Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

# Aproximačné algoritmy na určenie priesečníkového čísla grafov

(bakalárska práca)

Martin Námešný

**Vedúci:** doc. RNDr. Rastislav Královič, PhD

Bratislava, 2008

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

.....

Ďakujem svojmu školiteľovi doc. RNDr. Rastislav Královič, PhD za cenné rady a postrehy, ktorými mi pomohol pri písaní tejto práce.

**Abstrakt**

V tejto práci sa venujem heuristikám, ktoré minimalizujú lineárne priesečníkové číslo a lineárne priesečníkové číslo s fixovanou permutáciou vrcholov. V prvej časti popisujem princíp a popis algoritmov a v ďalšej časti porovnávam jednotlivé heuristiky podľa výsledkov, ktoré dosiahli na niekoľkých triedach grafov. V práci sú použité algoritmy rôznych druhov od jednoduchých greedy algoritmov, cez neurónové siete a semidefinitné programovanie až po genetický algoritmus a simulované žíhanie.

**Kľúčové slová** lineárne priesečníkové číslo, fixované lineárne priesečníkové číslo, heuristiky, aproximácie

# Obsah

# Kapitola 1

# Úvod

Cieľom tejto bakalárskej práce je porovnať rôzne aproximačné algoritmy na určenie priesečníkového čísla grafov. Minimalizácia priesečníkového čísla nakreslenia grafu je jeden z najstarších problémov automatizovaného kreslenia grafov a návrhu VLSI. Počet priesečníkov má veľký vplyv na estetický vzhľad grafu a tak isto čitateľnosť.

Problém priesečníkového čísla bol prvý krát uvedený Pálom Turánom, ktorý si kládol otázku: "Aké je priesečníkové číslo $cr(K_{n,m})$ úplného bipartitného grafu?" Garey a Johnson ukázali[18], že všeobecný rozhodovací problém: "Pre daný graf $G$ a prirodzené číslo $k$ rozhodnúť či $cr(G) \leq k$" je NP-úplný problém.

Kvôli komplexnosti všeobecného problému sa študovali rôzne varianty ako napríklad konvexné priesečníkové číslo, lineárne priesečníkové číslo, book crossing number ale aj pre tieto zostáva problém stále NP-ťažký. Pravdepodobnosť, že sa nájde polynomiálny algoritmus je veľmi malá, ale prax vyžaduje riešenia týchto problémov. Preto sa rozvíjajú rôzne heuristiky a aproximácie, ktoré dodávajú aspoň dobré odhady a riešenia.

## 1.1 Definície

Nech graf $G = (V(G), E(G))$, kde $V(G)$ je konečná množina vrcholov grafu a $E(G)$ je množina hrán grafu, pričom $E(G) \subseteq [V(G)]^2$. Pre ľubovoľnú množinu $A$ je množina $[A]^2$ množinou 2-prvkových podmnožín množiny $A$. $G$ je neorientovaný graf bez násobných hrán a slučiek. Počet vrcholov označme $n = |V(G)|$ a počet hrán $m = |E(G)|$. Pre daný graf $G$, priesečníkové číslo grafu, ktoré sa značí aj $cr(G)$, je najmenší počet priesečníkov hrán s ktorými môže byť nakreslený do roviny.

Planárne grafy majú priesečníkové číslo rovné nule. $cr(G)$ môžeme považovať za mieru neplanárnosti grafu. Existujú rôzne varianty priesečníkového čísla, ktoré kladú na nakreslenie grafu rôzne podmienky. Niektoré z nich sú tieto:

- Rektilineárne priesečníkové číslo $\overline{cr}(G)$. Hrany sú časti priamky, teda úsečky a vrcholy sú vo všeobecnej pozícii.
- Konvexné priesečníkové číslo $v_1(G)$. Vrcholy sú umiestnené na kružnici a hrany sú úsečky. Tiež sa nazýva aj outerplanar crossing number alebo circular crossing number a tiež 1-page book crossing number.
- k-page Book Crossing Number $v_k(G)$. Vrcholy sú umiestnené na spoločnej priesečnici k rovín. Pričom hrany sú polkružnice, ktoré celé ležia[1] práve v jednej z polrovín. Priamka na ktorej ležia vrcholy sa nazýva chrbtica(chrbát knihy) a polroviny sú stránky tejto knihy. Takže hrana je nakreslená na jednej zo stránok. Špeciálny prípad je už spomínané konvexné priesečníkové číslo. Pre nakreslenie grafu $D$ na k-stránkovú knihu, $v_k(D)$ označuje počet priesečníkov hrán v nakreslení $D$. Potom k-page book crossing number grafu $G$ označme $v_k(G)$ a označuje minimálne priesečníkové číslo medzi všetkými k-stránkovými nakresleniami $G$.
- Linear layout crossing number $v_2(G)$. Je inštancia 2-page Book crossing number.
- Fixed linear layout crossing number $v_L(G)$ Je linear layout crossing number ale poradie vrcholov na priamke je pevne dané. Problém minimalizácie fixovaného lineárneho priesečníkového čísla sa niekedy označuje skratkou FLCNP.

V tejto práci sa skúmajú algoritmy na určenie $v_1(G)$, $v_2(G)$ a $v_L(G)$. Prehľad ostatných typov priesečníkových čísel sa dá nájsť v [16].

### 1.1.1 Lineárne priesečníkové číslo

Nakreslenie grafu do roviny nazývame lineárne, ak spĺňa nasledujúce podmienky:

(i) vrcholy ležia na horizontálnej priamke $p$

(ii) hrany sú nakreslené ako polkružnice alebo polelipsy

V prípade fixovaného priesečníkového čísla pridávame podmienky:

(iii) poradie vrcholov na priamke $p$ je pevne dané.

V tejto práci spĺňa lineárne nakreslenie aj ďalšie predpoklady:

(i) hrana nepretína samú seba

(ii) hrany, ktoré majú spoločný vrchol sa nepretínajú

[1] Samozrejme okrem vrcholov, ktoré hrana spája, pretože tie ležia na priesečnici.

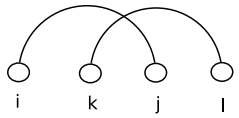
(iii) spoločný bod dvoch hrán, okrem koncových, je priesečník a nie bod dotyku

(iv) žiadne tri hrany nemajú spoločný priesečník

(v) ľubovoľná dvojica hrán sa pretína najviac raz

### 1.1.2 Počet priesečníkov

Očíslujme vrcholy grafu $G$ v poradí v akom ležia na priamke zľava doprava.[2] Vrchol $v \in V$ budeme značiť $v_i$, kde $i$ je poradové číslo vrcholu. Hranu $e = v_i v_j$ budeme značiť ako $e_{i,j}$ pričom $i < j$. Potom hrany $e_{i,j}$ a $e_{k,l}$ sa pretínajú práve vtedy, keď $(i < k < j \wedge l > j) \vee (k < i \wedge i < l < j)$.




Obrázok 1.1: Znázornenie polohy vrcholov pri priesečníku dvoch hrán

Zaveďme funkciu $cross(e_{i,j}, e_{k,l}) = 1$ ak $e_{i,j}$ a $e_{k,l}$ pretínajú inak 0. Celkový počet priesečníkov bude:

$$v_1(G) = \frac{1}{2} \sum_{e_{i,j} \in E} \sum_{e_{k,l} \in E} cross(e_{i,j}, e_{k,l}) \tag{1.1}$$

Pre $v_2(G)$ musíme ešte rozoznávať v ktorej polrovine hrany ležia. Nech $u_{i,j} \in \{0, 1\}$ a $u_{i,j} = 0$ ak hrana $e_{i,j}$ leží v prvej polrovine. Analogicky 1 ak leží v druhej.

$$v_2(G) = \frac{1}{2} \sum_{e_{i,j} \in E} \sum_{e_{k,l} \in E} cross(e_{i,j}, e_{k,l}) * ((u_{i,j} u_{k,l} + (1 - u_{i,j})(1 - u_{k,l}))) \tag{1.2}$$

### 1.1.3 Teoretický horný odhad

Takmer na každú otázku, ktorú sa môžme opýtať ohľadom priesečníkového čísla, nepoznáme odpoveď. Pre účely tejto práce sa v literatúre dajú nájsť niektoré horné odhady pre špeciálne triedy grafov. V niektorých prípadoch aj predpokladaný vzťah pre rovnosť.

$$v_2(K_n) \leq \frac{1}{4} \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{n-2}{2} \right\rfloor \left\lfloor \frac{n-3}{2} \right\rfloor \tag{1.3}$$

[2]Pri $v_2(G)$ sa priamka na ktorej ležia vrcholy najčastejšie znázorňuje vodorovne.

Kde pre $n \leq 10$ bola dokázaná rovnosť. Podrobnosti sa dajú nájsť v [8]. Pre pravidelný torus sa v [9] predpokladá vzťah:

$$v_2(T_{n,n}) = n(n-2) \tag{1.4}$$

# Kapitola 2

# Aproximačné algoritmy

## 2.1 Úvod

Aproximačné algoritmy sú momentálne jediná možnosť ako znížiť počet priesečníkov v nakreslení grafu. Z praktického hľadiska pomáhajú zlepšovať návrhy VLSI, či estetickosť automaticky nakresleného grafu a iné. Z teoretického hľadiska pomáhajú pri zlepšovaní horných odhadov.

## 2.2 Testované typy algoritmov

Na získanie dobrého riešenia ťažkého problému sa používajú rôzne techniky a postupy. V literatúre sa dajú nájsť greedy postupy, algoritmy založené na dynamickom programovaní, bisekcií [3] [1] [10] [19] [21] [23], neurónové siete [12] [17] [11], genetické algoritmy [13], simulované žíhanie [20],.

V tejto práci sú popísané niektoré publikované heuristiky, ktoré boli aj implementované a porovnávané. Z typov algoritmov sú zastúpené greedy, neurónové siete, genetické algoritmy a simulované žíhanie.

## 2.3 Algoritmy pre fixované lineárne priesečníkové číslo

Keďže vrcholy sú v tomto type nakreslenie zafixované, zostáva rozmiestniť hrany do polrovín. Veľkosť prehľadávaného priestoru je rádovo $\mathcal{O}(2^m)$.

### 2.3.1 Greedy metódy

Základné heuristické metódy boli čerpané od Cimikowského [3]. Vo svojej práci uvádza až 8 možných metód. V tejto práci sú rozobrané tieto:

### Znáhodnený greedy

Začína sa s grafom bez hrán a postupne sa do neho pridávajú hrany, ktoré sa pridajú do jednej z polrovín. Pričom poradie pridávania hrán je náhodne. Zistiť, koľko priesečníkov má hrana v prvej alebo druhej polrovine trvá $\mathcal{O}(m)$. Celkový čas behu $\mathcal{O}(m^2)$

**Algorithm 1** Znáhodnený greedy

```
náhodne preusporiadaj poradie hrán
for i = 0 to m − 1 do
    pridaj hranu e_i do tej polroviny, v ktorej spôsobí najmenší prírastok
    priesečníkov
end for
```

### Maximálny planárny podgraf

Maximálny planárny podgraf, ďalej len MPlanar, sa snaží vytvoriť planárne grafy v obidvoch polrovinách. V prvej fáze pridáva hrany do prvej polroviny, ktoré nespôsobia kríženie, tie, ktoré by mohli spôsobiť, sa odložia. V druhej fáze sa odložené hrany umiestňujú takým istým spôsobom do druhej polroviny. Ak potom zostanú nejaké hrany umiestnia sa do tej polroviny, kde spôsobia menej priesečníkov. Celkový čas behu $\mathcal{O}(m^2)$

### Edge length heuristic

Táto heuristika zoradí hrany v nerastúcom poradí podľa dĺžky a postupne ich pridáva do tej polroviny v ktorej spôsobia menší prírastok priesečníkov. Intuícia za týmto prístupom je taká, že dlhšia hrana má väčšiu pravdepodobnosť priesečníka a preto ju treba umiestniť skôr. Celkový čas behu $\mathcal{O}(m^2)$. Ďalej bude mať označenie E-len.

### One page

One page heuristika umiestni všetky hrany do prvej polroviny. Potom nasleduje fáza vylepšovania, v ktorej sa hrana presunie do druhej polroviny, ak to zníži priesečníkové číslo. Hrany na vylepšenie sa berú v nerastúcom poradí lokálnych priesečníkov. Celkový čas behu $\mathcal{O}(m^2)$. Počas prvých testov sa ukázalo,že niekoľkokrát zopakovať zlepšovaciu fázu prináša lepšie výsledky aj keď hrany zostanú v nezmenenom poradí.

### Dĺžka hrany 2

Táto heuristika vychádza z toho že hrany dĺžky $n/2+1$ môžu spôsobiť najviac priesečníkov. Zoradí preto hrany v nerastúcom poradí podľa hodnoty výrazu

**Algorithm 2** MPlanar

1: $E_1 = \emptyset$, $E_2 = \emptyset$, $E_{pom} = \emptyset$
2: **while** neprázdna $E$ **do**
3: odober hranu z $e \in E$
4: **if** ma hrana $e$ priesečník s nejakou hranou z $E_1$ **then**
5: pridaj $e$ do $E_{pom}$
6: **else**
7: pridaj $e$ do $E_1$
8: **end if**
9: **end while**
10: $E = E_{pom}$, $E_{pom} = \emptyset$
11: **while** neprázdna $E$ **do**
12: odober hranu z $e \in E$
13: **if** ma hrana $e$ priesečník s nejakou hranou z $E_2$ **then**
14: pridaj $e$ do $E_{pom}$
15: **else**
16: pridaj $e$ do $E_2$
17: **end if**
18: **end while**
19: **while** neprázdna $E_{pom}$ **do**
20: odober hranu z $e \in E_{pom}$
21: pridaj hranu do $E_1$ alebo $E_2$ podľa toho, kde spôsobí menej priesečníkov.
22: **end while**

**Algorithm 3** Edge length heuristic

1: zostupne usporiadaj hrany podľa dĺžky
2: **for** $i = 0$ to $m - 1$ **do**
3: pridaj hranu $e_i$ do tej polrovinu, v ktorej spôsobí najmenší prírastok priesečníkov
4: **end for**

**Algorithm 4** One Page heuristic

1: umiestni všetky hrany do prvej polroviny
2: zostupne zoraď hrany podľa počtu priesečníkov, ktoré spôsobujú
3: **for** $j = 0$ to 10 **do**
4: **for** $i = 0$ to $m - 1$ **do**
5: umiestni hranu $e_i$ do opačnej polroviny ak to zníži počet priesečníkov
6: **end for**
7: **end for**

$|n/2+1-len|$ kde $len$ je dĺžka hrany. Časová zložitosť je $\mathcal{O}(m^2)$ a označenie E-Len2.

**Algorithm 5** Edge length 2 heuristika

1: zostupne usporiadaj hrany podľa hodnoty výrazu $|n/2+1-len|$
2: **for** $i = 0$ to $m-1$ **do**
3: pridaj hranu $e_i$ do tej polrovinu, v ktorej spôsobí najmenší prírastok priesečníkov
4: **end for**

#### Slope stratégia

Nakreslíme vrcholy na kružnicu a spojme hranami - úsečkami. Ak hrana zviera s horizontálnou osou väčší uhol ako 90 tak ho umiestnim do druhej polroviny. Časová zložitosť je $\mathcal{O}(m)$.

**Algorithm 6** Slope strategy heuristika

1: **for** $i = 0$ to $m-1$ **do**
2: **if** $(x+y < \frac{1}{2}n) \vee (n < x+y < \frac{3}{2}n)$ **then**
3: umiestni hranu $e^i_{x,y}$ do druhej polroviny
4: **else**
5: umiestni hranu $e^i_{x,y}$ do prvej polroviny
6: **end if**
7: **end for**

### 2.3.2 Neurónové siete

Nasledujúce algoritmy využívajú Hopfieldovu neurónovú sieť. V tejto časti nebude $v$ označovať vrchol ale neurón. Hopfieldova sieť je úplne spojená rekurentná jedno vrstvová sieť. Každý neurón je spojený so všetkými ostatnými pomocou synapsií a všetky neuróny sú vstupné aj výstupné. Každá synapsia má svoju váhu $w_{ij} \in R$, pričom tieto váhy sú symetrické $w_{ij} = w_{ji}$, kde $ij$ je synapsia medzi neurónmi $v_i$ a $v_j$. V obidvoch uvedených algoritmoch hrany z grafu $G$ zodpovedajú neurónom. Model obsahuje $m$ neurónov $v_0, v_1, ..., v_{m-1}$. Hrane $e^i_{xy}$ zodpovedá neurón $v_i$.

#### HeSykoraNeural

He, Sýkora používajú vo svojej práci[12] model synchrónnej Hopfieldovej siete s binárnymi neurónmi. Vnútorná hodnota neurónu $v_i \in 0, 1$. Ak $v_i = 1$ ( respektívne $v_p = 0$ ) tak hrana $e_p$ je nad priamkov(respektíve pod priamkov). Pre aktivované neuróny ($v_i = 1$) sa dá priesečníkové číslo

vypočítať takto:

$$v_L^{up}(G) = \sum_{i=0}^{m-1} \sum_{j=i+1}^{m-1} cross(e_{xy}^i, e_{zw}^j) * v_i v_j$$

Podobne aj pre neaktívne neuróny(hrany pod priamkov) sa dá priesečníkové číslo vypočítať takto:

$$v_L^{down}(G) = \sum_{i=0}^{m-1} \sum_{j=i+1}^{m-1} cross(e_{xy}^i, e_{zw}^j) * \bar{v_i}\bar{v_j}$$

kde $\bar{v_i} = 1 - v_i$. Z toho vyplýva energetická funkcia siete:

$$E = A \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} cross(e_{xy}^i, e_{zw}^j) * (v_i v_j + \bar{v_i}\bar{v_j})$$

Hirsch[14] popísal neurónovú sieť ako nelineárny dynamický systém a nazval ho neurodynamika. Predpokladajme dynamický systém s $m$ stavovými premennými $v_0, v_1, ..., v_{m-1}$, potom môžme písať motion equation ako $\frac{du_i}{dt} = -\frac{\partial E}{\partial v_i}$, kde $u_i$ a $v_i$ sú vstup a výstup $i$-teho neurónu. Z toho dostávame:

$$\frac{du_i}{dt} = -A \sum_{j=0, j\neq i}^{m-1} cross(e_{xy}^i, e_{zw}^j)(2v_j - 1)$$

Týmto vzťahom je popísaná celá dynamika systému. Avšak Takefuji [22] dokázal, že stav binárneho modelu konverguje do lokálneho minima. Aby sa zabránilo uviaznutiu v tomto minime, aplikuje sa učiaci algoritmus, ktorý v tomto prípade reprezentuje hill-climbing prístup. Jedno kolo výpočtu je, keď sa prepočítajú hodnoty všetkých neurónov. Výpočet nových hodnôt neurónov sa robí sekvečne. Počet kôl je obmedzený pomocou parametra $max\_iterations$. Sieť často krát dosiahne ustálený stav aj skôr. Túto heuristiku budem označovať v testoch HeNeural.

| parameter | hodnota |
|---|---|
| $max\_iterations$ | 100 |
| $A$ | 2,0 |

Tabuľka 2.1: Parametre použité pri testovaní algoritmu HeNeural

**Algorithm 7** He Sýkora neural algoritmus

```
1: k = 0, v_L = MAX_INT
2: while k < max_iterations do
3:    t = 0
4:    náhodne inicializuj stavy neurónov v_0, v_1, ..., v_{m-1}
5:    vypočítaj hodnotu E
6:    repeat
7:       for i = 0 to m - 1 do
8:          vypočítaj Δu_i pre Δt = 1 pomocou motion equation
9:          vypočítaj u_i(t + 1) pomocou Δu_i
10:         if u_i > 0 then
11:            v_i = 1
12:         else
13:            v_i = 0
14:         end if
15:      end for
16:      vyrátaj energiu E
17:      t = t + 1
18:   until ΔE = 0
19:   cur_v_L = calculate_crossings(G, v_0, v_1, ..., v_{m-1})
20:   if cur_v_L < v_L then
21:      v_L = cur_v_L
22:   end if
23:   k = k + 1
24: end while
```

### Gradient Ascent Learning

Rong-long Wang a Zheng Tang publikovali článok [17], v ktorom využívajú hopfieldovu neurónovú sieť. Narozdiel oproti predchádzajúcemu modelu využívajú štandardnú formu motion funkcie a majú inú techniku učenia. Každej hrane je priradený jeden neurón, ktorého stav indikuje umiestnenie hrany. Ak má neurón hodnotu 1 tak je hrana umiestnená v prvej polrovine a ak má 0 tak je v druhej polrovine. Priesečníkové číslo môže byť spočítané

takto:

$$cr_L(G) = \frac{1}{2}\sum_{i=0}^{m-1}\sum_{j=0}^{m-1} cross(e^i_{xy}, e^j_{zw}) * (v_i v_j + (1-v_i)(1-v_j))$$

Energetickú funkciu pre FLCNP môžme zapísať takto:

$$e = \frac{A}{2}\sum_{i=0}^{m-1}\sum_{j=0}^{m-1} cross(e^i_{xy}, e^j_{zw}) * (v_i v_j + (1-v_i)(1-v_j))$$

Štandardná energetická funkcia hopfieldovej siete má tvar

$$E = -\frac{1}{2}\sum_{i=0}^{m-1}\sum_{j=0}^{m-1} w_{ij} v_i v_j - \sum_{i=0}^{m-1} h_i v_i$$

kde $w_{ij}$ je váha synapsie medzi neurónmi $v_i$ a $v_j$. V hopfieldovej sieti platí $w_{ii} = 0$ a $w_{ij} = w_{ji}$. $h_i$ sa nazýva aj externý vstup neurónu alebo aj prah. Aby sme vyriešili problém minimalizácie počtu priesečníkov dajme do rovnosti predchádzajúce dva vzťahy. Z nich dostaneme hodnoty $w_{ij}$ a $h_i$

$$w_{ij} = \begin{cases} -2A * cross(e^i_{xy}, e^j_{zw}) & \text{pre } i \neq j \\ 0 & i = j \end{cases}$$

$$h_i = \sum_{j=0}^{m-1} cross(e^i_{xy}, e^j_{zw})$$

Motion rovnica je ako v predchádzajúcom prípade odvodená zo vzťahu $\frac{du_i}{dt} = -\frac{\partial E}{\partial v_i}$.

$$\frac{du_i}{dt} = \sum_{j=0}^{m-1} w_j v_j + h_i$$

Sigmoidná funkcia je použitá ako vstupno výstupná funkcia.

$$v_i = \frac{1}{1 + e^{(-u_i/T)}}$$

kde T je parameter, ktorý sa nazýva teplota. Hopfield [15] ukázal, že takáto sieť sa dá použiť ako aproximačná metóda na riešenie 0-1 optimalizačných problémov. Pretože váhy sú symetrické tak sieť konverguje k minimu energetickej funkcie. Navyše ak $w_{ii} = 0$ tak hodnoty neurónov v lokálnom minime budú blízko vektoru $\{0,1\}^m$.

Pri výpočte neurónovej siete sa energetická funkcia približuje k lokálnemu alebo globálnemu minimu. Keď ho dosiahne tak hodnoty neurónov sa ďalším výpočtom už nezmenia. Keďže neurónový model priamo korešponduje s FLCNP, tak takto získané riešenie je prípustné. T.j neurónová sieť

neskončí v stave, ktorý by nevyjadroval nejaké rozloženie hrán. Avšak ak sieť dospeje do lokálneho minima, tak získané priesečníkové číslo nemusí byť dostatočne malé. Preto má algoritmus aj učiacu fázu, ktorá mu má pomôcť uniknúť z lokálneho minima.

Učiaci algoritmus sa aplikuje na váhy $w_{ij}$ a prahy $h_i$, pretože tie určujú vlastnosti energetickej funkcie.

$$\Delta w_{ij} = -pv_iv_j$$

$$\Delta h_i = -qv_i$$

kde, p a q sú malé kladné konštanty. Minimalizácia priesečníkového čísla bohužiaľ nie je typ problému, pri ktorom vieme, že sme dosiahli optimálne riešenie [1], preto musíme počet opakovaní výpočtovej a učiacej fázy dopredu obmedziť. Pri mojich experimentoch som ho nastavili na 10, tak ako pôvodní autori. Heuristiku budem označovať ako GANeural.

**Algorithm 8** Gradient Ascent Learning

**Require:** pole *neurons* obsahuje hodnoty neurónov, pole $start\_w[i][j]$ obsahuje váhy medzi neurónmi $v_i$,$v_j$, $start\_h[i]$ je pole prahov

1: $w = start\_w$ , $h = start\_h$
2: náhodne inicializuj stavy neurónov hodnotami z intervalu $(0, 1)$
3: update_neurons()
4: $min\_cross$ = calc_current_crossings($neurons$,$G$)
5: **for** $learn\_times = 0$ to $learn\_limit$ **do**
6: $w = start\_w$ , $h = start\_h$
7: learn_net()
8: update_net($w$,$h$)
9: update_net($start\_w$,$start\_h$)
10: $cur\_cross$ = calc_current_crossings($neurons$,$G$)
11: **if** $cur\_cross < min\_cross$ **then**
12: $min\_cross = cur\_cross$
13: **end if**
14: **end for**

Sieť najprv počíta s nezmenenými váhami, keď dosiahne ustálený stav, aplikuje sa učiaci algoritmus. Vykoná sa nový výpočet s novými váhami a po ňom ešte raz s pôvodnými váhami, aby sa zabránilo posunu globálneho minima.

[1] Samozrejme ak dospejeme nulovému priesečníkovému číslu môžme skončiť

| parameter | hodnota |
|---|---|
| $learn\_limit$ | 10 |
| $A$ | 1,0 |
| $T$ | 1,0 |
| $p$ | 0,5 |
| $q$ | 0,8 |

Tabuľka 2.2: Parametre použité pri testovaní algoritmu GANeural

**Algorithm 9** Gradient Ascent Learning - update_net

**Ensure:** update_net(w,h) - prepočíta vnútorné hodnoty neurónov

```
1: change = true
2: while change = true do
3:    change = false
4:    for i = 0 to m − 1 do
5:       sum = 0
6:       for j = 0 to m − 1 do
7:          sum+= w[i][j] ∗ neurons[j]
8:       end for
9:       sum+= h[i] + neurons[i]
10:      y = 1/(1+e^(−sum/T))
11:      if y <> sum then
12:         change = true
13:      end if
14:      neurons[i] = y
15:   end for
16: end while
```

Procedúra update_net(w,h) - vykonáva vlastný výpočet na neurónovej sieti, až kým sa sieť neustáli. To, že sieť konverguje nám zaručí Hopfieldova veta. Pôvodný algoritmus bol navrhnutý ako paralelný, tento variant však ráta nové hodnoty neurónov postupne. Časová zložitosť je $\mathcal{O}(m^2)$.

**Algorithm 10** Gradient Ascent Learning - learn_net

**Ensure:** learn_net() - upraví pole $w$ a $h$

```
1: for i = 0 to m − 1 do
2:    for j = 0 to m − 1 do
3:       w[i][j] = start_w[i][j] − p ∗ neurons[i] ∗ neurons[j]
4:    end for
5:    h[i] = start_h[i] − q ∗ neurons[i]
6: end for
```

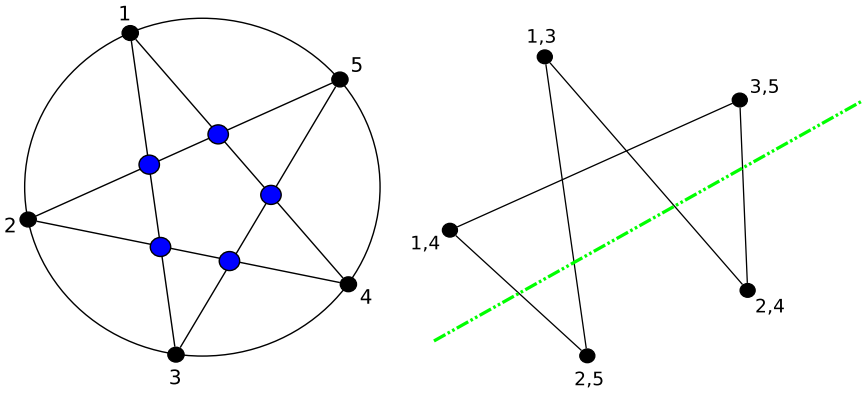
Časová zložitosť metódy learn_net je $\mathcal{O}(m^2)$.

### 2.3.3 Aproximácia pomocou MaxCut

#### Cimikowského redukcia na problém maximálneho rezu

Robert Cimikowski a Brendan Muney publikovali algoritmus [4], ktorý je založený na redukcií problému na problém maximálneho 2-rezu. Pre daný graf $G = (V, E)$, nájsť maximálny k-rez grafu znamená nájsť k - disjunktných podmnožín $S_1, S_2, \cdots, S_k$ množiny vrcholov grafu $G$ tak, že počet hrán, ktoré majú koncové vrcholy v rôznych množinách je maximálny. Problém je analogický s problémom nájsť maximálny k-partitný podgraf.

Problém minimalizácie fixovaného priesečníkového čísla môže byť transformovaný na maximálny 2-rez grafu takto: Nech $G = (V, E)$ je graf pre ktorý chceme riešiť FLCNP. Umiestnime všetky hrany do 1. polroviny. Potom množinu $C_G$ tvoria dvojice hrán, ktoré sa pretínajú. Presnejšie $(e_i, e_j) \in C_G \Leftrightarrow cross(e_i, e_j) = 1$. Nech graf $G' = (V', E')$. Potom každej hrane $e_i \in E$ priradíme vrchol $v_i \in V'$ a pre každú dvojicu hrán $(e_i, e_j) \in C_G$ priradíme hranu v $G'$ $v_i v_j \in E'$. 2-rez v $G'$ rozdelí vrcholy do množín $S_1, S_2$. Pričom vrcholy z $S_i$ zodpovedajú hranám v $G$, ktoré sa majú zobraziť v $i$-tej polrovine. Všimnime si, že priesečník dvojice hrán grafu $G$ vzniká iba v tom prípade, keď hrana z $G'$ nepatrí do rezu, t.j obidva koncové vrcholy patria do rovnakej množiny.




(a) $G$ (b) $G'$

Obrázok 2.1: Príklad transformácie FLCNP na maximálny rez. (a) zobrazuje graf $K_5$, modré body vyznačujú priesečníky. (b) Zobrazuje $G'$ a zelená čiara vyznačuje maximálny rez. Mimo rezu zostala iba jedna hrana a teda graf $K_5$ sa dá nakresliť iba s jedným priesečníkom($v_L(K_5) = 1$)

Problém maximálneho rezu patrí medzi 21 Karpových NP-úplných problémov. Goemans a Williamson[7] vyvinuli aproximáciu pre maximálny rez. Základná myšlienka je takáto: Pre každý vrchol $v_i \in V'$ definujme hodnotu $x_i$, ktorá sa rovná 1 ak $v_i$ patrí do $S_1$ a -1 ak do $S_2$. Hľadaniu maximálneho

rezu potom zodpovedá celočíselný kvadratický program

$$Max\frac{1}{2}\sum_{(v_i,v_j)\in E'}(1-x_ix_j)$$
$$x_i \in \{-1,1\} \quad \forall v_i \in V'$$

To je základný problém, ktorý relaxujeme na semidefinitné programovanie. Respektíve relaxujeme celočíselnú premennú $x_i$ na viacrozmerný vektor $u_i$ jednotkovej normy. Vektor $u_i$ patrí $m$-rozmernej jednotkovej guli $S_m$.

$$Max\ \frac{1}{2}\sum_{(v_i,v_j)\in E'}(1-u_i.u_j)$$
$$u_i \in S_m \quad \forall v_i \in V'$$

$u_i.u_j$ značí skalárny súčin. Popis algoritmu:

**Algorithm 11** Cimikowského heuristika $v_L(G)$ pomocou aproximácie maximálneho rezu

1: Preveď vstupný graf $G$ na $G'$
2: Vyrieš SDP a získaj optimálne vektory $u_i$
3: **for** $i = 1$ to 50 **do**
4: nech r je náhodný vektor z $S_m$
5: nech $S_1 = \{i|u_i.r \geq 0\}$ a $S_2 = S - S_1$
6: vyrátaj počet priesečníkov keď rozdelím hrany $S_1$ a $S_2$
7: **end for**
8: vráť najlepšie riešenie

Pri implementácii bola použitá knižnica csdp na riešenie úlohy semidefinitného programovania. Knižnica sa dá nájsť v [5]. Časová zložitosť je závislá na počte hrán vstupného grafu. Výpočtovo najnáročnejšia časť je v riešení úlohy semidefinitného programovania. Podľa [2] vo všeobecnosti počet iterácií nepresiahne 30, čo sa potvrdilo aj pri mojich testoch. Pri transformovaní problému sa počíta matica rozmeru $m \times m$ a počet podmienok je tiež $m$, kde $m$ je počet hrán grafu $G$, respektíve počet vrcholov $G'$. Matice podmienok majú len jeden 1 nenulový prvok. Potom časová zložitosť je $\mathcal{O}(m^3)$

## 2.4 Lineárne priesečníkové číslo

Výpočet lineárneho priesečníkového čísla sa skladá z dvoch podproblémov - nájdenie permutácie vrcholov a rozmiestnenie hrán do polrovín, pričom v obidvoch prípadoch treba minimalizovať počet priesečníkov. Nanešťastie obidva podproblémy sa ukázali byť NP-ťažké. Prvý spomínaný minimalizuje

konvexné priesečníkové číslo. A druhý minimalizuje lineárne priesečníkové číslo s fixovaným poradím vrcholov. Jednou z možných metód je skombinovať heuristiky pre obidva typy priesečníkových čísel. Druhou možnosťou je skúmať obidva podproblémy ako celok. V tejto kapitole sú popísané algoritmy, ktoré sú príkladom druhej možnosti.

### 2.4.1 Genetický algoritmus

Genetické algoritmy sa ukázali byť ako dobré aproximačné prostriedky pri výpočtovo nezvládnuteľných problémoch. Algoritmus [13] funguje nasledovne.

Jeden člen populácie - chromozóm reprezentuje jedno nakreslenie grafu $G$. Skladá sa z permutácie $\pi$ a m-bitového reťazca $u \in \{0,1\}^m$. Každý chromozóm ma svoju fitness hodnotu, ktorá sa dá vypočítať ako $\frac{1}{1+v_2^2}$. Čím je väčšia fitness hodnota tým je menšie priesečníkové číslo. Z populácie sa vygeneruje nová populácia pomocou operátorov selekcie, mutácie a crossover.

Selekcia - selekčný operátor vyberá chromozómy na následný crossover a mutáciu. Každé nakreslenie má určitú pravdepodobnosť, že bude vybrané. Pravdepodobnosť nakreslenia $prop(D_i)$ sa vypočíta takto:

$$prop(D_i) = \frac{\frac{1}{1+v_2^2(D_i)}}{\sum_{j=0}^{popSize-1} \frac{1}{1+v_2^2(D_j)}} * 100$$

**Algorithm 12** Genetický algoritmus - Operátor Selekcie

**Require:** *pop* - je populácia, $pop[i]$ - i-tý člen populácie
**Ensure:** vracia index chromozómu, ktorý bol vybraný

```
1: r = random()mod100
2: lastprob = pop[0].prop
3: i = 0
4: while lastprob < r do
5:     i++
6:     lastprob+ = pop[i].prop
7: end while
8: return i;
```

Procedúra crossover má na vstupe dva chromozómy a ako výstup vracia takisto dva chromozómy. Rekombinácia sa aplikuje na obidve časti chromozómu. V prípade rozmiestnenia hrán sa vyberie náhodne dlhý úsek bitového reťazca a tie sa vymenia. V prípade permutácií sa to urobí obdobne ale po výmene úsekov sa zvyšné prvky permutácie dopočítajú podľa toho ktoré ešte nie sú použité a v akom poradí boli v pôvodnej permutácií. Po rekombinácii sa s určitou pravdepodobnosťou aplikuje mutácia. Pravdepodobnosť mutácia bola nastavená na 40%.

| parameter | hodnota |
|---|---|
| $popSize$ | 16 |
| pravd. mutácie | 40% |
| počet generácií | $min(3n + 3m + 100, 1000)$ |

Tabuľka 2.3: Parametre použité pri testovaní genetického algoritmu

Na začiatku sa vygeneruje náhodná permutácia veľkosti $popSize$. Každá ďalšia generácia sa generuje tak, že pomocou selekcie sa vyberú dva chromozómy na crossover, z čoho vznikne nová dvojica na ktorú sa aplikuje mutácia. Následne sú pridaní do novej generácie. Toto sa opakuje až kým nová populácia nedosiahne veľkosť $popSize$. Na konci sa najhorší jedinec, s najmenšou fitness hodnotou, vymení za najlepšieho jedinca predchádzajúcej generácie.

Dôležitým faktorom genetického algoritmu sú jeho terminačné kritériá. V tomto prípade sa $duration$, počet generácií, stanoví ako $min(3n + 3m + 100, 1000)$ alebo priesečníkové číslo nejakého nakreslenia v populácií je 0.

**Algorithm 13** Genetický algoritmus

```
1: vytvor náhodnú populáciu
2: for i = 1 to duration do
3:    vyrátaj fitness pre každého člena populácie
4:    while size < popSize do
5:       c1 = selector()
6:       c2 = selector()
7:       crossover(c1, c2)
8:       aplikuj mutácie na nové chromozómy
9:       pridaj c1, c2 do novej populácie
10:      size++
11:   end while
12:   vymeň najhoršieho z novej za najlepšieho zo starej populácie
13: end for
```

### 2.4.2 Simulované žíhanie

Ďalším z algoritmov založených na stochastickom prístupe je simulované žíhanie[20]. Je založený na fyzikálnom princípe žíhania tuhého telesa, kde sa postupným ochladzovaním odstraňujú jeho vnútorné defekty. Základná idea algoritmu je začať z počiatočného štádia a postupne ho vylepšovať. Vylepšovanie prebieha náhodne - náhodne sa vyberie hrana, ktorá sa presunie do opačnej polroviny alebo sa vyberú dva vrcholy, ktoré si vymenia pozície v permutácii. Ak je nové riešenie lepšie tak sa príjme. Ak nie je lepšie, tak sa príjme iba z určitou pravdepodobnosťou, ktorá závisí od teploty. Na začiat-

| parameter | hodnota |
|---|---|
| $t_0$ | 1 |
| $t_1$ | 0,2 |
| $\alpha$ | 0,99 |

Tabuľka 2.4: Parametre použité pri testovaní simulovaného žíhania

ku majme riešenie $s_0 = (\pi, u)$, ktoré sme získali nejakou inou heuristikou. Autori použili Baur a Brandes heuristiku pre konvexné priesečníkové číslo a Cimikowského edge length heuristiku na rozmiestnenie hrán. Pri testovaní bola použitá heuristika AVSDF+GANeural.

## 2.5 Konvexné priesečníkové číslo

Konvexné priesečníkové číslo je veľmi úzko späté s 1-page priesečníkovým číslom. Obe sú určené permutáciou vrcholov a obe majú pri rovnakej permutácií rovnaké hodnoty. Často sa v literatúre chápu ako synonymá.

Existujú dve základné prístupy ako minimalizovať počet priesečníkov [10]:

- minimalizovať dĺžky hrán v grafe. Problém sa ukázal byť NP-ťažký[19]. V prípadne konvexného priesečníkového čísla minimalizujeme tento výraz:
$$\sum_{e_{xy} \in E} min(|x - y|, n - |x - y|)$$
- maximalizovať počet hrán ležiacich na kružnici.

Algoritmus navrhnutý Mäkinenom[19] je ukážkou prvého prístupu. Maximalizovať hrany ležiace na kružnici sa nachádza v[21]. Optimalizácia pomocou neurónovej siete sa dá nájsť v [11]. Pre účely tejto práce boli implementované a testované heuristiky od Baur a Brandes, ktorú budem označovať BB a Adjacent Vertex with Smallest Degree First, ktorá má označenie AVSDF.

### 2.5.1 Baur a Brandes

Baur a Brandes [1] vyvinuli heuristiku, ktorá sa skladá z dvoch fáz. V prvej fáze sa pridávajú vrcholy na jednu alebo druhú stranu zoznamu už umiestnených vrcholov. Takýto postup má tri stupne voľnosti: prvý umiestnený vrchol, poradie pridávania vrcholov a na ktorú stranu pridať vrchol. V druhej fáze aplikovali lokálnu optimalizáciu, ktorú nazývajú sifting. Každý vrchol sa posúva pozdĺž pevného zoznamu až kým sa nenájde jeho najlepšia pozícia (z hľadiska počtu priesečníkov).

**Algorithm 14** Simulované žíhanie pre $v_2(G)$

**Require:** nastav $t_0$ a $t_1$, koeficient chladenia $\alpha$ a parameter $r$

```
t = t_1
s = s_0
while t ≥ t_0 and calc_current_crossings(s, G) do
  while c ≤ 2r and calc_current_crossings(s, G) do
    c = c + 1
    if c ≤ r then
      vyber náhodnú hranu a umiestni do opačnej polroviny - u'
      cur_s = (π, u')
    else
      vyber náhodnú dvojicu vrcholov
      a vymeň ich pozície v permutácií - π'
      cur_s = (π', u)
    end if
    δ = calc_current_crossings(cur_s, G)
    - calc_current_crossings(s, G)
    if δ < 0 then
      s = cur_s
    else
      vyber náhodné číslo i ∈< 0, 1 >
      if i < e^{-δ/t} then
        s = cur_s
      end if
    end if
  end while
  t = αt
  c = 0
end while
```

### Prvá fáza

Základná myšlienka je v celku jednoduchá. Začať so zoznamom, ktorý obsahuje len jeden vrchol a postupne pridávať ostatné vrcholy na začiatok alebo koniec zoznamu. Počas behu algoritmu sú niektoré vrcholy umiestnené a niektoré nie. Hranu, ktorá spája umiestnený vrchol a ešte neumiestnený vrchol, budeme nazývať otvorená hrana a hranu medzi dvomi umiestnenými vrcholmi budeme volať zatvorená. Baur a Brandes experimentovali s rôznymi kritériami na odstránenie stupňov voľnosti. Najúspešnejšie boli tieto:

1. prvý umiestnený vrchol je vybraný náhodne - pri experimentoch nemal veľký vplyv na priesečníkové číslo.

2. Výber vrchola na pridanie do zoznamu: V každom kroku je vybratý

vrchol s najväčším počtom už umiestnených susedov ak nastane zhoda tak je vybratý ten, ktorý má menej ešte neumiestnených susedov.

3. Vybratý vrchol sa pridá na ten koniec zoznamu, na ktorom hrany, ktoré práve zatvára, vyprodukujú menší počet priesečníkov.

Prvá fáza môže byť implementovaná v čase $\mathcal{O}((n+m)log\ n)$

#### Druhá fáza

Princíp siftingu je priebežne upravovať hodnotu objektívnej funkcie tak, aby jej hodnota bola aktuálna a nemusela by byť stále prepočítavaná, čo stojí $\mathcal{O}(m^2)$ Každý vrchol sa výmenami s nasledujúcim vrchol presúva pozdĺž, inak nehybného, zoznamu ostatných vrcholov, pričom sa hľadá jeho optimálne umiestnenie. Umiestnením všetkých vrcholov takýmto spôsobom je jedno kolo siftingu. Jedno kolo siftingu môže byť implementované v čase $\mathcal{O}(nm)$. Experimenty ukázali, že na dosiahnutie lokálneho minima stačí niekoľko siftingových kôl.

### 2.5.2 Adjacent Vertex with Smallest Degree First

AVSDF je algoritmus, ktorý navrhli Hongmei He a Ondrej Sýkora v práci [10]. Je založený na minimalizácií dĺžok hrán, ktoré z geometrického hľadiska sú sečnicami kružnice, na ktorej sú usporiadané vrcholy. Aby to dosiahli upravili algoritmus na prehľadávanie grafu do hĺbky. Začne s prehladávaní vo vrchole s najmenším stupňom. Ako navštevuje jednotlivých susedov pridáva ich na kružnicu v poradí v akom boli navštívené. Pričom z viacerých susedov vrchola ma prednosť ten, ktorý má najmenší stupeň a ešte nebol navštívený.

Časová zložitosť $\mathcal{O}(m)$. Ak ešte pridáme počítanie priesečníkov tak $\mathcal{O}(m^2)$. Algoritmus sa dá vylepšiť druhou fázou. Nájdeme najlepšiu pozíciu vrchola z pomedzi jeho pozície a pozíciami jeho susedov. Vrcholy berieme v poradí podľa počtu priesečníkov spôsobených jeho incidentnými hranami.

**Algorithm 15** AVSDF fáza 1

Inicializuj int pole $positions[]$ a zásobník $S$
Nájdi vrchol s najmenším stupňom a vlož ho do $S$
$pos = 0$;
**while** $pos < n$ **do**
  vyber vrchol $v$ z $S$
  **if** $v$ je neumiestnený **then**
    pridaj $positions[pos] = v$
    pridaj všetkých susedov $v$ do $S$ v poradí od najväčšieho stupňa po najmenší, tak aby na vrchole zásobníka bol sused s najmenším stupňom
  **end if**
  $pos = pos + 1$
  **if** $S$ je prázdna **then**
    nájdi nenavštívený vrchol s najmenším stupňom a pridaj ho do $S$
  **end if**
**end while**

**Algorithm 16** AVSDF fáza 2

Pre každý vrchol vyrátaj počet priesečníkov hrán s ním incidentných
$Vlist$ je zoznam vrcholov usporiadaný podľa ich priesečníkového čísla. Premenná $currentV$ ukazuje na vrchol s najväčším počtom priesečníkov.
**for** each $currentV \in Vlist$ **do**
  vlož všetkých susedov $currentV$ do pomocného zoznamu.
  skús vymeniť pozíciu vrchola $currentV$ so všetkými z pomocného zoznamu a pre každú výmenu spočítaj počet priesečníkov.
  umiestni vrchol $currentV$ na najlepšiu pozíciu.
**end for**

# Kapitola 3

# Experimenty a merania

## 3.1 Úvod

Na porovnanie úspešnosti prezentovaných algoritmoch boli použité rôzne triedy grafov ako aj sada grafov z GDToolkit, ktoré sa tiež nazývajú Rímske grafy. Najprv sa porovnávali heuristiky pre lineárne priesečníkové číslo s fixovaným poradím vrcholov. V druhej časti sú testované heuristiky pre lineárne priesečníkové číslo, pričom sa využívajú aj heuristiky pre konvexné priesečníkové číslo. Porovnanie heuristík pre konvexné priesečníkové číslo sa dá nájsť v [10]. Všetky algoritmy boli implementované v C++. Operačný systém, v ktorom boli algoritmy testované bol Linux.

## 3.2 Lineárne priesečníkové číslo s fixovaným poradím vrcholov

Na určenie fixovaného priesečníkového čísla sa testovali všetky heuristiky spomínané v časti 2.1. Algoritmy na neurónových sieťach a randomizovaná greedy boli spúšťané 10 krát pre jedno meranie, pretože ich algoritmy pracujú priamo s náhodnými číslami. V tabuľke sú minimálne hodnoty, ktoré boli dosiahnuté.

Algoritmy pre fixované priesečníkové číslo boli testované na týchto grafoch:

- Kompletné grafy $K_5$ až $K_{70}$.
- Náhodné grafy s pravdepodobnosťou výskytu hrany $p = 0{,}01$ a $p = 0{,}03$
- Hyperkocky $Q_2$ až $Q_9$, pričom hrany boli usporiadané podľa hamiltonovskej kružnice.

V tabuľke 3.1 sú zobrazené výsledky, ktoré jednotlivé heuristiky dosiahli na úplných grafoch. Heuristika MPlanar dosiahla jasne najhoršie výsledky

| graf | Up[a] | Mplanar | E-len | OnePage | HeNeural | GANeural | E-len2 | Slope | RanGreedy | MaxCut |
|---|---|---|---|---|---|---|---|---|---|---|
| $K_5$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $K_6$ | 3 | 4 | 3 | 3 | 3 | 3 | 4 | 5 | 3 | 3 |
| $K_7$ | 9 | 11 | 9 | 9 | 9 | 9 | 11 | 9 | 9 | 9 |
| $K_8$ | 18 | 24 | 20 | 18 | 18 | 18 | 21 | 23 | 19 | 18 |
| $K_9$ | 36 | 46 | 36 | 36 | 36 | 36 | 42 | 36 | 36 | 36 |
| $K_{10}$ | 60 | 80 | 62 | 60 | 60 | 60 | 81 | 68 | 63 | 60 |
| $K_{11}$ | 100 | 130 | 100 | 100 | 100 | 100 | 122 | 100 | 100 | 100 |
| $K_{12}$ | 155 | 200 | 156 | 150 | 150 | 150 | 179 | 163 | 154 | 150 |
| $K_{13}$ | 225 | 295 | 231 | 225 | 225 | 225 | 271 | 225 | 233 | 225 |
| $K_{14}$ | 315 | 420 | 328 | 315 | 315 | 315 | 398 | 333 | 328 | 315 |
| $K_{15}$ | 441 | 581 | 449 | 441 | 441 | 441 | 543 | 441 | 443 | 441 |
| $K_{16}$ | 588 | 784 | 604 | 588 | 588 | 588 | 742 | 613 | 612 | 588 |
| $K_{17}$ | 784 | 1036 | 796 | 784 | 784 | 784 | 990 | 784 | 786 | 784 |
| $K_{18}$ | 1008 | 1344 | 1045 | 1008 | 1008 | 1008 | 1333 | 1040 | 1065 | 1008 |
| $K_{19}$ | 1296 | 1716 | 1318 | 1296 | 1296 | 1296 | 1604 | 1296 | 1318 | 1296 |
| $K_{20}$ | 1620 | 2160 | 1675 | 1620 | 1620 | 1620 | 2047 | 1661 | 1685 | 1620 |
| $K_{30}$ | 9550 | 12740 | 9724 | 9555 | 9555 | 9555 | 12370 | 9653 | 9741 | 9555 |
| $K_{40}$ | 32490 | 43320 | 33039 | 32490 | 32490 | 32490 | 40974 | 32671 | 32795 | 32490 |
| $K_{50}$ | 82800 | 110400 | 83896 | 82800 | 82800 | 82800 | 104936 | 83088 | 83699 | 82800 |
| $K_{60}$ | 176610 | 235480 | 178949 | 176610 | 176610 | 176610 | 224166 | 177031 | 178199 | 176610 |
| $K_{70}$ | 333795 | 445060 | 337672 | 333795 | 333795 | 333795 | 411696 | 334373 | 336373 | 333795 |

[a]Horný odhad 1.3

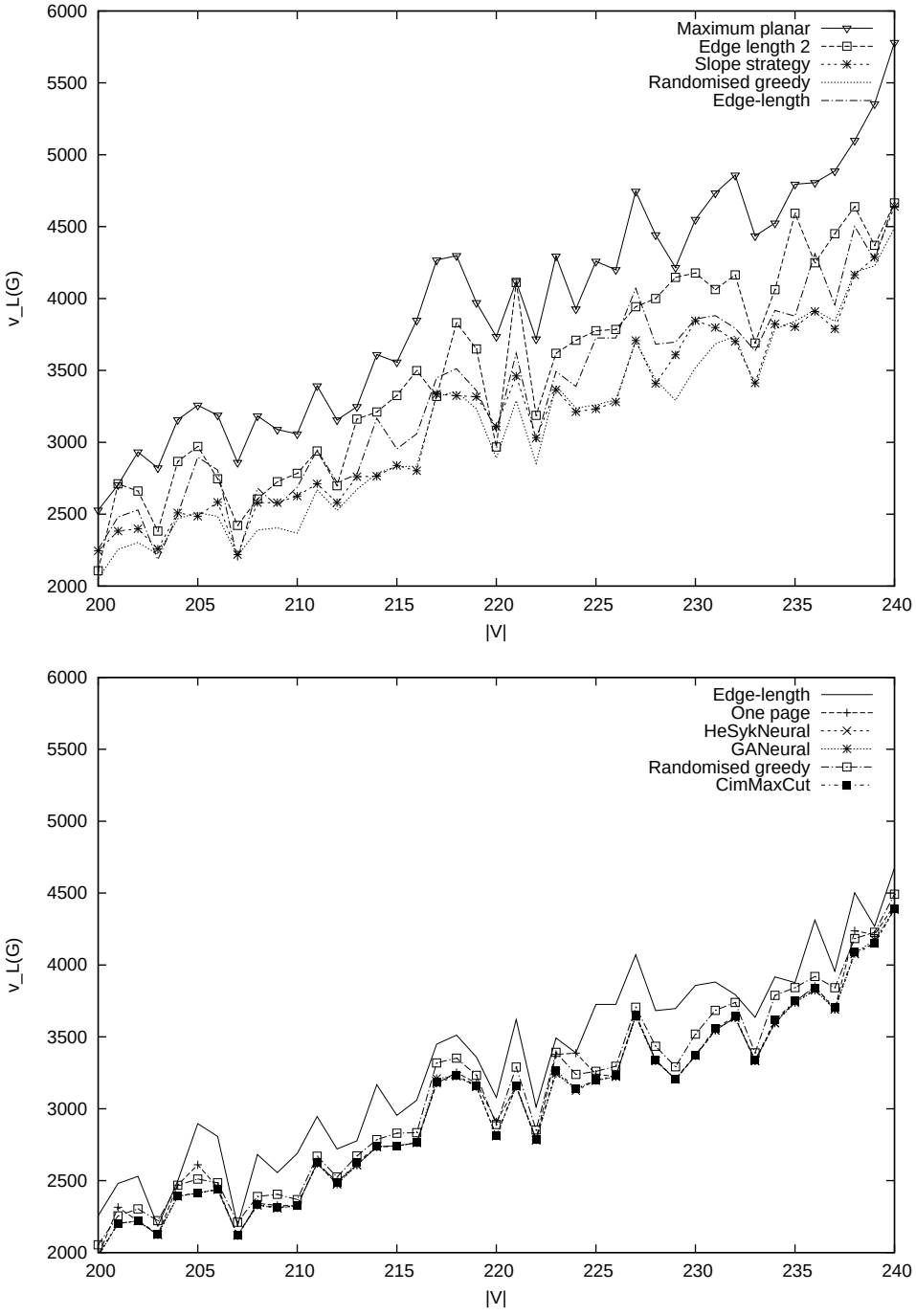
Tabuľka 3.1: Fixované lineárne priesečníkové číslo: Heuristiky na kompletných grafoch

nasledované Edge Length 2. Edge Length, Slope strategy a Randomized greedy vracali približne rovnaké výsledky ale od $K_{17}$ Slope stratégia dávala mierne lepšie výsledky. Zvyšná skupina heuristík OnePage, HeNeural, GANeural a MaxCut dosahujú rovnaké počty priesečníkov, ktoré sa zhodujú s horným odhadom 1.3 pre kompletné grafy.

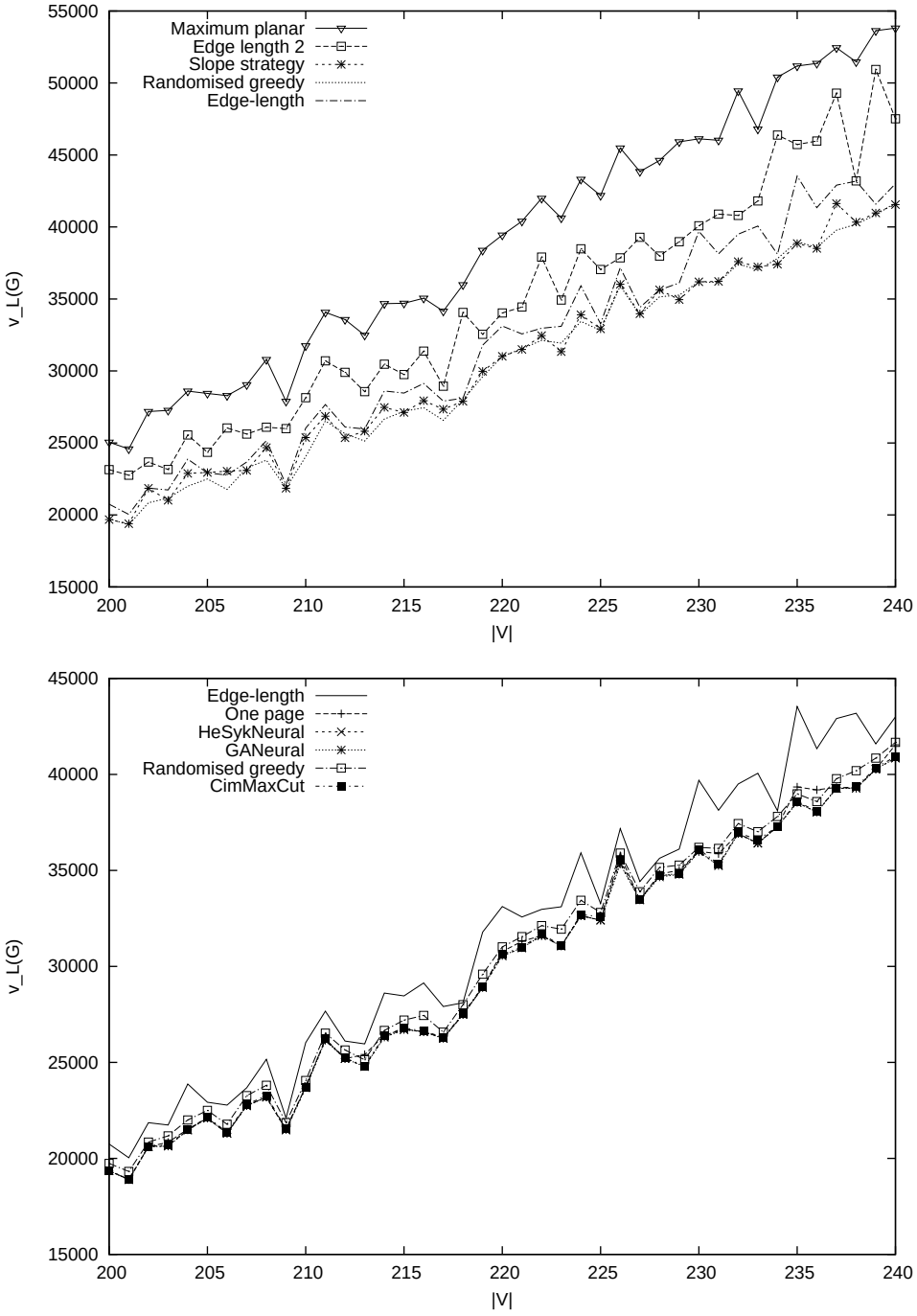
Graf 3.1 znázorňuje výsledky, ktoré dosiahli heuristiky na náhodných grafoch s pravdepodobnosťou výskytu hrany 0,01. MPlanar s relatívne veľkým odstupom zaostáva za ostatnými. Ďalšie v poradí Edge length 2, Edge length, Slope strategy a Randomized greedy. V druhej časti grafu sa ukazuje odstup randomised greedy od ostatných heuristík, ktoré v grafe splývajú do jednej čiary.

V grafe 3.2 boli aproximácie testované na náhodných grafoch s pravdepodobnosťou 0,03. Oproti predchádzajúcemu testu sa poradie nezmenilo, ale sú viditeľné väčšie rozostupy, medzi jednotlivými heuristikami. Najlepšie výsledky dosiahli heuristiky v poradí GANeural, HeSykoraNeural, CimMaxCut a One Page.

Výsledky testov na hyperkocke mierne prekonali predchádzajúce výsledky Cimikowského publikované [3]. Najlepšie vychádzala heuristika GANeural, ktorá dáva najlepšie výsledky až po $Q_8$. Pre $Q_9$ najlepší výsledok dosiahla HeNeural. One page a MaxCut dosiahli porovnateľné hodnoty.




Obrázok 3.1: Fixované lineárne priesečníkové číslo: Náhodné grafy s pravdepodobnosťou výskytu hrany $p = 0{,}01$




Obrázok 3.2: Fixované lineárne priesečníkové číslo: Náhodné grafy s pravdepodobnosťou výskytu hrany $p = 0{,}03$

| graf | Mplanar | E-len | OnePage | HeNeural | GANeural | E-len2 | Slope | RanGreedy | MaxCut |
|---|---|---|---|---|---|---|---|---|---|
| $Q_2$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $Q_3$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $Q_4$ | 8 | 8 | 8 | 8 | 8 | 10 | 8 | 8 | 8 |
| $Q_5$ | 80 | 64 | 64 | 62 | 60 | 72 | 80 | 64 | 60 |
| $Q_6$ | 512 | 396 | 372 | 374 | 368 | 412 | 512 | 380 | 372 |
| $Q_7$ | 2688 | 2038 | 2012 | 1878 | 1848 | 2100 | 2688 | 1952 | 1876 |
| $Q_8$ | 12672 | 9372 | 8700 | 8674 | 8632 | 9426 | 12672 | 8982 | 8712 |
| $Q_9$ | 56064 | 40864 | 37598 | 37488 | 37662 | 41086 | 56064 | 38878 | 37972 |

Tabuľka 3.2: Lineárne priesečníkové číslo hyperkocky. Poradie vrcholov určené hamiltonovskou knižnicou

## 3.3 Lineárne priesečníkové číslo

Pri testovaní heuristík pre lineárne priesečníkové číslo som vynechal MPlanar a Edge Length 2, ktoré v predchádzajúcich testoch nedosiahli dobré výsledky. Testované algoritmy boli 2 druhov:

- Algoritmus zložený z určenia vhodnej permutácie(konvexné priesečníkové číslo) a následne sa použije heuristika pre fixované priesečníkové číslo.
- Algoritmus, ktorý rieši permutáciu aj rozmiestnenie hrán súčasne. Popísané v kapitole 2.4

Testované heuristiky pre konvexné priesečníkové číslo boli popísané v kapitole 2.5.

Triedy grafov na ktorých sa testovalo:

- Cirkulantný graf $C_n(a_1, a_2, \cdots, a_k)$, kde $0 < a_1 < a_2 < \cdots < a_k < (n+1)/2$, je pravidelný hamiltonovský graf s $n$ vrcholmi. Pre každé $i$ vrcholy $a_1 \pm i, a_2 \pm i, \cdots a_k \pm i (mod\ n)$ sú susedné s vrcholom $i$.
- Kompletný p-partitný graf $K_n(p)$. Vznikne zovšeobecnením kompletného bipartitného grafu, ktorý má $p$ partícií a každá má $n$ vrcholov.
- Torus $T_{n,n}$. Torus vznikne ako produkt kartézskeho súčinu cyklov $C_n \times C_n$
- grafy z GDToolKit [6]. Sada neorientovaných grafov obsahuje 11,582 grafov, ktoré boli vygenerované z jadra 112 grafov, ktoré sa používali v skutočných aplikáciách.

V tabuľke 3.3 sú zhrnuté výsledky pre cirkulantné grafy. Podľa celkového počtu priesečníkov najlepšie výsledky dosiahla heuristika založená na simulovanom žíhaní. Za ňou nasledujú BB+MaxCut a BB+GANeural. Genetický algoritmus napriek očakávaniam z [13], nedosiahol najlepšie výsledky. Aj keď napríklad výsledok 356 genetického algoritmu pre graf $C_{26}(1,$

| | Baur a Brandes | | | | | | | AVSDF | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| graf | E-len | OnePage | HeNeural | GANeural | Slope | RanGreedy | MaxCut | E-len | OnePage | HeNeural | GANeural | Slope | RanGreedy | MaxCut | GA | SA |
| $C_{20}(1, 2)$ | 2 | 0 | 2 | 0 | 16 | 2 | 0 | 0 | 0 | 2 | 0 | 16 | 2 | 0 | 4 | 0 |
| $C_{20}(1, 2, 3)$ | 24 | 24 | 26 | 24 | 72 | 22 | 22 | 24 | 24 | 24 | 22 | 72 | 24 | 22 | 26 | 20 |
| $C_{20}(1, 2, 3, 4)$ | 92 | 84 | 76 | 70 | 180 | 80 | 70 | 70 | 84 | 78 | 70 | 180 | 76 | 70 | 88 | 70 |
| $C_{22}(1, 2)$ | 0 | 0 | 2 | 0 | 18 | 2 | 0 | 0 | 0 | 2 | 0 | 18 | 2 | 0 | 2 | 0 |
| $C_{22}(1, 2, 3)$ | 26 | 24 | 26 | 24 | 82 | 26 | 24 | 24 | 24 | 28 | 24 | 82 | 26 | 24 | 26 | 24 |
| $C_{22}(1, 3, 5, 7)$ | 252 | 213 | 208 | 208 | 277 | 222 | 208 | 260 | 200 | 200 | 200 | 276 | 212 | 200 | 212 | 199 |
| $C_{24}(1, 3)$ | 16 | 17 | 12 | 12 | 34 | 14 | 12 | 16 | 16 | 14 | 12 | 36 | 14 | 14 | 13 | 12 |
| $C_{24}(1, 3, 5)$ | 87 | 82 | 78 | 74 | 126 | 84 | 74 | 98 | 78 | 78 | 72 | 148 | 78 | 72 | 76 | 72 |
| $C_{24}(1, 3, 5, 7)$ | 275 | 239 | 214 | 214 | 289 | 235 | 214 | 282 | 228 | 224 | 216 | 328 | 230 | 216 | 236 | 216 |
| $C_{26}(1, 3)$ | 18 | 17 | 14 | 12 | 36 | 14 | 12 | 18 | 16 | 16 | 16 | 40 | 14 | 16 | 14 | 12 |
| $C_{26}(1, 3, 5)$ | 99 | 91 | 93 | 88 | 151 | 95 | 86 | 102 | 82 | 82 | 82 | 168 | 88 | 82 | 93 | 81 |
| $C_{26}(1, 4, 7, 9)$ | 408 | 377 | 366 | 367 | 380 | 388 | 370 | 440 | 364 | 364 | 364 | 430 | 382 | 364 | 356 | 359 |
| $C_{28}(1, 2, 3, 4)$ | 128 | 126 | 114 | 98 | 292 | 112 | 98 | 98 | 126 | 116 | 98 | 292 | 114 | 98 | 120 | 99 |
| $C_{28}(1, 3)$ | 18 | 19 | 16 | 12 | 40 | 16 | 12 | 18 | 18 | 16 | 16 | 44 | 16 | 16 | 14 | 14 |
| $C_{28}(1, 3, 5)$ | 109 | 100 | 91 | 89 | 182 | 98 | 90 | 110 | 92 | 92 | 86 | 188 | 94 | 86 | 97 | 85 |
| $C_{28}(1, 3, 5, 7, 9)$ | 659 | 618 | 620 | 617 | 717 | 657 | 626 | 740 | 560 | 560 | 560 | 768 | 570 | 560 | 719 | 559 |
| $C_{30}(1, 2, 4, 5, 7)$ | 466 | 394 | 400 | 392 | 728 | 414 | 392 | 426 | 394 | 400 | 392 | 728 | 422 | 392 | 413 | 395 |
| $C_{30}(1, 3, 5)$ | 118 | 113 | 103 | 106 | 208 | 113 | 100 | 120 | 96 | 100 | 96 | 208 | 100 | 96 | 108 | 90 |
| $C_{30}(1, 3, 5, 8)$ | 399 | 337 | 343 | 351 | 453 | 363 | 341 | 348 | 298 | 302 | 298 | 474 | 320 | 298 | 358 | 297 |
| $C_{32}(1, 2, 4, 6)$ | 204 | 160 | 178 | 190 | 408 | 204 | 160 | 160 | 160 | 190 | 190 | 408 | 198 | 160 | 204 | 160 |
| $C_{34}(1, 3, 5)$ | 136 | 116 | 116 | 111 | 218 | 116 | 109 | 138 | 110 | 110 | 104 | 248 | 112 | 104 | 114 | 103 |
| $C_{34}(1, 4, 8, 12)$ | 443 | 401 | 393 | 379 | 511 | 414 | 381 | 646 | 572 | 572 | 572 | 666 | 586 | 574 | 401 | 564 |
| $C_{36}(1, 2, 4)$ | 50 | 36 | 52 | 36 | 172 | 50 | 36 | 36 | 36 | 52 | 36 | 172 | 56 | 36 | 58 | 36 |
| $C_{36}(1, 3, 5, 7)$ | 420 | 335 | 319 | 318 | 618 | 338 | 319 | 428 | 328 | 328 | 328 | 664 | 364 | 328 | 371 | 327 |
| $C_{38}(1, 4, 7)$ | 188 | 183 | 179 | 178 | 343 | 188 | 179 | 236 | 194 | 190 | 190 | 378 | 206 | 190 | 184 | 187 |
| $C_{38}(1, 7)$ | 118 | 106 | 94 | 92 | 151 | 101 | 93 | 98 | 96 | 84 | 84 | 144 | 86 | 84 | 97 | 78 |
| $C_{40}(1, 5)$ | 57 | 50 | 45 | 45 | 108 | 51 | 41 | 64 | 64 | 58 | 56 | 120 | 60 | 56 | 50 | 51 |
| $C_{42}(1, 2, 4, 6)$ | 254 | 210 | 234 | 210 | 598 | 256 | 210 | 210 | 210 | 210 | 210 | 598 | 258 | 210 | 274 | 210 |
| $C_{42}(1, 3, 6)$ | 176 | 161 | 146 | 146 | 360 | 154 | 147 | 168 | 158 | 158 | 154 | 342 | 160 | 150 | 169 | 149 |
| $C_{42}(1, 4)$ | 46 | 42 | 42 | 41 | 110 | 45 | 41 | 42 | 42 | 42 | 42 | 102 | 44 | 42 | 44 | 39 |
| $C_{44}(1, 4, 5)$ | 190 | 144 | 137 | 140 | 370 | 151 | 138 | 196 | 190 | 182 | 180 | 448 | 192 | 180 | 158 | 167 |
| $C_{44}(1, 4, 7, 10)$ | 866 | 743 | 691 | 687 | 913 | 756 | 694 | 726 | 634 | 634 | 632 | 1116 | 658 | 632 | 728 | 631 |
| $C_{46}(1, 4)$ | 47 | 45 | 43 | 41 | 104 | 42 | 41 | 46 | 46 | 48 | 46 | 114 | 48 | 46 | 47 | 43 |
| $C_{46}(1, 5, 8)$ | 376 | 346 | 329 | 328 | 457 | 348 | 332 | 356 | 310 | 296 | 294 | 594 | 310 | 294 | 358 | 291 |
| # priesečníkov | 6767 | 5953 | 5802 | 5700 | 9722 | 6171 | 5672 | 6744 | 5850 | 5852 | 5742 | 10610 | 6122 | 5712 | 6232 | 5640 |

Tabuľka 3.3: Lineárne priesečníkové číslo pre grafy $C_{20}$ až $C_{46}$

| | Baur a Brandes | | | | | | | AVSDF | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| graf | E-len | OnePage | HeNeural | GANeural | Slope | RanGreedy | MaxCut | E-len | OnePage | HeNeural | GANeural | Slope | RanGreedy | MaxCut | GA | SA |
| $K_3(2)$ | 1 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 1 |
| $K_3(3)$ | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 18 | 16 | 16 | 16 | 16 | 16 | 16 | 15 | 16 |
| $K_3(4)$ | 97 | 90 | 87 | 86 | 91 | 88 | 86 | 93 | 90 | 91 | 86 | 91 | 90 | 86 | 85 | 86 |
| $K_4(2)$ | 4 | 6 | 6 | 4 | 4 | 4 | 4 | 4 | 6 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| $K_4(3)$ | 72 | 68 | 68 | 68 | 72 | 68 | 68 | 72 | 68 | 66 | 68 | 72 | 68 | 68 | 67 | 66 |
| $K_4(4)$ | 375 | 343 | 334 | 334 | 343 | 338 | 333 | 380 | 336 | 338 | 336 | 340 | 344 | 336 | 353 | 336 |
| $K_5(2)$ | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| $K_5(3)$ | 204 | 196 | 196 | 196 | 196 | 200 | 196 | 204 | 196 | 196 | 196 | 196 | 204 | 196 | 196 | 196 |
| $K_5(4)$ | 978 | 915 | 915 | 918 | 933 | 940 | 918 | 977 | 916 | 916 | 917 | 937 | 946 | 919 | 966 | 917 |
| # priesečníkov | 1763 | 1651 | 1639 | 1641 | 1672 | 1671 | 1638 | 1765 | 1645 | 1644 | 1642 | 1673 | 1689 | 1642 | 1703 | 1638 |

Tabuľka 3.4: Lineárne priesečníkové číslo pre grafy $K_n(p)$

4, 7, 9) prekonal ostatné aproximácie. Podobne aj simulované žíhanie sa nevyrovnalo meraniam, ktoré získali v [20]. Ani u kombinovaných heuristikách nie je jednoznačne lepší algoritmus. Napríklad pre graf $C_{44}(1, 4, 7, 10)$ sú lepšie heuristiky založené na AVSDF ale pre graf $C_{46}(1, 4)$ je to opačne.

Pre p-partitné kompletné grafy heuristiky BB+MaxCut, BB+HeNeural a SA dosiahli najlepšie výsledky ako je vidieť v tabuľke 3.4. Naopak najhoršie si počínali obidve kombinácie E-len a genetický algoritmus, pričom obidva algoritmy presiahli hranicu 1700 celkového počtu priesečníkov. Zvyšné heuristiky dosiahli mierne lepšie výsledky pri kombinácií s BB ako AVSDF.

Tabuľka 3.5 obsahuje výsledky heuristík na tóruse. Konvexné priesečníkové číslo lepšie minimalizovala heuristika AVSDF. Najlepšie výsledky dosiahli algoritmy založené na neurónových sieťach(HeNeural, GANeural) spolu s redukciou na maximálny rez(MaxCut). Simulované žíhanie(SA) ani v jednom prípade nezlepšilo svoje vstupné riešenie. V šiestich prípadoch najlepšie riešenie dosiahlo predpokladané hodnoty ($T_{4,4}$, $T_{6,6}$, $T_{8,8}$, $T_{10,10}$,

| | | Baur a Brandes | | | | | | | AVSDF | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| graf | Opt[a] | E-len | OnePage | HeNeural | GANeural | Slope | RanGreedy | MaxCut | E-len | OnePage | HeNeural | GANeural | Slope | RanGreedy | MaxCut | GA | SA |
| $T_{4,4}$ | 8 | 12 | 12 | 8 | 8 | 16 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 11 | 8 |
| $T_{5,5}$ | 15 | 20 | 22 | 20 | 20 | 39 | 21 | 20 | 26 | 24 | 20 | 20 | 38 | 20 | 20 | 21 | 20 |
| $T_{6,6}$ | 24 | 60 | 69 | 56 | 56 | 99 | 63 | 56 | 24 | 36 | 24 | 24 | 62 | 24 | 24 | 56 | 24 |
| $T_{7,7}$ | 35 | 110 | 84 | 84 | 83 | 144 | 91 | 83 | 60 | 68 | 48 | 48 | 134 | 50 | 48 | 88 | 48 |
| $T_{8,8}$ | 48 | 179 | 153 | 156 | 153 | 285 | 168 | 155 | 48 | 48 | 48 | 48 | 184 | 48 | 48 | 161 | 48 |
| $T_{9,9}$ | 63 | 307 | 281 | 257 | 261 | 453 | 281 | 251 | 108 | 124 | 88 | 88 | 314 | 98 | 88 | 267 | 88 |
| $T_{10,10}$ | 80 | 345 | 273 | 278 | 278 | 603 | 315 | 274 | 80 | 80 | 80 | 80 | 398 | 86 | 80 | 289 | 80 |
| $T_{11,11}$ | 99 | 519 | 461 | 439 | 431 | 900 | 482 | 437 | 170 | 196 | 140 | 140 | 602 | 146 | 140 | 497 | 140 |
| $T_{12,12}$ | 120 | 687 | 576 | 565 | 556 | 1056 | 647 | 562 | 120 | 120 | 120 | 120 | 728 | 160 | 120 | 654 | 120 |
| $T_{13,13}$ | 143 | 926 | 875 | 823 | 820 | 1588 | 905 | 829 | 246 | 284 | 204 | 204 | 1022 | 240 | 204 | 953 | 204 |
| $T_{14,14}$ | 168 | 1285 | 1147 | 1071 | 1087 | 2137 | 1183 | 1070 | 168 | 518 | 168 | 168 | 1198 | 192 | 168 | 1249 | 168 |
| $T_{15,15}$ | 195 | 1539 | 1362 | 1285 | 1269 | 2762 | 1402 | 1296 | 336 | 388 | 280 | 280 | 1598 | 340 | 280 | 1552 | 280 |
| # priesečníkov | 998 | 5989 | 5315 | 5042 | 5022 | 10082 | 5566 | 5041 | 1394 | 1894 | 1228 | 1228 | 6286 | 1412 | 1228 | | |

[a]Optimálna hodnota predpokladaná podľa vzťahu 1.4

Tabuľka 3.5: Lineárne priesečníkové číslo pre grafy $T_{4,4}$ až $T_{15,15}$

| | Baur a Brandes | | | | | | | AVSDF | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | E-len | OnePage | HeNeural | GANeural | Slope | RanGreedy | MaxCut | E-len | OnePage | HeNeural | GANeural | Slope | RanGreedy | MaxCut | GA | SA |
| dosiahol min. hodnotu | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 36 | 54 | 44 | 2 | 33 | 57 | 0 | 58 |
| # priesečníkov | 2929 | 2565 | 2410 | 2349 | 4513 | 2553 | 2304 | 939 | 769 | 722 | 733 | 1188 | 753 | 719 | 2552 | 718 |

Tabuľka 3.6: Zhrnuté výsledky pre grafy s 50 vrcholmi zo sady GDToolKit

$T_{12,12}$, $T_{14,14}$).

Z testovanej sady neorientovaných grafov som vyhodnotil heuristiky na grafoch s 50 vrcholmi, ktorých spolu bolo 59. Tabuľka 3.6 obsahuje zhrnutie. Prvý riadok obsahuje koľko krát dosiahla heuristika najmenšiu hodnotu. Simulované žíhanie dosiahlo minimálne hodnoty až 58 krát. Keďže počiatočné riešenie bolo získané pomocou heuristiky AVSDF+GANeural, ktorá dosiahla iba 44 krát minimum, SA sa ukázala ako prínos. Druhá v poradí kombinácia AVSDF+MaxCut. Algoritmy založené na BB dosiahli o rád horšie výsledky a takisto aj genetický algoritmus.

## 3.4 Zhrnutie

Z greedy heuristík určených na lineárne priesečníkové číslo s pevným poradím vrcholov je relatívne najlepšou OnePage. Nedosahovala síce vždy minimálne hodnoty ale na rôznych triedach grafov si zachovávala rovnaké umiestnenie v poradí. Je nutné podotknúť, že veľmi pomohla fáza opakovaného umiestnenia hrán, ktorá konštantne (10-krát) zvýši čas behu algoritmu. Neurónové siete, ktoré sa spolu MaxCut algoritmom striedali vo vedení, dobre aproximovali kompletné grafy a hyperkocky a celkovo patria k najlepším aproximáciám, ktoré som testoval. MaxCut stabilne dosahoval najlepšie alebo veľmi dobré výsledky na všetkých testovaných grafov. Heuristika AVSDF dosiahla rovnaké alebo lepšie výsledky ako BB pri heuristikách pre lineárne priesečníkové číslo. Genetický algoritmus nepodával výsledky podľa očakávaní. Iba v niekoľkých prípadoch sa mu podarilo dodať najlepší výsledok, inak vykazoval podobné výsledky ako greedy algoritmy. Simulované žíhanie ako zlepšujúca heuristika úspešne zvládla test na rímskych a cirkulantných

grafoch, ale nepriniesla žiadne zlepšenie na tórusoch. Čas behu je okrem veľkosťou grafu aj silne ovplyvnený parametrami ochladzovania, ktoré môžu čas značne predĺžiť. V praktických aplikáciách nemusí byť takéto predĺženie vhodné.

# Kapitola 4

# Záver

V prvej kapitole som uviedol do problému minimalizácie priesečníkového čísla grafu a poukázal na jeho NP-úplnosť. V dôsledku zložitosti všeobecného problému som spomenul niektoré varianty priesečníkového čísla s istými obmedzeniami ako sú napríklad rektilineárne priesečníkové číslo a konvexné priesečníkové číslo. Podrobnejšie som definoval problém minimalizácie lineárneho priesečníkového čísla s voľným a pevným poradím vrcholov (tzv. fixované lineárne priesečníkové číslo).

V druhej kapitole som popísal niekoľko heuristík a aproximácií na riešenie problému stanoveného v predchádzajúcej kapitole. V prvej časti sa nachádzajú algoritmy pre fixované poradie vrcholov. Tento problém sa zdal byť jednoduchší, pretože algoritmy musia iba rozmiestniť hrany do dvoch polrovín. Študované boli základné greedy metódy, ktoré rozdeľovali hrany podľa zvolených kritérií ako dĺžka hrany, sklon hrany pri nakreslení grafu na kružnicu alebo v snahe oddialiť vznik priesečníka prípadne postupné zlepšovanie najhoršieho možného usporiadania. Aproximácie s iným prístupom sú popísané v ďalšej časti. Využívajú neurónovú sieť, presnejšie Hopfieldovu symetrickú neurónovú sieť a jej schopnosť minimalizovať jej energetickú funkciu. Obidva algoritmy založené na neurónovej sieti obdobne formulujú energetickú funkciu, prvý z algoritmov využíva vzťahy neurodynamiky, druhý sa presnejšie pridŕža hopfieldovho modelu. Pri minimalizácií môže neurónová sieť uviaznuť v lokálnom minime, čo oba algoritmy riešia rozdielne. Prvý používa techniku hill-climbingu, zatiaľ čo druhý sa snaží "vypĺňať údolia energetickej funkcie" tzv gradient ascent learning metódou. Problém minimalizácie fixovaného lineárneho priesečníkového čísla sa dá transformovať na problém nájdenia maximálneho rezu v grafe. Čo je známy NP-úplný problém. Na aproximáciu tohoto problému som použil znáhodnený algoritmus,ktorý je založený na semidefinitnom programovaní a ktorého stredná hodnota riešenia je najmenej 0,87856 krát hodnota optimálneho riešenia.

V druhej časti druhej kapitoly sa venujem algoritmom, ktoré sa snažia riešiť aj usporiadanie vrcholov na priamke. Základným nástrojom v tom-

to prípade bola pravdepodobnosť, pretože obidva algoritmy sú založené na stochastickom princípe. Prvý z nich je genetický algoritmus. Algoritmy tejto triedy sa zaraďujú medzi globálne optimalizátory. Genetický algoritmus simuluje správanie sa živých organizmov, čo sa týka dedičnosti. Na začiatku sú v populácií jedince, ktoré predstavujú zakódované riešenie problému a postupne sa z nich pomocou rekombinácie a mutácie vytvárajú nové jedince. Pravdepodobnosť, že jedinec bude vybratý na takýto proces priamo úmerne závisí od jeho fitness hodnoty. V tomto prípade fitness funkcia zohladňuje počet priesečníkov. Druhým algoritmom je simulované žíhanie. Na začiatku máme nakreslenie grafu s určitým počtom priesečníkov. Náhodne urobíme zmenu v tomto nakreslení. Ak táto zmena zníži počet priesečníkov tak nové riešenie akceptujeme. Ak nezníži tak ho akceptujeme iba s určitou pravdepodobnosťou. Tretia časť druhej kapitoly obsahuje aproximácie konvexného priesečníkového čísla. Konvexné priesečníkové číslo je podproblém lineárneho priesečníkového čísla pretože určuje permutáciu vrcholov. Vybral som dva algoritmy. Prvý z nich sa snaží postupne pridávať vrcholy na kružnicu, tak aby susedné vrcholy boli k sebe čo najbližšie, čo znižuje pravdepodobnosť že hrany medzi nimi budú mať priesečník. Druhý sa tiež pokúša minimalizovať dĺžku hrán, pomocou modifikácie prehladávania grafu do hĺbky.

Experimentálne porovnanie je zhrnuté v tretej kapitole. Najprv som porovnával heuristiky pre fixované lineárne priesečníkové číslo. Porovnanie algoritmov som vykonal na kompletných grafoch, hyperkockách a náhodných grafoch. Algoritmy pre lineárne priesečníkové číslo som spustil na cirkulantných grafoch, tóruse, kompletných p-partitných grafoch a na sade rímskych grafov.

# Zoznam tabuliek

# Zoznam obrázkov

# Literatúra


[1] Baur, M., and Brandes, U. Crossing reduction in circular layouts.

[2] Borchers, B. CSDP, a C library for semidefinite programming. *Optimization Methods and Software 11* (1999).

[3] Cimikowski, R. Algorithms for the fixed linear crossing number problem. *Discrete Appl. Math. 122*, 1-3 (2002), 93–115.

[4] Cimikowski, R., and Mumey, B. Approximating the fixed linear crossing number. *Discrete Appl. Math. 155*, 17 (2007), 2202–2210.

[5] A C library for semidefinite programming. `https://projects.coin-or.org/Csdp/`.

[6] GDToolkit. `http://www.dia.uniroma3.it/~gdt/`.

[7] Goemans, M. X., and Williamson, D. P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM 42*, 6 (1995), 1115–1145.

[8] Guy, R., Jenkyns, T., and Schaer, J. The toroidal crossing number of the complete graph. *Comb. Theory 4* (1968), 376–390.

[9] Harary, F., Kainen, P., and Schwenk, A. Toroidal graphs with arbitrarily high crossing numbers. *Nanta Math 6* (1997), 58–67.

[10] He, H., and Sýkora, O. New circular drawing algorithms. In *Proceedings of the Workshop on Information Technologies - Applications and Theory (ITAT)* (September 2004).

[11] He, H., and Sýkora, O. A hopfield neural network model for the outerplanar drawing problem. *Lecture Notes in Engineering and Computer Science Number 62* (2007).

[12] He, H., Sýkora, O., and Mäkinen, E. An improved neural network model for the two-page crossing number problem. *IEEE transactions on neural networks 17*, 6 (2006).

[13] He, H., Sýkora, O., Salagean, A., and Mäkinen, E. Parallelisation of genetic algorithms for the 2-page crossing number problem. *J. Parallel Distrib. Comput. 67*, 2 (2007), 229–241.

[14] Hirsch, M. W. Convergent activation dynamics in continuous time networks. *Neural Netw. 2*, 5 (1989), 331–349.

[15] Hopfield, J. J. Neurons with graded response have collective computation properties like those of two-state neurons. *Proc. Nat. Acad. Sci. U.S. 81* (1982).

[16] Kocúrová, H. Analysis of algorithms for computing the crossing number. Master's thesis, Univerzita Komenského, Bratislava, 2007.

[17] long Wang, R., and Tang, Z. A parallel algorithm for fixed linear crossing number problem. *International Journal of Computer Science and Network Security 6*, 11 (2006).

[18] M. R. Garey, D. S. J. Crossing number is np-complete. *SIAM Journal on Algebraic and Discrete Methods 4*, 3 (1983), 312–316.

[19] Mäkinen, E. On circular layouts. *International Journal of Computer Mathematics 24* (1988).

[20] Poranen, T., Mäkinen, T., and He, H. A simulated annealing algorithm for the 2-page crossing number problem. *Proceedings of INOC* (2007).

[21] Six, J., and Tollis, I. Circular drawings of biconnected graphs. *LNCS 1619* (1999).

[22] Takefuji, Y. Design of parallel distributed cauchy machines. *Neural Network Parallel Computing* (1992).

[23] Winterbach, W. The crossing number of a graph in the plane. Master's thesis, University of Stellenbosch, South Africa, 2005.