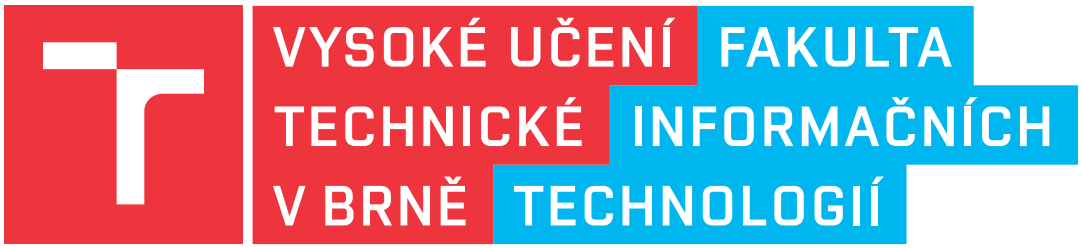


Grafové algoritmy – Projekt

Porovnání přesného a aproximačního algoritmu

Tomáš Dvořák (xdvora3r)

Zdeněk Lapeš (xlapes02)

12. prosince 2024

1 Úvod

Cílem projektu je porovnat dvě řešení jednoho algoritmu s časovou složitostí alespoň $\mathcal{O}(n^3)$, který jsme si měli sami vybrat. S kolegou jsme zvolili Floyd-Warshallův algoritmus, který slouží pro hledání nejkratších cest ze všech uzlů do ostatních uzlů. Tím pádem je nutné nejen generovat graf, ale také do hran zakomponovat váhovou funkci. Způsob implementace grafu s váhami je podrobnější popsán v kapitole 3.

2 Srovnání algoritmů

Na začátek je nutné porozumět složitosti algoritmů a detailnější implementace. Pro exaktní algoritmus jsme zvolili jednoduchý trojnásobný for cyklus, který je do detailu popsán na stránce GeeksforGeeks[1]. Kódová implementace algoritmu zjednodušeně vypadá takto:

Algorithm 1: Exaktní Floyd-Warshall

```
for i in range (n) do
    for j in range (n) do
        for k in range (n) do
            dist[i, j] = min(dist[i, j], dist[i, k] + dist[k, j])
```

Z tohoto kódu je patrná časová složitost algoritmu, která je $\mathcal{O}(n^3)$. Procházíme všechny vrcholy (cyklus *i*) a následně páry vrcholů (cykly *j* a *k*), přičemž pro každý pár aktualizujeme vzdálenost mezi vrcholy podle pravidla: pokud je cesta přes vrchol *k* kratší než přímá cesta mezi *j* a *k*, aktualizujeme vzdálenost. Díky tomu lze postupně zahrnout všechny možné prostřední vrcholy (*k*) do analýzy nejkratších cest. Tím je také zajištěno, že výsledná vzdálenost mezi libovolnými dvěma vrcholy odpovídá skutečně nejkratší možné cestě v grafu. A teď tedy pro srovnání aproximační algoritmus:

Algorithm 2: Aproximační Floyd-Warshall

```
for _ in range (n * iteration_modifier) do
    k = numpy.random.randint (0, n)
    for i in range (n) do
        for j in range (n) do
            dist[i, j] = min(dist[i, j], dist[i, k] + dist[k, j])
```

Už na první pohled je zřejmé, že se provádí náhodná procházka v cyklu *k*, kde je v každé iteraci vybrán vrchol (pseudo)náhodně jako prostřední vrchol - `intermediate_node`. Toto řešení usnadňuje výpočet matice, ovšem za cenu přesnosti. Parametr `intermediate_node` pracuje v rozmezí 0-1, kde např. 0.5 symbolizuje polovinu procházených vrcholů. Detailnější popis parametru naleznete v sekci 5.1.

Tímto můžeme vytvořit předpoklad, že čím větší parametr `intermediate_node` zadáme, tím více se budeme blížit „ideálnímu“ Floyd-Warshallovu algoritmu. Tím pádem také můžeme očekávat, že dostaneme tolika násobné zrychlení (např. pro 0.5 by odpovídalo dvojnásobné zrychlení). Složitost bychom tedy mohli přepsat následovně:

$$\mathcal{O}(\frac{|V|}{\texttt{intermediate\_node}} * |V^2|)$$

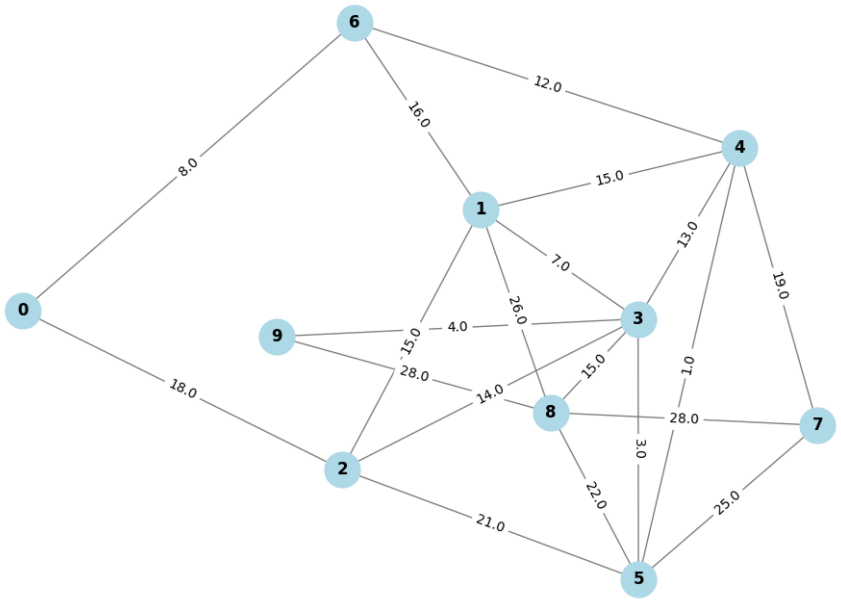
Tato složitost je amortizovaně stále $\mathcal{O}|V^3|$, ale testy ukázaly, že toto řešení není tak špatné jak se může na první pohled zdát.

[1] https://www.geeksforgeeks.org/floyd-warshall-algorithm

3 Generování grafů

Ještě před testováním bylo nutné vytvořit grafy samotné. Grafy generuje skript `graph_generator.py` a podporuje i vizuální zobrazení grafů. Pro hustší grafy (20 vrcholů a více) vizualizace nefunguje nejlépe. Váhy hran se překrývají, občas vrcholy překrývají i samotné váhy, a obecně se graf do okna nevejde. Vizualizace proto slouží jenom jako orientační zobrazení grafu, není nijak nutná. Skript je plně modulární, tedy je možné zvolit různý počet vrcholů, maximální váhu hrany, pravděpodobnost hran i náhodné semínko pro reprodukci grafů. Například tedy lze vytvořit graf:

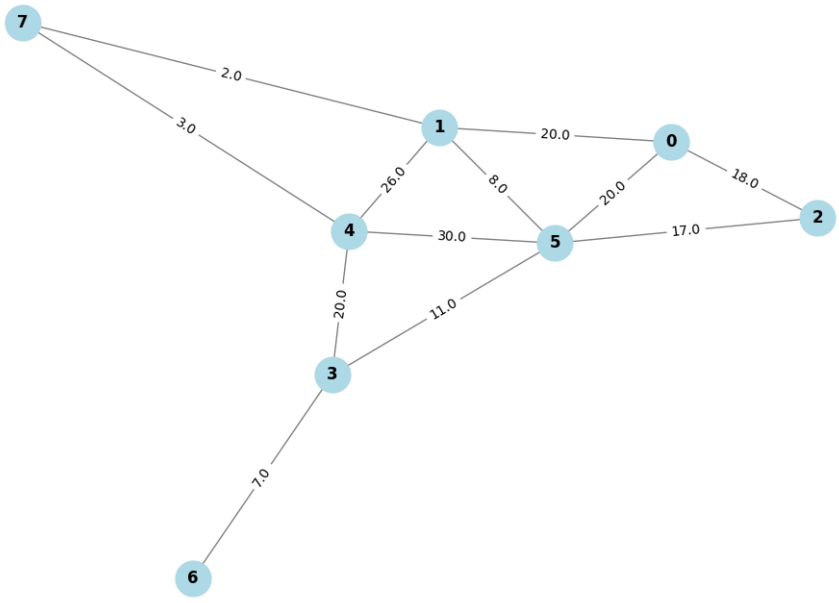
```
python3 ./src/graph_generator.py --nodes 10 --probability 0.4 --
   max_weight 30 --seed 101 --output_txt datasets/graph-1.txt --
   output_png datasets/graph-1.png --visualize
```



Obrázek 1: Ukázka grafu s deseti vrcholy

Na tomto grafu lze vidět jednotlivé parametry generování. Celkem máme deset vrcholů, ne všechny vrcholy mají hrany do všech vrcholu, maximální váha hrany je až 30 a díky semínku lze tento graf generovat znovu. Obrázek 2 ukazuje, jak pouhá změna semínka obmění celý graf.

```
python3 ./src/dataset_generator.py --nodes 10 --probability 0.4 --
   max_weight 30 --seed 102 --output_txt datasets/graph-2.txt --
   output_png datasets/graph-2.png --visualize
```



Obrázek 2: Ukázka grafu s deseti vrcholy ale vytvořený jiným semínkem

Jediná úprava je v semínku z hodnoty **100** na **101** a graf je úplně odlišný. Rozdíl vidíme především v počtu vrcholů. I přes vynucení vygenerování grafu s deseti vrcholy, kvůli pravděpodobnosti nebyly nevygenerovány hrany, které by spojovaly dané vrcholy. Takto může být vygenerovaný obecně jakýkoli graf.

4 Testování a výsledky

Teď tedy k té nejzajímavější části. Pro masové testování jsme volili datové sady o velikosti 200 a 300 vrcholů a pro tvorbu menších sad jsme volili velikost 1000 vrcholů. Testování probíhalo tak jak popisuje kapitola 5. Veškeré testy a parametry jsou reprodukovatelné, stačí zadat správné zdrojové příkazy. Pro přehlednost budou spouštěcí příkazy vynechány a zaměříme se především na výsledné informace z histogramů. U malých sad lze velmi jednoduše vyčíst jaké parametry byly použity, tyto hodnoty byly naschvál ponechány v obrázcích. Masové testy jsou složitější na reprodukci.

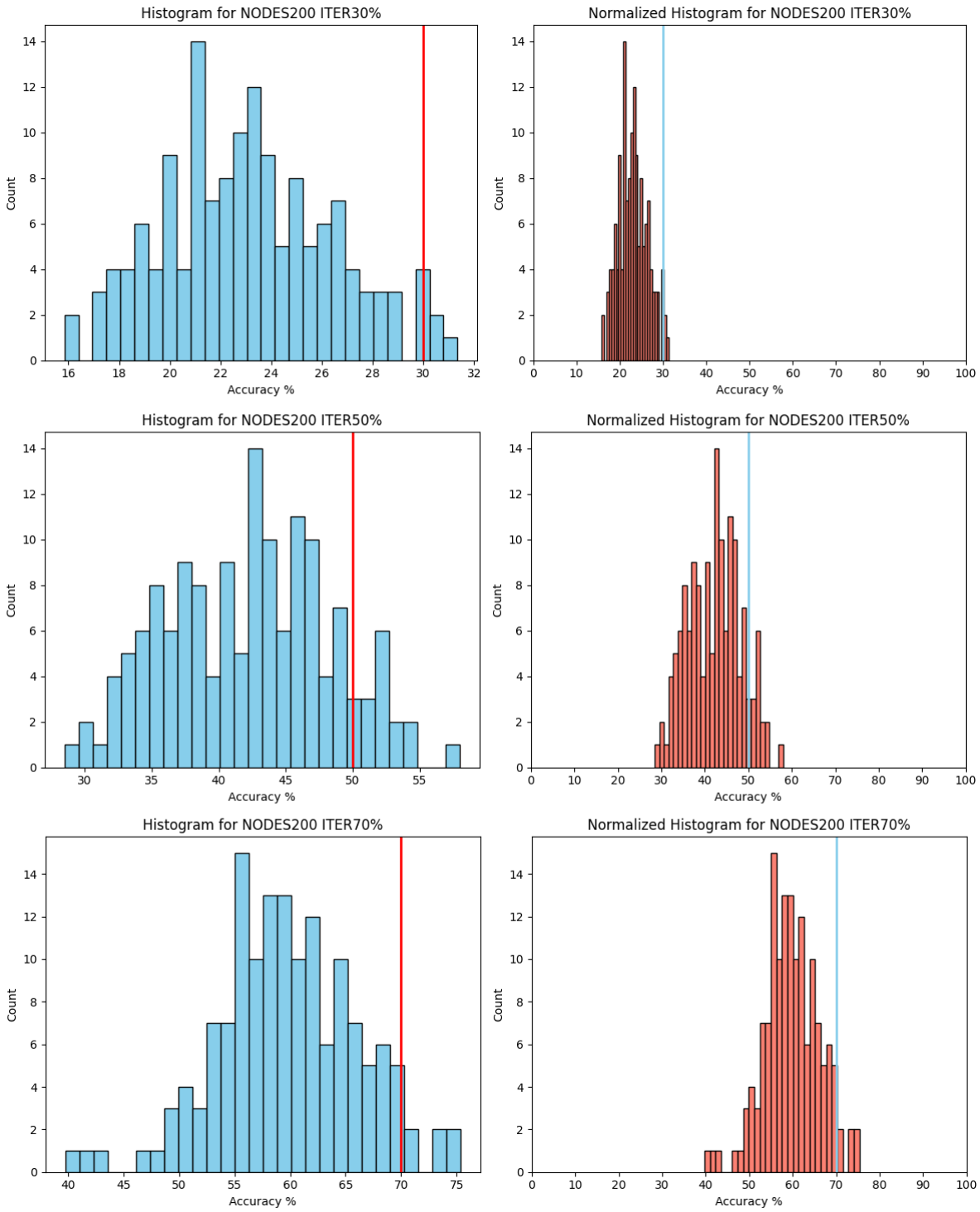
4.1 Masové testy

Nejprve byly spouštěny masové testovací sady. Každá sada (tedy jednotlivý histogram) obsahuje přibližně 150 grafů a dohromady bylo otestováno necelých 1300 grafů. Ze statistického hlediska nás především zajímaly tyto parametry:

- Přesnost výpočtu - neboli jak moc se aproximační algoritmus liší od exaktního.
- Velikost `iteration_modifier` - neboli kolik iterací jsme provedli.
- Faktor zrychlení - Kolikrát se nám podařilo zrychlit výpočet daného grafu.

- Interval spolehlivosti - s jakou spolehlivostí se podobá aproximovaná hodnota exaktní.

Histogramy v hlavičce obsahují počet vrcholů a kolik (procentuální) části grafu jsme prošli. **V této části je interval spolehlivosti vždy 95%. Pravděpodobnost tvorby vrcholu je 40%.**

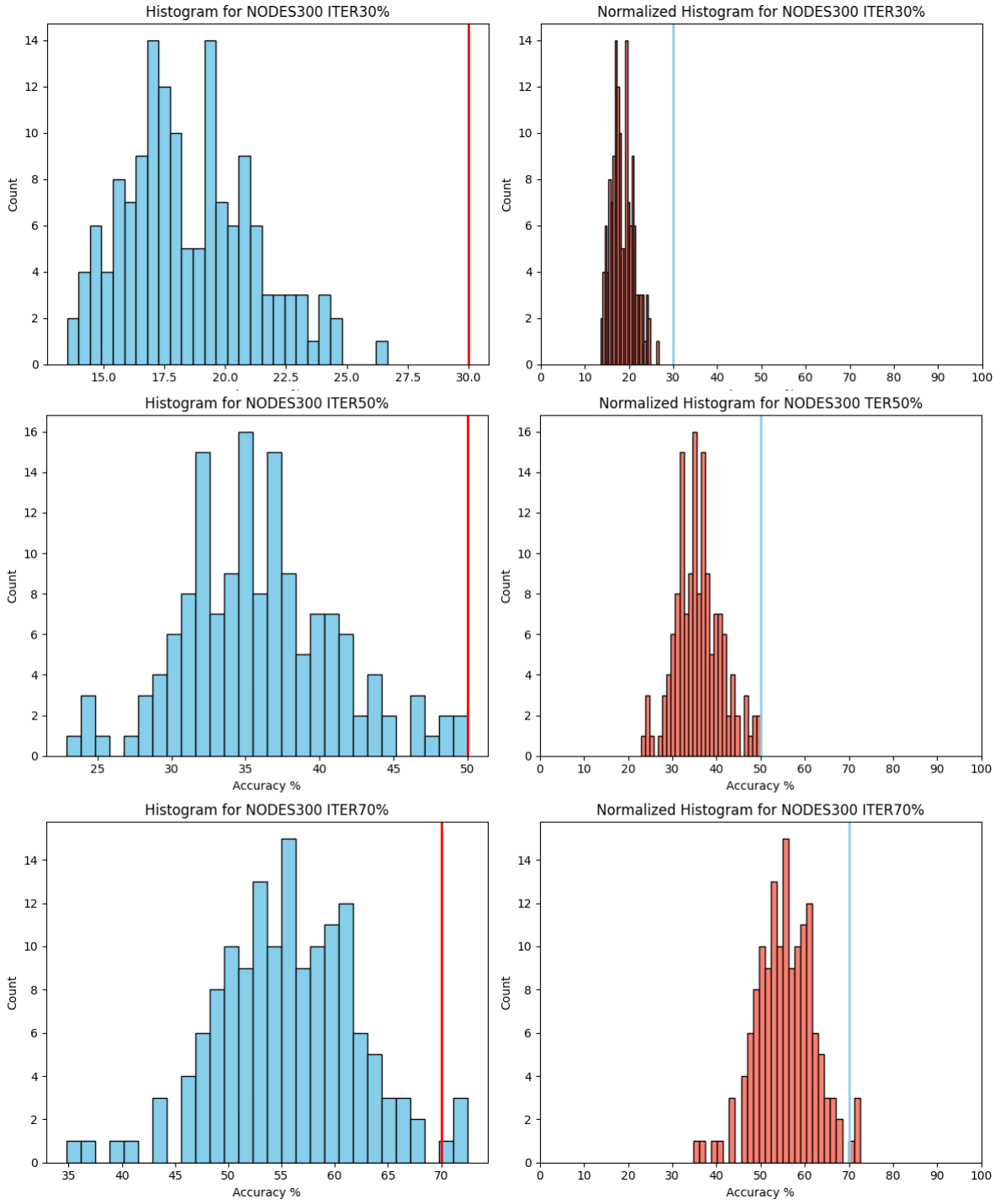


Obrázek 3: Jednotlivé histogramy pro 200 vrcholů

Obrázek obsahuje dva grafy, jeden přiblížený pro detailnější popis os a druhý s normalizovanou osou x na rozsah 0-100%. Na ose x leží procentuální přesnost výpočtu a na ose y celkový počet grafů s touto

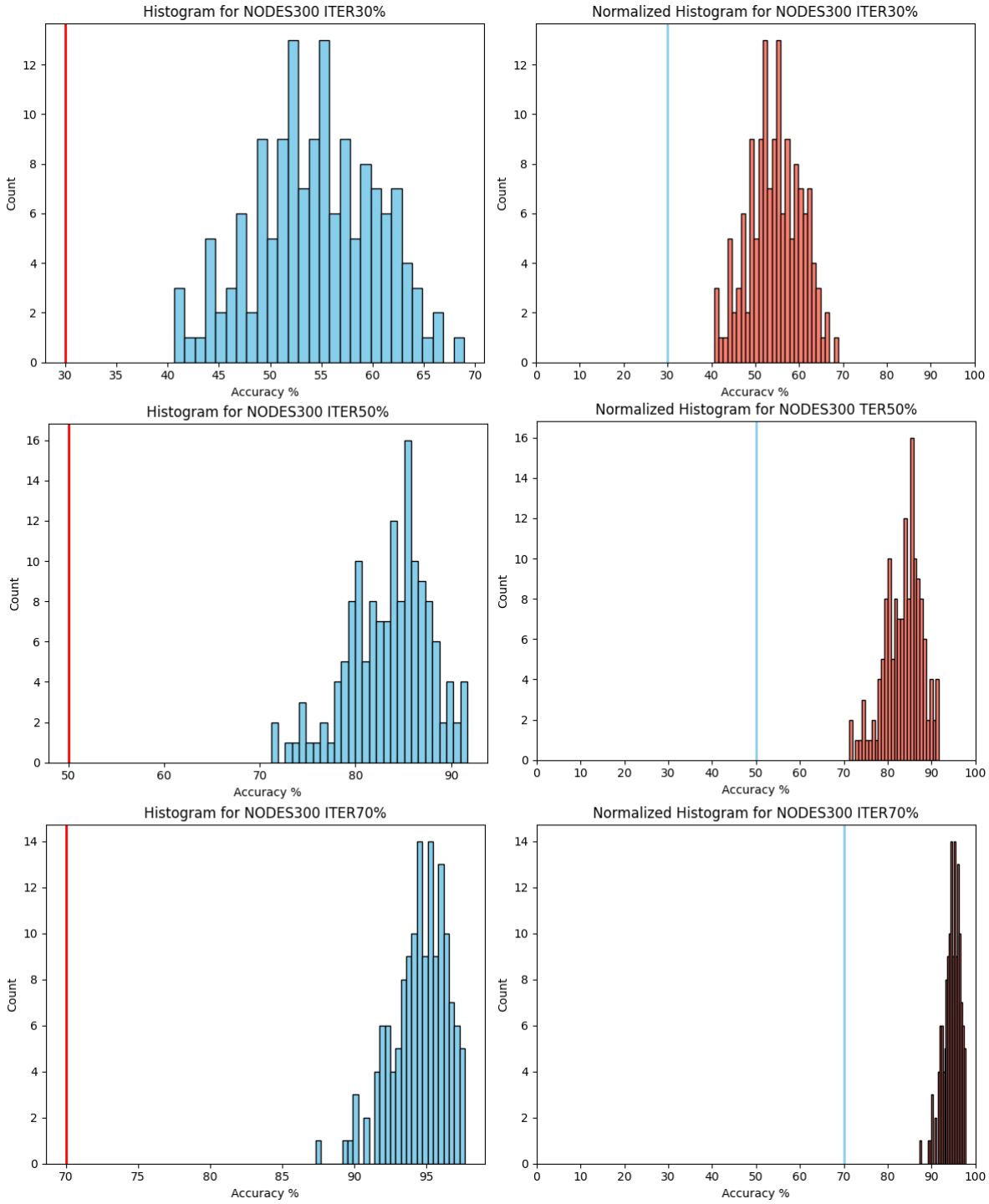
hodnotou. Barevná čára vyznačuje „předpokládaný“ střed Gaussova rozdělení, kolem kterého by se histogram měl pohybovat.

V jednotlivých histogramech lze vidět horší výsledky než by se dalo očekávat. Např. při průchodu 70% grafu máme nejčastější přesnost pouhých 55%. U 300 vrcholů je tento jev ještě více viditelný. V tomto testu jsme ještě snížili **pravděpodobnost hran na 30%** a zejména v nejméně prozkoumaném grafu je přesnost opravdu špatná.



Obrázek 4: Jednotlivé histogramy pro 300 vrcholů

Protože se zhoršila přesnost u grafů s nižším počtem hran, zajímalo nás, jestli se přesnost nezlepší s větším výskytem hran. Následující grafy zobrazují výsledky pro 70% pravděpodobnost vytvoření hrany.

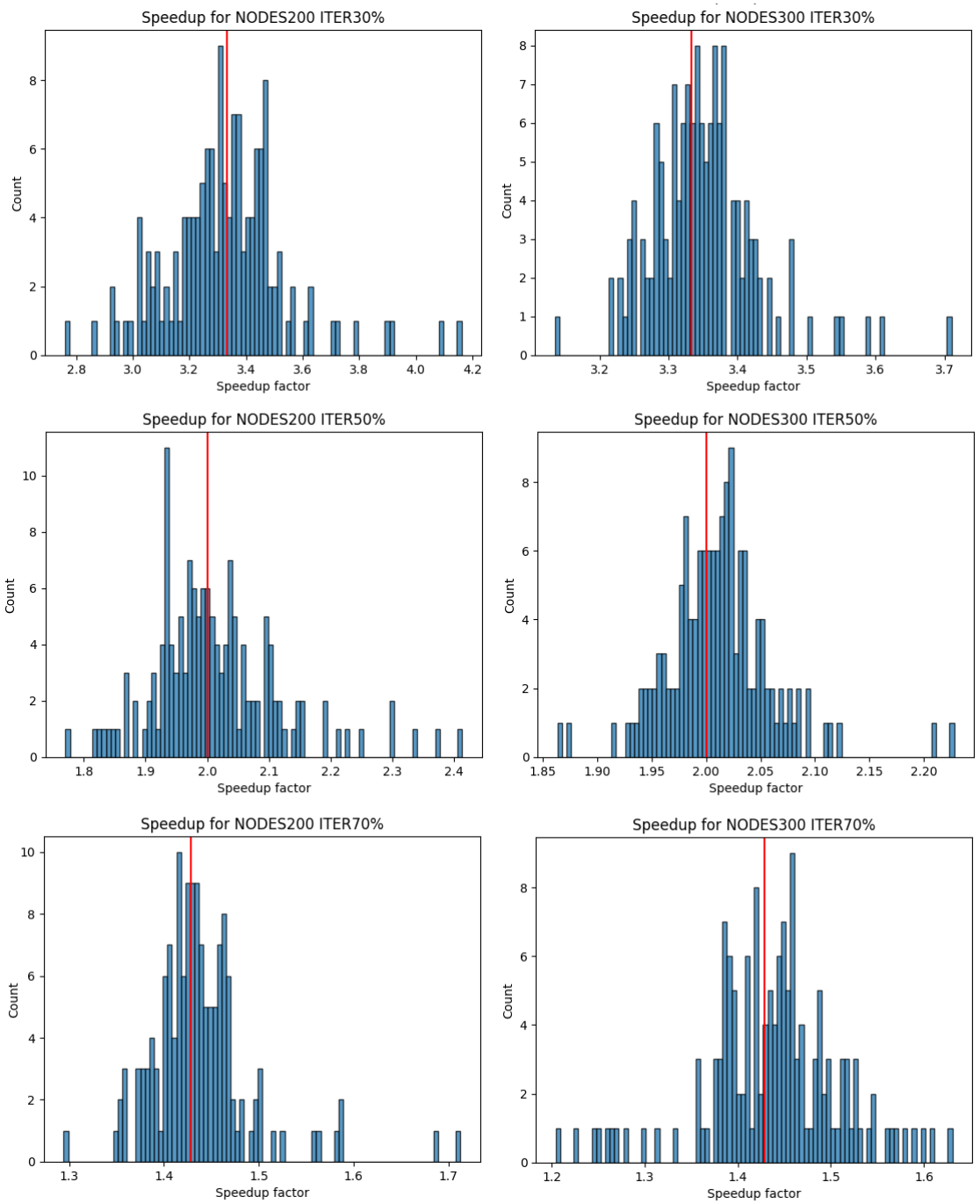


Obrázek 5: Jednotlivé histogramy pro grafy s větším výskytem vrcholů

Na první pohled je hned viditelný rozdíl. Očekávaná přesnost je značně převýšena a tudíž nám algoritmus dává lepší výsledky. U průchodu 70% grafu máme u většiny grafů 95% přesnost. U poloviního průchodu se přesnost pohybuje okolo 85% a u 30% průchodu činí přesnost přibližně 55%. Jedná se ted o mnohem lepší výsledky, než očekáváno.

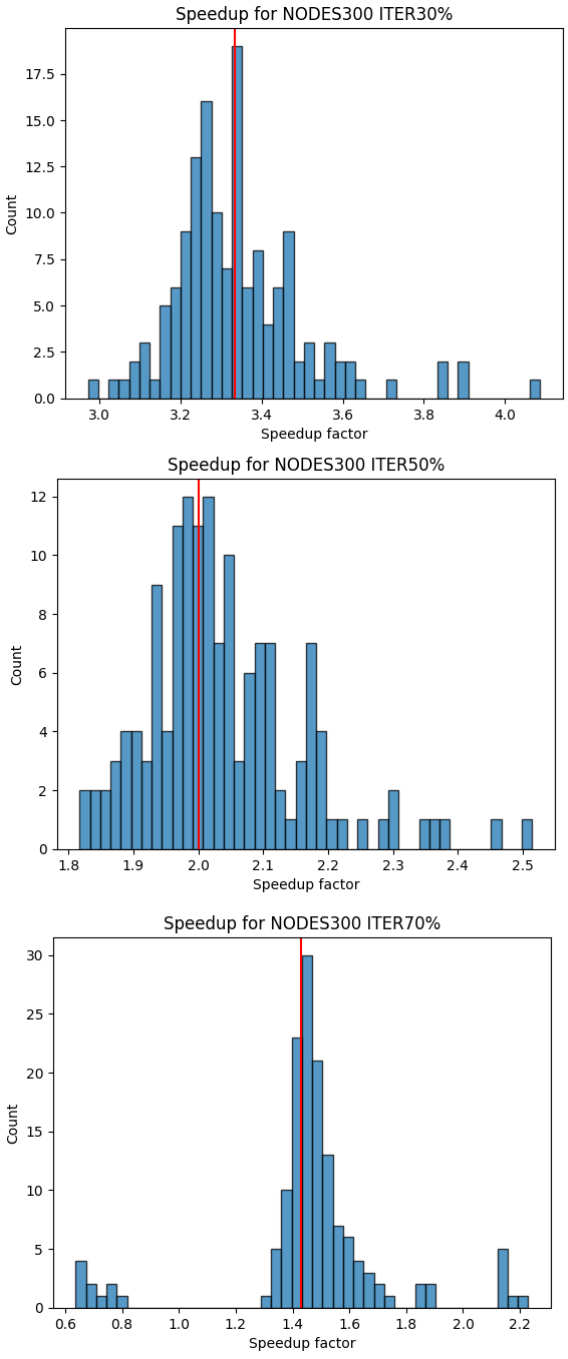
Teď trochu více k faktoru zrychlení. Opět platí stejná pravidla jako pro dřívejší histogramy. Barevná čára značí předpokladané průměrné zrychlení, osa x značí faktor zrychlení (neboli kolikrát je skript rychlejší) a osa y značí četnost jednotlivých zrychlení.

Zde už hodnoty odpovídají očekávaným hodnotám. Nejčastěji je zrychlení tolikrát větší, o kolik bylo ušetřeno při „neprůchodu“ grafu. Samozřejmě jsou případy, kdy byl aproximační algoritmus lepší, ale také horší.



Obrázek 6: Jednotlivé histogramy faktoru zrychlení

Taktéž tomu je i u grafů s větší pravděpodobností výskytu hran. Vrchol Gaussovské křivky je nejvyšší v bodě barevné čáry.



Obrázek 7: Histogramy faktoru zrychlení

Je ovšem zajímavé, že u průchodu 70% grafu dostáváme jisté odchylky v čase. Je možné, že tyto hodnoty jsou čistě chybná měření použitým programem.

4.2 Malé sady o hustých grafech

V této části byly procházeny grafy o 1000 vrcholech s pravděpodobností výskytu hrany 45%. Z důvodu dlouhé časové náročnosti (jeden test exaktní metodou trval přes 10 minut) nebylo generované veliké množství grafů v

datové sadě a proto jsou výsledky popsány numericky. Jednotlivé obrázky zobrazují základní statistické hodnoty jako jsou medián, průměr, odychlka a maximální a minimální hodnota. . .

V první sada obsahuje čtyři grafy a pracuje s intervalem spolehlivosti (I.S.) 95%. Přesnost aproximačního algoritmu je téměř perfektní, průměrně je přesnot 98.5% a zrychlení dvojnásobné.

	Exact algorithm time	Approximation algorithm time	Accuracy	Tolerance	Weight	Iteration modifier	Speedup	Confidence interval
count	4.000000	4.000000	4.000000	4.0	4.0	4.0	4.000000	4.0
mean	638.432862	316.228790	98.582500	15.0	300.0	0.5	2.018875	95.0
std	17.207258	7.682438	0.586082	0.0	0.0	0.0	0.022455	0.0
min	613.533911	307.305743	97.750000	15.0	300.0	0.5	1.996500	95.0
25%	635.009794	313.167135	98.417500	15.0	300.0	0.5	2.001075	95.0
50%	643.641735	315.787229	98.740000	15.0	300.0	0.5	2.020200	95.0
75%	647.064804	318.848884	98.905000	15.0	300.0	0.5	2.038000	95.0
max	652.914069	326.034958	99.100000	15.0	300.0	0.5	2.038600	95.0

Obrázek 8: Sada první - I.S. 95%, průchod 50% grafu

Druhou sadou skript prochází pouhými 30% grafu a stále se pohybuje na 83% přesnosti. Tyto hodnoty lze stále v jisté míře akceptovat, obzvláště pokud uvážíme tři-a-půl násobné zrychlení.

	Exact algorithm time	Approximation algorithm time	Accuracy	Tolerance	Weight	Iteration modifier	Speedup	Confidence interval
count	18.000000	18.000000	18.000000	18.0	18.0	18.0	18.000000	18.0
mean	730.056157	212.166703	83.241111	15.0	300.0	0.3	3.439650	95.0
std	52.067884	13.750538	3.656807	0.0	0.0	0.0	0.037199	0.0
min	649.520945	191.238577	77.200000	15.0	300.0	0.3	3.372900	95.0
25%	679.407944	198.233890	80.790000	15.0	300.0	0.3	3.412700	95.0
50%	756.992691	218.880178	83.310000	15.0	300.0	0.3	3.441000	95.0
75%	770.207262	223.021623	86.030000	15.0	300.0	0.3	3.470300	95.0
max	801.094987	230.645718	89.080000	15.0	300.0	0.3	3.505300	95.0

Obrázek 9: Sada druhá - I.S. 95%, průchod 30% grafu

U třetí sady byl testován přesnější interval spolehlivosti 99%. Výsledky jsou překvapivé. Téměř třetina hodnot je se spolehlivostí 99% stejná jako exaktní algoritmus a to s pouhým polovičním průchodem a náhodným výběrem vrcholů.

	Exact algorithm time	Approximation algorithm time	Accuracy	Tolerance	Weight	Iteration modifier	Speedup	Confidence interval
count	6.000000	6.000000	6.000000	6.0	6.0	6.0	6.000000	6.0
mean	635.086944	318.693695	29.253333	3.0	300.0	0.5	1.992650	99.0
std	15.282071	6.010162	2.495369	0.0	0.0	0.0	0.017582	0.0
min	615.397186	312.091090	24.740000	3.0	300.0	0.5	1.970400	99.0
25%	622.940302	314.559045	28.567500	3.0	300.0	0.5	1.978025	99.0
50%	637.105494	317.020842	30.165000	3.0	300.0	0.5	1.997150	99.0
75%	646.360096	323.247478	30.892500	3.0	300.0	0.5	2.006375	99.0
max	653.249958	326.967925	31.280000	3.0	300.0	0.5	2.010100	99.0

Obrázek 10: Sada třetí - I.S. 99%, průchod 50% grafu

U čtvrté sady byl cíl tzv. „vytěžit co nejvíce“ z aproximačního algoritmu. Tento test má ukázat jaká je horní limita. A i přes to, že se může 12% zdát jako špatná hodnota, nesmíme zapomenout, že se jedná o interval spolehlivosti 99% a průchod pouhých 30% grafu s trojnásobným zrychlením.

	Exact algorithm time	Approximation algorithm time	Accuracy	Tolerance	Weight	Iteration modifier	Speedup	Confidence interval
count	6.000000	6.000000	6.000000	6.0	6.0	6.0	6.000000	6.0
mean	629.444546	188.857088	12.060000	3.0	300.0	0.3	3.333267	99.0
std	8.063719	3.230290	0.941339	0.0	0.0	0.0	0.037902	0.0
min	615.958202	185.417130	11.180000	3.0	300.0	0.3	3.294000	99.0
25%	627.871879	186.747366	11.380000	3.0	300.0	0.3	3.305825	99.0
50%	630.090636	188.481583	11.830000	3.0	300.0	0.3	3.322400	99.0
75%	632.066010	189.692493	12.407500	3.0	300.0	0.3	3.359375	99.0
max	640.671040	194.496809	13.690000	3.0	300.0	0.3	3.388800	99.0

Obrázek 11: Sada čtvrtá - I.S. 99%, průchod 30% grafu

4.3 Shrnutí výsledků

Při testování řídších grafů (200 a 300 uzlů s 40% a 30% výskyt hran) s menší pravděpodobností tvorby hrany se aproximační algoritmus choval hůře než očekáváno. Jeho přesnost byla horší, než původní předpoklad, avšak zrychlení algoritmu se chovalo dle očekávání. Narozdíl grafy s velkou pravděpodobností výskytu hrany (70%) byly aproximovány s mnohem lepší přesností než byl původní předpoklad.

Při testování hustých grafů (1000 uzlů a 45% na výskyt hrany) se aproximační algoritmus choval výrazně lépe než očekáváno. Zrychlení se sice chovalo dle očekávání, ale přesnost algoritmu byla v podstatě zachována.

Lze tedy vyvodit, že na grafy s větší pravděpodobností výskytu hran je aproximační algoritmus časově výrazně lepší a zachovává velmi dobrou přesnost. Pro hustší grafy výskyt hrany hraje menší roli a aproximační algoritmus má téměř perfektní přesnost.

5 Použití skriptů

5.1 Instalace:

Instalace projektu vyžaduje vytvoření virtuálního prostředí a následnou instalaci závislostí uvedených v souboru `requirements.txt`.

1. Tvorba virtuálního prostředí:

```
python3 -m venv .venv
source .venv/bin/activate
```

2. Instalace závislostí:

```
pip install -r requirements.txt
```

Generování dat

Pro generování grafů slouží skript `auto_gen.py`, který umožňuje vytvořit náhodné grafy na základě zadaných parametrů.

```
python3 src/auto_gen.py --nodes 200 220 --probability 0.4 --max_weight 50
    --seeds 10 15 --batch_name "example_batch"
```

Popis argumentů:

- `--nodes [from to]`
  Definuje rozsah počtu uzlů grafu. Například `--nodes 200 220` znamená, že grafy budou generovány s počtem uzlů od 200 do 220.

- `--probability`
  Určuje pravděpodobnost vytvoření hrany mezi dvěma uzly v intervalu [0,1]. Hodnota `--probability 0.4` znamená, že každá hrana bude vytvořena s pravděpodobností 40 %.

- `--max_weight`
  Nastavuje maximální váhu hran v grafu. Váhy jsou náhodně generovány v rozsahu od 1 do `max_weight`. Příklad `--max_weight 50` znamená, že váhy hran budou v intervalu $[1, 50]$.

- `--seeds [from to]`
  Umožňuje zadat rozsah seedů pro náhodný generátor. Různé seedy zajišťují odlišné grafy pro stejná nastavení parametrů. Např. `--seeds 10 15` vygeneruje grafy s hodnotami seedů od 10 do 15.

- `--batch_name`
  Jméno složky, do které budou uloženy vygenerované grafy. Výsledky se uloží do složky `csv_data/raw/<batch_name>`. Například `--batch_name "example_batch"` vytvoří složku `csv_data/raw/example_batch`.

Spuštění algoritmů

Pro spuštění přesného a aproximačního algoritmu Floyd-Warshall je určen skript `run_test.py`. Tento skript umožňuje porovnat výsledky obou algoritmů.

```
python3 src/run_test.py --iteration_modifier 0.3 --tolerance 5 --
   batch_name "example_batch"
```

Popis argumentů:

- `--iteration_modifier`
  Modifikuje počet iterací aproximačního algoritmu v závislosti na počtu uzlů v intervalu [0,1]. Například `--iteration_modifier 0.3` znamená, že se provede pouze 30 % iterací.

- `--tolerance`
  Definuje maximální povolenou odchylku mezi výsledky přesného a aproximačního algoritmu. Například `--tolerance 5` umožňuje, aby výsledek aproximačního algoritmu odchyloval od přesného výsledku maximálně o 5 jednotek váhy hrany.

- `--batch_name`
  Název dávky grafových dat, na kterých bude algoritmus spuštěn. Tato data musí být předem vygenerována skriptem `auto_gen.py`. Například `--batch_name "example_batch"` spustí algoritmus na grafech uložených ve složce `autogen_datasets/example_batch`.

Sloučení výsledků

Pro sloučení výsledků více dávkových úloh do jednoho souboru slouží skript `make_csv.py`.

```
python3 src/make_csv.py --batches batch1 batch5
```

Popis argumentů:

- `--batches BATCHES [BATCHES ...]`
  Například `--batches batch1 batch5` sloučí výsledky ze složek `csv_data/raw/batch1` a `csv_data/raw/batch5` do jednoho souboru `csv_data/data.csv`.

Struktura projektu

Složka `src/` obsahuje následující skripty:

- `auto_gen.py` - Skript pro generování grafových datasetů.
- `run_test.py` - Skript pro spuštění testů přesného a aproximačního algoritmu.
- `make_csv.py` - Skript pro sloučení výsledků do jednoho souboru.
- `graph_generator.py` - Generátor grafů v maticové formě s vizualizací.
- `gal.py` - Implementace přesného a aproximačního Floyd-Warshallova algoritmu.