VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

INFORMATIKOS KATEDRA

Nulinis namų darbas

**Minimalus aibės denginys**

(Minimum set cover)

Atliko: 3 kurso, 2 grupės studentė

@kukimukiku

Darbo vadovas:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vilnius 2023

**TURINYS**

1. Užduoties formulavimas............................................................................................................2
2. Godus minimalaus aibės denginio algoritmas...........................................................................3
3. Algoritmų sudėtingumų apžvalga..............................................................................................4

Literatūros sąrašas............................................................................................................................5

**1. Užduoties formulavimas**

Užduoties *pradiniais duomenimis* yra baigtinė aibė *S* ir baigtinė aibė *C*, kurios elementais yra aibės *S* poaibiai.

Aibės *C* poaibis *C‘* vadinamas *S* denginiu, jei kiekvienas *S* elementas priklauso bent vienai aibei iš *C.*

*Tikslas*: rasti mažiausiai elementų turintį aibės *C* poaibį *C‘,* kuris būtų aibės *S* denginys. Toks denginys vadinamas *minimaliu*.

*Pavyzdys. S* = {1, 2, 3, 4, 5}. *C* = {{1, 2}, {3, 4}, {4, 5}, {1, 2, 3}}.

*C’* = {{1, 2}, {3, 4}, {4, 5}} yra aibės *S* denginys.

*C‘‘* = {{4, 5}, {1, 2, 3}} yra minimalus denginys.

Minimalaus denginio uždavinį galima išspręsti brutalios jėgos (pilno perrinkimo) algoritmo pagalba:

Nagrinėjame visus galimus *C* poaibius. Jų yra 2k (*k* – aibės *C* elementų skaičius).

Tikriname, kurie iš jų yra denginiai.

Iš gautų denginių išrenkame mažiausią (trumpiausią).

Kadangi praktiniuose uždaviniuose *k* dažniausiai būna pakankamai didelis, tai pilno perrinkimo algoritmas praktiškai nenaudojamas. Jis turi tik teorinę reikšmę. 1972 metais R.M.Karp [Kar72] įrodė, kad minimalaus denginio uždavinys yra NP-pilnas.

Todėl šio darbo tikslas: *aprašyti godų algoritmą ir įvertinti jo sudėtingumą*.

Godūs algoritmai, nors randa tik apytikslį bet gana gerą sprendinį, kiekviename žingsnyje imant lokaliai optimalų sprendimą.

Laikysime, kad aibės *C* elementų sąjunga lygi aibei *S.*

**2. Godus minimalaus aibės denginio algoritmas**

Pasirenkame tokį aibės *C* elementą *C*i1, kuriuo uždengiame daugiausia aibės *S* elementų, t.y. jame bus daugiausia aibės *S* elementų. Jį įtraukiame į būsimą sprendinį. Iš aibės *S* pašaliname visus tuos jos elementus, kurie priklauso aibei *C*i1. Iš aibei *C* priklausančių aibių taip pat pašaliname visus elementus, kurie priklauso *C*i1.

Po to vėl renkamės poaibį *C*i2, kuriuo uždengiame daugiausiai likusių aibės *S* elementų. Jį traukiame į būsimą sprendinį. Vėl šaliname tuos aibės *S* bei *C* priklausančiose aibėse visus tuos elementus, kurie priklauso *C*i2.

Minimalaus denginio algoritmą tęsiame tol, kol uždengiame visus aibės *S* elementus.

**function** *C‘* = set\_cover(S, C)

*A* := *S*;

*C’* := ∅;

**for** *i* = 1 : *k*  **do** *B*i :=*C*i ; **end** ;

**while** *A* ≠ ∅ **do**

rasti *j* : |*B*j| = max {|*B*1|, . . . , |*B*k|};

*C’* := *C’* ∪ {*C*j};

*A* := *A* \ *B*j ;

**for** *i* = 1 : *k*  **do** *B*i :=*B*i \ *B*j ; **end** ;

**end** ;

Įvertinsime algoritmo sudėtingumą. Tarkime aibėje *S* yra *n* elementų, o aibėje *C* – *k* elementų. Ciklas **while** bus vykdomas min(*n, k*) kartų (t.y. tol, kol *A* bus tuščia), o operacija *B*i :=*B*i \ *B*j cikle **for** gali pareikalauti n elementarių žingsniu. Todėl šio algoritmo sudėtingumas *O*(*kn*∙ min(*n, k*)).

*Pavyzdys. S* = {1, 2, 3, 4, 5, 6}. *C* = {{1, 2, 3}, {3, 4}, {4, 5}, {2, 3, 6}}.

Randame *j* . *j* = 1, nes |*B*1| = max {|*B*1| = 3, |*B*2| = 2, |*B*3| = 2, |*B*4| = 3}.

*C’* = *C’* ∪ {*C*1} = ∅ ∪ {{1, 2, 3}} = {{1, 2, 3}}.

*A* := *A* \ *B*1 = {1, 2, 3, 4, 5, 6}\ {1, 2, 3} = {4, 5, 6}.

{*B*1, *B*2, *B*3, *B*4} = {∅, {4}, {4, 5}, {6}}.

Randame *j* . *j* = 3, nes |*B*3| = max {|*B*1| = 0, |*B*2| = 1, |*B*3| = 2, |*B*4| = 1}.

*C’* = *C’* ∪ {*C*3} = {{1, 2, 3}} ∪ {{4, 5}} = {{1, 2, 3}, {4, 5}}.

*A* := *A* \ *B*3 = {4, 5, 6}\ {4, 5} = {6}.

{*B*1, *B*2, *B*3, *B*4} = {∅, ∅, ∅, {6}}.

Randame *j* . *j* = 4, nes |*B*4| = max {|*B*1| = 0, |*B*2| = 0, |*B*3| = 0, |*B*4| = 1}.

*C’* = *C’* ∪ {*C*4} = {{1, 2, 3}, {4, 5}} ∪ {{2, 3, 6}} = {{1, 2, 3}, {4, 5}, {2, 3, 6}}.

*A* := *A* \ *B*4 = {6}\ {6} = ∅.

*Atsakymas*: *C’* = {*C*1, *C*2, *C*4} = {{1, 2, 3}, {4, 5}, {2, 3, 6}}.

Aprašytasis godus algoritmas nėra optimalus. Pavyzdžiui, *S* = {1, 2, 3, 4, 5, 6}. *C* = {{1, 2, 3}, {1, 4}, {2, 5}, {3, 6}}.

Denginys taikant godų algoritmą susideda iš 4 aibių. *C’* = {{1, 2, 3}, {1, 4}, {2, 5, {3, 6}}.

Optimalus denginys susideda iš 3 aibių *C’* = {{1, 4}, {2, 5, {3, 6}}.

**3. Algoritmų sudėtingumų apžvalga**

Bene geriausias rezultatas aprašytas straipsnyje [Joh74]. Godaus algoritmo sudėtingumas *O*(1 + ln |*S*|). Idėja ta pati: kiekviename etape imama aibė padengianti maksimalų skaičių dar nepadengtų elementų. Tik algoritmas kitoks:

Greedy-Set-cover (*S, C*)

1. *A* := *S*

2. *C’* := ∅

3. **while** *A* ≠ ∅ **do**

1. rasti *C*j ∈ *C* su didžiausiu |*C*j  ∩ *A*|

2. *A* := *A* \ *C*j

3. *C’* := *C’* ∪ {*C*j}

4. **return** *C’*

Straipsnyje [Fei98] įrodyta, kad nėra minimalaus aibės denginio algoritmo su *O*((1 – ε) ln |*S*|) koks bebūtų ε > 0. Jei toks egzistuotų, tai NP sudėtingumas būtų lygus klasės TIME(*n O*(log log *n*)) sudėtingumui.

Kituose straipsniuose aprašytų godžių algoritmų sudėtingumų įverčiai problemai *minimalus aibės* *denginys* yra silpnesni. Pavyzdžiui, darbe [Mou17] sudėtingumas *O*(*n*∙*k*2).

**Literatūros sąrašas**

[Fei98] U.Feige. A threshold of ln *n* for approximating set cover. Journal of the ACM. vol.45, iss.4,

1998, p.634-652.

[Joh74] D.S.Johnson. Approximation algorithms for combinatorial problems. Journal of computer and

system sciences 9, 1974, p.256-278.

[Kar72] R.M.Karp. Reducibility among combinatorial problems. Knygoje: R.E.Miller, J.W.Thatcher,

J.D.Bohlinger. Complexity of computer computations. New York:Plenum, 1972, p.85-103.

[Mou17] D.Mount. CMSC451: Lecture 9 greedy approximation: set cover.

<URL:https://www.cs.umd.edu/class/fail2017/cmsc451-0101/Lects/lect09-set-cover.pdf>. 2017