Hľadanie perfektných párení

Filip Novák

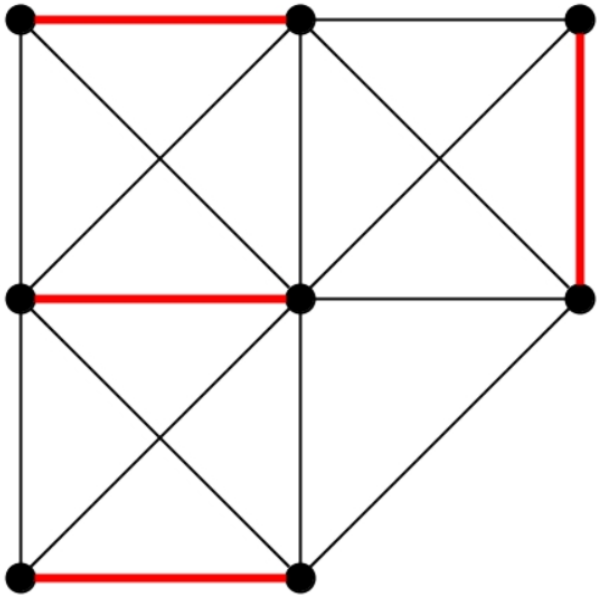
RNDr. Ján Mazák, PhD.

Ciele

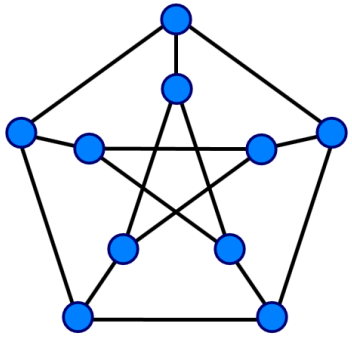
- Implementácia algoritmov
- Testovanie na kubických grafoch
- Vlastný algoritmus
- Použitie na snarkoch

Základné pojmy

- Perfektné párenie



- Snark



Petersenov graf

- AllSAT solver

Čiastkové známe algoritmy

- Tutteho matica
- Rabin-Vaziraniho algoritmus
- Zrýchlený R-V algoritmus

Tutteho matica

$$T[i,j] = \begin{cases} 0, & ak\ (i,\ j) \notin E \\ x_{ij}, & ak\ (i,\ j) \in E\ a\ zároveň\ i < j \\ -x_{ij}, & ak\ (i,\ j) \in E\ a\ zároveň\ i > j \end{cases}$$

- Graf má perfektné párenie práve vtedy, keď determinant Tutteto matice je rôzny od nuly

Zrýchlený R–V algoritmus

1: $A \leftarrow T_G^{-1}$
2: $M \leftarrow \emptyset$
3: **for** $i = 1$ to n **do**
4: nájdite j také, že $(v_i, v_j) \in G$ a $A_{i,j} \neq 0$
5: $M \leftarrow M \cup \{(v_i, v_j)\}$
6: $G \leftarrow G - \{v_i, v_j\}$
7: odstráňte i-ty riadok a j-ty stĺpec z A
8: odstráňte j-ty riadok a i-ty stĺpec z A
9: **end for**
10: **return** M

Porovnanie

- 26 kubických grafov, najviac 16 vrcholov

Údaje	Tutteho matica	Zrýchlený R-V
MIN	2	42
MAX	16	923
PRIEMER	6.53846	273.192
MEDIÁN	8	199.5

Hodnoty sú uvedené v mikrosekundách

Vlastný algoritmus

- Spracovanie vstupu
- Získanie informácií z grafu (*n, M, E, adj_list*)
- Rekurzívna funkcia:
  - Podmienka kontrolujúca nájdenie perf. párenia
  - Použitie Tutteho matice
  - Vyberanie hrany
  - Odstránenie susedných hrán
  - Lokálna podmienka slúžiaca na urýchlenie algoritmu
  - **Prvé** rekurzívne volanie
  - Vrátenie susedných hrán
  - **Druhé** rekurzívne volanie

- Podmienka kontrolujúca nájdenie perf. párenia
- Použitie Tutteho matice

```
if (M.size()*2 != n) {
    if (E.size() >= 1) {

        bool canContinue = true;

        if (!has_perfect_matching(n, M, E)) { //ma to perfektne parenie ?
            canContinue = false;
        }

        if (canContinue) {
            std::set<Location> edges_to_remove; //pomocny set v ktorom si budeme pamatat hrany, ktore budeme vymazavat pred prvou rekurziu,
            //a nasledne tento set pouzijeme aby sme dostali nase mnoziny do zaciatocneho stavu
```

- Vyberanie hrany
- Odstránenie susedných hrán

```
Location e; //striedave vyberanie
Number v1, v2;
if ((counter % 2) == 0) {
    e = *std::next(E.begin(), 0);
    v1 = e.n1();
    v2 = e.n2();

    counter++;
} else {
    e = *std::next(E.begin(), E.size() - 1);
    v1 = e.n1();
    v2 = e.n2();

    counter++;
}

M.push_back(e);
edges_to_remove.insert(e); //chceme odstranit z E aj hranu co je v perfektnom pareni, pridame ju do edges_to_remove
remove_neighbours(adj_list, edges_to_remove, v1, v2);
remove_neighbours(adj_list, edges_to_remove, v2, v1);
```

- Lokálna podmienka slúžiaca na urýchlenie algoritmu
- **Prvé** rekurzívne volanie

```
for (int i = 0; i < adj_list.size(); i++) { //lokalna podmienka, kde zistujeme ci je nejaky vrchol
    if (adj_list[i].size() == 1) {
        if (i <= adj_list[i][0].to_int()) {
            M.push_back(Location(Number(i), adj_list[i][0]));
            edges_to_remove.insert(Location(Number(i), adj_list[i][0]));
        } else {
            M.push_back(Location(adj_list[i][0], Number(i)));
            edges_to_remove.insert(Location(adj_list[i][0], Number(i)));
        }
        remove_neighbours(adj_list, edges_to_remove, adj_list[i][0], Number(i));
        adj_list[i].clear();
    }
}

for (auto edge : edges_to_remove) { //odstranujeme hrany z E
    auto pos = E.find(edge);
    E.erase(pos);
}

recursion(n, adj_list, E, M, callback, callbackParam, counter);
```

- Vrátenie susedných hrán
- **Druhé** rekurzívne volanie

```
std::vector<Location>::iterator itr = std::find(M.begin(), M.end(), e); //chceme z M odstranit aj tie hrany, ktore sme pridal
M.erase(itr, M.end());

auto pos = edges_to_remove.find(e);
edges_to_remove.erase(pos);

for (auto edge : edges_to_remove) { //vratenie do zaciatocneho stavu ale bez hrany "e" lebo sme ju vyhodili v prikaze vyssie
    E.insert(edge);
    adj_list[edge.n1().to_int()].push_back(edge.n2());
    adj_list[edge.n2().to_int()].push_back(edge.n1());
} //ked skonci for cyklus, tak je to pripravene na rekurziu

recursion(n, adj_list, E, M, callback, callbackParam, counter);

adj_list[v1.to_int()].push_back(v2); //vratenie adj_list do zaciatocneho stavu
adj_list[v2.to_int()].push_back(v1);
```

Vyberanie hrany

- 100 kubických grafov, najviac 38 vrcholov

a) zo začiatku *E*
b) z konca *E*
c) striedavé vybranie z *E*

Údaje	(a)	(b)	(c)
MIN	36	35	35
MAX	1 580 977 547	17 794 95 794	1 511 732 711
PRIEMER	83 700 100	96 115 200	79 959 600
MEDIÁN	130 970	100 272.5	107 257

Hodnoty sú uvedené v mikrosekundách

Použitie lokálnej podmienky

- 100 kubických grafov, najviac 38 vrcholov

Údaje	Bez lok. podmienky	S lok. podmienkou
MIN	35	35
MAX	1 511 732 711	323 192
PRIEMER	79 959 600	28 747.8
MEDIÁN	107 257	2 549.5

Hodnoty sú uvedené v mikrosekundách

Použitie Tutteho matice

- 100 kubických grafov, najviac 38 vrcholov

a) vždy medzi 30% – 70% hrán v *M*
b) vždy v polovici hrán *M*
c) vždy medzi 40% – 60% hrán v *M*
d) vždy medzi 50% – 90% hrán v *M*
e) vždy medzi 10% – 90% hrán v *M*
f) vždy, keď to bude možné (bez podmienky)
g) vždy, keď je deliteľný zaokrúhleným číslom znázorňujúcim 10% z celkového počtu hrán v *M*

Použitie Tutteho matice

Údaje	(a)	(c)	(e)	(f)	(g)
MIN	30	35	60	51	29
MAX	147 407	212 395	123 753	127 861	164 947
PRIEMER	16 384	21 670.3	12 211	11 959.3	15 230.4
MEDIÁN	2 489	2 861	1 761	1 800	1 714

Hodnoty sú uvedené v mikrosekundách

Vlastný alg. *vs.* AllSAT solver

- 100 kubických grafov, najviac 38 vrcholov

Údaje	Vlastný alg.	AllSAT solver
MIN	78	2 080
MAX	128 399	15 928
PRIEMER	11 953.8	4 332.72
MEDIÁN	1 778.5	3 199

Hodnoty sú uvedené v mikrosekundách

Výsledné pozorovanie

- Vlastný alg. rýchlejší na menších grafoch
- AllSAT solver rýchlejší na väčších grafoch
- Prehľadávanie všeobecných snarkov je rýchlejšie

Ďakujem za pozornosť

- 1. Prečo ste neimplementovali aj algoritmus hľadania všetkých perfektných párení založený na zrýchlenom Rabin-Vaziraniho algoritme? Nebude taký algoritmus rýchlejší ako opakované počítanie determinantu Tutteho matice? Aj keby vyšiel pomalší ako hľadanie nejakého perfektného párenia pomocou Tutteho matice, nemôže sa stať, že po prerobení na hľadanie všetkých perfektných párení by bol rýchlejší?

- 2. Prečo sa viac nevenujete Edmondsovmu alebo Micali-Vaziranihovmu algoritmu, ktoré len spomínate v úvode kapitoly 2?

- 3. Prečo pri implementácii výsledného algoritmu reprezentuje množiny hrán pomocou štruktúry set a taktiež neodovzdáva parametre M a V referenciou ? Nemohli by tieto úpravy zrýchliť algoritmus?

```
for (auto &r : G) { //inicializacia E a adj_list
    vertex = r.n();
    for (auto &i : G[vertex]) {
        v = i.n1();
        u = i.n2();
        e = i.l();

        if (v <= u) {
            E.insert(e);
        } else {
            E.insert(e.reverse());
        }
        adj_list[v.to_int()].push_back(u);
    }
}
```