
**Dokumentace projektu**
Předmět: **GAL** 2018/19
Projekt: **Heuristiky pro pokrytí uzlů**
Autoři: **Jan Kohút** xkohut08
**Vladimír Jeřábek** xjerab21

# 1 Úvod

Projekt je zaměřen na řešení problému Minimálního vrcholového pokrytí grafů pomocí aproximačních algoritmů, které využívají různé heuristiky. Problém spočívá v nalezení minimální množiny vrcholů tak, aby každá hrana měla alespoň jeden vrchol v této množině. Nalezení takové množiny je NP-úplný problém, je tudíž žádoucí výpočet urychlit pomocí aproximačních algoritmů za cenu suboptimálního řešení. Kvalitu těchto algoritmů lze vyjádřit pomocí tzv. aproximačního koeficientu, ten udává nejhorší možný poměr mezi řešením algoritmu a řešením optimálním. Některé algoritmy jsou schopny zaručit určitý koeficient tzn. pro libovolný graf poměr řešení nikdy nepřesáhne tento koeficient. Nejlepší známý konstantní koeficient je 2 [1]. Jiné algoritmy umožňují získání lepšího řešení za cenu potenciálně velmi špatných řešení v některých specifických případech. Výběr heuristiky tedy záleží na konkrétním případu použití.

# 2 Heuristiky

V rámci projektu jsme si zvolili dvě nejlépe hodnocené heuristiky popsané v článku [1] (MDG, GIC) a heuristiku popsanou v knize [2] (ME). Všechny algoritmy iteračně vybírají množinu vrcholů, kterou přidají do výsledného pokrytí. Na konci každé iterace je tato množina navíc odstraněna z původního grafu tzn. v další iteraci se vybírá z podgrafu původního. Pokud graf nemá žádné hrany znamená to, že byl pokryt a algoritmus končí. Heuristiky spočívají ve způsobu výběru množiny vrcholů v každé iteraci. Grafy jsou reprezentovány jako seznamy sousedů (adjacency list). Počet vrcholů je značen písmenem $n$ a počet hran písmenem $m$.

**Maximum Degree Greedy (MDG)** Algoritmus vybírá v každé iteraci pouze jeden vrchol a to vrchol s maximálním nenulovým stupněm. Pseudokód algoritmu Algoritmus 1. Hlavní cyklus se maximálně provede $n-1$ krát. Kontrola výskytu hrany v grafu, získání požadovaného vrcholu a smazání tohoto vrcholu z grafu má linearní časovou složitost. Získání vrcholu se provede spočítáním stupňů všech vrcholů ($O(n+m)$) a následným výběrem maxima ($O(n)$). Pro smazání vrcholu z grafu je potřeba projít celou reprezentaci grafu ($O(n+m)$). Celková složitost algoritmu je tedy $O(n^2+nm)$. Algoritmus po celou dobu výpočtu využívá pouze jeden graf, který se průběžně modifikuje a konstantní počet proměnných. Celková prostorová složitost je tedy $O(n+m)$.

**Algorithm 1** Maximum Degree Greedy

```
1: function MDG(graph)                                   ▷ O(n^2 + nm)
2:     cover ← ∅                                         ▷ O(1)
3:     while graph.HaveEdges() do                        ▷ O(n)
4:         maxVertex ← graph.GetMaxDegreeVertex()        ▷ O(n + m + n)
5:         cover ← cover ∪ maxVertex                     ▷ O(1)
6:         graph.RemoveVertex(maxVertex)                 ▷ O(n + m)
7:     return cover
```

**Greedy Independent Cover (GIC)** Algoritmus vybírá v každé iteraci vrcholy sousední s vrcholem o minimálním nenulovém stupni. Pseudokód algoritmu Algoristmus 2. Hlavní cyklus se maximálně provede $n/2$ krát.

Oproti algoritmu MDG se navíc provádí získání sousedních vrcholů ($O(n)$). Celková časová složitost algoritmu je $O(n^2 + nm)$. Stejně jako v případě algoritmu MDG je použit pouze jeden graf po celou dobu běhu. Počet sousedů je celkově maximálně $n - 1$. Celková prostorová složitost je tedy opět $O(n + m)$.

**Algorithm 2** Greedy Independent Cover

```
1: function GIC(graph)                                          ▷ O(n^2 + nm)
2:     cover ← ∅                                                ▷ O(1)
3:     while graph.HaveEdges() do                               ▷ O(n)
4:         minVertex ← graph.GetMinDegreeVertex()               ▷ O(n + m + n)
5:         neighbours ← graph.GetNeighbours(minVertex)          ▷ O(n)
6:         cover ← cover ∪ neighbours                           ▷ O(1)
7:         graph.RemoveVertices(neighbours)                     ▷ O(n + m)
8:     return cover
```

**Max Edge (ME)** Algoritmus vybírá v každé iteraci vrcholy, které jsou sousední a součet jejichž stupňů je největší. Pseudokód algoritmu Algoritmus 3. Hlavní cyklus se opět maximálně provede $n/2$ krát. Oproti předchozím algoritmům je výběr vrcholů asymptoticky složitější neboť se musí provádět srovnání každý s každým ($O(n + m + n^2)$). Celková časová složitost algoritmu je $O(n^3 + nm)$. Algoritmus využívá oproti předcházejícím pole uchovávající informace o stupni jednotlivých dvojic vrcholů, aby následně mohl vybrat maximální ($O(n^2)$). Celková prostorová složitost je tedy $O(n^2 + m)$[1].

**Algorithm 3** Max Edge

```
1: function ME(graph)                                           ▷ O(n^3 + nm)
2:     cover ← ∅                                                ▷ O(1)
3:     while graph.HaveEdges() do                               ▷ O(n)
4:         vertex_0, vertex_1 ← graph.GetMaxEdge()              ▷ O(n + m + n^2)
5:         cover ← cover ∪ {vertex_0, vertex_1}                 ▷ O(1)
6:         graph.RemoveVertices({vertex_0, vertex_1})           ▷ O(n + m)
7:     return cover
```

**Optimalizace** První optimalizace má název *pendant*. Libovolný aproximační algoritmus lze potenciálně zlepšit přidáním každého vrcholu jehož sousední vrchol má stupeň jedna do výsledného pokrytí. Tyto vrcholy jsou následně odstraněny z grafu, čímž je získán nový graf nad kterým se provede samotný algoritmus. Pokud má nějaká komponenta grafu pouze jednu hranu, je vybrán pouze jeden vrchol. Takto vybrané vrcholy patří do nějakého minimálního pokrytí grafu [3]. Časová složitost této optimalizace je $O(n+m)$. Optimalice *redundant* spočívá v odebrání z výsledného vrcholového pokrytí, jejichž všichni sousedé jsou také v tomto pokrytí [2]. Odebírání se provádí iteračně od nejmenšího počtu sousedů. Časová složitost je $O(n^3 + m)$.

**Srovnání** První dva algoritmy jsou srovnatelné, co se týče časové i prostorové složitosti. Jejich hlavní nevýhodou je, že nezaručují nejhorší možný výsledek vzhledem k optimálnímu pokrytí. Pro specifické grafy tedy může být výsledné pokrytí značně horší než optimální [1]. Poslední algoritmus na druhou stranu zaručuje, že výsledné pokrytí bude maximálně dvakrát horší než pokrytí optimální. Tato skutečnost plyne z nutnosti pokrýt každou hranu v grafu. Algoritmus ME pokrývá graf po hranách tak, že do vrcholového pokrytí přidá oba vrcholy hrany. V nejhorším případě lze všechny hrany pokryté tímto způsobem pokrýt pouze jedním vrcholem tzn. dvakrát horší pokrytí. Oproti ostatním dvěma algoritmům, je časová i prostorová složitost asymptoticky větší.

[1] Prostorovou složitost je možné snížit na $O(n + m)$, pro nalezení maximálního stupně dvojice vrcholů si není potřeba pamatovat ohodnocení všech možných dvojic.

# 3 Implementace

Algoritmy jsme implementovali v jazyce C++. Grafy jsou reprezentovány třídou `Graph`. Tato třída umožňuje odstranění vrcholů z grafu, získání vrcholu o maximálním/mninimálním stupni a získání sousedů. Jako reprezentaci jsme zvolili seznam sousedů (adjacency list), kterou jsme realizovali pomocí `std::list<std::list<int>>`. Tato reprezentace zajišťuje časovou i prostorovou složitost popsanou v kapitole 2, nalezení i odstranění má linearní časovou složitost.

Rodičovskou třídu algoritmů představuje abstraktní třída `Algorithms`. Tato třída požaduje po svých potomcích (třídách reprezentující jednotlivé algoritmy) implementaci metody `std::vector<int> Run(Graph graph, bool removePendant, bool removeRedundant)`. `Run` spočítá vrcholové pokrytí, pomocí argumentů typu `bool` lze zadat využití/nevyužití optimalizací. Implementace jednotlivých algoritmů odpovídají pseudokódum v kapitole 2.

Experiment časové složitosti implementuje třída `TimeSize`. Metoda `std::pair<double, int>Run(int numberOfInterations, Algorithms &algorithm, Graph &graph, bool removePendant, bool removeRedundant)` vrací dvojici průměrného času a průměrného pokrytí pro daný počet iterací, jedna iterace se rovná jednomu spuštění algoritmu nad daným grafem.

Postup pro překlad a spuštění programu je uveden v přiloženém souboru README.md.

# 4 Experimenty

V rámci experimentů jsme se především zaměřili na porovnání algoritmů a optimalizací uvedených v kapitole 2 mezi sebou. Kromě toho, že jsme u jednotlivých algoritmů ověřili jak časovou, tak i prostorovou složitost, tak jsme uskutečnili experimenty s cílem zjistit, který algoritmus dosahuje co nejlepšího minimálního vrcholového pokrytí. V neposlední řadě jsme také prováděli experimenty za účelem zjištění vazby mezi *hustotou grafu* a danými algoritmy.

**Použité grafy v experimentech** V jednotlivých experimentech jsme použili grafy[2] vygenerované vlastním programem `graph_gen`. Vstupem je počet vrcholů a počet hran výsledného grafu. Výstupem pak je `.txt` soubor, který specifikuje neorientovaný graf s určeným počtem vrcholů a určeným počtem hran. Hrany jsou k jednotlivým vrcholům přiřazeny náhodně. Struktura grafu v `.txt` souboru je následující:

- 1. řádek: Název grafu (`string`).
- 2. řádek: Počet vrcholů grafu (`int`).
- 3. řádek: Informace o sousedních vrcholech k vrcholu číslo 0. Čísla jednotlivých přilehlých vrcholů jsou odděleny mezerou. (*Pozn: Číslo vrcholu odpovídá číslu řádku - 3. Tedy na řádku 3 je informace o sousedních vrcholech pro vrchol 0.*)
- 4. řádek: Informace o sousedních vrcholech k vrcholu číslo 1. (*Pozn: Stejně jako 3. řádek*).
- 5. řádek: ...

Při generování testovacích grafů jsme rozdělili grafy do tzv. *tříd* dle *hustoty grafu*. Kde každá třída obsahuje grafy s *hustotou* v daném intervalu. Hustotu grafu jsme určili dle vztahu:

$$D = \frac{2|E|}{|V|(|V|-1)},$$

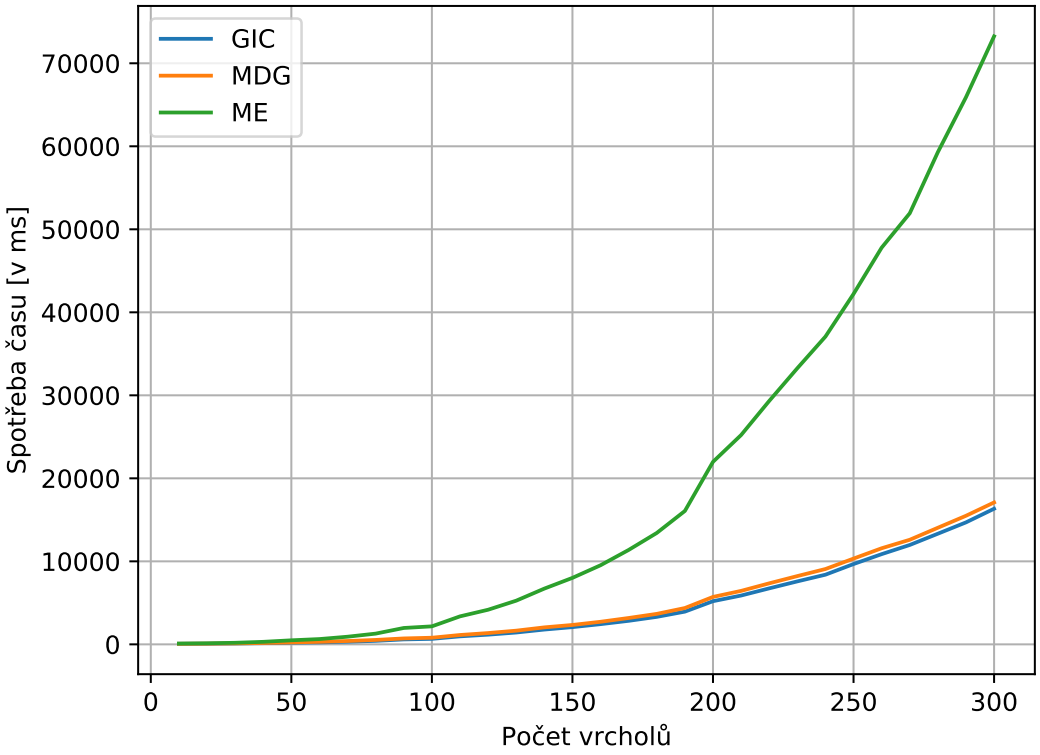
kde $D$ je *hustota* grafu, $|E|$ je počet hran v grafu a $|V|$ je počet vrcholů v grafu. Pro experimenty jsme použili

[2] Odkaz na grafy `https://drive.google.com/drive/folders/1QymWQCwExOFlnKIJnHOJLz4HCFZkdgE_?usp=sharing`

10 různých tříd. Každá třída má stejné rozmezí hustot a to $0.1$. Tzn: Třída 0 obsahuje grafy s hustotou $0.0 - 0.1$, třída 1 obsahuje grafy s hustotou $0.1 - 0.2$, atd. V jednotlivých třídách (s konkrétní hustotou) jsou složky s grafy s 10, 20, 30, ..., 100 vrcholy a dané složky obsahují až 300 různých grafů. Celkově bylo použito přes $10 \times 10 \times 300 = 30\,000$ grafů. Účelem takto sestavené sady je zjistit vhodnost použití jednotlivých algoritmů v závislosti na třídě hustoty a počtu vrcholů grafů. Počet vzorků u jednotlivých tříd hustoty o počtu vrcholů by měl dostatečně reprezentovat danou skupinu grafů.

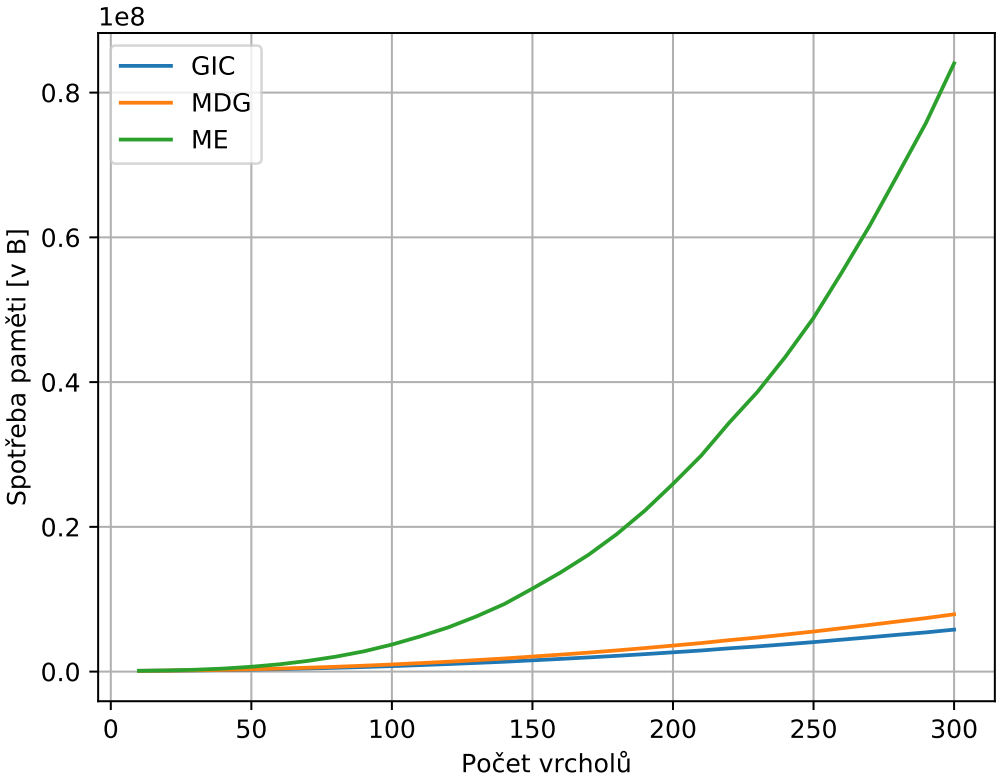
**Časová a prostorová složitost algoritmů** V teoretické části jsme se zabývali teoretickými hodnotami pro časovou a prostorovou složitost jednotlivých algoritmů a nyní se podíváme na výsledky získané z experimentů.

Pro experimenty jsme použily grafy s *hustotou* $0.4 - 0.6$ a počtem vrcholů 10, 20, ... , 300, abychom dosáhli větší přesnost a také mohli snáze odhadnout tendenci. Pro určení časové složitosti jednotlivých algoritmů bylo použito hodnot, které vracela konzolová aplikace `vc`, která měřila přesně na milisekundy běh daného algoritmu nad konkrétním grafem. Pro určení prostorové složitosti byl použit program `valgrind`, který zaznamenával celkovou alokovanou paměť programu. To bylo možné použít s ohledem na zanedbatelnou režii samotného programu (samotný program alokoval vždy konstantní množství paměti).




Obrázek 1: Časová náročnost jednotlivých algoritmů v závislosti na počtu vrcholů.

Na obrázku 1 lze vidět, že algoritmus GIC a MDG jsou na tom velmi podobně, naproti tomu časová složitost algoritmu ME je viditelně horší. Tyto výsledky potvrzují teoretické předpoklady, které jsme uvedli v kapitole 2. Podobně jako časová složitost, dopadl i graf s prostorovou složitostí jednotlivých algoritmů, viz obrázek 2. Vidíme, že algoritmus ME je asymptoticky složitější, než algoritmy MDG a GIC. Tyto výsledky korelují s teoretickými předpoklady viz kapitola 1.




Obrázek 2: Prostorová náročnost jednotlivých algoritmů v závislosti na počtu vrcholů.

**Experimenty velikosti vrcholového pokrytí** Dále jsme se při experimentech zaměřili na porovnání vhodnosti jednotlivých algoritmů na konkrétní typy grafů, kde jsme se především soustředili na minimalitu vrcholového pokrytí nad řídkými a hustými grafy. Jelikož neexistuje jednoznačné rozlišení mezi řídkými a hustými grafy, tak jsme pro účely tohoto projektu nazvali řídkými grafy skupinu grafů z třídy 0 (tedy grafy s hustotou $0.0 - 0.1$) a jako husté grafy jsme nazvali skupinu grafů z třídy 9 (tedy grafy s hustotou $0.9 - 1.0$)

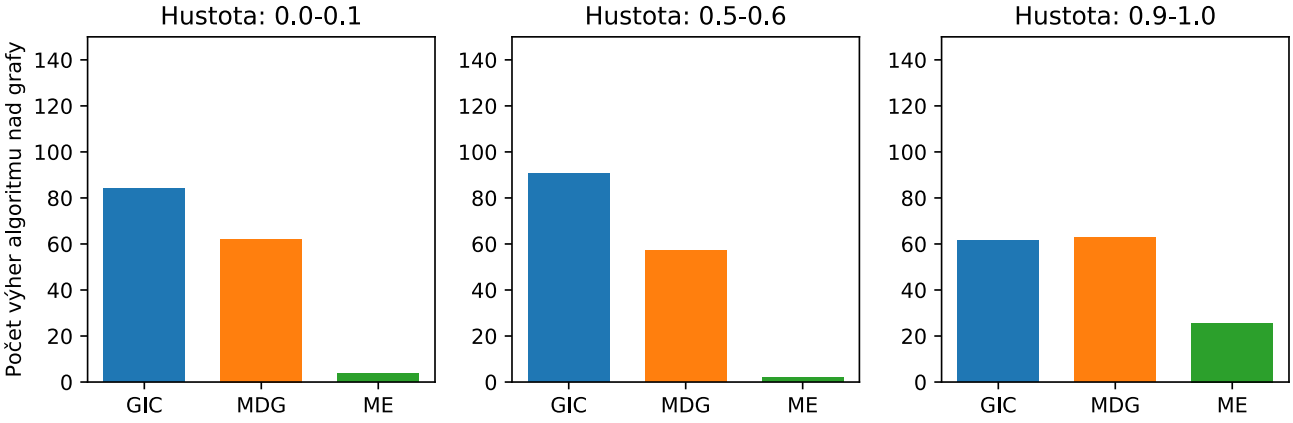
V tabulce 4 lze vidět výsledky experimentu (pouze část), který sloužil pro zjištění velikosti průměrného vrcholového pokrytí v různých třídách hustoty a při různých velikostech grafu. Z výsledků experimentu lze vidět, že algoritmus GIC generuje nejlepší vrcholové pokrytí z implementovaných algortimů. Dále je pozorovatelné, že algoritmy GIC a MDG jsou velmi blízké, co se týče velikosti průměrného vrcholového pokrytí. V neposlední řadě lze z grafu vyčíst, že s rostoucí hustotou grafu se mezery mezi velikostmi průměrných vrcholových pokrytí u jednotlivých algoritmů zmenšují (nejlépe je to viditelné u gtafů s počtem vrcholů 100, kde v posledním řádku se průměrné vrcholové pokrytí pro GIC, MDG a ME skoro rovnají).

**Experiment lepšího vrcholového pokrytí (*soutěž*)** Následující experimenty porovnávají jednotlivé algoritmy podle množství *výher*. Výhra daného algoritmu znamená, že v rámci daného grafu našel nejmenší vrcholové pokrytí. Vítěznému algoritmu je přidělen hlas. Pokud vyhraje více než jeden algoritmus, rozdělí se hlas rovnoměrně mezi všechny výherce. Tento způsob porovnání byl zvolen z důvodu, že nemáme k dispozici optimální vrcholové pokrytí a hlavním cílem je vybrat nejlépe fungující algoritmus pro danou skupinu grafů. Experimenty jsou prováděny nad grafy o velikosti 20 ve třech třídách hustoty kde D je rovno $0.0 - 0.1$, $0.5 - 0.6$ a $0.9 - 1.0$. Na základě tabulky 4 lze předpokládat, že chování pro větší/menší počet vrcholů by bylo podobné.

Grafy na obrázku 3 prezentují výsledky soutěže pro algoritmy bez optimalizací. Pro řídké a středně husté grafy dominuje algoritmus GIC. V rámci hustých grafů se rozdíly zmenšují, což lze zdůvodnit tím, že pokud je

| Počet vrcholů | 10 | | | 20 | | | 50 | | | 100 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 - 0.1 | 1.4 | 1.4 | 2.6 | 5.1 | 5.2 | 8.1 | 19.0 | 19.6 | 27.1 | 54.4 | 55.8 | 68.4 |
| 0.1 - 0.2 | 3.3 | 3.4 | 5.3 | 9.6 | 9.9 | 13.2 | 33.0 | 33.5 | 39.1 | 77.6 | 79.1 | 86.7 |
| 0.5 - 0.6 | 6.0 | 6.2 | 7.4 | 15.2 | 15.4 | 17.2 | 43.8 | 44.1 | 46.4 | 92.6 | 93.1 | 95.8 |
| 0.8 - 0.9 | 7.3 | 7.3 | 8.2 | 17.1 | 17.3 | 18.3 | 46.5 | 46.7 | 48.0 | 96.1 | 96.3 | 97.7 |
| 0.9 - 1.0 | 7.7 | 7.7 | 8.4 | 17.9 | 17.9 | 18.6 | 47.4 | 47.5 | 48.3 | 97.2 | 97.2 | 98.4 |
| Algoritmus | GIC | MDG | ME | GIC | MDG | ME | GIC | MDG | ME | GIC | MDG | ME |

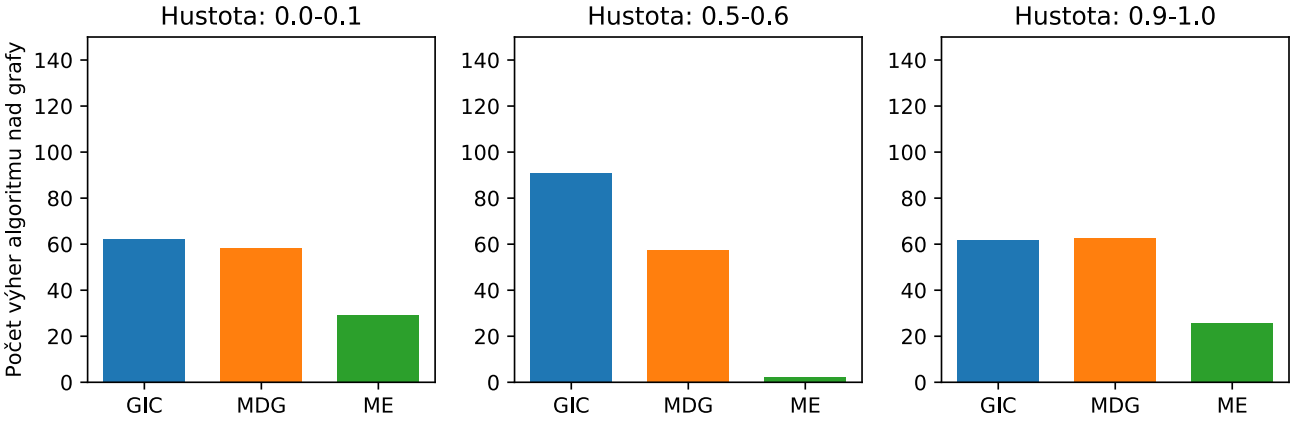
Tabulka 1: Ukázka průměrných velikostí vrcholového pokrytí pro jednotlivé algoritmy v různých třídách hustoty (*vlevo*) a pro grafy s různými počty vrcholů.




Obrázek 3: Ukázka výsledků *soutěže* pro algoritmy bez zapnutých optimalizací.

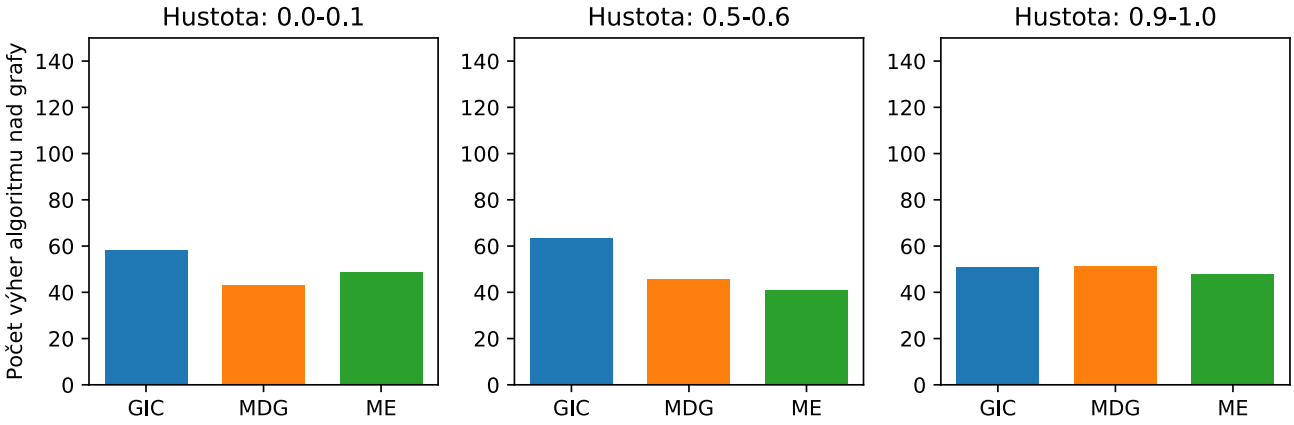
graf velmi hustý je jen malá pravděpodobnost, že některý z vrcholů nebude ve vrcholovém pokrytí tzn. možnost špatného pokrytí je nízká a každý algoritmus vrátí uspokojující pokrytí. Algoritmus ME je značně horší než ostatní, zvláště pak pro grafy o nižších hustotách.

Grafy na obrázku 4 prezentují výsledky soutěže pro algoritmy s optimalizací *pendant*. Tato optimalizace má vliv zejména na řídké grafy. Pro hustší grafy je velmi nepravděpodobné, aby obsahovaly vrcholy, které mají pouze jednoho souseda tzn. pravděpodobnost uplatnění této optimalizace je velmi malá.

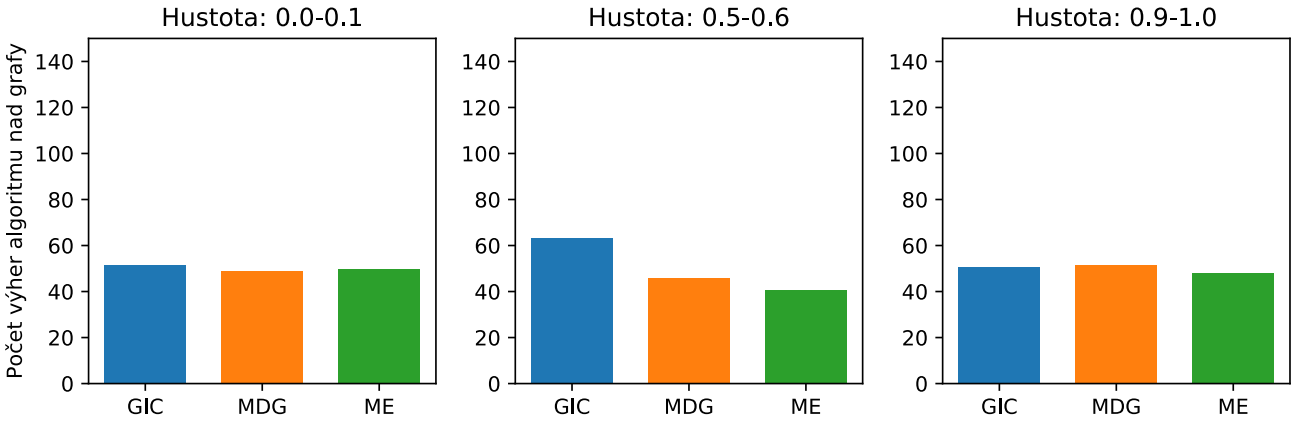



Obrázek 4: Ukázka výsledků *soutěže* pro algoritmy se zapnutou optimalizací *pendant*.

Grafy na obrázku 5 prezentují výsledky soutěže pro algoritmy s optimalizací *redundant*. Dopad této optimalizace je nejznatelnější a to ve všech třídách hustoty. Patrné je zejména zlepšení algoritmu ME, který s touto optimalizací dává srovnatelné výsledky s algoritmem MDG.




Obrázek 5: Ukázka výsledků *soutěže* pro algoritmy se zapnutou optimalizací *redundant*.




Obrázek 6: Ukázka výsledků *soutěže* pro algoritmy se zapnutou optimalizací *pendant* a *redundant*

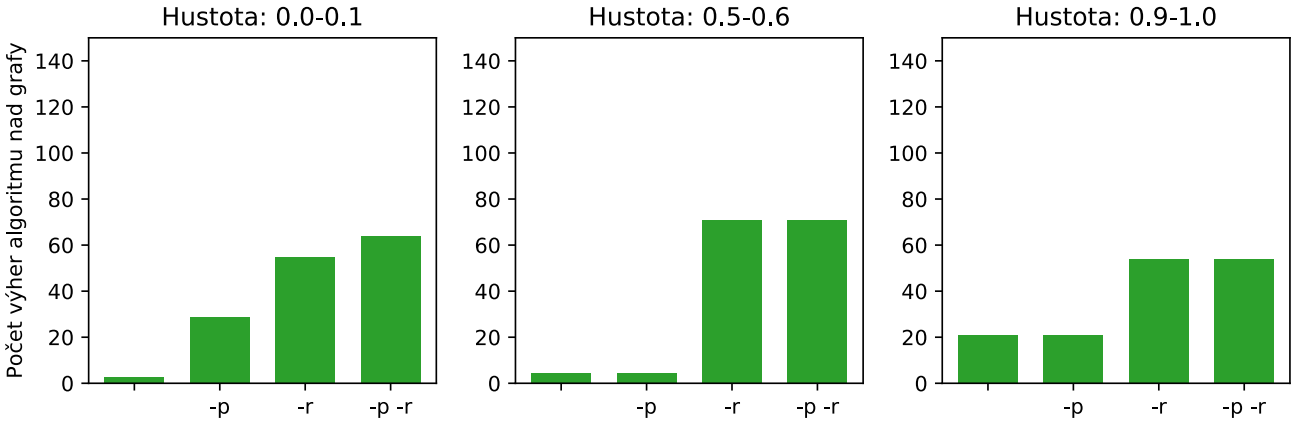
Grafy na obrázku 6 prezentují výsledky soutěže pro algoritmy s optimalizací *pendant* i *redundant*. Oproti grafům na obrázku 5 je pozorovatelný rozdíl pouze u řídkých grafů, kde dojde téměř k úplnému srovnání jednotlivých algoritmů. Tato skutečnost potvrzuje vhodnost použití optimalizace *pendant* pro řídké grafy i pokud je použita optimalizace *redundant*.

Jednotlivé optimalizace nemají na algoritmus GIC žádný vliv, z tohoto důvodu neuvádíme graf potvrzující tuto skutečnost. Vliv na algoritmus MDG je znatelný, ale malý. Vliv na algoritmus ME je graficky znázorněn na obrázku 7, tento vliv je nejvíce znatelný, toto rovněž odpovídá předchozím experimentům.

# 5 Závěr

Tento projekt porovnává tři heuristiky a dvě optimalizace pro řešení problému minimálního vrcholového pokrytí grafu. Experimenty jasně potvrzují výsledky z článku [1] a to sice, že algoritmus GIC je většinou nejlepším. Algoritmus MDG má velmi podobnou úspěšnost a průměrná velikost pokrytí, které nalezá je velmi blízká algoritmu GIC. Nevýhodou těchto algoritmů je nezaručení konstantního aproximačního koeficientu tzn. existují specifické grafy pro které tyto algoritmy vrátí velmi špatné pokrytí. Pro algoritmus ME je tento koeficient rovný dvěma, algoritmus má však téměř vždy značně horší výsledky než předchozí dva a zároveň má asymptoticky horší časovou i prostorovou složitost.

První optimalizace algoritmů spočívá v předzpracování. Před samotným spuštěním algoritmu jsou do výsledného pokrytí přidány sousední vrcholy vrcholů, které mají pouze jednoho souseda. Druhá optimalizace zpracovává výsledné vrcholové pokrytí tak, že odebere zbytečné vrcholy v tomto pokrytí. Optimalizace mají značný vliv na algoritmus ME. Na algoristmus GIC není vliv prakticky žádný a na algoritmus MDG velmi malý.




Obrázek 7: Vliv optimalizací na kvalitu řešení algoritmu ME.

Pokud nám jde hlavně o rychlost výpočtu a průměrné nejlepší výsledky je vhodné použít algoristmus GIC. Pokud nám naopak nevadí delší doba výpočtu a vyžadujeme záruku určité kvality řešení, pak je vhodné použít algoritmus ME s oběma optimalizacemi.

## Reference

[1] Francois Delbot and Christian Laforest. Analytical and experimental comparison of six algorithms for the vertex cover problem. *ACM Journal of Experimental Algorithms - JEA*, 15, 03 2010.

[2] Steven S. Skiena. *The Algorithm Design Manual*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

[3] Sonika Dahiya. A new approximation algorithm for vertex cover problem. *Proceedings - 2013 International Conference on Machine Intelligence Research and Advancement, ICMIRA 2013*, pages 472–478, 10 2014.