KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Dainius Šuopys

AUTOMATO TESTINĖS SEKOS MINIMIZAVIMO METODAI

Magistro darbas

Vadovas: habil. dr. R. Šeinauskas

KAUNAS, 2005

Turinys

1. Įvadas

Automatas – tai yra matematinis realiųjų sistemų aprašymo modelis.

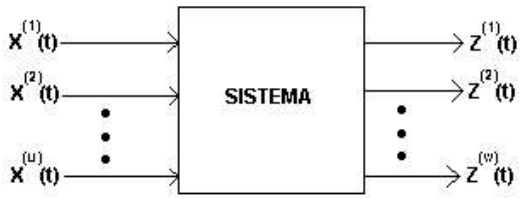
Daugelį problemų, sutinkamų moksle, galima suskirstyti į dvi kategorijas:

- analizės,
- sintezės.

Analizės atveju reikia nustatyti duotojo objekto (automato) funkcionavimo dėsnius. Sintezės atveju reikia suprojektuoti sistemą pagal duotus funkcionavimo dėsnius. Tiek sintezės, tiek analizės požiūriu sistemą charakterizuojančius kintamuosius galima suskirstyti į tokias grupes:

- Įėjimo kintamieji – tai kintamieji, generuojami kitos sistemos ir veikiantys tiriamąją sistemą.
- Išėjimo kintamieji – tai sistemos reakcija, charakterizuojanti sistemos poelgį, ji domina sistemos tyrėją.
- Tarpiniai kintamieji – tai dydžiai, nėra nei įėjimo, nei išėjimo kintamieji.

Schemiškai sistemą galima pavaizduoti kaip „juodą dėžę“ (1 pav.), turinčią baigtinį išvadų skaičių, prieinamų tos sistemos tyrėjui.



1 pav. Automato schema

Sekanti prielaida yra ta, kad kintamieji gali įgauti tik baigtinį reikšmių skaičių. Aibė reikšmių, kurias gali įgyti kintamasis v, vadinama alfabetu ir žymima V. Alfabeto V elementas v vadinamas simboliu. Jeigu i-tojo įėjimo alfabetą pažymėtume X^i, o i-tojo išėjimo alfabetą Z^i, tai galima parašyti tokias išraiškas:

$$X=X^1xX^2x...xX^u=\{(x^1, x^2, \ldots, x^u) \mid x^1 \varepsilon X^1, x^2 \varepsilon X^2, \ldots, x^u \varepsilon X^u\}$$

Sistemą, turinčią daug įėjimų ir išėjimų, suvedame į sistemą, turinčią vieną įėjimą ir išėjimą.

Tarpiniai kintamieji turi įtaką ryšiui tarp įėjimo ir išėjimo kintamųjų ir vadinami sistemos būsena. Sistemos būseną laiko momentu t_ν žymėsime s_ν. Sistemos visų galinių būsenų rinkinį vadinsime būsenų aibe ir žymėsime S. Baigtiniu automatu M vadinama sinchroninė sistema, kurioje naudojamas baigtinis įėjimo alfabetas X, baigtinis išėjimo alfabetas Z, baigtinė vidinių būsenų aibė S ir aprašoma dviem charakteringom funkcijom f_z (išėjimų funkcija) ir f_s (perėjimų funkcija).

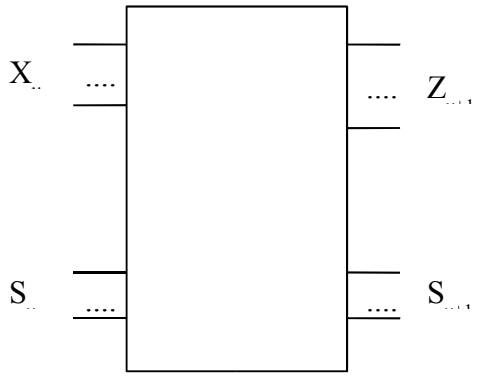
$$\begin{cases} z_{\nu+1} = f_z(x_\nu, s_\nu) \\ s_{\nu+1} = f_s(x_\nu, s_\nu) \end{cases}$$ Mili automatas (*Mealy Machine*)

Kai $f_z(x_\nu, s_\nu) = f_z(x_\nu) = z_\nu$, toks automatas vadinamas trivialiu (be atminties).

Kitas baigtinio automato tipas yra Muro automatas (*Moore Machine*).

$$\begin{cases} z_{\nu+1} = f_z(s_\nu) \\ s_{\nu+1} = f_s(x_\nu, s_\nu) \end{cases}$$

Mes automatą patogumo dėlei vaizduosime su vidinėm būsenom laiko momentu ν ir ν+1 (2 pav.).



2 pav. Automato schema su vidinėmis būsenomis

X_ν – automato įėjimai

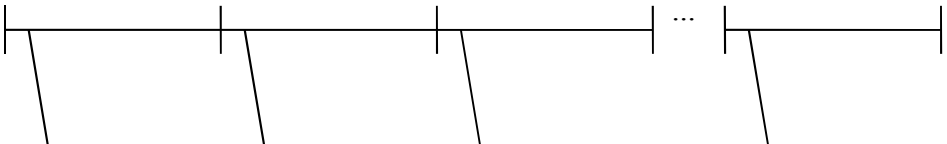
S_ν – automato vidinės būsenos, laiko momentu ν

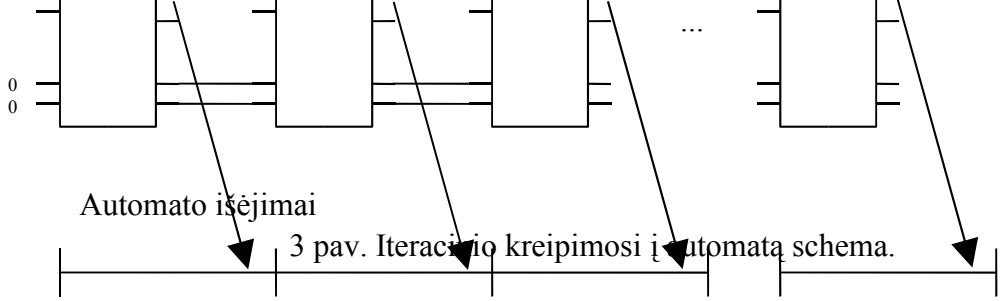
$Z_{\nu+1}$ – automato išėjimai, laiko momentu ν+1

$S_{\nu+1}$ – automato vidinės būsenos, laiko momentu ν+1

Tarp automato įėjimų, išėjimų ir vidinių būsenų egzistuoja sąryšiai. Sąryšis tarp įėjimų ir išėjimų egzistuoja tada, kai pakeitus vieną įėjimo reikšmę keičiasi bent viena išėjimo reikšmė. Pagal tokius sąryšius ir nustatoma kiek tų sąryšių yra ištestuota.

Automato įėjimai





3 pav. Iteracinio kreipimosi į automatą schema.

Pirmą kartą kreipiantis į automatą vidinės būsenos yra lygios nuliui. Sekantį kartą kreipiantis su kitu testinių duomenų rinkiniu vidinės būsenos yra paimamos iš prieš tai suskaičiuotų (3 pav.).

Sąryšių matricos skaičiavimo metodika. Invertuojama viena pradinių duomenų būsena ir lyginama kaip pasikeitė rezultatai su neinvertuota būsena. Jeigu rezultatai skiriasi, vadinasi egzistuoja ryšys tarp invertuotos pradinių duomenų būsenos ir pakitusių rezultatų būsenos.

Automato testavimui aktualūs tik tie duomenys, kurie tikrino bet vieną automato sąryšį. Testiniai duomenys atrenkami naudojant „Automato testavimo analizės programą". Šia programa atrinkti duomenys yra pertekliniai t.y. sąryšiai tikrinami daugiau nei vieną kartą. Todėl šiuos duomenis reikia optimizuoti. Tai yra standartinis aibės padengimo uždavinys (SCP – *set covering problem*).

Aibės padengimo uždavinys yra vienas iš tipinių uždavinių kombinatoriniame optimizavime ir yra plačiai studijuojamas keletą dešimtmečių. Jis yra pritaikomas įvairiuose atvejuose, pvz.: efektyvumo testavime, eksperimentų statistiniame projektavime, aerolinijų tvarkaraščių sudaryme, mašinų mokyme ir kompiuterinių virusų tikrinime. Pavyzdžiui, IBM T. J. Watson Research Center tyrinėtojai pritaikė aibės padengimo uždavinį atpažinti kompiuterinius virusus. Šiame pavyzdyje elementai, kurie turi būti padengti yra žinomi kompiuteriniai virusai (apie 5000), o padengimo aibės (apie 9000) tai yra 20 ar daugiau nuoseklių baitų eilutės iš virusų, kurios negali būti randamos „gerame" kode. Atlikus optimizavimą buvo rasta 180 eilučių padengimo aibė. Pakanka ieškoti tų 180 eilučių, norint patikrinti ar nėra žinomo viruso. Aibės padengimo uždavinys yra priskiriamas NP - sunkumo (*NP-hard*) uždaviniams, ir vienas iš geriausių polinominio aproksimavimo laikų yra gerai žinomo „Gobšusis" (*Greedy*) algoritmo. Algoritmo principas yra toks: kiekviename žingsnyje stengiamasi paimti nenaudojamą rinkinį, kuris padengia didžiausią likusių (neuždengtų) elementų aibę.

Formalus uždavinio aprašymas:

Duota:

- n elementų aibė U,
- aibės U poaibių rinkinys $\mathbf{S} = \{S_1,....,S_k\}$,
- kainos funkcija c: $\mathbf{S} \to Q^+$.

Rasti mažiausios kainos kolekciją $C \subseteq \mathbf{S}$, kuris padengtų visus U elementus, tai yra $\bigcup_{S \in C} S = U$.

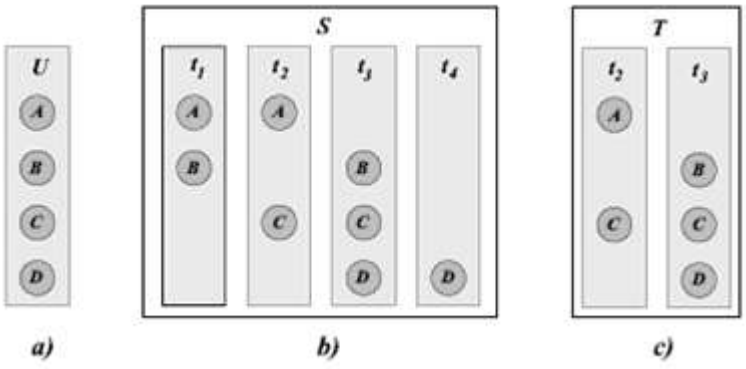
Programinis formulavimas:

Minimizuoti $\sum_{S \in S} c\,(\mathrm{S})\, x_S$,

Iki $\sum_{S:e \in S} x_S \geq 1$, $e \in U$,

$x_S \in \{0,1\}$, $S \in \mathbf{S}$.

Padengimo uždavinio pavyzdys yra pateikiamas 4 paveikslėlyje. 4a parodyta elementų aibė U, kurią reikia padengti. 4b duoti elementų poaibiai, kuriais reikia padengti aibę U. 4c pateiktas uždavinio sprendimas $\{t_2, t_3\}$.



4 paveikslėlis. Aibės padengimo uždavinio pavyzdys

Mūsų atveju padengimo aibės uždavinys susideda iš nuliukų-vienetukų matricos *m x n* eilučių poaibio radimo, kuris padengtų visus matricos stulpelius mažiausia kaina. Tegul M = {1,2,..,m} ir N = {1,2,..,n} yra atitinkamai matricos stulpeliai ir eilutės. A = (a_{ij}) yra nuliukų - vienetukų matrica, o c = (c_i) yra n dimensijų sveikasis vektorius, kuris simbolizuoja eilutės i kainą (mūsų atveju visos eilutės turi vienodą kainą). Mes sakome, kad eilutė i padengia stulpelį j, jei a_{ij}=1. Problema formaliai gali būti aprašyta taip:

Sumažinti v(SCP) = $i = \sum_{n=1}^{n} c_i x_i$

t.y. $\sum_{i=1}^{n} a_{ij} x_i \geq 1$, i= 1,...,n.

$x_i \in \{0,1\}$,

kur x_i=1, jei eilutė yra sprendimas, ir x_i=0, priešingu atveju.

Aibės padengimo uždavinio paprastumas daro jį idealiu sprendžiant tikras kombinatorinio optimizavimo problemas. Norint pritaikyti SCP (*Set Covering Problem*) reikalinga tiktai turėti elementus (eilutes), kurie turi būti padengti (patenkinti, priskirti) minimalia tų elementų (stulpelių) poaibių kaina. Tos eilutės gali būti reikšti keliones arba darbo dalis įgulos tvarkaraščio sudaryme, ir t.t.

Kadangi problema yra priskiriama NP - sunkumo uždaviniams ir praktinių pritaikymų yra labai daug, tai yra pasiūlyta daug šio uždavinio sprendimo variantų tarp pačių geriausių algoritmų yra apytikriai algoritmai (*approximate algorithms*).

Tikslas. Šiame darbe ištirsiu kelis aibės padengimo algoritmus. Pagrindinis dėmesys bus skirtas „Gobšaus“ (*Greedy*) algoritmo tyrimams bei jo tobulinimams. Norint patikrinti ar uždavinio sprendinys priklauso nuo pradinio poaibių kiekio dydžio, su tuo pačiu automatu bus atlikti 4 tyrimai. Kiekvienu atveju skirsis pradinis poaibių dydis. Pasirinkau poaibių dydžius 1x, 3x, 5x ir 10x. Skaičius reiškia kiek minimaliai yra tikrinamas vienas automato sąryšis: pavyzdžiui 10x, jeigu rinkinys tikrina bent vieną automato sąryšį dešimtą kartą, jis yra įtraukiamas į pradinį poaibių sąrašą (minimalus sąryšio tikrinimų kiekis yra 10 ir daugiau).

2. Aibės padegimo uždavinio algoritmai

2.1.Tikslūs algoritmai (Exact algorithms)

Tikslūs metodai gali išspręsti pavyzdžius iki kelių šimtų eilučių ir kelių tūkstančių stulpelių. Fisher ir Kedia pristatė algoritmą B&B, kuris gali spręsti pavyzdžius su 200 eilučių ir 2000 stulpelių. Kristofides ir Paixao pristatė „Medžio Paieškos“ algoritmą pagrįstą būsenų erdvės sumažinimu ir dinaminiu programavimu, kuris gali išspręsti problemas iki 400 eilučių ir 4000 stulpelių.

Aibės padengimo uždavinys yra tiksliai sprendžiamas naudojant šakojimosi ir ribojimo („branch & bound“) algoritmą. Skaldymas yra vykdomas pagal stulpelius, t.y. problemos yra generuojamos svarstant ar pasirinktas stulpelis yra sprendimo aibėje, ar ne. Ši metodas formuoja paieškos medį iteratyviai imdamas kintamuosius. Generuodamas du sub-medžius su kintamuoju atitinkamai nustatytu į 0 ir 1. dvi skirtingos technikos apkarpo paieškos erdvę. Pirma mažinimo technika, darant išvadą iš dabartinio dalinio uždavinio tam, kad sumažinti problemą. Antra, apatinė riba potencialiam sprendiniui yra suskaičiuojama esamame kelyje. Visas sub-medis gali būti išgenėtas, jei geriausio dabartinio sprendinio kainos riba yra viršijama.

Kainos riba yra kaina dabartinio sprendinio plius apytikriai įvertinant eilučių, kurių reikės padengti stulpelius, skaičių. Standartinė apatinės ribos skaičiavimo technika yra suskaičiuoti maksimaliai nepriklausomų eilučių aibę, kuri vis dar padengia visus stulpelius. Tai yra sunki problema, todėl padengimo uždaviniams taikomi euristiniai metodai.

Daugiausia patobulinimų yra padaryta taikant papildomą ir sudėtingesnį mažinimą, ribojimo schemas, ir euristikas pasirenkant kintamuosius.

2.2.Apytikriai algoritmai (Approximate algorithms)

Apytikriai algoritmai yra svarbūs aibės padengimo uždaviniuose, dėl tikslių metodų apribojimų ir didelio sąrašo programų naudojančių didelio dydžio SCP. Pasiūlyti euristiniai metodai SCP sprendimui yra pagrįsti labai plačia technikų sfera (grupe, aibe) nuo „*Greedy*“ konstrukcijos algoritmų iki Lagrangian‘o meta euristinių metodų arba genetinių algoritmų. Labai populiarus Lagrangrian pagrįstas euristinis metodas, kuris skaičiuoja stulpelio svorius susijusius su galimybe, kad stulpelis yra galima dalis optimalaus sprendinio. Per kelius

pastaruosius metus sukurta keletas meta euristinių SCP sprendimo metodų, kurie pasiekė nepaprastai gerą sprendimą. Kol kas, geriausi rezultatai yra genetinio algoritmo sukurto Chu ir Beasley.

Čia pateikiami galimi metodai ir keliai algoritmams tobulinti. Padengimo uždavinio sprendimo galimybės:

- Atrinkimas (*selection*). Individai (eilutės) yra sužymimos pagal tinkamumą. Aukštesnio rango eilutės turi didesnę tikimybę priklausyti sprendiniui.
- Keitimas (*mutation*). Nedideli atsitiktiniai elementų keitimai.
- Pakeitimas (*replacement*). „Elitist“ strategija, t.y. geriausio seno sprendinio 20% yra pasiliekama. Kita dalis yra pakeičiama.
- Populiacija (*population*). Naujas sprendimas paveldi geras seno sprendimo dalis.
- Tinkamumas (*fitness*). Geresnis senas sprendimas turi didesnę tikimybę pagerinti naują sprendimą.

Galimi kriterijai kaip pasirinkti eilutę:

- Pagal jos kainą
- Kiek neuždengtų stulpelių jinai padengs
- Ar jos įtraukimas padaro kitą eilutę pertekline
- Kiek stulpelių jinai uždengs iš viso.
- Kiek stulpelių eilutė dalinasi su jau atrinktomis eilutėmis

Jei eilučių tvarka yra gera (tinkama), tai ir reliatyviai paprastas euristinis algoritmas gali optimaliai išspręsti padengimo uždavinį.

2.2.1. Paprastasis (*Simply*) algoritmas

Algoritmas pereina pradinę rinkinių aibę vieną sykį. Atrenka tik tuos rinkinius, kurie tikrina bent vieną automato sąryšį. Kitais žodžiais sakant iš 10x arba kokios kitos rinkinių aibės padaro 1x rinkinio aibę.

Šis algoritmas yra įgyvendintas norint jo rezultatus palyginti su kitais algoritmais. Kadangi kai paduodami rinkiniai, kurie tikrina automatą 3x, 5x arba 10x kartų, yra sunku spręsti apie optimizavimo kokybę. Nėra su kuo palyginti rezultatų.

2.2.2. Paparastas2 (*Simply_other*) algoritmas

Šis algoritmas į uždavinį žvelgia iš kitos pusės nei „*Simply*“. Algoritmas ieško pirmo nepadengto elemento, o po to ieško rinkinio, kuris tą elementą padengia.

Pradiniai duomenys: baigtinės aibės U padengimas $S = \{S_1,....,S_n\}$.
Rezultatai: aibės U padengimas $\hat{S} = \{ \hat{S}_1,...., \hat{S}_k\} \subseteq S$
Pradžia

1. $\hat{S} \leftarrow Ø$
2. kol $U \neq Ø$
3. paimti $Uj \in U$, kur Uj = 1 (nepadengtas)
4. paimti $Si^* \in S$, kuri padengia Uj elementą
5. $U \leftarrow U - Si^*$
6. $\hat{S} \leftarrow \hat{S} \cup \{\hat{S}i\}$
7. gražinti $\hat{S}$

Pabaiga.

2.2.3. Rikiavimo (*Sort*) algoritmas

Algoritmo pagrindinė mintis yra ta, kad pirmiausia atrenkami tie rinkiniai, kurie turi didžiausią poveikį padengimo uždaviniui.

Pradiniai duomenys: baigtinės aibės U padengimas $S = \{S_1,....,S_n\}$.
Rezultatai: aibės U padengimas $\hat{S} = \{ \hat{S}_1,...., \hat{S}_k\} \subseteq S$
Pradžia

1. $\hat{S} \leftarrow Ø$
2. vieną kartą skanuojamas S, norint suskaičiuoti kiekvieno elemento poveikį U aibei
3. išrikiuoti S pagal poveikį U nedidėjančia tvarka
4. kol $U \neq Ø$
5. imti iš aibės S po vieną elementą S_i, kuris turi poveikį U
6. $U \leftarrow U - S_i$
7. $\hat{S} \leftarrow \hat{S} \cup \{S_i\}$

8. gražinti $\hat{S}$

Pabaiga.

2.2.4. „Gobšusis“ (*Greedy*) algoritmas

Gobšusis (*greedy*) algoritmas yra metodas skirtas optimalaus problemos sprendinio paieškai apimant dideles homogenines duomenų struktūras. Gobšus algoritmas yra pagrįstas tuo, kad kiekviename žingsnyje paima tokį elementą, kuris padengia daugiausiai nepadengtų elementų. Mūsų atveju bus pirmas bus paimamas tas elementas, kuris ištestuoja daugiausiai dar netestuotų automato sąryšių. Po kiekvieno žingsnio perskaičiuojamas nepaimtų elementų naudingumas. Greedy algoritmas pagrįstas tuo, kad lokalus optimumas yra globalaus optimumo dalis. Pagrindinis algoritmo privalumas yra tas, kad jis yra lengvai suprantamas ir lengvai programuojamas. Deja, daugeliu atvejų nėra garantijų, kad lokalus optimumas yra globalaus optimumo dalis.

Skirtumas nuo rikiavimo (*sort*) algoritmo yra, kad po kiekvieno elemento atrinkimo yra perskaičiuojami netrinktų eilučių poveikiai. Atrenkama ta eilutė, kuri turi didžiausią poveikį.

Pradiniai duomenys: baigtinės aibės U padengimas $S = \{S_1,....,S_n\}$.

Rezultatai: aibės U padengimas $\hat{S} = \{ \hat{S}_1,...., \hat{S}_k\} \subseteq S$

Pradžia

1. $\hat{S} \leftarrow Ø$
2. kol $U \neq Ø$
3. paimti $S_{i*} \in S$, kuris turi didžiausią poveikį $|S_{i*}|$
4. $U \leftarrow U - S_{i*}$
5. $\hat{S}_i \leftarrow$ originalų S_{i*} rinkinį
6. $\hat{S} \leftarrow \hat{S} \cup \{\hat{S}_i\}$
7. kiekvienam $S_i \in S$, $S_i \leftarrow Si - S_{i*}$
8. gražinti $\hat{S}$

Pabaiga.

Pagrindinis algoritmo trūkumas yra tas, kad ne visada lokalus optimumas veda į globalų optimumą.

2.2.5. SaGaR (*Sort-and-Greedy-and-Remove*) algoritmas

Yra du pagrindiniai skirtumai tarp Greedy ir SaGaR algoritmų. Pirmiausia, prieš kviečiant Greedy algoritmą, mes 2 – 4 žingsniais apdorojame originalų padengimą S. Rikiavimo procesas verčia SaGaR algoritmą pasirinkti didžiausią rinkinį iš kelių rinkinių, kurie maksimaliai padengia U. Vietoj to Greedy algoritmas pasirenka rinkinius jų natūralia tvarka.

Trečias žingsnis pirmiausia parenka tokius rinkinius, kurie padengia unikalius langelius (kurių nepadengia joks kitas rinkinys). Šiuo žingsniu tokie rinkiniai paimami pirmiausia. Jie bet kokiu atveju vis tiek bus paimti.

Po Greedy algoritmo kvietimo, mes 6 – 8 žingsniais pašaliname perteklinius rinkinius. Pastebime, kad paskutinio rinkinio tikrinti nereikia, nes be to rinkinio $\hat{S}$ nebus padengimo aibė. Eksperimentai rodo kad tai yra svarbus žingsnis tobulinant sprendinį.

Pradiniai duomenys: baigtinės aibės U padengimas $S = \{S_1,....,S_n\}$.

Rezultatai: aibės U padengimas $\hat{S} = \{ \hat{S}_1,...., \hat{S}_k\} \subseteq S$

Pradžia

1. $\hat{S} \leftarrow Ø$
2. vieną kartą skanuojamas S, norint suskaičiuoti kiekvieno U elemento dažnumą ir kiekvieno $S_i \in S$ dydį
3. iš S pašalinti visus poaibius S_j, kur dažnis didesnis už 1, $x \in S_j$ ir $| S_j| = 1$, rezultatą pažymime S’
4. išrikiuoti S’ pagal dydi nedidėjančia tvarka
5. kviesti algoritmą Greedy su pradiniais duomenimis (U,S’), rezultatą pažymime $\hat{S}$
6. $r \leftarrow |\hat{S}|$
7. i ← nuo 1 iki r-1
8. jei $\left| \bigcup_{\substack{j=1 \\ j \neq i}}^{r} \hat{S}_j \right| = |U|$ tada $\hat{S} \leftarrow \hat{S} - \hat{S}_i$
9. gražinti $\hat{S}$

Pabaiga.

2.2.6. SaG (*Sort-and-Greedy*) algoritmas

Šis algoritmas yra SaGaR algoritmo sumažintas variantas. Jis paimtas norint išsiaiškinti kokį realų poveikį algoritmui SaGaR turi šalinimo procedūra.

Pradiniai duomenys: baigtinės aibės U padengimas $S = \{S_1,....,S_n\}$.

Rezultatai: aibės U padengimas $\hat{S} = \{ \hat{S}_1,...., \hat{S}_k\} \subseteq S$

Pradžia

1. $\hat{S} \leftarrow Ø$
2. vieną kartą skanuojamas S, norint suskaičiuoti kiekvieno U elemento dažnumą ir kiekvieno $S_i \in S$ dydį
3. iš S pašalinti visus poaibius S_j, kur dažnis didesnis už 1, $x \in S_j$ ir $| S_j| = 1$, rezultatą pažymime S'
4. išrikiuoti S' pagal dydi nedidėjančia tvarka
5. kviesti algoritmą Greedy su pradiniais duomenimis (U,S'), rezultatą pažymime $\hat{S}$
6. gražinti $\hat{S}$

Pabaiga.

2.2.7. GaR (*Greedy-and-Remove*) algoritmas

Šis algoritmas yra SaGaR algoritmo sumažintas variantas. Jis paimtas norint išsiaiškinti kokį realų poveikį algoritmui SaGaR turi unikalių (tikrinančių tik vieną automato sąryšį) rinkinių atrinkimo ir rikiavimo veiksmai šalinimo procedūra.

Pradiniai duomenys: baigtinės aibės U padengimas $S = \{S_1,....,S_n\}$.

Rezultatai: aibės U padengimas $\hat{S} = \{ \hat{S}_1,...., \hat{S}_k\} \subseteq S$

Pradžia

1. $\hat{S} \leftarrow Ø$
2. kviesti algoritmą Greedy su pradiniais duomenimis (U,S'), rezultatą pažymime $\hat{S}$
3. $r \leftarrow |\hat{S}|$
4. i ← nuo 1 iki r-1

5. jei $\left|\bigcup_{\substack{j=1 \\ j\neq i}}^{r} \hat{S}_j\right| = |U|$ tada $\hat{S} \leftarrow \hat{S} - \hat{S}_i$
6. gražinti $\hat{S}$

Pabaiga.

2.2.8. Šalinimo (*Remove*) algoritmas

Vienas esminių sprendinio optimizavimo etapų yra perteklinių rinkinių pašalinimas. Toks šalinimas yra atliekamas SaGaR algoritme trečiame etape.

Pradiniai duomenys: baigtinės aibės U padengimas $S = \{S_1,....,S_n\}$.

Rezultatai: aibės U padengimas $\hat{S} = \{ \hat{S}_1,...., \hat{S}_k\} \subseteq S$

Pradžia

1. $S \leftarrow Ø$
2. $r \leftarrow |S|$
3. $i \leftarrow$ nuo 1 iki r-1
4. jei $\left|\bigcup_{\substack{j=1 \\ j\neq i}}^{r} S_j\right| = |U|$ tada $S \leftarrow S - S_i$
5. gražinti S

Pabaiga.

2.2.9. Atsitiktinis – gobšusis (*Random Greedy*) algoritmas

Rinkiniai yra paimami atsitiktinai. Rinkinio atrikimo tikimybė yra skaičiuojama pagal jo poveikį sąryšio matricai (kiek nepadengtų elementų jis padengs).

Pradiniai duomenys: baigtinės aibės U padengimas $S = \{S_1,....,S_n\}$.

Rezultatai: aibės U padengimas $\hat{S} = \{ \hat{S}_1,...., \hat{S}_k\} \subseteq S$

Pradžia

1. $\hat{S} \leftarrow Ø$

2. kol $U \neq \emptyset$
3. kiekvienam $Si \in S$,
4. jei $x^*_j \geq \frac{1}{f}$
5. $\hat{S} \leftarrow \hat{S} \cup \{\hat{S}_i\}$
6. gražinti $\hat{S}$

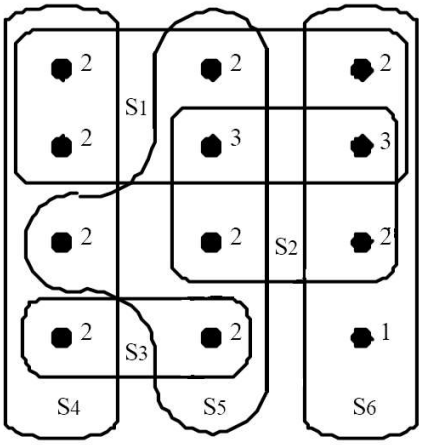
Pabaiga.

3. Aibės padengimo uždavinio algoritmų tyrimas

3.1.Algoritmų veikimo pavyzdžiai

Pateikiami keli realizuotų algoritmų veikimo pavyzdžiai. Kiekviename paveikslėlyje pateikiamos matricos, kurių yra žinomas optimalus padengimas. Prie kiekvieno taško parašytas skaičius reiškia to taško padengimo dažnumą. Pavyzdžiui 1 pav. matricos pirmas taškas (pirma eilutė, pirmas stulpelis) padengtas 2 kartus rinkinių S_1 ir S_4. Pavyzdžiuose sprendimo tvarka yra nekeista, t.y. rinkiniai yra paimami būtent tokia tvarka kaip ir pateikta. Kiekvienam pavyzdžiui taip pat pateikiamas realizuotų algoritmų sprendimas su rinkinių paėmimo tvarka.

1 pavyzdyje (5 pav.) matricos optimalus padengimas yra $S^*=\{ S_4, S_5, S_6\}$.

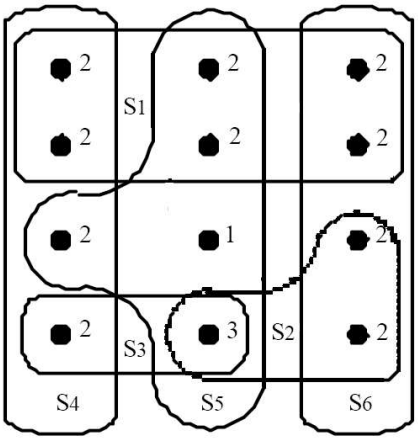


5 pav. Matrica sudaryta iš 12 taškų padengtų rinkiniais 6 rinkiniais: $S=\{S_1, S_2, S_3, S_4, S_5, S_6\}$

1 lent. Detalūs pirmo pavyzdžio sprendiniai

Algoritmas	Rinkinių paėmimo eiliškumas	Rezultatas
Simply	$S^*=\{S_1, S_2, S_3, S_4, S_6\}$	5/6
Simply_other	$S^*=\{S_1, S_4, S_2, S_3, S_6\}$	5/6
Sort	$S^*=\{S_1, S_5, S_2, S_4, S_6\}$	5/6
Greedy	$S^*=\{S_1, S_5, S_6, S_3\}$	4/6
SaGaR	1. Rikiavimas (Sort) – $S^*=\{S_6\}$ 2. Gobšus (Greedy) – $S^*=\{S_5, S_4\}$ 3. Išmetimas (Remove) – neišmetė jokio elemento	3/6
SaG	1. Rikiavimas (Sort) – $S^*=\{S_6\}$ 2. Gobšus (Greedy) – $S^*=\{S_5, S_4\}$	3/6
GaR	1. Gobšus (Greedy) – $S^*=\{ S_1, S_5, S_6, S_3\}$ 2. Išmetimas (Remove) – neišmetė jokio elemento	3/6
R	Išmetė rinkinius $\{S_1, S_2, S_3\}$, rezultatas $S^*=\{S_4, S_5, S_6\}$	3/6
Probability	$S^*=\{S_3, S_1, S_2, S_4, S_6\}$	5/6
SaGaR_be_Sort	1. Rikiavimas (Sort) – $S^*=\{S_6\}$ 2. Gobšus (Greedy) – $S^*=\{S_5, S_4\}$	3/6

2 pavyzdyje (6 pav.). Pakeičiame pirmo pavyzdžio 2 rinkinio vietą. Matricos optimalus padengimas yra $S^*=\{ S_5, S_6, S_4\}$.

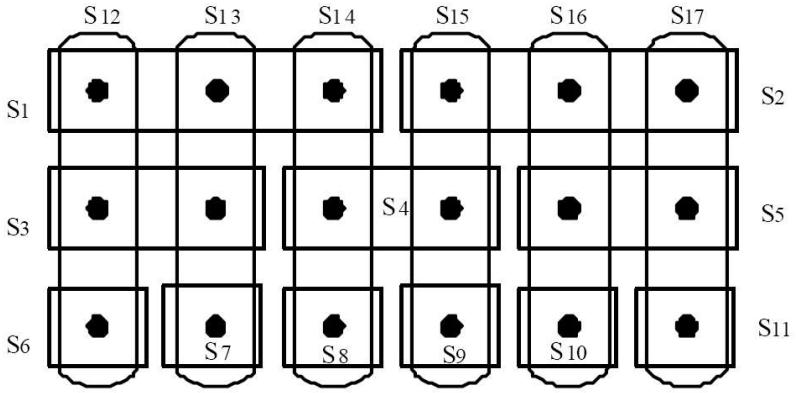


6 pav. Optimalus sprendimas yra $S^*=\{S_5, S_6, S_4\}$

2 lent. Detalūs antro pavyzdžio sprendiniai

Algoritmas	Rinkinių paėmimo eiliškumas	Rezultatas
Simply	$S^*=\{S_1, S_2, S_3, S_4, S_5\}$	5/6
Simply_other	$S^*=\{S_1, S_4, S_5, S_2\}$	4/6
Sort	$S^*=\{S_1, S_5, S_4, S_6\}$	4/6
Greedy	$S^*=\{S_1, S_2, S_4, S_5\}$	4/6
SaGaR	1. Rikiavimas (Sort) – $S^*=\{S_5\}$ 2. Gobšus (Greedy) – $S^*=\{ S_1, S_6, S_4\}$ 3. Išmetimas (Remove) – $\{S_1\}$	3/6
SaG	1. Rikiavimas (Sort) – $S^*=\{S_5\}$ 2. Gobšus (Greedy) – $S^*=\{ S_1, S_6, S_4\}$	4/6
GaR	1. Gobšus (Greedy) – $S^*=\{ S_1, S_2, S_4, S_5\}$ 2. Išmetimas (Remove) – neišmetė nieko	4/6
R	Išmetė rinkinius $\{S_1, S_2, S_3\}$, rezultatas $S^*=\{S_4, S_5, S_6\}$	3/6
Probability	$S^*=\{S_1, S_3, S_4, S_5, S_6\}$	5/6
SaGaR_be_Sort	1. Rikiavimas (Sort) – $S^*=\{S_5\}$ 2. Gobšus (Greedy) – $S^*=\{S_1, S_2, S_3\}$	4/6

3 pavyzdys.



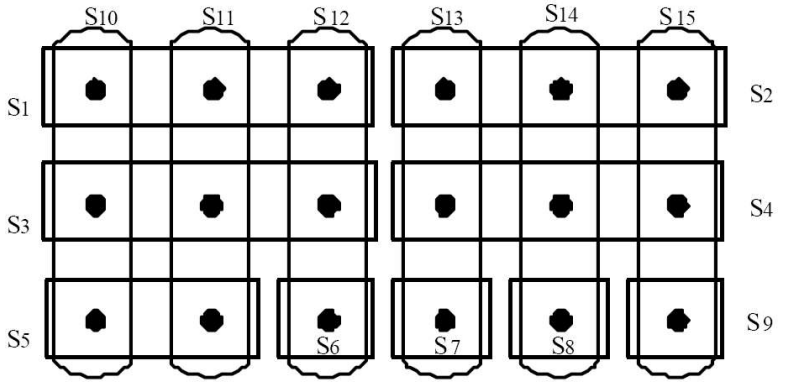
7 pav. Optimalus sprendimas yra $S^*=\{S_{12}, S_{13}, S_{14}, S_{15}, S_{16}, S_{17}\}$

3 lent. Detalūs trečio pavyzdžio rezultatai

Algoritmas	Rinkinių paėmimo eiliškumas	Rezultatas
Greedy	$S^*=\{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}\}$	11/17
SaGaR	1. Rikiavimas (Sort) – S*={} 2. Gobšus (Greedy) – $S^*=\{S_1, S_2, S_{12}, S_{13}, S_{14}, S_{15}, S_{16}, S_{17}\}$ 3. Išmetimas (Remove) – $\{S_1, S_2\}$	6/17
SaG	1. Rikiavimas (Sort) – S*={} 2. Gobšus (Greedy) – $S^*=\{ S_1, S_2, S_{12}, S_{13}, S_{14}, S_{15}, S_{16}, S_{17}\}$	8/17
GaR	1. Gobšus (Greedy) – $S^*=\{ S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}\}$ 2. Išmetimas (Remove) – neišmetė nieko	11/17
R	Išmetė rinkinius $\{ S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}\}$, rezultatas $S^*=\{ S_{12}, S_{13}, S_{14}, S_{15}, S_{16}, S_{17}\}$	6/17

3x lentelė ....

4 pavyzdys

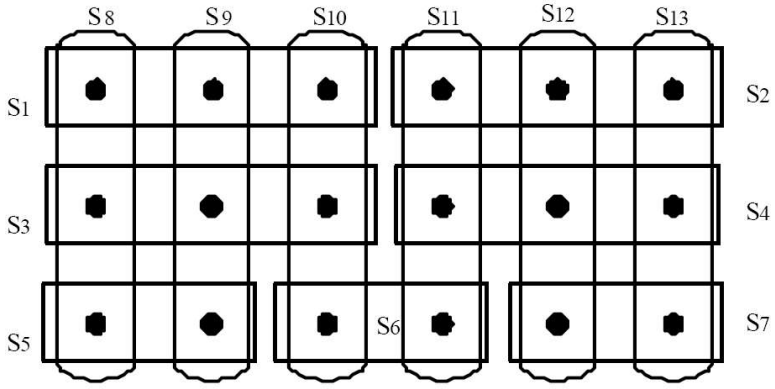


8 pav. Optimalus sprendimas yra $S^*=\{ S_{10}, S_{11}, S_{12}, S_{13}, S_{14}, S_{15}\}$

4 lent. Detalūs rezultatai

Algoritmas	Rinkinių paėmimo eiliškumas	Rezultatas
Greedy	$S^*=\{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9\}$	9/15
SaGaR	1. Rikiavimas (Sort) – $S^*=\{\}$ 2. Gobšus (Greedy) – $S^*=\{S_1, S_2, S_3, S_4, S_5, S_{12}, S_{13}, S_{14}, S_{15}\}$ 3. Išmetimas (Remove) – $\{S_2, S_4\}$	7/15
SaG	1. Rikiavimas (Sort) – $S^*=\{\}$ 2. Gobšus (Greedy) – $S^*=\{ S_1, S_2, S_3, S_4, S_5, S_{12}, S_{13}, S_{14}, S_{15}\}$	9/15
GaR	1. Gobšus (Greedy) – $S^*=\{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9\}$ 2. Išmetimas (Remove) – neišmetė nieko	9/15
R	Išmetė rinkinius $\{ S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9\}$, rezultatas $S^*=\{ S_{10}, S_{11}, S_{12}, S_{13}, S_{14}, S_{15}\}$	6/15

5 pavyzdys

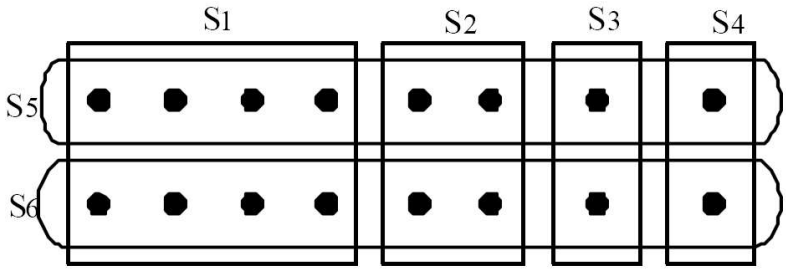


9 pav. Optimalus sprendimas yra $S^*=\{ S_8, S_9, S_{10}, S_{11}, S_{12}, S_{13}\}$

5 lent. Detalūs penkto pirmo pavyzdžio rezultatai

Algoritmas	Rinkinių paėmimo eiliškumas	Rezultatas
Greedy	$S^*=\{S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$	7/13
SaGaR	1. Rikiavimas (Sort) – $S^*=\{\}$ 2. Gobšus (Greedy) – $S^*=\{ S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$ 3. Išmetimas (Remove) – $\{\}$	7/13
SaG	1. Rikiavimas (Sort) – $S^*=\{\}$ 2. Gobšus (Greedy) – $S^*=\{ S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$ 3. Išmetimas (Remove) – $\{\}$	7/13
GaR	1. Rikiavimas (Sort) – $S^*=\{\}$ 2. Gobšus (Greedy) – $S^*=\{ S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$ 3. Išmetimas (Remove) – $\{\}$	7/13
R	Išmetė rinkinius $\{ S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$, rezultatas $S^*=\{S_8, S_9, S_{10}, S_{11}, S_{12}, S_{13}\}$	6/13

6 pavyzdys



10 pav. Optimalus sprendimas yra $S^*=\{S_5, S_6\}$

6 lent. Detalūs šešto pavyzdžio rezultatai

Algoritmas	Rinkinių paėmimo eiliškumas	Rezultatas
Greedy	$S^*=\{S_1, S_2, S_3, S_4\}$	4/6
SaGaR	1. Rikiavimas (Sort) – S*={} 2. Gobšus (Greedy) – S*={S_1, S_5, S_6 } 3. Išmetimas (Remove) – {S_1}	2/6
SaG	1. Rikiavimas (Sort) – S*={} 2. Gobšus (Greedy) – S*={S_1, S_5, S_6}	3/6
GaR	1. Gobšus (Greedy) – S*={S_1, S_2, S_3, S_4} 2. Išmetimas (Remove) – neišmetė nieko	4/6
R	Išmetė rinkinius { S_1, S_2, S_3, S_4}, rezultatas $S^*=\{S_5, S_6\}$	2/6

Kaip matome pateiktuose pavyzdžiuose išsiskiria SaGaR algoritmas. Jis patobulina „Gobšaus“ algoritmo sprendinį. SaGaR algoritmas atrenka unikalius rinkinius, o pabaigoje patikrina ar rezultate nėra perteklinių rinkinių.

3.2. Tyrimo atlikimo tvarka

Pradinis poaibių rinkinys yra sugeneruotas „Automato testavimo analizės programa“. Pradinis poaibių rinkinys generuojamas naudojant atsitiktinį generavimą. Pirmiausiai sugeneruojami 10x duomenys. Toliau 10x rinkinių pradinis poaibis sumažinamas iki 5x, 3x ir 1x. Taip elgiantis siekiama panaikinti atsitiktinio generavimo poveikį.

Kiekvienas algoritmas yra atliekamas 4 kartus su kiekvienu automatu, naudojant 10x, 5x, 3x ir 1x dydžių pradinių elementų poaibį. Lentelėse pateiksime optimizavimo rezultatus ir laiką, per kurį buvo atliktas optimizavimas. Vienoje lentelėje pateiksime vienos bibliotekos rezultatus, t.y. bus pateikti visi keturi tyrimai (poaibiai skirtingais dydžiais) vienoje lentelėje. Optimalūs rezultatai bus pajuodinti.

Optimizavimo algoritmai buvo realizuoti C++ programavimo kalba, naudojant DEV C++ 4.9.9.0 kompiliatorių. Tyrimas atliktas, naudojant AMD Athlon(tm) XP 2800+ 2,08 GHz procesoriaus ir 512 darbinės atminties, turintį kompiuterį. Padengimo aibės buvo sugeneruotos atsitiktinai naudojant „Automato testinių sekų tyrimo programą". Pirmiausia buvo generuojama sekų rinkiniai, kurie vieną sąryšį tikrino bent 10 sykių (10x). Siekiant pašalinti atsitiktinio generavimo poveikį, iš 10x aibės buvo padaryto 5x, 3x ir 1x aibės. Kiekvieno rinkinio svoris (kaina) yra tas pats.

3.3.Naudojamų bibliotekų duomenys

Tyrimas atliktas su dešimčia testinių (*benchmark*) schemų. Schemų pradiniai duomenys yra pateikti 7 lentelėje. Lentelėje nurodomi testinių schemų įėjimų ir išėjimų kiekis bei maksimalus sąryšių kiekis schemose (matricos užpildymas). Schemos c7552 pilnas sąryšių kiekis nėra pasiektas. Schemų užpildymo tyrimas yra pateiktas 1 priede.

7 lent. Testinių schemų duomenys

Schemos pavadinimas	Įėjimų skaičius	Išėjimų skaičius	Maksimalus matricos užpildymas
c432	36	7	540
c499	41	32	5184
c880	60	26	1326
c1355	41	32	5184
c1908	33	25	3004
c2670	157	64	3320
c3540	50	22	2588
c5315	178	123	10540
c6288	32	32	3068
c7552	206	107	>12000

3.4.Tyrimo rezultatai

Toliau pateikiame lenteles, kuriose nurodyti algoritmų tyrimo rezultatai. Lentelės viršuje nurodyta tiriamoji schema. Toje pačioje eilutėje nurodyta pradinės duomenų aibės dydis (schemai c432 pradinių duomenų dydis yra 264, kai duomenų aibės dydis yra 10x). Pradiniai duomenys yra paduodami keturių dydžių (10x, 5x, 3x ir 1x). Toliau kiekvienai pradinių

duomenų pateikiami algoritmų rezultatai (rezultatų aibės dydis ir algoritmo atlikimo laikas, atitinkamai stulpeliai „rez“ ir „laikas, ms“). Pajuodintas yra daugiausiai suminimizuotas rezultatas.

8 lent. Schemos c432 rezultatai ir algoritmų atlikimo laikai

c432	264		138		90		45	
	10x		5x		3x		1x	
	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms
Simply	45	15	45	16	45	15	45	15
Simply_other	34	15	34	15	35	15	37	15
Sort	35	47	35	16	34	15	39	15
Greedy	**22**	157	24	47	28	16	35	16
SaGaR	**22**	235	**23**	78	**25**	47	**34**	16
SaG	23	203	**23**	63	27	31	**34**	0
GaR	**22**	187	**23**	78	26	47	35	31
Remove	36	1015	33	265	30	110	35	47
Random-Greedy	43	47	40	16	41	15	41	15
SaGaR_bS	**22**	188	**23**	78	26	47	**34**	31

9 lent. Schemos c499 rezultatai ir algoritmų atlikimo laikai

c499	2059		1269		878		429	
	10x		5x		3x		1x	
	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms
Simply	429	532	429	312	429	203	429	78
Simply_other	335	328	335	203	335	141	349	47
Sort	279	7500	292	2766	312	1312	352	234
Greedy	**182**	108578	211	40015	238	17937	326	3437
SaGaR	183	154640	211	68422	**233**	39563	**317**	19813
SaG	185	114656	214	42109	243	18922	318	109
GaR	**182**	155859	**208**	72703	**233**	46219	319	22172
Remove	321	1107813	337	350979	313	149297	323	26906
Random-Greedy	444	8140	436	2812	426	1469	379	7562
SaGaR_bS	**182**	153359	**208**	72485	**233**	47719	318	18688

10 lent. Schemos c880 rezultatai ir algoritmų atlikimo laikai

c880	848		473		314		142	
	10x		5x		3x		1x	
	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms
Simply	142	93	142	62	142	31	142	15
Simply_other	104	31	107	16	107	15	116	16
Sort	108	1266	111	406	104	187	112	47
Greedy	**61**	848	67	1188	73	562	97	141
SaGaR	**61**	5937	**66**	2469	72	1562	**93**	1187
SaG	**61**	4734	**66**	1531	74	735	**93**	1187
GaR	**61**	5015	67	2171	**71**	1375	95	1219
Remove	108	57156	94	16781	93	7469	94	1703
Random-Greedy	140	1141	135	391	127	219	125	203
SaGaR_bS	**61**	5000	67	2140	**71**	1437	**93**	969

11 lent. Schemos c1355 rezultatai ir algoritmų atlikimo laikai

c1355	2046		1184		824		426	
	10x		5x		3x		1x	
	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms
Simply	426	297	426	172	426	109	426	62
Simply_other	347	156	348	94	350	62	357	47
Sort	281	6282	306	2125	308	953	351	187
Greedy	**181**	92735	207	30938	232	13218	318	2687
SaGaR	184	114797	208	49281	**228**	23406	**306**	13469
SaG	185	98593	212	33281	230	13891	**306**	78
GaR	**180**	118140	**204**	46437	230	25063	312	16266
Remove	334	552203	323	167735	316	68219	310	19984
Random-Greedy	443	6312	381	2078	411	1156	371	2063
SaGaR_bS	**180**	112234	**204**	46734	230	25563	307	13266

12 lent. Schemos c1908 rezultatai ir algoritmų atlikimo laikai

c1908	2276		1203		807		413	
	10x		5x		3x		1x	
	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms
Simply	413	250	413	125	413	78	413	47
Simply_other	274	62	275	47	278	32	285	32
Sort	273	17985	295	5109	308	2407	309	328
Greedy	**167**	71094	183	21797	201	10093	268	1578
SaGaR	170	104234	**178**	34469	**196**	17172	**261**	6437
SaG	173	90656	185	29063	204	12359	262	47
GaR	166	85406	180	30734	197	15859	263	7922
Remove	267	430438	268	88969	250	37985	263	11344
Random-Greedy	380	18563	368	6047	357	2391	331	890
SaGaR_bS	166	82312	180	29781	197	15844	262	6406

13 lent. Schemos c2670 rezultatai ir algoritmų atlikimo laikai

c2670	548		308		206		93	
	10x		5x		3x		1x	
	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms
Simply	93	297	93	172	93	102	93	47
Simply_other	72	63	72	47	74	47	78	47
Sort	95	1297	94	484	91	234	83	94
Greedy	53	3125	58	1156	61	609	70	219
SaGaR	**49**	3657	**55**	2625	**56**	2500	**66**	2797
SaG	50	1687	**55**	625	58	375	**66**	125
GaR	52	5357	56	3282	57	2844	67	3016
Remove	78	94469	68	21579	69	15275	66	4047
Random-Greedy	104	1359	99	485	97	375	87	484
SaGaR_bS	50	3468	**54**	2500	**56**	2343	**66**	2719

14 lent. Schemos c3540 rezultatai ir algoritmų atlikimo laikai

c3540	1085		594		406		188	
	10x		5x		3x		1x	
	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms
Simply	188	78	188	47	188	32	188	15
Simply_other	149	31	149	31	150	15	157	16
Sort	156	4203	152	922	160	406	152	94
Greedy	88	20703	95	4532	104	1594	136	375
SaGaR	**82**	11829	**88**	3704	**96**	2469	**133**	1703
SaG	85	12593	92	2969	99	1125	**133**	31
GaR	86	18250	94	5313	100	3031	**133**	2015
Remove	129	90953	131	21765	113	9969	135	2515
Random-Greedy	178	12766	177	860	174	406	156	750
SaGaR_bS	**82**	10656	91	4125	100	2422	**133**	1578

15 lent. Schemos c5315 rezultatai ir algoritmų atlikimo laikai

c5315	2519		1628		1171		562	
	10x		5x		3x		1x	
	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms
Simply	562	4813	562	6797	562	4062	562	922
Simply_other	427	1407	429	2500	432	1953	440	656
Sort	445	130688	450	54985	452	23954	460	3328
Greedy	285	391360	294	365766	319	467406	389	16704
SaGaR	**268**	488469	**276**	395640	**301**	597484	**379**	215750
SaG	270	71875	284	35563	310	29625	**379**	1062
GaR	274	644922	289	724187	309	768141	386	196687
Remove	360	5351250	354	2934672	362	4062453	385	370922
Random-Greedy	573	84547	553	50844	550	62360	475	20734
SaGaR_bS	272	242266	283	206766	304	607140	**379**	215171

16 lent. Schemos c6288 rezultatai ir algoritmų atlikimo laikai

c6288	536		315		223		93	
	10x		5x		3x		1x	
	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms
Simply	93	47	93	16	93	16	93	15
Simply_other	66	15	67	16	68	16	69	15
Sort	90	2172	86	781	85	422	80	78
Greedy	47	4672	52	1562	54	797	69	141
SaGaR	**44**	1328	**48**	813	**52**	641	**63**	406
SaG	45	735	**48**	297	54	172	**63**	15
GaR	45	5047	50	1984	54	1188	**63**	532
Remove	62	15187	59	5562	59	2718	65	594
Random-Greedy	90	2203	97	766	91	375	79	94
SaGaR_bS	47	1140	**48**	672	53	531	**63**	453

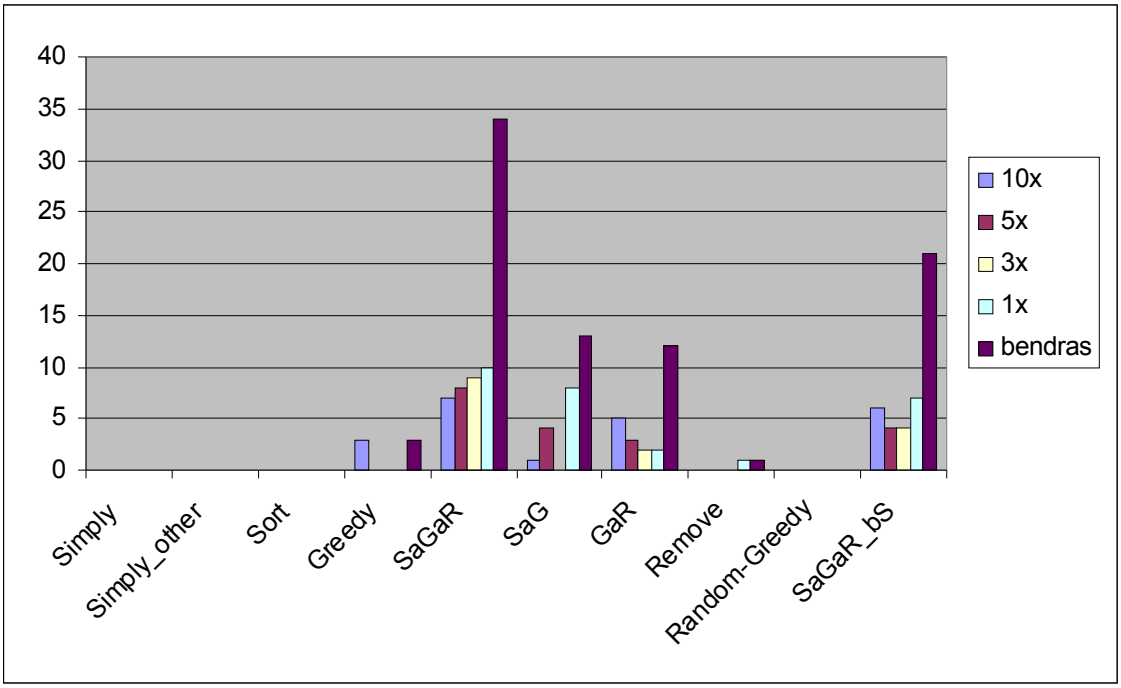
17 lent. Schemos c7552 rezultatai ir algoritmų atlikimo laikai

c7552	2048		1306		947		472	
	10x		5x		3x		1x	
	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms	rez.	laikas, ms
Simply	472	12828	472	11391	472	8828	472	4750
Simply_other	376	4265	378	4156	379	15360	381	3375
Sort	440	213860	435	119719	424	57719	441	14438
Greedy	278	471750	287	408219	303	228640	367	51375
SaGaR	**262**	544125	**281**	511157	**294**	330375	**359**	128359
SaG	266	28078	283	15953	299	7500	**359**	109012
GaR	272	1033797	284	681610	298	365281	361	2218282
Remove	317	6175859	324	4442344	330	1622203	**359**	512438
Random-Greedy	515	61141	487	94547	473	34094	419	25125
SaGaR_bS	265	170719	285	434157	**294**	292984	**359**	353641

Lentelėje 18 pateikta kiek sykių konkretus algoritmas daugiausiai suminimizavo pradinę duomenų aibę. Rezultatai yra apibendrinami pagal duomenų aibės dydį (10x, 5x, 3x, 1x) ir bendrai (nepriklausomai nuo pradinės aibės dydžio).

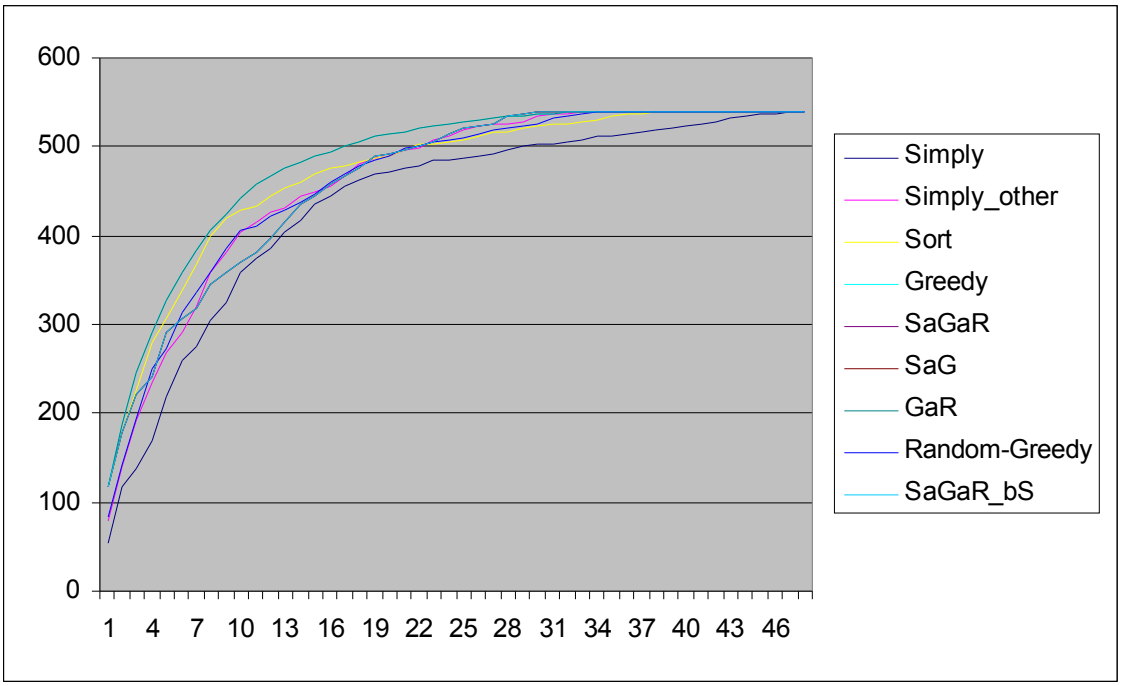
18 lent. Apibendrinta rezultatų lentelė

	10x	5x	3x	1x	bendras
Simply	0	0	0	0	0
Simply_other	0	0	0	0	0
Sort	0	0	0	0	0
Greedy	3	0	0	0	3
SaGaR	7	8	9	10	34
SaG	1	4	0	8	13
GaR	5	3	2	2	12
Remove	0	0	0	1	1
Random-Greedy	0	0	0	0	0
SaGaR_bS	6	4	4	7	21
				bendras:	40



11 pav. pateiktas 18 lentelės grafikas. Jame gerai matosi, kad geriausiai minimizuoja SaGaR algoritmas. Iš grafiko matosi, kad SaGaR algoritmas yra nepriklausomas nuo pradinės aibės dydžio.

Schemai c432 buvo atliktas papildomas tyrimas. Schema yra pasirinkta atsitiktinai, norint parodyti kaip greitai algoritmai užpildo nepadengtus elementus. Schemos c432 maksimalus padengimas yra 540 elementų. Grafike x ašyje yra pavaizduotas elementų kiekis, o y ašyje padengtas elementų kiekis. Su kiekvienu elemento parinkimu padengtų sąryšių skaičius didėja. Algoritmai yra vertinami pagal tai, kaip greitai pasiekia tikslą, t.y. maksimaliai padengia sąryšio matricą. Kuo greičiau algoritmas pasiekia tikslą tuo jis geresnis.



12 pav. Algoritmų pildymo greitis

4. Išvados

- Kaip matome iš lentelės ir iš grafiko bendru atveju SaGaR algoritmas yra žymiai geresnis už kitus algoritmus. Jis pateikia geriausią rezultatą net 34 atvejais iš 40. Tuo tarpu kai Greedy algoritmas 3/40. Tarpiniai algoritmai tarp SaGaR ir Greedy (SaG ir GaR) gerumu yra maždaug apylygiai.
- Iš pateiktos lentelės galima daryti išvadą, kad SaGaR algoritmas yra pastoviai geras su bet kokiu pradiniu duomenų poaibiu.
- SaG algoritmo efektyvumas didėja kai pradinių duomenų aibė mažėja.
- GaR algoritmas pateikia geresnius rezultatus, kai pradinės duomenų aibė yra didelė (10x ir t.t.).
-

5. Literatūra

[1] Chaval V., Freeman W. H. Linear Programming. New Yorl, NY, 1983.

[2] Alon N., Spencer J. H. The probabilistic method. New York, 1992.

[3] Yu Jiang, Binxing Fang, Mingzeng Hu. Embillishment on Greedy algorithm for Set Cover Problem – 2004: Proceedings of the 2nd International Conference on Informatikon Technology for Application [ICITA 2004]

[4] Aprara A., Fishetti M., Toth P. Algorithms for the set covering problems. DEIS, University of Bologna, Italy, 1998.

[5] Goemans M., Williamson D. The primal-dual method for approximation algorithms and its application to network design problems. Chapter 3 *In Approximation Algorithms for NP-Hard Problems*, PSW, Boston, MA, 1997.

[6] Quan Guanri, Hong Bingrong, Ye Fei ir Ren Jianjum, "A heuristic function algorithms for minimum set-covering problem", *Journal of software*, vol. 9, no 2, Feb. 1998.

[7] Petr Slavy, "A tight analysis of the greedy algorithm for set cover", *Journal of Algorithms*, vol. 25, no. 2, Nov. 1997.

[8] Duh R. ir Furer M. "Approximation of k-set cover by semi-local optimization", *Proceeding of the 29th Annual ACM Symposium on Theory of Computing (ACM STOC 1996)*, Philadelphia, Pennsylvania, USA, May 1996.

[9] Nemhauser G. L. ir Wolsey L. A. Integer and Combinatorial Optimization. John Wiley & Sons, New York, 1988.

[10] Coudert O. ir Madre J. C. New ideas for solving covering problems. In *The Proceeding of the Design Automation Conference*, June 1995.

[11] Hochbaum D. S. Approximation covering and packing problems: set cover, vertex cover, independent set and related problems. Chapter 4 *In Approximation Algorithms for NP-Hard Problems*, PSW, Boston, MA, 1997.

[12] Papadimitriou C. H. ir Steiglitz K. *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, 1982.

6. Terminų ir santrumpų žodynas

1. Atrinkti pradiniai duomenys	Tai duomenys, kurie turi poveikį automato ištestavimo procentui.
2. Automatas	Tai yra matematinis modelis realių sistemų aprašymui
3. Automato failas	tai yra C++ programavimo kalbos failas, kuriame aprašytas analizuojamas automatas
4. Automato įėjimai	automato įėjimų skaičius
5. Automato išėjimai	automato išėjimų skaičius
6. Automato pradiniai duomenys	automato pradiniai duomenys yra automato įėjimai ir vidinės būsenos prieš vykdymą
7. Automato rezultatai	tai automato išėjimai ir visinės būsenos po automato vykdymo
8. Automato vidinės būsenos	automato vidinių būsenų skaičius. Jos keičiasi po kiekvieno automato vykdymo
9. Iteracijų skaičius	apibrėžiama kiek sykių bus kreipiamasi į automatą
10. Matricos pildymo greitis	Jis priklauso nuo matricos sąryšio tarp pradinių duomenų ir rezultatų. Jei sąryšis yra mažas, tai mažas ir matricos pildymo greitis, i atvirkščiai
11. Papildymo laikas	per šį laiką programa ieško nerastų automato sąryšių, o kai randa papildo atrinktų duomenų failą
12. Pilna automato matrica	Tai matrica, kurioje sužymėti visi galimi sąryšiai tarp automato pradinių duomenų ir rezultatų
13. Sąryšis	Tai sąryšis tarp pradinių duomenų ir rezultatų. Sąryšis egzistuoja tada, kai pakeitus bent vieną pradinių duomenų būsena, keičiasi rezultatas.
14. Testinė automato matrica	Tai matrica, kurioje sužymėti visi sąryšiai tarp automato pradinių duomenų ir rezultatų, kurie yra ištestuojami testinio duomenų failo
15. Testinių duomenų failas	šiuo failu automatui nurodomi įėjimai
16. Tikrieji automato įėjimai	Tai automato įėjimai be vidinių būsenų
17. Tikrieji automato išėjimai	Tai automato išėjimai be vidinių būsenų
18. SCP	Aibės padengimo problema (Set covering problem)

7. Priedai

7.1.Pabaigos sąlygos tyrimas

Atsitiktinė paieška naudoja paieškos pabaigos sąlygą, kuri apsprendžia gauto sprendinio tikslumą. Atsitiktinis testinių rinkinių generavimas juodos dėžės gedimams taip pat turi remtis pabaigos sąlyga, kuri turi apibūdinti kiek pilnai yra tikrinami juodos dėžės gedimai. Atsitiktinio generavimo metu atrenkami įėjimo poveikiai, kurie tikrina bent vieną dar netikrintą juodos dėžės gedimą. Siekiama patikrinti maksimalų gedimų kiekį ir apibūdinti kiek pilnai tikrinami juodos dėžės gedimai. Tai gali apibūdinti tikimybė nusakanti galimybę surasti įėjimo poveikį tikrinantį naują juodos dėžės gedimą. Ši tikimybė $p = \frac{S}{N}$ gali būti išreikšta santykiu atrinktų įėjimų poveikių S su visų nagrinėtų įėjimo poveikių kiekiu N. Kadangi tikrinamų juodos dėžės gedimų kiekis yra ribotas tai kuo daugiau atsitiktinai generuojami įėjimo poveikių tuo tikimybė p mažėja. Kitas svarbus parametras yra tikimybės p pasikliaunamumas (confidence), kad pavyks surasti įėjimo poveikį tikrinantį naują juodos dėžės gedimą. Pasikliaunamumas apskaičiuojamas pagal formulę $C = 1 - (1 - p)^N$. Pavyzdžiui, kad pasikliaunamumas C viršytų 0,9999 esant tikimybei p=0,0001 reikia išnagrinėti nemažiau kaip milijoną atsitiktinai sugeneruotų įėjimo poveikių. Galima įvertinti sprendžiamo uždavinio specifiką, kad pagrindinai įėjimo poveikiai atrenkami atsitiktinio generavimo pradžioje, o atrinkus įėjimo poveikius tikrinančius visus juodos dėžės gedimus, toliau nauji įėjimo poveikiai nebeatrenkami. Todėl svarbiu faktoriumi demonstruojančiu juodos dėžės gedimų tikrinimo išsamumą yra santykis r1 paskutinio atrinkto įėjimo poveikio generavimo eilės numeris su bendru sugeneruotų įėjimo poveikių kiekiu. Pavyzdžiui šis santykis r1 lygus 10 rodo, kad po paskutinio atrinkto įėjimo poveikio dar buvo generuota 9 kartus daugiau įėjimo poveikių ir jie netikrino naujų juodos dėžės gedimų. Kitas svarbus faktorius demonstruojantis, ar tikrinami visi juodos dėžės gedimai yra kelių nepriklausomų generavimo bandymų tikrinamų juodos dėžės gedimų palyginamas. Jeigu atsitiktinis įėjimo paveikių generavimas yra pakankamas, kad būtų atrinkti poveikiai tikrinantys visus juodos dėžės gedimus, tai kelių generavimo bandymų metu atrinkti įėjimo poveikiai tikrins tuos pačius juodos dėžės gedimus. Sakykim, kad iš dešimties nepriklausomų generavimo bandymų visų dešimt bandymų atrinkti įėjimo poveikiai tikrina tuos pačius juodos dėžės gedimus, tai

yra rimta paraiška, kad tikrinami visi juodos dėžės gedimai. Tokių situacijų kiekį žymėsime r2.

Norėdami charakterizuoti ar visi juodos dėžės gedimai yra tikrinami kiekvienai schemai nurodysime juodos dėžės naujo gedimo patikrinimo tikimybę p, pasikliaunamumą C, reikšmę r1 – rodančią kiek buvo generuota įėjimo poveikių po paskutinio atrinkto ir reikšmę r2 rodančią kiek iš n bandymų buvo gauti tie patys rezultatai. Šios reikšmės benchmark schemoms nurodytos lentelėje 19 ir 20.

19X lentelė. Pilnos matricos paieškos pildymo rezultatai

Schemos pavadinimas	Bandymų skaičius	Vidutinis sugeneruotų poveikių kiekis	Paskutinis elementas, kuris koregavo matrica (vidurkis)	Vidutinis atrinktų poveikių kiekis	*Tikimybė surasti naują tinkantį poveikį*	Pasikliaunamumo koeficiento devyniukių kiekis po nulio	*Vidutiniška i kiek kartų viršytas paskutinio atrinkto poveikio numeris*	Matricos užimtumo maksimalių reikšmių kiekis	Didžiausias matricos užpildymas iš visų bandymų
c432	5	706000,00	3739,00	70,60	*0,0001*	30	*188,82*	5	540
c499	5	5052000,00	73325,00	505,20	*0,0001*	219	*68,90*	5	5184
c880	5	1852000,00	103242,60	185,20	*0,0001*	80	*17,94*	5	1326
c1355	5	4952000,00	61635,20	495,20	*0,0001*	215	*80,34*	5	5184
c1908	5	3230000,00	94943,00	323,00	*0,0001*	140	*34,02*	5	3004
c2670	5	1928000,00	1871528,00	192,80	*0,0001*	83	*1,03*	1	3164
c3540	5	2494000,00	210280,00	249,40	*0,0001*	108	*11,86*	5	2588
c5315	5	5726000,00	154308,40	572,60	*0,0001*	248	*37,11*	5	10540
c6288	5	1190000,00	214112,00	119,00	*0,0001*	51	*5,56*	5	3068
c7552	5	7211666,67	7043246,50	721,17	*0,0001*	313	*1,02*	1	9846

20X lentelė. Schemų pagrindiniai parametrai

Schemos pavadinimas	Įėjimų skaičius	Išėjimų skaičius	Maksimalus matricos užpildymas
c432	36	7	540
c499	41	32	5184
c880	60	26	1326
c1355	41	32	5184
c1908	33	25	3004
c2670	157	64	3320
c3540	50	22	2588
c5315	178	123	10540
c6288	32	32	3068
c7552	206	107	>12000

Iš rezultatų matome, kad schemoms c2670 ir c7552 nepavyko pilnai užpildyti matricų. Tai matoma iš stulpelių „Matricos užimtumo maksimalių reikšmių kiekis“ ir „Vidutiniškai kiek kartų viršytas paskutinio atrinkto poveikio numeris“. Šioms schemoms patariama mažinti tikimybę, vietoj naudojamos 0,0001 tikimybės. Pagal šiuos rezultatus galima sakyti, kad pasiūlytos lentelės pilnai užtenka patikrinti ar nurodytos schemoms patikrinami visi galimi sąryšiai.