracking(graph* g)`, jež je volána z vnějšího prostředí, spravuje proměnné `int* current_clique`, která je polem vrcholů tvořících právě analyzovanou kliku, `int** largest_cliques`, která je polem ukazatelů na pole vrcholů tvořících největší nalezené kliky, `int max_size`, která je velikost největší nalezené kliky, a `int clique_count`, která slouží jako počítadlo největších nalezených klik.

Tyto proměnné jsou použity při volání `find_clique_backtracking(g, current_clique, 0, &largest_cliques, &max_size, &clique_count, 0)`. V této funkci se po ověření vzniku nové kliky (funkcí `is_clique_backtracking`) rekurzivně volá opět funkce `find_clique_backtracking` s aktualizovanými parametry. Pro každou iteraci je ověřeno, zda právě analyzovaná klika má být uložena jako největší.

Princip zpětného vyhledávání je zde zaopatřen vynořením z rekurze a přepisem dříve zkoumaných `current_clique`. Díky tomu je na konci implementace zapotřebí uvolňovat pouze paměť alokovanou pro všechna nalezená řešení a `current_clique`.

```
1  void find_clique_backtracking(graph* g, int* current_clique, int clique_size, int***
       largest_cliques, int* max_size, int* clique_count, int start) {
2      if (clique_size > *max_size) {
3          *max_size = clique_size;
4          for (int i = 0; i < *clique_count; i++) {
5              free((*largest_cliques)[i]);
6          }
7          free(*largest_cliques);
8          *largest_cliques = NULL;
9          *clique_count = 0;
10     }
11     if (clique_size == *max_size) {
12         *largest_cliques = realloc(*largest_cliques, (*clique_count + 1) * sizeof(int*))
               ;
13         (*largest_cliques)[*clique_count] = malloc(clique_size * sizeof(int));
14         for (int i = 0; i < clique_size; i++) {
15             (*largest_cliques)[*clique_count][i] = current_clique[i];
16         }
17         (*clique_count)++;
18     }
19     for (int i = start; i < g->size; i++) {
20         if (is_clique_backtracking(g, current_clique, clique_size, i)) {
21             current_clique[clique_size] = i;
22             find_clique_backtracking(g, current_clique, clique_size + 1, largest_cliques
                   , max_size, clique_count, i + 1);
23         }
24     }
25 }
```

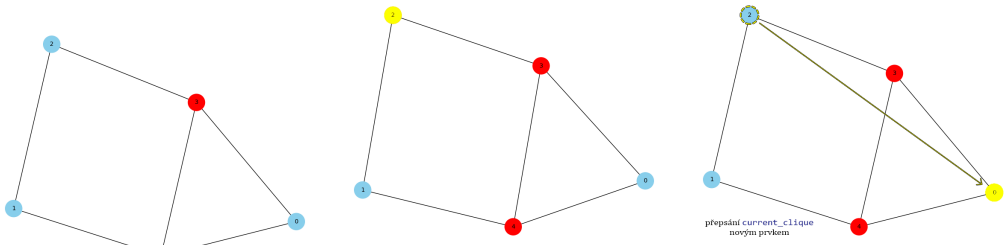
Listing 3: Zjednodušený kód pro rekurzivní funkci využitou při implementaci metody zpětným vyhledáváním.

Jednoduchá grafická reprezentace přepisu prvků podgrafu v proměnné `current_clique` je znázorněna na obrázku 5. Důležitou poznámkou však je, že pořadí zpracování indexů se nemusí shodovat s pořadím v programu.

5 Experiment

Tato kapitola se zabývá experimentálním ověřením vlastností implementace algoritmů v jazyce C (kapitola 4) pro jejich následné porovnání s teoretickými předpoklady vznesenými v kapitole 3.2.

Pro tyto účely byl sepsán kód souboru `experiment.c`. Bližší popis tohoto kódu je shrnut v komentářích souboru. Pro účely této kapitoly je důležitá funkce `time_comparison_experiment`. Její zjednodušený kód je uveden v úryvku 4. Z tohoto úryvku vyplývá, že může být volán experiment pro různě husté



(a) Podgraf s uzly 3 a 4.

(b) Přidání uzlu 2 nezachová vlastnost kliky podgrafu.

(c) Přepis falešného uzlu kliky dalším podezřelým.

Obrázek 5: Ilustrace k přepisu uzlů v proměnné `current_clique` pro hledání zpětným vyhledáváním (pořádí uzlů nemusí odpovídat pořadí v programu).

grafy se zvolenými velikostmi pro oba, či jen jeden z algoritmů.

```
void time_comparison_experiment() {
    FILE* file = fopen("experiment.csv", "w");
    fprintf(file, "size,density,bruteforce,backtracking\n");
    for (int size = 5; size <= 30; size += 1) {
        for (double density = 0.5; density <= 0.5; density += 0.3) {
            graph* g = generate_random_graph(size, density);

            // bruteforce
            double bruteforce_time = measure_execution_time(bruteforce, g);

            // backtracking
            // double backtracking_time = measure_execution_time(backtracking, g);

            // fprintf(file, "%d,%.2f,%.6f,%.6f\n", size, density, bruteforce_time,
                backtracking_time);  // pro oba algoritmy
            fprintf(file, "%d,%.2f,%.6f,na\n", size, density, bruteforce_time);  //
                pouze bruteforce
            // fprintf(file, "%d,%.2f,na,%.6f\n", size, density, backtracking_time);  //
                 pouze backtracking

            graph_delete(g);
        }
    }
    fclose(file);
}
```

Listing 4: Zjednodušený kód hlavní funkce souboru `experiment.c` nasteven pro generaci grafů velikosti 5 až 30 s hustotou 0,5 pro algoritmus metody hrubé síly.

Důležitou poznámkou je, že funkce `measure_execution_time`, která měří délku časového intervalu potřebného pro nalezení všech řešení, používá standardní knihovnu `time.h` a v ní funkci `clock`. Přesnost této metody závisí na čase využitém procesorem, předpokládáme rozlišení metody větší než μs. Proto právě μs byly stanoveny jako rozlišení experimentů.

Kód souboru `experiment.c` byl spouštěn pro následující podmínky a algoritmy, kde n vyjadřuje počet uzlů, ρ hustotu hran v matici (viz kapitolu 4.1):

1. $\forall n \in \langle 30; 300\rangle \cap \mathbb{N}$, $\rho = 0,5$, algoritmus metody zpětného vyhledávání
2. $\forall n \in \langle 5; 30\rangle \cap \mathbb{N}$, $\rho = 0,5$, algoritmus metody hrubé síly
3. $\forall n \in \langle 5; 59\rangle \cap \mathbb{N}$, $\forall \rho \in \bigcup_{k=1}^{9} 0,1 \cdot k$, algoritmus metody zpětného vyhledávání
4. $\forall n \in \langle 5; 92\rangle \cap \mathbb{N}$, $\forall \rho \in \bigcup_{k=1}^{9} 0,1 \cdot k$, algoritmus metody hrubé síly

5.1 Srovnání závislosti časové náročnosti algoritmů na počtu uzlů n

Tato podkapitola se zabývá výsledky z experimentů výše označených jako 1. a 2. Díky vhodnému nastavení kódu souboru `experiment.c` byla získána data délek časových intervalů potřebných k nalezení řešení úlohy hledání největší kliky v závislosti na počtu uzlů n vstupního neorientovaného grafu. Závislosti pro oba tyto algoritmy jsou vyneseny na obrázku 6. Protože závislost projevuje exponenciální charakter a výstup mění řád mnohem rychleji než vstup, byla tato data vynesena i v logaritmickém měřítku, to je zachyceno na obrázku 7.

5.2 Srovnání závislosti časové náročnosti algoritmů na počtu uzlů n a hustotě matice sousednosti

Tato podkapitola se zabývá výsledky z experimentů výše označených jako 3. a 4. Na základě těchto experimentů byly vykresleny dvě teplotní mapy srovnávající vliv počtu uzlů n a hustoty matice sousednosti na časový interval potřebný k nalezení všech řešení. Protože výsledná data vykazují exponenciální chování, byl vynesen tentýž graf také v logaritmickém měřítku. Na něm byly nulové body zanedbány. Zmíněné grafy jsou na obrázcích 8 a 9.

5.3 Vyhodnocení experimentu

Experimenty srovnávající délku časového intervalu pro vyřešení úlohy v závislosti na velikosti matice sousednosti n mezi algoritmy, jež byly provedeny s hustotou této matice o hodnotě $\frac{1}{2}$, prokázaly předpokládané chování (podle teoretických předpokladů z kapitoly 3.2), a to sice, že algoritmus metody hrubé síly je z dvou porovnávaných více časově náročný.

Je patrné, že se oba algoritmy chovají exponenciálně, což vychází z vlastností implementace matice sousednosti (kapitola 4.1). Rozdíl je však v horní hranici časových složitostí. V kapitole 3.2 byl vznesen předpoklad, že metoda hrubé síly má tuto funkci n-krát větší než metoda zpětného vyhledávání. I experiment potvrzuje tuto vlastnost.

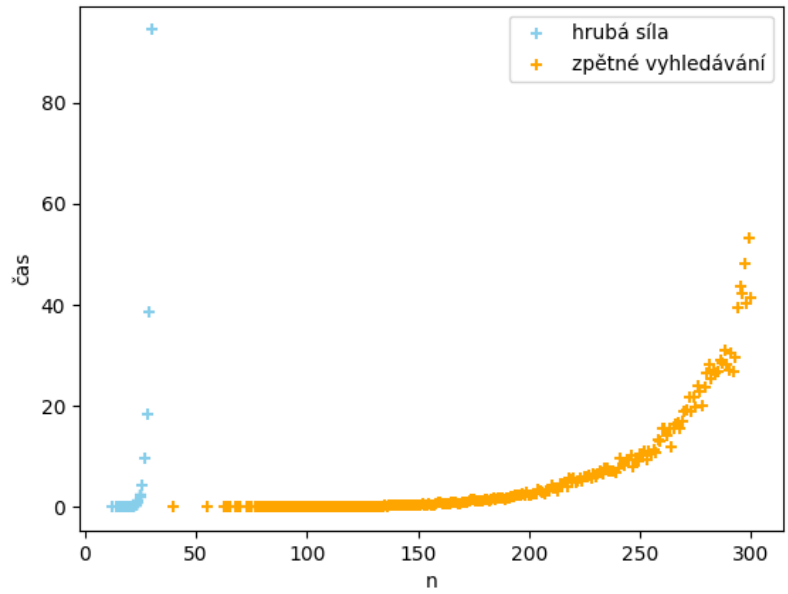
Další metoda, kterou by bylo možné přesněji ověřit shodu s předpokladem časové složitosti, by bylo vyjádřit limitu poměru logaritmizovaných časových intervalů. To vyplývá z následujících rovnic:

$$\begin{aligned} \text{Pro hrubou sílu: } & \log\left(n^2 \cdot 2^n\right) = 2\log(n) + n\log(2) = f(n) \\ \text{Pro zpětné vyhledávání: } & \log\left(n \cdot 2^n\right) = \log(n) + n\log(2) = g(n) \end{aligned} \tag{3}$$

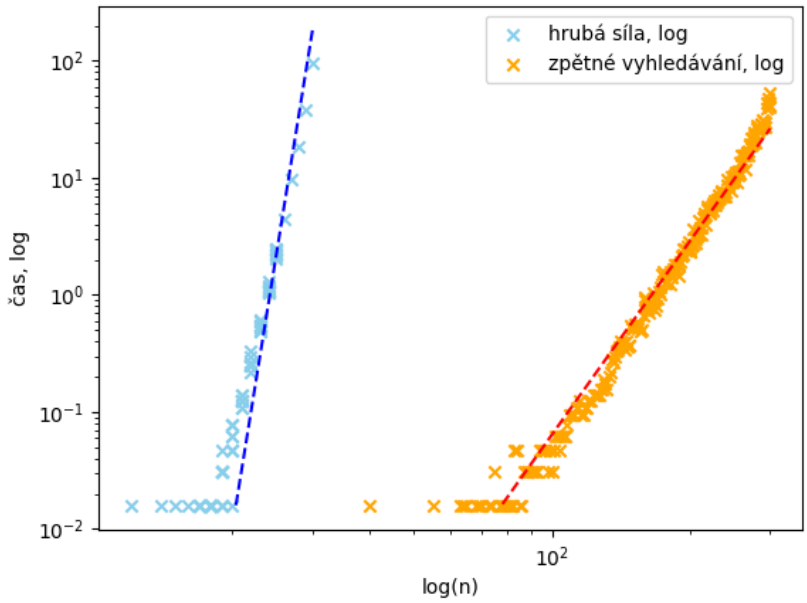
Protože se výrazy liší pouze o koeficient 2 u jednoho ze sčítanců, jakýsi pomyslný vliv tohoto koeficientu bude s $n \to \infty$ klesat, a tak bude čitatel i jmenovatel nabývat téže hodnoty. Tento jev je shrnut v rovnici 4 a obrázku 10.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1 \tag{4}$$

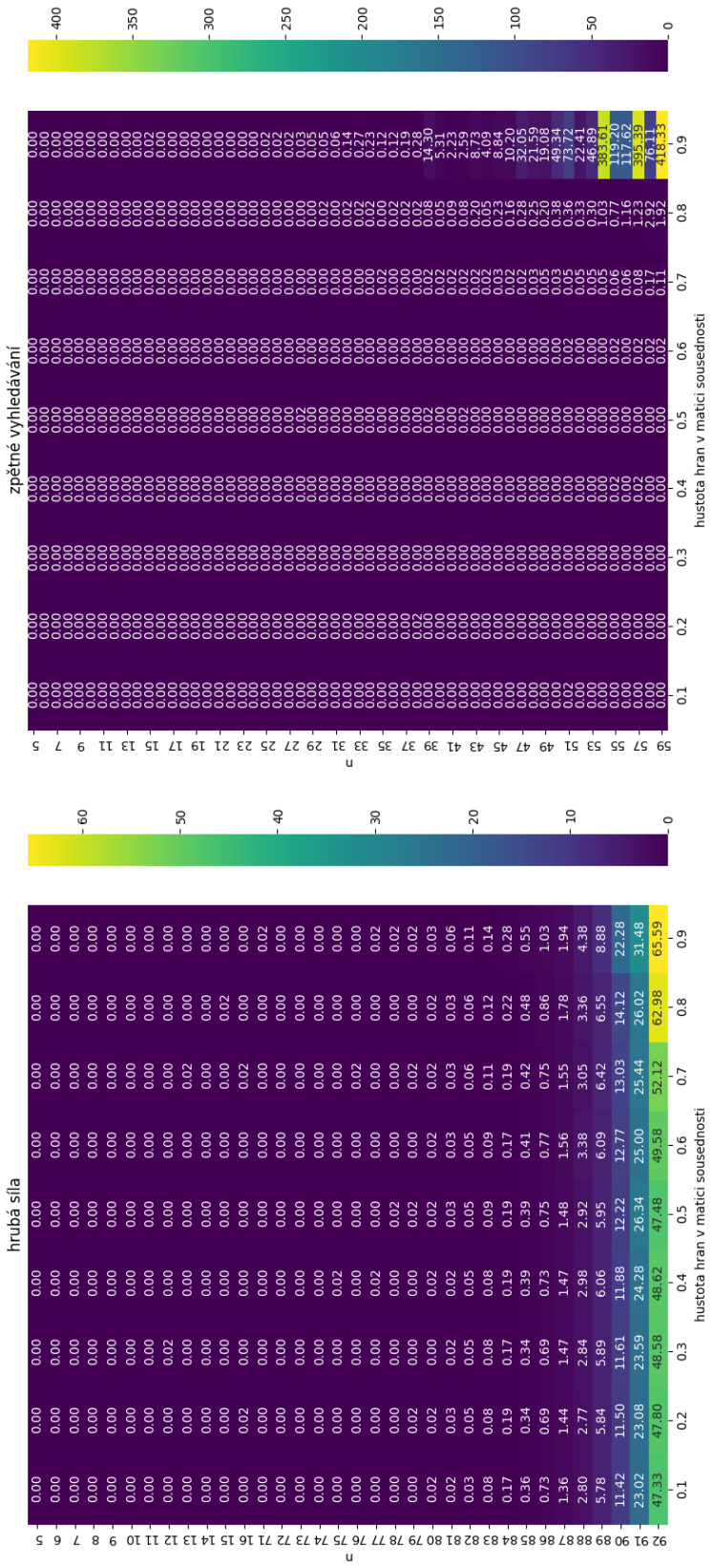
Toto srovnání však v rámci experimentu neproběhlo, jelikož vybraná implementace (viz kapitolu 4) má kvůli vysoké prostorové složitosti (viz kapitolu 3.2) problémy s velkými maticemi a pro hustotu matice 0,5 je tedy problematické najít takovou velikost této matice, pro níž by nedetekovatelně rychlý algoritmus zpětného vyhledávání byl vykonán v měřitelném čase spolu s paměťově náročným algoritmem hrubé síly.
V rámci experimentů 3. a 4. bylo na základě dat na obrázcích 8 a 9 zjištěno, pro jaké neorientované grafy jsou navržené algoritmy vhodné. Protože graf pro algoritmus metody hrubé síly vykazuje relativně oproti druhému algoritmu nezávislost na hustotě matice sousednosti, můžeme učinit závěr, že je tento algoritmus univerzálnější, zatímco algoritmus metody zpětného vyhledávání je efektivní, pokud není matice zaplněna s hustotou $< 0,7$. Pro hustotu 0,9 dokonce velmi silně zaostává za algoritmem hrubé síly.



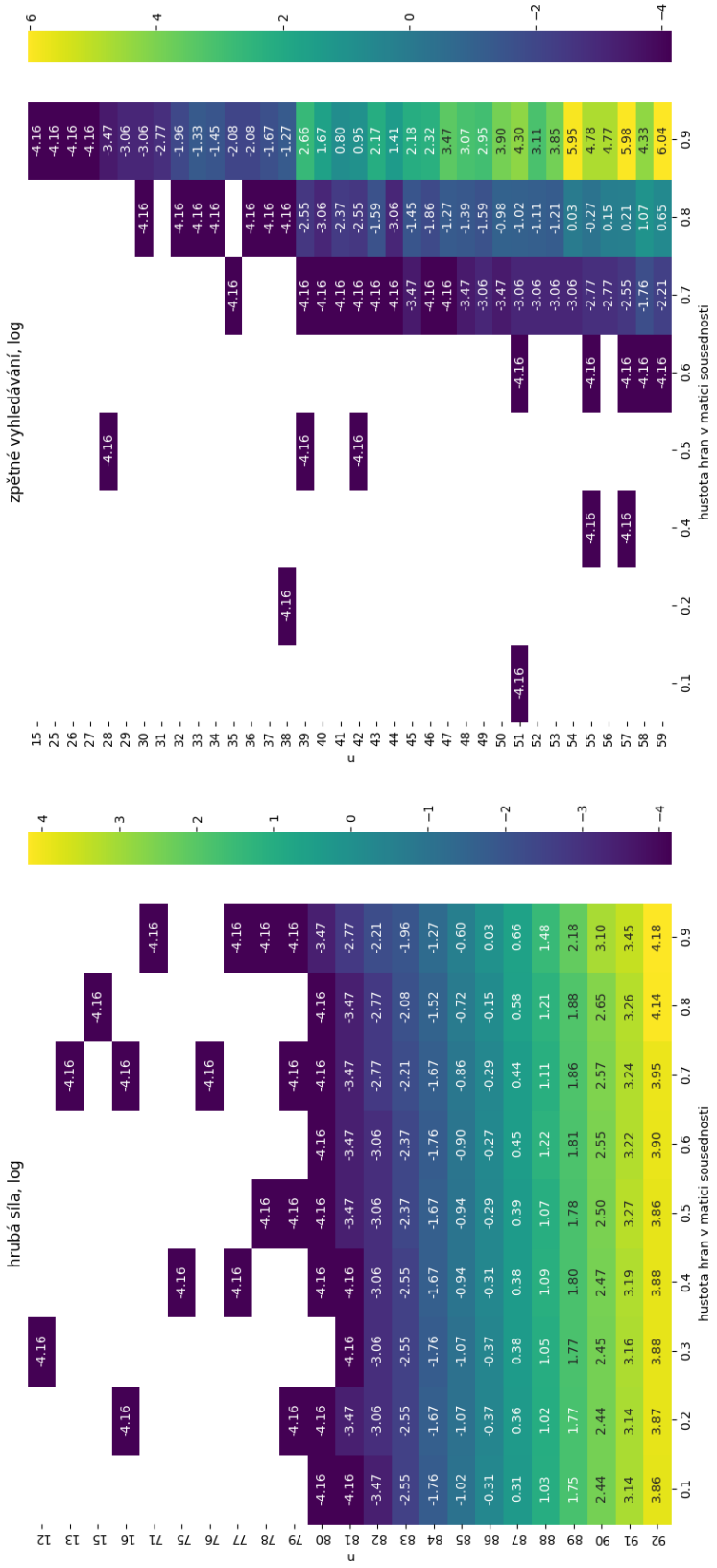
Obrázek 6: Výsledek experimentálního měření závislosti délky časového intervalu pro dokončení úlohy na počtu uzlů vstupního grafu pro oba algoritmy.



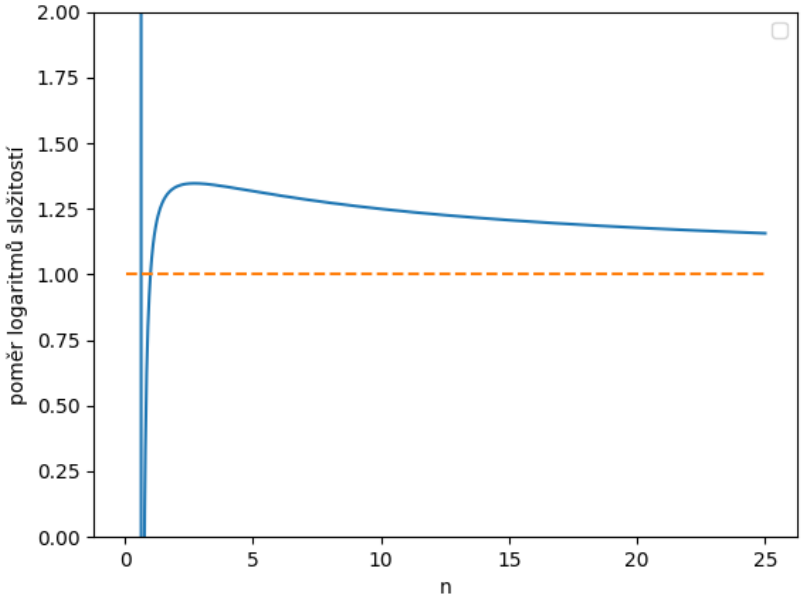
Obrázek 7: Výsledek experimentálního měření závislosti délky časového intervalu pro dokončení úlohy na počtu uzlů vstupního grafu pro oba algoritmy s logaritmickým škálováním na osách.



Obrázek 8: Graf závislosti časové náročnosti algoritmů na počtu uzlů n a hustotě matice sousednosti.



Obrázek 9: Graf závislosti časové náročnosti algoritmů na počtu uzlů n a hustotě matice sousednosti v logaritmickém měřítku.



Obrázek 10: Závislost poměru horních hranic logaritmů složitostí obou algoritmů na velikosti matice sousednosti n.

6 Závěr

Byl vytvořen jednoduchý program v jazyce C, který implementuje dva způsoby hledání všech největších klik v neorientovaném grafu. Těmito způsoby jsou použití metody hrubé síly a metody zpětného vyhledávání. Implementace těchto metod je shrnuta v kapitole 4. V téže kapitole je implementace neorientovaných grafů maticí sousednosti.

Pro oba vybrané algoritmy byly stanoveny časové a prostorové složitosti teoreticky v kapitole 3.2. Teoretické výroky byly nakonec experimentálně ověřeny pomocí programu v jazyce C. Postup i výsledky tohoto měření jsou uvedeny v kapitole 5. V jejich dvou sekcích jsou pak stanoveny závěry rozebírající časovou náročnost a vhodnou aplikaci každé metody.

Kapitola 5 pak také nastiňuje možnosti další optimalizace či metody přesnější analýzy zkoumaných dat, které nejsou obsahem této práce.

Reference

[1] Azizullah Paeyndah and Adela Murid. The hard and easy way to determine clique (a comparative review). *International Journal of Mathematics and Physical Sciences Research*, 2021.

[2] Jue Xue Panos Pardalos. The maximum clique problem. *Journal of Global Optimization*, 1994.

[3] Radomír Perzina. Základní pojmy z teorie grafů. `https://is.slu.cz/el/opf/zima2021/INMBPOAE/3113831/Prednaska_9.pdf`. [Online 24. 11. 2024].

Listings

Seznam obrázků