**KAUNO TECHNOLOGIJOS UNIVERSITETAS**
**INFORMATIKOS FAKULTETAS**

Vaidotas Ragaišis

# Dvimačio giljotininio pjaustymo sprendimo metodai ir jų tyrimas

**Magistro darbas**

**Darbo vadovas** doc. dr. Dalius Rubliauskas

**KAUNAS, 2008**

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**
**INFORMATIKOS FAKULTETAS**

# Dvimačio giljotininio pjaustymo sprendimo metodai ir jų tyrimas

**Informatika**

**MAGISTRO DARBAS**

**Magistrantas .....**

Vaidotas Ragaišis IFM-2/1

2008 m. gegužės 21 d.

**Vadovas**_____

doc. dr. Dalius Rubliauskas

2008 m. gegužės 21 d.

**Recenzentas**_____

doc. dr. Kęstutis Motiejūnas

2008 m. gegužės 21 d.

**KAUNAS, 2008**

# Turinys

PAVEIKSLĖLIŲ SĄRAŠAS ... 4
LENTELIŲ SĄRAŠAS ... 5
SANTRUMPŲ SĄRAŠAS ... 6
ĮVADAS ... 7
1. DVIMATIS GILJOTININIS PJAUSTYMAS ... 8
2. DVIMAČIO GILJOTININIO PJAUSTYMO SPRENDIMO METODAI ... 10
2.1. DAŽNIAUSIAI NAUDOJAMI GILJOTININIO PJAUSTYMO ALGORITMAI ... 11
2.2. KITI OPTIMIZAVIMO ALGORITMAI ... 13
2.3. PJAUSTYMO UŽDAVINIAMS NAUDOJAMŲ ALGORITMŲ PRIVALUMAI BEI TRŪKUMAI ... 16
3. DAŽNIAUSIAI NAUDOJAMOS DVIEJŲ DIMENSIJŲ GILJOTININIO PJAUSTYMO EURISTIKOS IR JŲ PALYGINIMAS ... 17
3.1. LYGIO ALGORITMAI ... 18
3.2. ETALONINIS TESTAS ... 26
3.3. ALGORITMŲ REZULTATŲ PALYGINIMAS ... 28
4. SIŪLOMOS EURISTIKOS DVIEJŲ DIMENSIJŲ PAKAVIMO PROBLEMAI SPRĘSTI ... 33
4.1. DVIEJŲ STULPELIŲ EURISTIKA ... 34
4.2. VIENO STULPELIO EURISTIKA ... 36
4.3. RŪŠIAVIMO EURISTIKA ... 37
5. PASIŪLYTŲ EURISTIKŲ PALYGINIMAI ... 38
IŠVADOS IR REKOMENDACIJOS ... 49
LITERATŪRA ... 50
SANTRAUKA ANGLŲ KALBA ... 51

# PAVEIKSLĖLIŲ SĄRAŠAS

1 pav. Giljotininio pjaustymo pavyzdys. Rodyklėmis pažymėtos giljotininių pjūvių vietos ..... 8

2 pav. Pastarųjų metų publikuotų algoritmų išskiriamos grupės pagal globaliai optimizacijai naudojamus metodus ..... 10

3 pav. Kairiojo apatinio kampo euristikos veikimo schema ..... 12

4 pav. Kairiojo apatinio kampo pildymo euristikos darbo rezultatai ..... 13

5 pav. (a) ass algoritmo pradžia, (b) tuščios stačiakampės sritys išnyko, kraudamos plačius stačiakampius su ass algoritmo pagalba, (c) krovimas siauro stačiakampio srityje rj. ..... 23

6 pav. Pakavimai, pagaminti lygmens algoritmų. ..... 25

7 pav. Giljotinos ir ne giljotinos įpjovimai; (a) giljotinos pjovimas su 4 stačiakampiais, (b) ne giljotinos pjovimas su 5 stačiakampiais, (c) giljotinos pjovimas su 2 stačiakampiais. ..... 27

8 pav. Dažniai, su kuriais algoritmai pasiekė mažiausią ruožo aukštį ..... 29

9 pav. Ff klasės algoritmų efektyvumo lygių grafikas ..... 30

10 pav. Kvadratų išdėstymas su pasukimu ..... 33

11 pav. 2c algoritmo schema ..... 35

12 pav. 1c algoritmo schema ..... 36

13 pav. Rūšiavimo algoritmų schema ..... 38

14 pav. Hoppert1 duomenų klasės rezultatų stulpelinė diagrama ..... 38

15 pav. Hoppert2 duomenų klasės rezultatų stulpelinė diagrama ..... 39

16 pav. Hoppert3 duomenų klasės rezultatų stulpelinė diagrama ..... 40

17 pav. Hoppert4 duomenų klasės rezultatų stulpelinė diagrama ..... 40

18 pav. Hoppert5 duomenų klasės rezultatų stulpelinė diagrama ..... 41

19 pav. Hoppert6 duomenų klasės rezultatų stulpelinė diagrama ..... 42

20 pav. Hoppert7 duomenų klasės rezultatų stulpelinė diagrama ..... 42

21 pav. Vidutinių aukščių stulpelinė diagrama ..... 43

22 pav. Kiek kartų ir kuris algoritmas parodė geriausią rezultatą ..... 44

23 pav. Algoritmų vidutinis efektyvumo lygis t1 duomenų klasėje. ..... 45

24 pav. Algoritmų vidutinis efektyvumo lygis t2 duomenų klasėje. ..... 46

25 pav. Algoritmų vidutinis efektyvumo lygis t3 duomenų klasėje. ..... 46

26 pav. Algoritmų vidutinis efektyvumo lygis t4 duomenų klasėje. ..... 47

27 pav. Algoritmų vidutinis efektyvumo lygis t5 duomenų klasėje. ..... 47

28 pav. Algoritmų vidutinis efektyvumo lygis t6 duomenų klasėje. ..... 48

29 pav. Algoritmų vidutinis efektyvumo lygis t7 duomenų klasėje. ..... 48

# LENTELIŲ SĄRAŠAS

1 lentelė stačiakampių matmenų pavyzdys, kuriuos reikia supakuoti į 40 vienetų pločio juostą ..... **18**

2 lentelė dispersinės analizės rezultatai gauti panaudojus algoritmų nf, ff, bf klasių euristikas ..... **28**

3 lentelė chi-kvadrato testo rezultatai, su kuriais algoritmai gavo mažiausią pakavimo aukštį atlikus 542 bandymus ir kur $\chi^2_{df}$ ir $\chi_{critical}$ reikšmės aprašo skirtumus tarp dažnių. ..... **29**

4 lentelė dispersinės analizės ir chi-kvadrato testo rezultatai ..... **31**

5 lentelė algoritmų vidutinių efektyvumo lygių lentelė su hopper and turton (2002) testo duomenimis ..... **31**

6 lentelė algoritmų vidutinių efektyvumo lygių lentelė su hopper and turton (2002) testo duomenimis ..... **32**

7 lentelė eksperimento rezultatai hoppert1 duomenų klasėje ..... **38**

8 lentelė eksperimento rezultatai hoppert2 duomenų klasėje ..... **39**

9 lentelė eksperimento rezultatai hoppert3 duomenų klasėje ..... **39**

10 lentelė eksperimento rezultatai hoppert4 duomenų klasėje ..... **40**

11 lentelė eksperimento rezultatai hoppert5 duomenų klasėje ..... **41**

12 lentelė eksperimento rezultatai hoppert6 duomenų klasėje ..... **41**

13 lentelė eksperimento rezultatai hoppert7 duomenų klasėje ..... **42**

14 lentelė f-testo rezultatai ..... **43**

15 lentelė chi-kvadrato testo rezultatai ..... **44**

16 lentelė algoritmų vidutinis efektyvumo lygis ..... **45**

## SANTRUMPŲ SĄRAŠAS

NFDH - Sekančio tinkančio mažėjančio aukščio algoritmas (The Next-Fit Decreasing Height algorithm)
NFDHIW - Sekančio tinkančio mažėjančio aukščio didėjančio pločio algoritmas (The Next- Fit Decreasing Height Increasing Width algorithm)
NFDHDW - Sekančio tinkančio mažėjančio aukščio mažėjančio pločio algoritmas (The Next- Fit Decreasing Height Decreasing Width algorithm)
FFDH - Pirmo tinkančio mažėjančio aukščio algoritmas (The First-Fit Decreasing Height algorithm)
FFDHIW - Pirmo tinkančio mažėjančio aukščio didėjančio pločio algoritmas (The First-Fit Decreasing Height Increasing Width algorithm)
FFDHDW - Pirmo tinkančio mažėjančio aukščio mažėjančio pločio algoritmas (The First-Fit Decreasing Height Decreasing Width algorithm)
BFDH - Geriausiai tinkančio mažėjančio aukščio (The Best-Fit Decreasing Height algorithm)
BFDHIW - Geriausiai tinkančio mažėjančio aukščio didėjančio pločio algoritmas (The Best- Fit Decreasing Height Increasing Width algorithm)
BFDHDW - Geriausiai tinkančio mažėjančio aukščio mažėjančio pločio algoritmas (The Best- Fit Decreasing Height Decreasing Width algorithm)
SF - Dalinimo algoritmas (The Split Fit algorithm)
ASS - Kintamo sąrašo (Alternate Size Stack algorithm)
FCNR - Žemiausios-aukščiausios ribos be sukimosi algoritmas (The Floor-Ceiling No Rotation algorithm)
KP01 - Kuprinės algoritmas (The Knapsack algorithm)
2C – Dviejų stulpelių euristika (Two Columns heuristic)
1C – Vieno stulpelio euristika (One column heuristic)
SH – Rūšiavimas pagal aukštį (Sort by Height heuristic)
SW – Rūšiavimas pagal plotį (Sort by Width heuristic)
SS – Rūšiavimas pagal plotą (Sort by area heuristic)
SSN – Rūšiavimas pagal kvadrantingumą (Sort by Squareness heuristic)

# ĮVADAS

Visais laikais žaliavų taupymo problema buvo svarbus klausimas. Optimaliai suplanavus gamybą galima sumažinti išlaidas žaliavų pirkimui, pagerėja įmonės rodikliai ir taupomi gamtos resursai. Šios problemos yra sutinkamos daugelyje pramonės šakų - medžio, stiklo, popieriaus, porolono ir kituose.

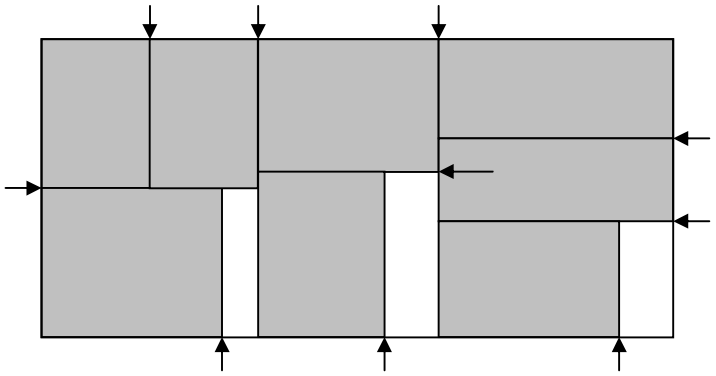
Pjaustymas yra priskiriamas optimizavimo uždaviniams, kurių tikslas yra rasti gera keleto objektų (ruošinių) išdėstymą dideliame stačiakampiame objekte. Paprastai išdėstymo proceso tikslas yra minimizuoti žaliavos panaudojimą kartu  minimizuojant nepanaudotą (atliekų) plotą. Šios problemos sprendimas yra aktualus masinės gamybos pramonės šakoms, kadangi nedidelis išdėstymo pagerinimas gali leisti sutaupyti nemažai žaliavų bei sumažinti gaminio savikainą. Pjaustiniai dažniausiai būna stačiakampio formos.

Kadangi šios problemos priklauso kombinatorinio optimizavimo uždavinių klasei, todėl turi būti rastas sprendinys iš daugelio galimų sprendinių, kuris optimizuoja gerumo funkciją esant aibei apribojimų. Tokios problemos yra priskiriamos NP sudėtingumo uždaviniui klasei. NP sudėtingumo uždaviniai, tai uždaviniai, kuriems nėra rasta algoritmo, kuris sprendinį pateiktų per polinominį laiką. Todėl tokių uždavinių sprendimui yra kuriami bei naudojami euristiniai metodai, kurie pateikia sprendinį artimą optimaliam per priimtiną laiko tarpą.[7]

# 1. Dvimatis giljotininis pjaustymas

Daugelyje pramonės šakų, tokių kaip popieriaus, metalo, baldų, stiklo, siuvimo bei kitose produkcija yra gaminama pjaustant tam tikro dydžio ruošinius paprastai iš stačiakampio pavidalo plokščių, juostų ar audeklo rulonų (siuvimo pramonėje). Ekonominių bei aplinkos apsaugos požiūriu yra labai svarbu, kad gaminant tokį patį kiekį produkcijos, būtų suvartojama kiek galima mažiau žaliavų. Jei žaliavų būtų sunaudojama mažiau, tai ir gaminamo produkto savikaina sumažėtų. Taip pat būtų daroma mažesnė žala aplinkai, nes būtų tausojami gamtiniai resursai bei į aplinką išmetama mažiau teršalų. Todėl nuo seno (anksčiau tik dėl ekonominių, o dabar ir dėl aplinkosauginių paskatų) ieškoma efektyvių sprendimo būdų, kurie leistų atlikti greitą ruošinių supjaustymą ir po supjaustymo likusių atliekų plotas būtų minimalus. Žmogus yra nepajėgus fiziškai atlikti gero ruošinių supjaustymo per trumpą laiką siekiant, kad atliekų kiekis butų mažas. Todėl tokio pobūdžio uždaviniams spręsti yra pasitelkiama skaičiavimo techniką. Anksčiau kuriami algoritmai dėl silpnų kompiuterinės įrangos techninių charakteristikų pjaustymo uždavinius spręsdavo ilgai. Dėl tos priežasties, kad skaičiavimo laikas netruktų keletą valandų buvo atliekami tam tikri apribojimai, kurie leisdavo skaičiavimo laiką pagreitinti, tačiau atliekų plotas dėl to dažnai išaugdavo. Esant dabartiniai skaičiavimo technikai yra tikslinga kurti (tobulinti) algoritmus skirtus pjaustymo uždaviniams spręsti.

Pjaustymo uždaviniuose ruošiniai dažniausiai yra stačiakampio formos, sprendimo radimui naudojami taip vadinami dvimačio supjaustymo algoritmai. Tokiose pramonės šakose kaip stiklo ar baldų ruošinių pjaustymui yra naudojami pjaustymo įrenginiai galintys atlikti tik giljotininius pjūvius, todėl šiose veiklos srityse naudojami taip vadinami giljotininio pjaustymo algoritmai. Giljotina pasižymi tuo, kad pjūvis gali būti daromas tik nuo vieno plokštės krašto iki kito, be to pjūvio linija turi būti statmena pjaunamai kraštinei. Giljotininio supjaustymo pavyzdys pateiktas 1 pav.[7]



Šaltinis: Andreas Fritsch, Oliver Vornberger, 2008

**1 pav. Giljotininio pjaustymo pavyzdys. Rodyklėmis pažymėtos giljotininių pjūvių vietos**

Akivaizdu, kad ruošinių pjaustymui naudojant giljotininius įrenginius, lyginant su dvejetainio supjaustymo uždaviniais, yra sunkiau rasti gerą ruošinių išdėstymą plokštėje, kad atliekų plotas būtų minimalus. Todėl moksline prasme šis uždavinys yra labiau įdomus nei dvejetainio supjaustymo bei jam panašūs uždaviniai.

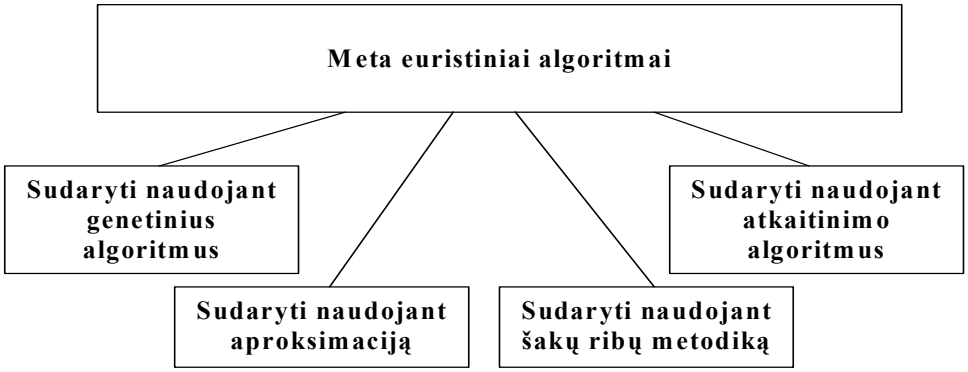
Giljotininio supjaustymo uždavinį būtų galima formuluoti taip:

Duotas baigtinis ruošinių kiekis $n$, kurių ilgiai $h_i$ ir pločiai $w_i$ yra žinomi. Taip pat turime teoriškai begalinę aibę pjaustomų plokščių $\{R_1, R_2, ...\}$, kurių ilgiai $H_j$ ir pločiai $W_j$ taip pat žinomi. Čia $i = 1, 2, ..., n$, $j = 1, 2, ..., \infty$. Reikia supjaustyti ruošinius taip, kad būtų sunaudotas minimalus pjaustomų plokščių kiekis, o atliekų kiekis būtų minimalus. Galimi tik giljotininiai pjūviai, t.y. turimą plokštę galima pjauti nuo kraštinės iki kraštinės, o pjūvio linija privalo būti statmena pjaunamai kraštinei. Siekiant gauti mažą atliekų plotą ruošinius galima sukioti 90º kampu.[1]

Giljotininio supjaustymo uždavinį išspręsti galima atlikus pilną perrinkimą, tačiau toks sprendimo būdas nėra įmanomas, kadangi uždavinio sudėtingumas auga eksponentiškai, didėjant ruošinių kiekiui. Tokios problemos yra priskiriamos NP sudėtingumo uždaviniui klasei, todėl jų sprendimui yra naudojami euristiniai metodai.[1]

# 2. Dvimačio giljotininio pjaustymo sprendimo metodai

Pastarųjų metų literatūroje pateikiamus pjaustymo metodus būtų galima pavadinti meta euristiniais algoritmais, kadangi paprastai probleminės srities sprendiniui gauti yra naudojama keletas algoritmų. Dažnai vienas algoritmas, kuris būna kaip taisyklė euristinis, atlieka ruošinių supjaustymą, o kitas neretai būna globalios optimizacijos algoritmas, kurio pagalba stengiamasi pagerinti pirmojo darbo rezultatus [1]. Pagal tai kokie metodai yra naudojami globaliai optimizacijai realizuoti šiuos algoritmus būtų galima suskirstyti į keturias grupes: algoritmus sudarytus panaudojant genetinius ir atkaitinimo algoritmus bei panaudojant aproksimacijos bei šakų ribų metodiką 2 pav.




Šaltinis: sudarytas autoriaus

**2 pav. Pastarųjų metų publikuotų algoritmų išskiriamos grupės pagal globaliai optimizacijai naudojamus metodus**

Pjaustymo uždaviniuose keleto algoritmų naudojimą lemia tai, kad paprastai vienas algoritmas būna atsakingas už ruošinių išdėstymą plokštėje, o kitas naudojamas optimizacijai arba ruošinių rikiavimui pagal tam tikrą kriterijų vykdyti. Remiantis publikuojamų straipsnių kiekiu būtų galima teigti, kad didžiąją dalį pjaustymo metodų sudaro euristikos apjungtos su genetiniais algoritmais. Dažniausiai naudojamas euristikas būtų galima supaprastinus priskirti taip vadinamos kairiojo apatinio kampo bei godžiai euristikoms.

## 2.1. Dažniausiai naudojami giljotininio pjaustymo algoritmai

Ruošinių vietos nusakymui yra sudaromi euristiniai algoritmai. Kuo naudojama euristika geresnė tuo mažesnį atliekų kiekį pavyksta gauti naudojant globalius optimizavimo metodus. Optimizacijos algoritmai yra naudojami tik galutiniam rezultatui pagerinti. Labiausiai paplitusias euristikas būtų galima suskirstyti į šias grupes:

- Godi euristika.
- Apatinio kairiojo kampo euristikos naudojimas.

***Godi euristika***

Ši euristika yra gerai žinoma ir dažnai naudojama kuprinės uždaviniui spręsti. Godi euristika teikia pirmumą tokiam elementui, kurio svoris yra didžiausias. Tokios euristikos būtų galima išskirti pagrindinius žingsnius šiuos:

1. Iš pradinio sąrašo pasirinkti elementą turintį didžiausią svorį.
2. Perkelti pasirinktą elementą iš pradinio sąrašo į kitą sąrašą.
3. Patikrinti rinkinio tinkamumą į kurį įtrauktas šis elementas.
4. Jei rinkinys netinkamas, pasirinktas elementas šalinamas iš pradinio sąrašo.
5. Tikrinti ar pradinis sąrašas nėra tuščias.
6. Jei pradinis sąrašas netuščias, tai grįžtame į pirmą žingsnį.
7. Jei pradinis sąrašas tuščias, tai stabdome procesą . Gautas rinkinys yra geriausias.

Tokį algoritmą galima pritaikyti ir dvimačiam atvejui. Tokiu atveju reikia atlikti pjaustomos plokštės dalinimą į tam tikro pločio juosteles. Juostele galima vertinti kaip vienmatį atveją, kadangi sprendiniui gauti naudojamas kuprinės uždavinys. Naudojamas algoritmas išrinks kainos atžvilgiu geriausius ruošinius, kurie tilps į nustatyto pločio juostelę.

Godžios euristikos algoritmas yra greitas, bet negarantuoja gero sprendinio.[3]

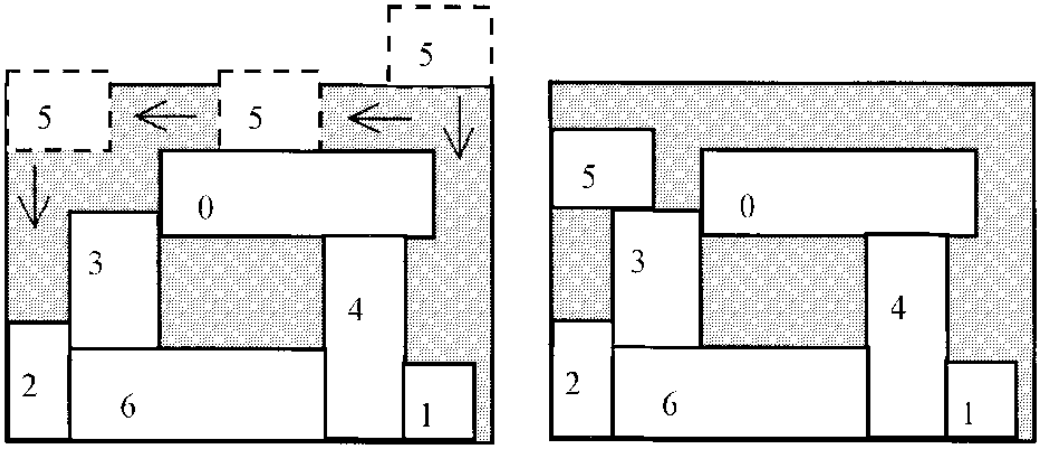
***Apatiniojo kairiojo kampo euristika***

Apatinio kairiojo kampo euristika [1]. Ši euristika yra skirta pakavimo uždaviniams spręsti, tačiau įvedus papildomas sąlygas gali būti pritaikyta ir giljotininio supjaustymo problemoms spręsti. Apatinio kairiojo kampo metodas pasižymi tuo, kad ruošiniai yra pradedami dėlioti nuo apatinio

kairiojo pjaustomos plokštės kampo. Taip pat šis algoritmas pasižymi ir tuo, kad ruošiniui nustačius padėti, jos keisti negalima. Apatinė kairiojo kampo euristika dirba tokia tvarka:

1. Pradedant nuo dešinio viršutinio kampo kiekvienas ruošinys yra stumiamas kiek galima žemiau.
2. Pasiekus galima žemiausią poziciją ruošinys stumiamas kiek galima kairiau.
3. Ruošinio teisinga padėtis yra tuomet, kai savo apatine ir kaire kraštinėmis glaudžiasi su kitais ruošiniais arba su pakavimo pagrindo kraštais, bet nekerta jų.

Pagrindinis šio metodo trūkumas yra tas, kad yra sudaroma nemažai tuščių erdvių dėliojant ruošinius. Tai įvyksta dėl to, kad didesni ruošiniai blokuoja analogiško dydžio ruošinių galimą stumdymą. Kita vertus tokio algoritmo sudėtingumas yra tik $O(N^2)$, kuomet N – ruošinių kiekis. Dėl žemo šio algoritmo sudėtingumo jis yra mėgiamas ir dažnai naudojamas hibridinėse kombinacijose su meta euristikomis. Apatinio kairiojo kampo euristikos veikimas pavaizduotas 3 pav.



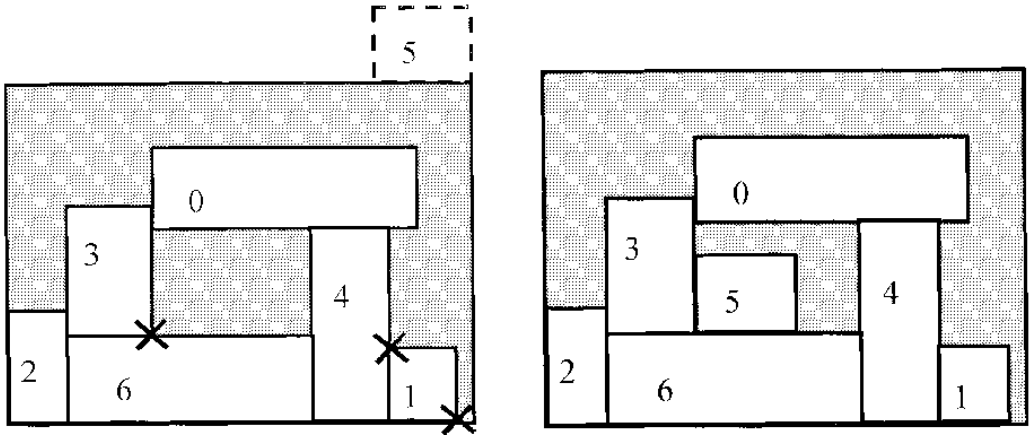

Šaltinis: sudarytas autoriaus

**3 pav. Kairiojo apatinio kampo euristikos veikimo schema**

Apatinio kairiojo kampo euristika turi patobulinimą, kuris vadinamas apatinio kairio kampo pildymo euristika. Natūralu, kad modifikacija buvo padaryta dėl to, kad apatinio kairiojo kampo euristika palikdavo didelius nepanaudotus plotus, todėl buvo pasiūlyta šios euristikos sudėtingesnis variantas, kuris gali užpildyti susidariusius tuščius tarpus. Šio algoritmo veikimas:

1. Pradedant dešiniu viršutiniu kampu ruošinys talpinamas į žemiausią galimą vietą.
2. Pasiekus žemiausią įmanomą poziciją ruošinys yra stumiamas kiek galima kairiau.
3. Kuomet ruošinio nėra kur stumti jo vieta yra fiksuojama.

Kadangi pakuojamo stačiakampio padėties paieška vyksta per visą pagrindo plokštę, tai algoritmas gali užpildyti likusius nepanaudotus plotus. Tam, kad šis metodas būtų išskirtas iš apatinio kairiojo kampo euristikos, jis yra pavadintas apatinio kairiojo kampo pildymo euristika. Palyginus apatinio kairiojo kampo euristiką su apatinio kairiojo kampo pildymo euristika, galima pastebėti, kad pastaroji sugeba atlikti tankesnį ruošinių išdėliojimą. Pagrindinis šios euristikos trūkumas yra tas, kad tokio algoritmo vykdymo laikas yra $O(N^3)$. Apatinio kairiojo kampo pildymo euristikos veikimas pavaizduotas 4 pav.[3]




Šaltinis: sudarytas autoriaus

**4 pav. Kairiojo apatinio kampo pildymo euristikos darbo rezultatai**

## 2.2. Kiti optimizavimo algoritmai

Išdėstymo kokybė priklauso nuo sekos, kuria ruošiniai yra pateikiami pjaustymui. Kadangi visų galimų kombinacijų skaičius yra per didelis, kad būtų atlikta išsami kiekvieno atvejo analizė, trunkanti per priimtiną laiko tarpą, tam yra naudojami meta euristiniai algoritmai, kad gauti efektyvesnę paieškos strategiją. Meta euristikų naudojimo tikslas yra rasti gerą ruošinių surikiavimą pagal tam tikrą kriterijų. Šie paieškos metodai paprastai pateikia gerą, bet nebūtinai optimalų sprendinį per priimtiną laiko tarpą.

***Genetiniai algoritmai.***

Genetiniai algoritmai atvaizduoja derinių ir atsitiktinumų kelią sprendimo paieškoje. Šie algoritmai tai metodologija, kuri sprendinių aibėje leidžia atlikti paiešką, panašų į natūralios atrankos

procesą būdu, vykstanti gamtoje. Tai yra pasiekiama įvertinant labiausiai pageidautinas savybes iš sprendinių sekos ir kombinuojant juos į sekančią formą. Kiekvieno sprendinio kokybė yra įvertinama ir "pritaikyti" individai yra atrenkami. Šio proceso vyksmas trunkantis keletą kartų pateiks rezultatą, kuris gali būti artimas optimaliam. Kiekvienas sprendinio variantas yra realizuotas kaip simbolių eilutė. Aibė sprendinių tam tikroje būsenoje m yra vadinama m-tosios kartos populiacija. Populiacijai leidžiama daugintis iš anksto apibrėžtą kiekį M kartų. Nesudėtingą genetinį algoritmą sudaro 2 punktai:

1. Tinkamumo nustatymas - atranka.
2. Kryžminimasis ir mutacija.

Atrankos tikimybė yra nustatoma pagal individo tinkamumą.

Vykstant kryžminimo operacijai simbolių eilutė yra padalinama į dvi. Dalinimo vieta yra nustatoma pagal tam tikrą atsitiktinį skaičių s. Nustačius dalinimo vietą yra sukuriamos dvi naujos eilutės, sukeičiant visus simbolius esančius iki dalinimo taško s. Vykstant mutacijos operacijai kai kurie eilutės simboliai yra keičiami atsitiktine tvarka. Mutacija yra k-tosios eilės jei jai vykstant yra pakeičiama k elementų per vieną mutacijos operaciją.

Panaudojant šias genetinių algoritmų savybes ir apjungiant jas su euristikomis, atliekančiomis ruošinių pjaustymą yra gaunami neblogi rezultatai. Mūsų nagrinėjamo tipo uždaviniuose genetiniai algoritmai yra naudojami nustatyti sekai, kuri nusako ruošinių supjaustymo eiliškumą bei apibrėžia kokia tvarka ruošiniai turi būti išdėlioti plokštėje.

Sudėtingų euristinių metodų naudojimas reikalauja nemažo skaičiavimo laiko, kas ne visada yra priimtina. Nežiūrint į tai, naudojant sudėtinga euristika, galima gauti geresnius rezultatus minimalių atliekų atžvilgiu.

Egzistuoja globalios optimizacijos algoritmų grupė, kuri savo elgesiu ir savybėmis yra panaši į genetinius algoritmus - tai taip vadinami naivūs evoliuciniai algoritmai. Naivios evoliucijos algoritmo idėja skiriasi nuo genetinio algoritmo tuo, kad čia nėra naudojama kryžminimo operacija. Sekančios kartos sukūrimui yra naudojama tik mutacijos operacija.[2]

***Šakų ribų metodas.***

Šakų ribų metodo taikymas taip pat gali pagerinti naudojamos euristikos sprendinį. Šis algoritmas yra taikomas toms euristikoms, kurios atlieka beveik pilną perrinkimą. Ši metodologija nėra sudėtinga:

1. Sudaryti sprendimų medį.

2. Nustatyti viršutines šakų ribas.
3. Kirsti šaką, jei viršutinė riba mažesnė nei dabartinis geriausias sprendinys.
4. Baigti skaičiavimus, jei daugiau nėra šakų, kurias būtų galima nagrinėti.

Ieškant sprendinio šio algoritmo pagalba iteracijų kiekis yra $K \le 2^n$, o skaičiavimo laikas $T \le C2^n$. Paprastai laikas $T$ yra daug kartų mažesnis nei viršutinė riba $C2^n$. Tačiau galimi atvejai, kai ši riba gali būti pasiekta. Pavyzdžiui jei visi elementai $c_i$ turi panašų svorį $g_i$. Tokiu atveju nepavyks nukirsti šakų ir skaičiavimo trukmė pailgės.[2]

***Euristinis dekompozicijos metodas.***

Naudojant euristinį dekompozicijos metodą, visų pirma reikia rasti geriausią vienos dimensijos pjovimo planą visoms $w_i$-juostomis, atsižvelgiant į maksimalų leistiną dalių skaičių $b_i$. Turėdami pjovimo planus kiekvienai juostai, galime juos išdėstyti išilgai pjaunamo lakšto pločio. Euristinio dekompozicijos metodo veikimas žingsniais:

*Pirmas žingsnis:* kiekvienam j=1,...,r (t.y. kiekvienai juostai) išspręsti kuprinės su apribojimais užduotį:

$V_j = Max\sum v_i\alpha_j^i$, atsižvelgiant į: $\sum_{k\in W_j} l_i\alpha_j^i \le L$

$0 \le \alpha_j^i \le b_i$ ir $i \in W_j = \{i \,|w_i \le w_j$.

*Antras žingsnis.* Išspręsti sveikaskaitinio tiesinio optimizavimo užduotį:

Maksimizuoti $\sum V_j\beta_j$ (23)

Atsižvelgiant į: $\sum_{j=1}^{r} w_j\beta_j \le W$ (24)

$\sum_{j=1}^{r} \alpha_j^i\beta_j \le b_i$, i=1,…,m (25)

$\beta_j \ge 0$ ir j=1,2,...,r. (26)

Šio algoritmo procedūra yra euristinė, nes imamas tik vienas (geriausias) pjovimas kiekvienai juostai, o sprendimas gali būti sudarytas iš skirtingų pjovimų tai pačiai juostai. Be to antrojo žingsnio

modelis nėra lengvai sprendžiamas ir reikalauja labai sudėtingo kodo palyginus su kuprinės užduotimi.[2]

***Sukeitimo***

Kuomet dar neturime jokio sprendinio patogu yra naudotis godžia ar kokia kita euristika, kuri pateikia nebloga rezultatą. Tačiau turint tam tikrą sprendinį galima bandyti jį pagerinti naudojant sukeitimo algoritmą. Toks algoritmas gautame rinkinyje atlieka elementų sukeitimus bandydamas gauti dar geresnį rinkinį. Sukeitimai yra vykdomi atsitiktine tvarka. Algoritmas baigia darbą kuomet iteracijų skaičius tampa lygus elementų skaičiui.[2]

## 2.3. Pjaustymo uždaviniams naudojamų algoritmų privalumai bei trūkumai

Sprendžiant pjaustymo uždavinius yra naudojama eilė įvairių algoritmų, metodikų ar jų modifikacijų. Tai susiję su tuo, kad pasireiškiant vienai ar kitai uždavinio specifikai tenka naudoti būtent tam atvejui skirtą metodiką ar jos modifikaciją. Tai taip pat susiję ir su kai kurių algoritmų trūkumais bei jų specifika.

Dažniausiai globaliam optimizavimui naudojami genetiniai algoritmai ar jų modifikacijos. Šių algoritmų grupė pasižymi galimybe parinkti gerą sprendinį, remdamasi natūralios evoliucijos, vykstančios gamtoje, principu. Dėl šio principo naudojimo ir kyla kai kurios problemos. Kadangi galutinis sprendinys priklauso nuo prieš tai buvusių algoritmo žingsnių, tai pagrindinė problema yra išsigimimu eliminavimas proceso eigoje. Tai reiškia, kad kiekviename žingsnyje reikia stebėti ar geros chromosomos yra parenkamos naujos kartos sudarymui vykdant mutacijos ir kryžminimo operacijas.

Dauguma giljotininio supjaustymo uždavinių yra naudojami realiame laike, todėl svarbu yra skaičiavimo rezultatus pateikti per trumpą laiko tarpą.

Publikuotose algoritmų aprašymuose nėra užsimenama apie susidariusių nuopjovų panaudojimą. Tai yra aktualu įmonėms, kadangi susidariusios nuopjovos, priklausomai nuo jų dydžių gali būti sėkmingai panaudotos tolimesniuose pjaustymuose. Todėl siekiant sumažinti atliekų kiekį yra tikslinga specialiai sudaryti tokias nuopjovas, kurios galėtų būti panaudotos ateityje.

Šio tyrimo tikslas yra sudaryti keletą algoritmų, kurie nenaudotų sudėtingų ir skaičiavimo laikui imlių globalios optimizacijos algoritmų, o pateiktų skaičiavimo rezultatus realiame laike.[2]

# 3. Dažniausiai naudojamų dviejų dimensijų giljotininio pjaustymo euristikos ir jų palyginimas

Šių euristikų tikslas yra sudaryti algoritmą tokį, kurio kainos funkcija (matuojanti erdvinį nuostolį) būtų minimizuota. Paprastai neįmanoma surasti optimalų sprendinį per priimtiną laiko tarpą, nes giljotininio pjaustymo uždavinys yra NP sunkumo. Šiuo metu egzistuoja keletas metodų šiam uždaviniui spręsti ir juos galima sugrupuoti į tris plačias klases: euristika, meta euristika ir tikslūs metodai. Euristika yra metodai, pagrįsti intuityviais ir (arba) tikėtinais argumentais, kurie duoda pakankamai gerus sprendimus esant tam tikromis sąlygomis, bet negarantuoja optimalaus. Meta euristika apytikriai yra aproksimavimo procedūros, kurios produktyviai ir efektyviai ieško sprendimo erdvėje (ar poerdvėje), naudodamos įvairius rinkinių euristikas. O tikslus algoritmas garantuoja optimalų sprendinį.

Dviejų dimensijų giljotininio pjaustymo uždavinį galima spręsti pritaikius dviejų dimensijų pakavimo uždavinį. Pastarasis gali būti klasifikuojamos į dvi svarbias klases – ruožo pakavimo *(strip packing problems)* ir dėžės pakavimo *(bin packing problems)*. Ruožo pakavime visi pakuojami daiktai yra dedami į vieną dėžę su pastoviu pločiu W (pavadintas ruožu) ir su begaliniu aukščiu H siekiant jį sumažinti. Dėžės pakavime visi daiktai yra dedami į daugialypes pastovaus pločio W ir aukščio H dėžes ir siekiama sumažinti panaudotų dėžių skaičių. Abiejuose klasėse pakuojamiems „gaminiams“ neleidžiama persikloti.

Daugiausiai dėmesio skirsime ruožo pakavimo uždaviniui spręsti, kuriose daiktai gali būti pakuojami tik su pastovia orientaciją (negalima sukinėti pakuojamų stačiakampių). Šio uždavinio puikus pavyzdys būtų popieriaus pramonė, kur ritinių formos žaliavoje yra spausdinama informacija, o pagrindinis tikslas būtų minimizuoti šio popieriaus ilgį.

Ruožo pakavimo uždavinio euristika gali būti klasifikuota į tris klases – lygmens algoritmai *(level algorithms)*, pakopos algoritmai *(shelf algorithms)*, plokštumos algoritmai *(plane algorithms)*. Algoritmų pirmoji klasė naudojama tam, kad išspręstų pakavimo uždavinius, kuriose visas stačiakampių sąrašas yra iš anksto žinomas. Šiems algoritmams būdingas ruožo dalijimas į horizontalius lygmenis su tam tikru pagrindu ir dedančius daiktus ant jų. Antrojoje algoritmų klasėje yra žinoma tik einamo, kuris buvo supakuotas, stačiakampio matmenys ir šių algoritmų uždaviniai vadinami tiesioginio pakavimo uždaviniai. Pakopos algoritmai taip pat dalo ruožą į horizontalias juostas, tačiau jų aukščiai parenkami palyginti didesni negu reikalingas patalpinti stačiakampiui. Taip

yra sukuriama didesnė erdvė jei vėliau atsirastų aukštesnis stačiakampis. Plokštumos algoritmuose ruožas nėra padalytas ir stačiakampiai yra padedami bet kokioje pasiekiamoje erdvėje, kur jie dera ruožo viduje.[4]

**1 lentelė**

**Stačiakampių matmenų pavyzdys, kuriuos reikia supakuoti į 40 vienetų pločio juostą**

| | $L_1$ | $L_2$ | $L_3$ | $L_4$ | $L_5$ | $L_6$ | $L_7$ | $L_8$ | $L_9$ | $L_{10}$ | $L_{11}$ | $L_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $h(L_i)$ | 6 | 16 | 20 | 24 | 4 | 4 | 6 | 16 | 4 | 6 | 3 | 3 |
| $w(L_i)$ | 8 | 32 | 3 | 24 | 13 | 2 | 7 | 11 | 8 | 12 | 13 | 28 |

Pagrindinis mūsų tikslas yra apžvelgti ir įvertinti dviejų dimensijų pakavimo euristikas uždaviniams spręsti bei pasiūlyti pagerinimus kai kurioms šitų procedūrų.

## 3.1. Lygio algoritmai

Visuose lygio algoritmuose, tam tikruose horizontaliuose lygmenyse, trumpesnėmis kraštinėmis stačiakampiai L yra dedami ruožo viduje. Kiekvieno lygmens aukštis nustatomas pagal didžiausių stačiakampių padėtų ankstesniuose lygmenyse. Pabandysime apžvelgti keletą standartinių lygmens algoritmų.

***Sekančio tinkančio mažėjančio aukščio algoritmas (The Next-Fit Decreasing Height algorithm-NFDH)***

Taip vadinamajame sekančio tinkančio mažėjančio aukščio algoritme stačiakampių sąrašas, kurie turėtų būti supakuoti yra iš anksto sudėliotas pagal nedidėjantį aukštį. Tai reiškia, kad pirmas stačiakampis padėtas konkrečiame lygmenyje nustato sekančio lygmens aukštį. Jei stačiakampis padėtas einamajame lygyje ir telpa į plotį – tai viskas gerai. O jei netelpa (netinkamas), tai sukuriamas naujas lygmuo, aukščiau esamo (kuris tampa einamuoju lygmeniu) ir stačiakampis dedamas jame. Taip pakavimo procesas tęsiasi iš kairės į dešinę ir aukštyn pagal lygius kol nebelieka pakuojamų stačiakampių. Lygiai, esantys žemiau esamo, niekada neperžiūrimi. To paties aukščio stačiakampiai išlaiko originalią tvarką pakuojamų daiktų sąraše atitinkamai vienas kito atžvilgiu. Ši procedūra sudeda pavyzdinius duomenis į 56 vienetų aukščio juostą.[4]

***Sekančio tinkančio mažėjančio aukščio didėjančio pločio algoritmas (The Next- Fit Decreasing Height Increasing Width algorithm NFDHIW)***

Tai naujai pasiūlytas algoritmas, kuris yra gautas truputi pakeitus NFDH algoritmą. Pagrindinis skirtumas yra tas, kad pakuojami stačiakampiai yra išrikiuojami į seką mažėjančiu aukščiu didėjančiu pločiu. Ši procedūra sutalpina pateiktus duomenis į 53 vienetų aukščio juostą.[4]

***Sekančio tinkančio mažėjančio aukščio mažėjančio pločio algoritmas (The Next- Fit Decreasing Height Decreasing Width algorithm NFDHDW)***

Tai taip pat panašus algoritmas į jau pateiktą NFDH. Jo pagrindinis skirtumas yra tas, kad pakuojami stačiakampiai rikiuojami į seką mažėjančio pločio mažėjančio aukščio. Ši procedūra pateiktus testinius duomenis sudeda į 66 vienetų pakavimo aukštį.[4]

***Pirmo tinkančio mažėjančio aukščio algoritmas (The First-Fit Decreasing Height algorithm FFDH)***

Taip vadinamajame pirmo tinkančio mažėjančio aukščio algoritme, stačiakampių sąrašas, kurie turėtų būti supakuoti, yra iš anksto surūšiuotas pagal nedidėjantį aukštį. Tai reiškia, kad pirmas stačiakampis padėtas lygmenyje nustato kito lygmens aukštį. Pakavimo sąraše to paties aukščio stačiakampiai išlaiko originalią tvarką atitinkamai vienas kito atžvilgiu. Pakuojamas objektas dedamas kairėje pusėje išlygiuotai žemiausiame lygyje, kur yra pakankamai vietos. Stačiakampio padėjimo pozicija yra ieškoma per visus esamus lygius nuo apačios į viršų, t.y. ten kur yra jam laisvos vietos pakavimui. Jei stačiakampis yra padedamas kažkuriame lygyje ir telpa į plotį – tai viskas gerai. O jei netelpa į jokį esamą lygmenį, tai sukuriamas naujas lygmuo virš aukščiausio lygio ir stačiakampis dedamas jame. Pagrindinis skirtumas tarp HFDH ir FFDH yra tas, kad pastarajame visi pakavimo lygmenys yra peržiūrimi, kai tuo tarpu pirmajame anksčiau pakuoti lygiai neperžiūrimi. Ši procedūra pavyzdinius duomenis sudeda į 53 aukščio juostą.[4]

***Pirmo tinkančio mažėjančio aukščio didėjančio pločio algoritmas (The First-Fit Decreasing Height Increasing Width algorithm FFDHIW)***

Tai naujai pasiūlytas algoritmas, kuris yra gautas truputi pakeitus FFDH algoritmą. Pagrindinis skirtumas yra tas, kad pakuojami stačiakampiai yra išrikiuojami į seką mažėjančiu aukščiu didėjančiu pločiu. Ši procedūra sutalpina pateiktus duomenis į 53 vienetų aukščio juostą.[4]

***Pirmo tinkančio mažėjančio aukščio mažėjančio pločio algoritmas (The First-Fit Decreasing Height Decreasing Width algorithm FFDHDW)***

Tai taip pat panašus algoritmas į jau pateiktą FFDH. Jo pagrindinis skirtumas yra tas, kad pakuojami stačiakampiai rikiuojami į seką mažėjančio pločio mažėjančio aukščio. Ši procedūra pateiktus testinius duomenis sudeda į 52 vienetų pakavimo aukštį.[4]

***Geriausiai tinkančio mažėjančio aukščio (The Best-Fit Decreasing Height algorithm BFDH)***

Geriausiai tinkančio mažėjančio aukščio algoritmas yra analogiškas pirmo tinkančio mažėjančio aukščio algoritmui (FFDH), išskyrus tai, kad šiame algoritme stačiakampiai yra dedami kairėje išlygiuotai, dešinėje pusėje prieš paskutinį supakuotą stačiakampį ir lygyje, kuris turi mažiausią likusią erdvę. Tai reiškia, kad norint supakuoti sekanti stačiakampį, iš naujo paieškoma likusi mažiausia horizontali laisva erdvė visuose esamuose lygiuose ir stačiakampis dedamas joje. Kai nesurandama tinkama vieta, tada sukuriamas naujas lygmuo. Ši procedūra pateiktus testinius duomenis sudeda į 53 vienetų pakavimo aukštį.[4]

***Geriausiai tinkančio mažėjančio aukščio didėjančio pločio algoritmas (The Best- Fit Decreasing Height Increasing Width algorithm BFDHIW)***

Tai naujai pasiūlytas algoritmas, kuris yra gautas truputi pakeitus BFDH algoritmą. Pagrindinis skirtumas yra tas, kad pakuojami stačiakampiai yra išrikiuojami į seką mažėjančiu aukščiu didėjančiu pločiu. Ši procedūra sutalpina pateiktus duomenis į 53 vienetų aukščio juostą.[4]

***Geriausiai tinkančio mažėjančio aukščio mažėjančio pločio algoritmas (The Best- Fit Decreasing Height Decreasing Width algorithm BFDHDW)***

Tai taip pat panašus algoritmas į jau pateiktą BFDH. Jo pagrindinis skirtumas yra tas, kad pakuojami stačiakampiai rikiuojami į seką mažėjančio pločio mažėjančio aukščio. Ši procedūra pateiktus testinius duomenis sudeda į 52 vienetų pakavimo aukštį.[4]

***Dalinimo algoritmas (The Split Fit algorithm SF)***

Dalinimo algoritme visų stačiakampių ilgiai ir pločiai turi būti sveiko skaičiaus, o pakavimas vykdomas į vieno vieneto pločio ruožą. Taip pat, šioje procedūroje didžiausias sveikas skaičius m>=1 yra nustatytas tada, kai visi pakuojami stačiakampiai L turės plotį mažiau arba lygu 1/m. Tada pakuojamų objektų eilė yra dalijama į du sąrašus $L_{wide}$ ir $L_{narrow}$ , kurie bus išrūšiuoti mažėjimo tvarka taip, kad $L_{wide}$ turėtų savyje visus stačiakampius, kurių pločiai yra didesni negu 1 / (m + 1) ir $L_{narrow}$, kuriame stačiakampių pločiai mažesni nei 1 / (m + 1). Pirmiausiai sąrašas $L_{wide}$ pakuojamas panaudojant FFDH algoritmą ir visi stačiakampiai, kurie padėti atskiruose lygmenyse yra sugrupuojami į blokus. Šie blogai surūšiuojami taip, kad jei bendras juose esančių stačiakampių plotis yra didesnis už (m + 1)/(m + 2) būtų žemiau už blokus, kurių plotis mažesnis už (m + 1)/(m + 2). Šis sukeitimo procesas sukuria stačiakampę sritį R su pločiu 1 / (m + 2) į dešinę nuo paskutinių blokų. Toliau panaudojant tą patį FFDH algoritmą yra pakuojamas sąrašas $L_{narrow}$ pradedant nuo srities R. Jei stačiakampis netelpa į R, tai pakavimas tęsiamas virš $L_{wide}$ pakavimo. Ši procedūra pateiktus duomenis sutalpina į 63 matavimo vienetų aukščio juostą.[4]

***Žemiausios-aukščiausios ribos be sukimosi algoritmas (The Floor-Ceiling No Rotation algorithm FCNR)***

Žemiausios-aukščiausios ribos be sukimosi algoritme stačiakampiai yra iš anksto surūšiuojami pagal nedidėjantį aukštį. Lygmens viduje žemiausia riba *(floor)* yra apibrėžta kaip horizontali linija, sutampanti su apatiniais stačiakampių kraštais, supakuotų tame lygmenyje, o aukščiausia riba *(ceiling)* yra horizontali linija, sutampanti su aukščiausio stačiakampio viršutiniu kraštu, supakuoto tame lygmenyje. Jei yra pakankama erdvė stačiakampiui ant žemiausios ribos, tada sakoma, kad tai yra įmanoma žemiausia riba. Stačiakampiai yra supakuoti ant žemiausios ribos nuo kairės į dešinę pusę taip, kad kairė kraštinė sutaptų su prieš tai buvusio stačiakampio dešine kraštine. Jei pirmas stačiakampis, kuris padėtas ant aukščiausios ribos, yra supakuotas taip, kad jo dešinė kraštinė sutampa su pakuojamos juostos dešine kraštine, tai sakoma, kad inicijuota lygmens aukštutine riba. Stačiakampiai pakuojami ant aukštutinės ribos iš dešinės į kairę gali turėtu tuščiu erdvių, kad tenkintų giljotininio pjaustymo sąlygas. Aukščiausios ribos inicializavimas visada turi pirmenybę prieš žemiausios ribos inicializavimą, nes tai užlaiko naujų lygmenų sukūrimą.[4]

Žemiausios-aukščiausios be sukimosi algoritmas naudoja tą patį principą kaip ir geriausio tinkamo algoritmų klasė (BFDH), kurioje stačiakampis yra pakuojamas į lygmenį su minimalia likusia horizontalia erdve. Šiame metode pirmiausiai yra apskaičiuojama likusi horizontali aukščiausios ribos

erdvė kiekviename lygmenyje ir jei nei vienas iš stačiakampių negali būti inicijuotas ir padėtas joje, tai skaičiavimai atliekami žemiausioje horizontalioje erdvėje. Jei atstumas tarp aukščiausio stačiakampio ir aukščiausios ribos krašto yra nepakankami tam, kad patalpintų bet kurį iš pakuojamų stačiakampių, tai tokio stačiakampio dešinė kraštinė suformuoja kairę sieną. Aukštutinės ribos horizontali erdvė bus atstumas tarp kairės ruožo sienos ir kairiojo stačiakampio krašto. Naujas lygmuo yra kuriamas, kai nei vienas iš pakuojamų stačiakampių negali tikti nei ant aukščiausios nei ant žemiausios ribos visuose egzistuojančiuose lygiuose. Ši procedūra pateiktus duomenis sutalpina į 47 matavimo vienetų aukščio juostą.[4]

***Kuprinės algoritmas (The Knapsack algorithm KP)***

Iš pradžiu kuprinės algoritmas buvo išvystytas dėžės pakavimo problemai spręsti. Tačiau šį metodą aprašysime ruožo pakavimo kontekste.

Kuprinės algoritme stačiakampiai yra iš anksto surūšiuoti pagal mažėjantį aukštį. Kiekvienas naujas lygmuo yra inicijuojamas, kai pakuojamas stačiakampis ($L_{j*}$) ir jo aukštis yra didesnis už tam tikrą nustatytą skaičių. Po iniciavimo kuprinės uždavinį reikia išspręsti:

$$\left.\begin{aligned} &\text{maximise} \sum_{i=1}^{n} h(L_i)w(L_i)x_i, \\ &\text{subject to} \sum_{i=1}^{n} w(L_i)x_i \leq W - w(L_{j*}), \\ &\qquad x_i \in \{0,1\} \quad (i = 1,\ldots,n), \end{aligned}\right\}$$

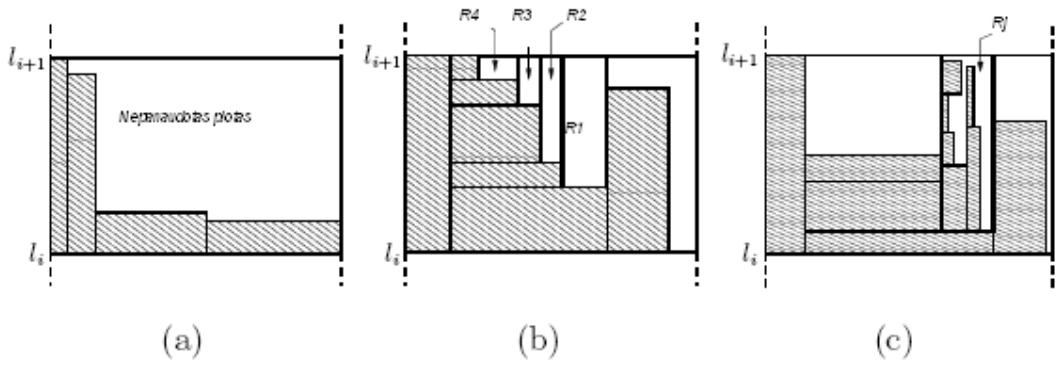
Kuprinės problemos sprendimas identifikuoja tuos stačiakampius, kurie turi būti supakuoti tam tikrame lygmenyje (pavyzdžiui $x_2 = 1$, $x_3 = 0$ reiškia, kad stačiakampis $L_2$ turi būti supakuotas ir $L_3$ neturi būti supakuotas tam tikrame nagrinėjamame lygmenyje). Algoritmas tęsiamas, kol visi stačiakampiai supakuojami. Šis algoritmas pateiktus testinius duomenis supakuoja į 52 vienetų aukščio ruožą.[4]

***Kintamo sąrašo* (*Alternate Size Stack algorithm - ASS*)**

Buvo nustatyta, kad kai skirtumas tarp stačiakampių aukščių, įtelpančių į vieną lygmenį, tampa ekstremalus, FFDH algoritmas labai blogai ieško sprendinio. Dėl šio pastebėjimo atsirado naujas algoritmas, kuris pavadintas Kintamo sąrašo algoritmas *(Alternate Size Stack algorithm - ASS)*. Šiame metode, stačiakampių eilė $L_n$ yra padalyta į du sąrašus $L_1$ ir $L_2$ atitinkamai tenkinančius sąlygas $h(L_i) > w(L_i)$ ir $h(L_j) < w(L_j)$. Stačiakampiai $n_1$ iš $L_1$ pavadinti siaurais ir $n_2$ iš $L_2$ plačiais. Stačiakampiai sąraše $L_1$ yra surūšiuojami pagal nedidėjantį aukštį, o sąraše $L_2$ - išrūšiuojami pagal nedidėjantį plotį.

Kiekvienas pakavimo lygmuo yra inicijuojamas, lyginant aukščius pirmo stačiakampio abiejuose sąrašuose (t.y. aukščiausio stačiakampio sąraše $L_1$ ir plačiausias stačiakampis sąraše $L_2$) - didžiausio aukščio stačiakampio pakavimas. Parinkto stačiakampio aukštis tampa lygmens aukščiu ir horizontali linija yra nupiešiama sutampanti su aukščiausiu kraštu stačiakampio į ruožo dešinę kraštinę, kad atribotų viršutinę lygmens sieną.

Svarbiausia mintis šiame algoritme yra ta, kad vykdomas keitimas tarp siaurų ir plačių stačiakampių, pakuojant daiktus iš kairės į dešinę kiekviename ruožo lygmenyje. T.y. jei stačiakampis, inicijuojantis lygmenį yra iš $L_1$, tai sekantys stačiakampiai imami iš $L_2$ ir atvirkščiai. Kai tik sąrašas iš kurio bus pakuojama identifikuotas, stačiakampiai yra kraunami vienas ant kito prasidedančio nuo žemesnės lygmens sienos, kol viršutinė siena nėra pasiekta arba kol vertikali erdvė tarp viršutinės sienos lygmens ir aukščiausio krašto aukščiausio stačiakampio yra nepakankama, kad sutalpintų bet kurį iš pakuojamų stačiakampių tame sąraše.[4]




Šaltinis: N. Ntene, J.H. van Vuuren, 2008

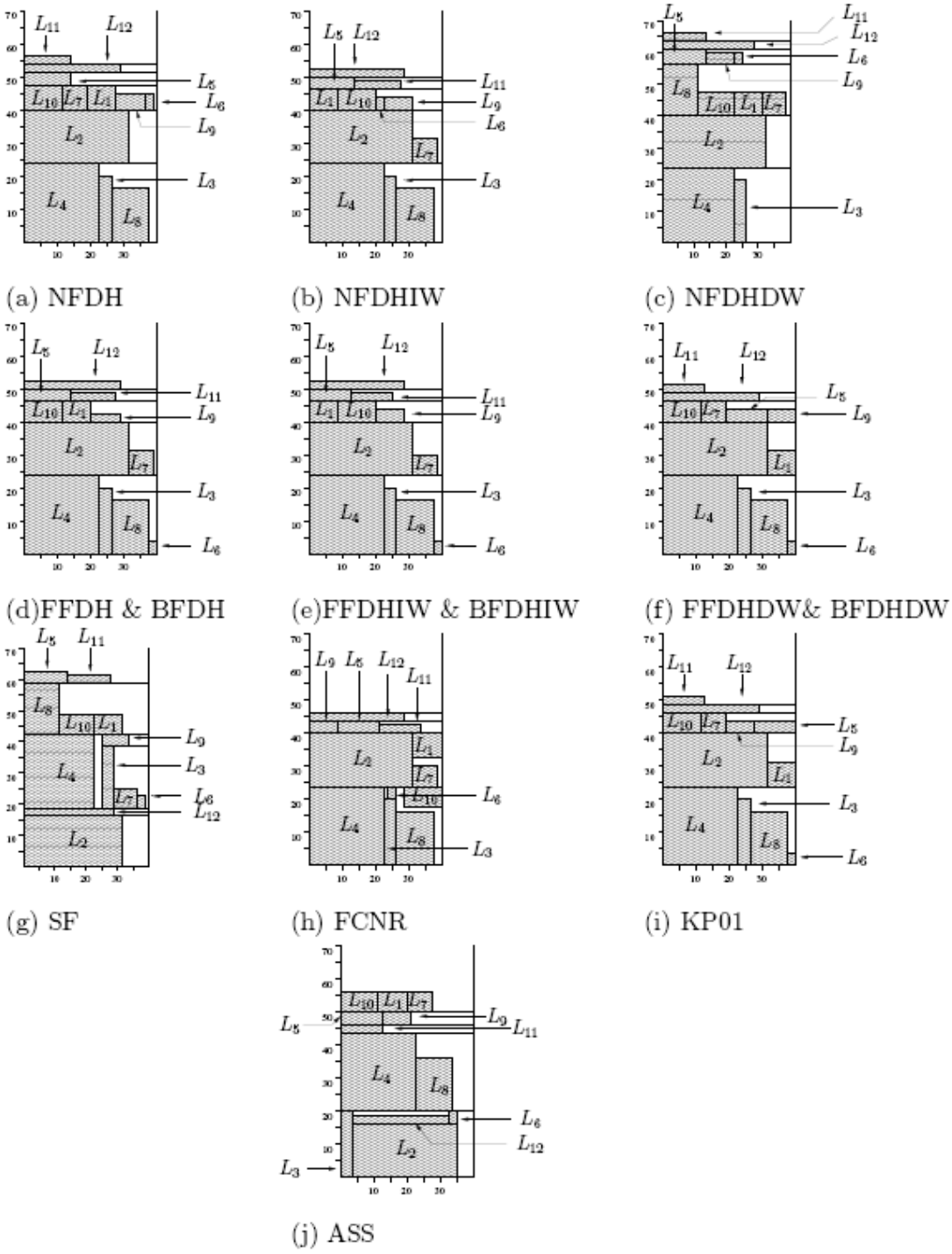
**5 pav. (a) ASS algoritmo pradžia, (b) Tuščios stačiakampės sritys išnyko, kraudamos plačius stačiakampius su ASS algoritmo pagalba, (c) Krovimas siauro stačiakampio srityje Rj.**

Dedant plačius stačiakampius, jei einamų stačiakampių pločiai $L_i$ ir $L_{i+1}$ nėra lygūs, paliekama tuščia stačiakampė sritis $R_j$, kurios kairė siena yra nuo dešinės stačiakampio kraštinės, kaip parodyta 5 paveiksliuke (b). Kiekvienos stačiakampės srities aukštis tęsiasi nuo aukščiausio stačiakampio krašto $L_i$ į viršutinę lygmens sieną, esant kiekvienos nepanaudotos srities pločiui $w(R) = w(L_i) - w(L_{i+1})$. Siauri stačiakampiai yra sukrauti išrenkant juos iš visų, kurių plotis neviršija siauriausio stačiakampio pagrindo pločio ir tinkamas aukščiu lygmens viduje. Siaurų stačiakampių krovimas srityje $R_j$ pavaizduotas 5 iliustracijoje (c), kur stačiakampiai yra sudėlioti taip, kad užpildytų $R_j$ plotį, kol nėra pakankamos horizontalios erdvės ar nėra jokių siaurų stačiakampių pakavimui.

Šis algoritmas yra lankstus – jei per apsikeitimą sąrašais yra nepakankama horizontali erdvė lygmenyje, kad supakuotų bet kurį iš stačiakampių pažymėtame sąraše, tai pasirenkamas alternatyvus

sąrašas pakavimui, jei tam yra pakankama erdvė. Naujas lygmuo yra inicijuojamas, jei niekas iš stačiakampių ar $L_1$, ar $L_2$ netilpo į horizontalią erdvę tarp dešiniosios ruožo sienos ir labiausiai dešinėje supakuoto stačiakampio dešinės kraštinės.

ASS algoritmas formuoja giljotinos pakavimą. Šis metodas pateiktus testinius duomenis supakuoja į 57 vienetų aukščio ruožą.[4]




Šaltinis: N. Ntene, J.H. van Vuuren, 2008

**6 pav. Pakavimai, pagaminti lygmens algoritmų.**

## 3.2. Etaloninis testas

Etaloninio testo naudojimas yra standartinis metodas naujų algoritminių procedūrų įvertinimui, palengvindamas palyginimus tarp erdvės ir laiko produktyvumo bei sprendimo kokybių. Daug internetinių bibliotekų publikuoja etaloninio testo duomenis internete algoritmų išbandymams, suprojektuotus tam, kad išspręstų didelę įvairovę gerai dokumentuotų uždavinių. Tačiau dažnai tyrėjai savo sukurtus algoritmus bando ant naujų testinių duomenų rinkinių ir jų nepadaro prieinamu visiems, kai publikuoja savo metodų atlikimo įvertinimus. Tai žlugdo etaloninio testo išbandymo tikslą, vietoj to, kad tyrinėtų su egzistuojančiais duomenimis. Etaloninio testo duomenys yra surinkti tam, kad išbandytų įvairias ruožo pakavimo euristikas.

***Mumford-Valenzuela testo duomenys***

Tai dvi duomenų rinkinio klasės. Pirmoji klasė pavadinta Gražaus *(Nice)* rinkinio duomenų klase ir kiekvienas duomenų rinkinys šioje klasėje susideda iš panašių dydžių ir formų stačiakampių. Antroji klasė pavadinta Kelio *(Path)* duomenų rinkinio klase ir kiekvienas duomenų rinkinys šioje klasėje susideda iš labai įvairių formų ir dydžių stačiakampių. Procedūra, kuri generavo duomenis apibrėžė stačiakampių ploto santykius. Gražus ir Kelio duomenų rinkinys turi santykį su dydžiais diapazone atitinkamai 1/4<= H/W<=4 ir 1/100<=H/W<=100. Taip pat buvo nustatyti maksimalūs bet kokių dviejų stačiakampių plotų santykiai 7 ir 100 vienetų atitinkamai Gražiam ir Kelio duomenų rinkiniui. Giljotininiam variantui suformuotas duomenų rinkinys dydžio n, t.y. n – 1 giljotinos pjovimai ant 100 x 100 kvadrato.[4]

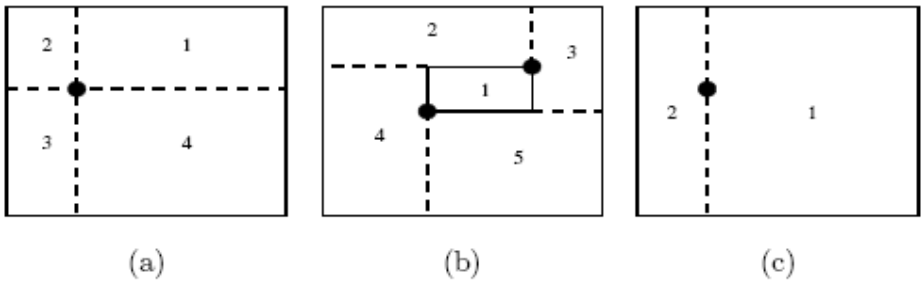
***Hopper and Turton testo duomenys***

Šis bandomasis duomenų rinkinys padalintas į septynias kategorijas (i = 1,..,7), kur kiekviena kategorija apima tris bandomuosius pavyzdžius (j = 1,2,3). Stačiakampių skaičius kiekvienoje kategorijoje svyruoja nuo 17 iki 197. Šitas duomenų rinkinys buvo sukurtas atsitiktinai, išlaikant maksimalų aukščio ir pločio santykį 7.[4]

***Hopper testo duomenys***

Buvo sukurtos dvi klasės – duomenų rinkinys giljotininiam pjaustymui ($T_i^j$) ir duomenų rinkinys ne giljotininiam pjaustymui ($H_i^j$). Kiekviena klasė (i = 1,..,7) apima penkis pavyzdžius (j = 1,..,5), kurie turi išsipjauti iš 200 x 200 kvadrato. Šie duomenys buvo sukurti giljotininio pjaustymo būdu, parinkinėjant atsitiktinį tašką stačiakampyje ir darant vertikalius ir horizontalius pjovimus per tą

tašką taip, kad susidarytų keturi nauji stačiakampiai, kaip pavaizduota 7 paveikslo iliustracijoje (a). Bandomųjų ne giljotininio problemų atveju stačiakampio viduje buvo parinkinėjami du atsitiktiniai taškai, kaip pavaizduota paveikslo 7 iliustracijoje (b). Mažesnio stačiakampio kraštinės buvo ištęstos , kur jie perkerta didesnio stačiakampio kraštines – tokiu būdu suformuoti 5 stačiakampiai.[4]




Šaltinis: N. Ntene, J.H. van Vuuren, 2008

**7 pav. Giljotinos ir ne giljotinos įpjovimai; (a) giljotinos pjovimas su 4 stačiakampiais, (b) ne giljotinos pjovimas su 5 stačiakampiais, (c) giljotinos pjovimas su 2 stačiakampiais.**

***Burke testo duomenys***

Sukurta trylika bandomųjų pavyzdžių. Šitie duomenų rinkiniai buvo sukurti atsitiktinai, pakartotinai sumažindami didelį stačiakampį ar vertikaliai, ar horizontaliai taip, kad du nauji stačiakampiai būtų sukurti po kiekvieno įpjovimo, kaip pavaizduota 7 paveiksle iliustracijoje (c). Čia garantuojama, kad matmenys nebus mažesni negu kažkoks apibrėžtas minimalus matmuo.[4]

***Chistofides and Whitlock testo duomenys***

Duomenų rinkinys susideda iš trijų pavyzdžių $G_1$,$G_2$,$G_3$ ir buvo numatyti giljotinos uždaviniui. Ploto m mažesni stačiakampiai diapazone [0, 0.25A], kur A reiškia pradinio didelio stačiakampio plotą. Po gautų stačiakampių su plotu m, kiekvieno aukštis $w(L_j)$ buvo apskaičiuotas pagal formulę $w(L_j)=m_i / h(L_i)$. Galimas optimalus ruožo pakavimo sprendimas šitiems duomenų rinkiniams nėra žinomas.[4]

***Beasley testo duomenys***

Buvo sukurta daug duomenų rinkinių, bet tiktai keturi ($U_1$,$U_2$,$U_3$,$U_4$) tinka giljotinos uždaviniui analizuoti. Sveikieji skaičiai buvo gauti po vienodo dalijimo [H/4, 3H/4] ir [W/4, 3W/4] atitinkamai aukščiui $h(L_j)$ ir pločiui $w(L_j)$ kiekvienam sukurtam stačiakampiui. Šiems duomenų rinkiniams galimi optimalūs sprendiniai taip pat nėra žinomi.[4]

### 3.3. Algoritmų rezultatų palyginimas

Šiame poskyryje pateiksim bandomuosius rezultatus, kurie buvo gauti pritaikius ankščiau aprašytus algoritmus. Kiekvienas metodas išbandytas su 542 etaloninio testo duomenų rinkiniais. Kiekvienas bandomasis atvejis ir dažnis, su kuriuo algoritmas pasiekia mažiausia pakavimo aukštį, buvo panaudoti kaip kriterijai lyginant ir vertinant algoritmus. Idealiu atveju geriausias euristinis metodas sugebėtų gauti mažiausia ruožo aukštį per trumpiausią laiką. Pradžioje kiekvienas standartinis euristinis algoritmas yra palyginamas su jo pasiūlytais pagerinimais ir pagaliau visi algoritmai yra palyginami išreiškiant jų efektyvumu ir charakteristika. Standartinės statistinės analizės, tokios kaip dispersinė analizė *(Analyses Of Variance (ANOVA))* ir $\chi^2$ (chi-kvadrato) testai buvo atlikti tam, kad išmatuoti statistiškai reikšmingus skirtumus tarp vidutinių ruožo aukščių gautų skirtingais klasių algoritmais, su kuriais pasiektas mažiausias ruožo aukštis. Taip pat buvo apskaičiuotas algoritmų efektyvumo lygis PR(A)=A(L) / OPT(L), kur A(L) – algoritmo gautas ruožo aukštis, kai taikytas etaloninio testo pavyzdžiui L ir kur OPT(L) yra optimalus aukštis susietas su duomenų rinkinių L.[4]

**2 lentelė**

**Dispersinės analizės rezultatai gauti panaudojus algoritmų NF, FF, BF klasių euristikas**

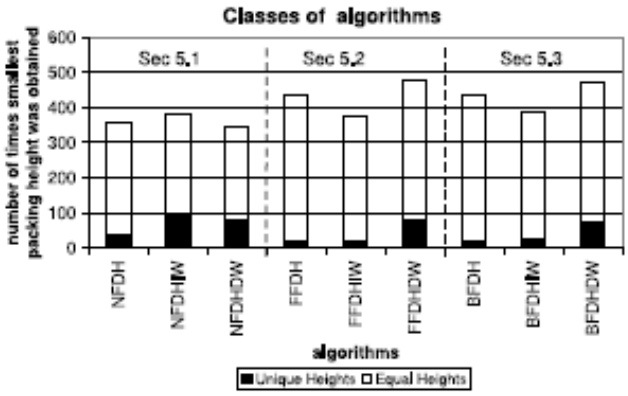
| NFDH | NFDHIW | NFDHDW | FFDH | FFDHIW | FFDHDW | BFDH | BFDHIW | BFDHDW | $F_{value}$ | $F_{critical}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 169.542 | 169.488 | 169.708 | | | | | | | 0.00016 | 3.00127 |
| | | | 161.570 | 161.875 | 161.399 | | | | 0.00104 | 3.00127 |
| | | | | | | 161.360 | 161.607 | 161.240 | 0.00063 | 3.00127 |

***Palyginimas algoritmų Sekančioje tinkančioje klasėje (Next-Fit class NF)***

Reliatyvus NDFH, NFDHIW ir NFDHDW algoritmų atlikimas buvo įvertintas kaip kiekvieno algoritmo gebėjimas gauti mažiausią ruožo aukštį. Šių algoritmų gauti aukščiai, atlikus 542 bandymus, pavaizduoti 2 lentelėje kartu su dviem vertėmis, $F_{value}$ ir $F_{critical}$ kiekvienu atveju. Čia $F_{value}$ reiškia skirtumą tarp supakuojančių daiktus aukščių gautų algoritmų ir skirtumo ruožo aukščių kiekvieno algoritmo viduje, o $F_{critical}$ yra testo teorinė-statistinė reikšmė, gauta iš F-dalijimo plokštumos. Dispersinės analizės rezultatai sekančio tinkamo algoritmų klasės palyginime rodo, kad nėra jokio reikšmingo skirtumo tarp vidutinių ruožo aukščių, gautų algoritmų klasėje 5% reikšmės lygmenyje. Šis rezultatas nėra geras, atsižvelgiant į panašią svarstomų algoritmų prigimtį.

Sekančiu žingsniu įvertinama, kaip dažnai kiekvienas algoritmas įgavo mažiausią aukštį (8 pav.). Pirmos trys juostos pavaizduoja dažnius NF algoritmų klasei. Nenuspalvotų juostų aukščiai reiškia kartus kiek kiekvienas algoritmas įgavo mažiausią pakavimo aukštį, o nuspalvotų juostų aukščiai reiškia kiek kartų kiekvienas algoritmas buvo vienintelė procedūra gavusi mažiausią

pakavimo aukštį (t.y. kiek kartų algoritmas rado mažiausią pakavimo aukštį unikaliai). NFDHIW algoritmas sugebėjo gauti mažiausią ruožo aukštį dažniau negu kiti du algoritmą (8 pav.).[4]




Šaltinis: N. Ntene, J.H. van Vuuren, 2008

**8 pav. Dažniai, su kuriais algoritmai pasiekė mažiausią ruožo aukštį**

**3 lentelė**

**Chi-kvadrato testo rezultatai, su kuriais algoritmai gavo mažiausią pakavimo aukštį atlikus 542 bandymus ir kur $\chi^2_{df}$ ir $\chi_{critical}$ reikšmės aprašo skirtumus tarp dažnių.**

| NFDH | NFDHIW | NFDHDW | FFDH | FFDHIW | FFDHDW | BFDH | BFDHIW | BFDHDW | $\chi^2_{df}(0.05)$ | $\chi_{critical}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 359 | **380** | 344 | | | | | | | **1.812** | **5.990** |
| | | | 433 | 377 | **477** | | | | 11.711 | 5.990 |
| | | | | | | 435 | 385 | **474** | 9.229 | 5.990 |

Chi-kvadrato testo reikšmės 5% reikšmės lygmenyje matomas 3 lentelėje. Kadangi $\chi_{critical} > \chi^2_{df}(0.05)$, tai statistiškai nėra jokio skirtumo tarp dažnių. Tai reiškia, kad joks algoritmas kitoje tinkamoje (NF) klasėje nėra statistiškai geresnis už kitą išreiškiant dažniu, su kuriuo pasiektas geriausias rezultatas 5% reikšmės lygmenyje.[4]

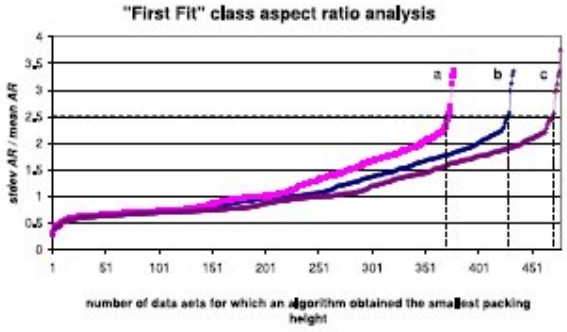
***Algoritmų palyginimas pirmo tinkančio klasėje (First-Fit class FF)***

Šios metodų klasės analizė panaši į prieš tai atliktą. Reliatyvus algoritmų FFDH, FFDHIW ir FFDHDW atlikimas buvo taip pat pagrįstas ruožo aukščiais, gautais kiekvieno algoritmo. Dispersinė analizė parodė, kad nėra jokio reikšmingo skirtumo tarp vidutinių ruožo aukščių gautų trijų algoritmų per visus 542 bandymus 5% reikšmės lygmenyje. Apskaičiuotas (2 lentelė) $F_{value}$ (0.00104) yra mažiau, negu $F_{critical}$ (3.00127). Vadinasi, išreiškiant visa apimančia sprendimo kokybe, nėra jokio

statistinio skirtumo tarp algoritmų FF klasėje, kur tikimasi, kad visi algoritmai pasirodytų maždaug lygiai efektyviai.[4]

Iš 8 paveiksliuko akivaizdu, kad FFDHDW algoritmas dirbo geriau už kitus du algoritmus išreiškiant dažniu, su kuriuo gavo mažiausią pakavimo aukštį. To priežastis yra ta, kad FFDHDW algoritme platesni stačiakampiai yra supakuoti iš pradžių ir egzistuojančiuose lygmenyse yra visada paieškoma pakankamai erdvės mažesniems stačiakampiams sutalpinti. Tai mažina tikimybę kurti naujus lygmenis, kuris savo ruožtu yra mažesnių ruožo aukščių priežastis.[4]

Chi-kvadrato testas panaudotas tam, kad nustatytų ar yra koks nors statistinis reikšmingas skirtumas tarp dažnių prie bet kurio iš tinkamų algoritmų klasės. Kadangi $\chi_{critical} < \chi^2_{df}(0.05)$ (3 lentelės antra eilutė), tai leidžia suprasti, kad yra reikšmingų skirtumų tarp dažnių. Iš šių rezultatų sprendžiame, kad FFDHDW yra pirmas, FFDH antras ir FFDHIW trečias statistiškai pagal gerumą algoritmas.[4]




Šaltinis: N. Ntene, J.H. van Vuuren, 2008

**9 pav. FF klasės algoritmų efektyvumo lygių grafikas**

***Algoritmų palyginimas geriausio tinkančio klasėje (Best-Fit class BF)***

Dispersinės analizės rezultatai, palyginus ruožo aukščius gautus algoritmų BF klasėje, rodo, kad nėra jokio skirtumo tarp ruožo aukščių, pasiektų algoritmų 5% reikšmės lygmenyje (2 lentelė $F_{value}(0.00063) < F_{critical}(3.00127)$). Vadinasi, išreiškiant sprendimo kokybe, joks algoritmas nėra geresnis už kitą.[4]

Pagal chi-kvadrato testą (3 lentelės trečia eilutė) nustatome, kad reikšmingas yra skirtumas tarp dažnių, kuriuose gautas mažiausias pakavimo aukštis tarp BF klasės algoritmų todėl, kad $\chi_{critical}$ $(5.990) < (\chi^2_{df}=9.229)$. BFDHDW algoritmas įgavo mažiausią ruožo aukštį dažniau negu kiti du algoritmai šioje klasėje. Iš gautų rezultatų sprendžiame, kad BFDHDW yra pirmas, BFDH antras ir BFDHIW trečias statistiškai pagal gerumą algoritmas.[4]

*Visų euristikų palyginimas*

Pagal 4 lentelės pirmos eilutės dispersinę analizę, sprendžiam, kad nėra jokio skirtumo tarp ruožo aukščių, gautų dešimties algoritmų 5% reikšmės lygmenyje. O chi-kvadrato testas rodo (4 lentelės antra eilutė), kad yra reikšmingas skirtumas tarp dažnių. Palyginus rezultatus matome, kad FCNR algoritmas duoda geresnį rezultatą nei bet kuris kitas, įskaitant net ir ASS metodą. Tai lauktas rezultatas todėl, kad abu šie paminėti algoritmai kiekviena kartą peržiūri buvusius pakavimo lygmenis. Vadinasi abi procedūros yra pranašesnės už kitus pasiūlytus.[4]

**4 lentelė**

**Dispersinės analizės ir chi-kvadrato testo rezultatai**

| ASS | BFDH | BFDHDW | BFDHIW | FCNR | FFDH | FFDHDW | FFDHIW | KP01 | SF | $F_{value}$ | $F_{critical}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 157.046 | 161.360 | 161.240 | 161.607 | 148.872 | 161.570 | 161.399 | 161.875 | 164.292 | 173.289 | 0.610 | 1.882 |
| ASS | BFDH | BFDHDW | BFDHIW | FCNR | FFDH | FFDHDW | FFDHIW | KP01 | SF | $\chi^2_{df}(0.05)$ | $\chi_{critical}$ |
| 136 | 33 | 40 | 24 | 399 | 32 | 39 | 22 | 30 | 8 | 1661.05 | 16.92 |

Paskaičiuoti algoritmų vidutiniai efektyvumo lygiai *(Average Performance Ratios)* APR(A)=A(T) / OPT(T), kur A(T) – algoritmo gautas vidutinis ruožo aukštis, atliekant etaloninį testą su duomenų rinkiniu $T_i$ ir kur OPT(T) yra menamas optimalus aukštis duomenų rinkiniui $T_i$, pavaizduoti 5 ir 6 lentelėse.[4]

**5 lentelė**

**Algoritmų vidutinių efektyvumo lygių lentelė su Hopper and Turton (2002) testo duomenimis**

| Duomenų rinkinys | Rinkinių skaičius | NFDH | NFDHIW | NFDHDW | FFDH | FFDHIW | FFDHDW |
|---|---|---|---|---|---|---|---|
| | | PR | PR | PR | PR | PR | PR |
| T1 | 5 | 1,5870 | 1,5800 | 1,5870 | 1,4650 | 1,4650 | 1,4650 |
| T2 | 5 | 1,5090 | 1,5080 | 1,4920 | 1,4080 | 1,4080 | 1,4080 |
| T3 | 5 | 1,5970 | 1,4970 | 1,4800 | 1,4070 | 1,4070 | 1,4080 |
| T4 | 5 | 1,3170 | 1,3170 | 1,3330 | 1,2750 | 1,2770 | 1,2760 |
| T5 | 5 | 1,3370 | 1,3400 | 1,3310 | 1,2730 | 1,2730 | 1,2720 |
| T6 | 5 | 1,3800 | 1,3740 | 1,3820 | 1,3170 | 1,3120 | 1,3170 |
| T7 | 5 | 1,3150 | 1,3200 | 1,3120 | 1,2740 | 1,2760 | 1,2730 |

**6 lentelė**

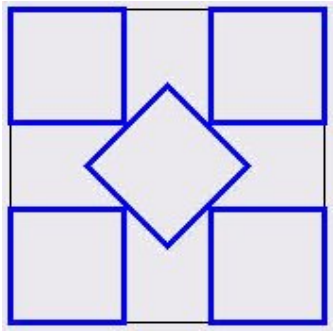
**Algoritmų vidutinių efektyvumo lygių lentelė su Hopper and Turton (2002) testo duomenimis**

| Duomenų rinkinys | Rinkinių skaičius | BFDH | BFDHIW | BFDHDW | SF | ASS | FCNR | KP01 |
|---|---|---|---|---|---|---|---|---|
| | | PR | PR | PR | PR | PR | PR | PR |
| T1 | 5 | 1,4650 | 1,4650 | 1,4650 | 1,4860 | 1,4340 | 1,3660 | 1,4650 |
| T2 | 5 | 1,4080 | 1,4080 | 1,4080 | 1,4480 | 1,4090 | 1,3200 | 1,4390 |
| T3 | 5 | 1,4060 | 1,4060 | 1,4070 | 1,4350 | 1,3620 | 1,2650 | 1,4200 |
| T4 | 5 | 1,2750 | 1,2770 | 1,2760 | 1,3020 | 1,2500 | 1,1840 | 1,2940 |
| T5 | 5 | 1,2720 | 1,2720 | 1,2720 | 1,3090 | 1,2010 | 1,1480 | 1,2870 |
| T6 | 5 | 1,3170 | 1,3170 | 1,3160 | 1,3280 | 1,1680 | 1,1800 | 1,3340 |
| T7 | 5 | 1,2740 | 1,2760 | 1,2730 | 1,3120 | 1,1260 | ? | 1,2790 |

## 4. Siūlomos euristikos dviejų dimensijų pakavimo problemai spręsti

Pagrindinė sprendžiamo uždavinio sąlyga yra tokia, kad turimą tam tikros rūšies plotą reikia užpildyti duotais įvairių dydžių stačiakampiais. Pavyzdžiui, turime kelis lapus faneros, nuo kurios reikia išpjauti įvairių dydžių ruošinius. Tikslas - išpjauti dalis ir kaip galima mažiau sunaudoti faneros. Jei pjautume 1x8 išmatavimo figūras nuo 4x8 faneros lapų - uždavinys būtų parastas, bet kai pjaunamos formos nėra reguliarios arba pagrindas nėra pjaunamo stačiakampio kartotinis - problema tampa apgaulingai sudėtinga ir turi didžiulį galimų variantų skaičių. Kai kurie iš galimų variantų:

- Leidžiama pasirinktinai sukti dalis taip padidinant galimas padėtis ir orientacijas iki begalinio skaičiaus. 10 paveikslas pavaizduoja penkis kvadratus padėtus ant didesnio kvadrato. Centrinis kvadratas netiks, jei jis nebus pasuktas kaip parodyta.
- Leidžiama sukti dalis, bet tiktai apibrėžtai, pavyzdžiui, 90 laipsnių kartotiniu.
- Nuopjovos gali būti įgaubtos. Pavyzdžiui, pusmėnulio formos gali būti įdėtos viena kitame, kad užimtų mažiau ploto.
- Leidžiama mažinti inventorių, naudojama tiktai giljotininiuose pjovimuose.



Šaltinis: sudarytas autoriaus

**10 pav. Kvadratų išdėstymas su pasukimu**

Sprendžiamas uždavinys yra sudėtingas dviejuose matmenyse, bet didesnė problema susidaro tada, kai nagrinėjamas trimatis atvejis. Kad sumažinti pakuojančių daiktų problemą, ją reikia padaryti kuo labiau valdomą, t.y. kuo labiau susiprastinti uždavinį ir parinkti tinkamą euristiką.

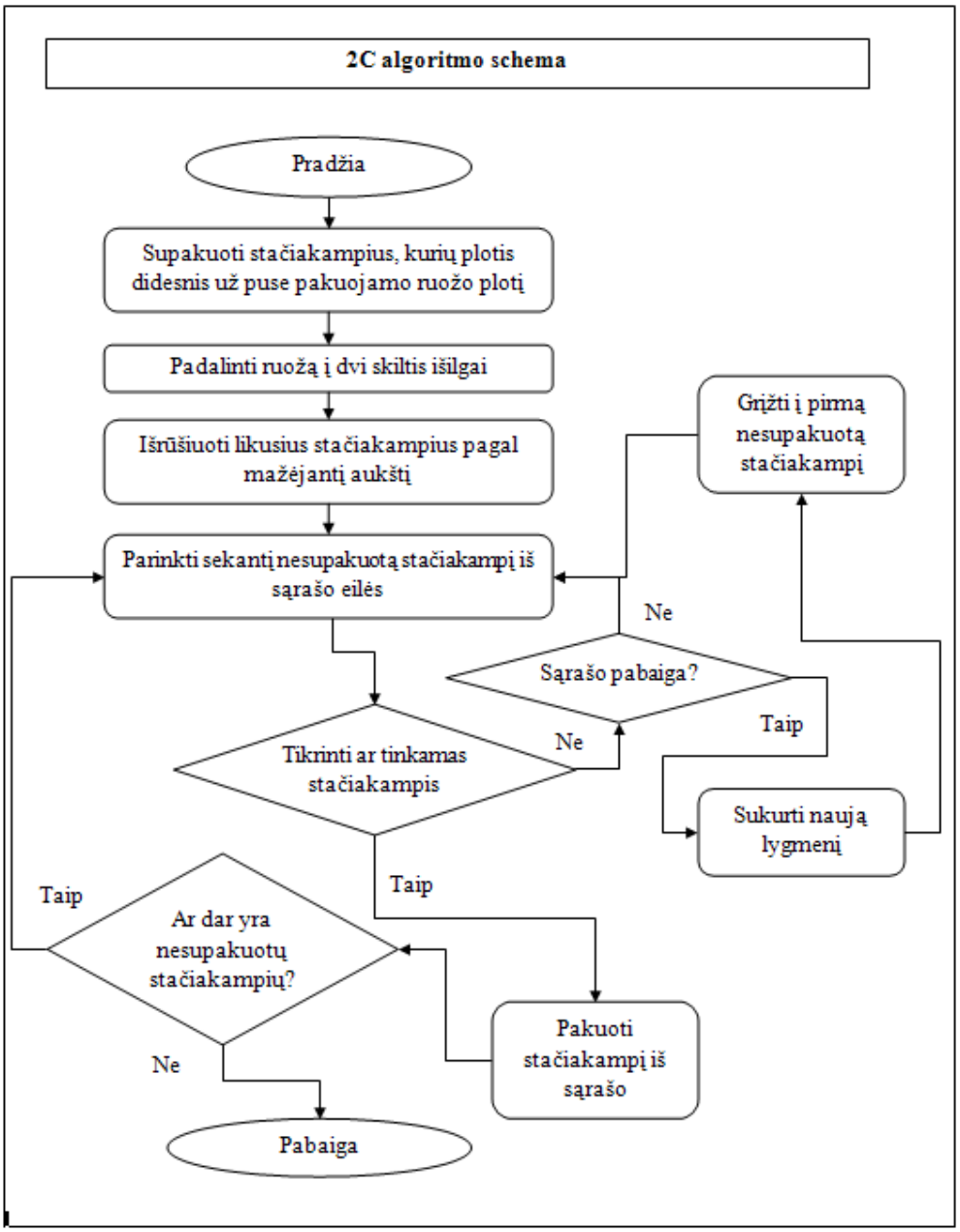
Pabandysim realizuoti keletą metodų tam, kad išspręstume dviejų dimensijų giljotininį pjaustymo uždavinį, naudodami šias prielaidas:

- Stačiakampių matmenys priklauso sveikų skaičių aibei.
- Negalima sukioti jokių stačiakampių.
- Pjaunamas pagrindas turi plotį, bet yra be galo ilgas. Tai reikia įsivaizduoti kaip idealizuotą kokio nors audeklo ar popieriaus ritinį.

Vienas iš būdų spręsti šiai problemai, tai panaudoti pilną parinkimą, t.y. nustatyti visas įmanomas pjaunamų objektų padėtis, kur jie galėtų būti. Bet tai yra labai sudėtinga, nes galimų leistinų kombinacijų skaičius priklauso nuo pjaunamo pagrindo ir pjaunamų stačiakampių dydžių. Pavyzdžiui, pagrindas yra 10 metrų ilgio ir 10 metrų pločio bei turime 8 ruošinius su 10 kartų mažesniais išmatavimais. Tada ruošinį galime padalinti į 10 * 20 = 200 kvadratų, kur teoriškai galima padėti kiekvieną iš 8 formų. Gausim 200 galimų padėčių pirmai formai, 199 - antrai, 198 - trečiai ir taip toliau. Kombinacijų bendras skaičius sieks 200 * 199 * 198 * ... * (200 - 7) ~= $22 * 10^{18}$. Jei būtų įmanoma įvertinti vieną milijoną kombinacijų per sekundę, tai mums prireiktų daugiau kaip 70 000 metų tam, kad patikrintume visus variantus. Todėl šis sprendimas buvo atmestas.

### 4.1. Dviejų stulpelių euristika

Vienas iš pasirinktų metodų yra dviejų stulpelių euristika. Iš pradžių šis metodas supakuoja bet kokias formas, kurios yra platesnės negu pusė pjaustomo ruožo plotis ir deda juos viršuje - taip suformuojama tiek ruožo lygmenų, kiek yra padėtų stačiakampių. Paskui algoritmas dalo ruožą į dvi - lygiai matuojamas vertikalias skiltis bei surūšiuoja likusias formas mažėjančio aukščio tvarka. Toliau dedamas aukščiausias stačiakampis pirmoje skiltyje ir taip nusakomas sekančio lygmens aukštis šioje skiltyje. Tokiu būdu stačiakampiai dedami iš kairės į dešinę kol telpa vertikaliai. Kai nebetelpa, tai sukuriamas naujas lygmuo antroje skiltyje ir pakuojami stačiakampiai joje. Kai užpildyta ir ši pusė, tai pakavimas persikelia į pirmą skilti ir taip tęsiama kol baigiasi pakuojam stačiakampiai. Šis algoritmas nusako koks maksimalus turi būti ruožo aukštis. Įrodyta, kad $H_{alg} <= 2 * H_{opt} + H_{tall} / 2$, kur $H_{alg}$ – algoritmo gautas aukštis, $H_{opt}$ – optimalus sprendimas, $H_{tall}$ – aukščiausio stačiakampio aukštis. Problema su dviejų stulpelių euristikos yra ta, kad du gretimi ruožai kartu gali turėti pakankamai daug tuščios erdvės. Šis metodas parodo sąsaja su optimaliu sprendimu, bet nebūtinai jis bus geriausias.[5]

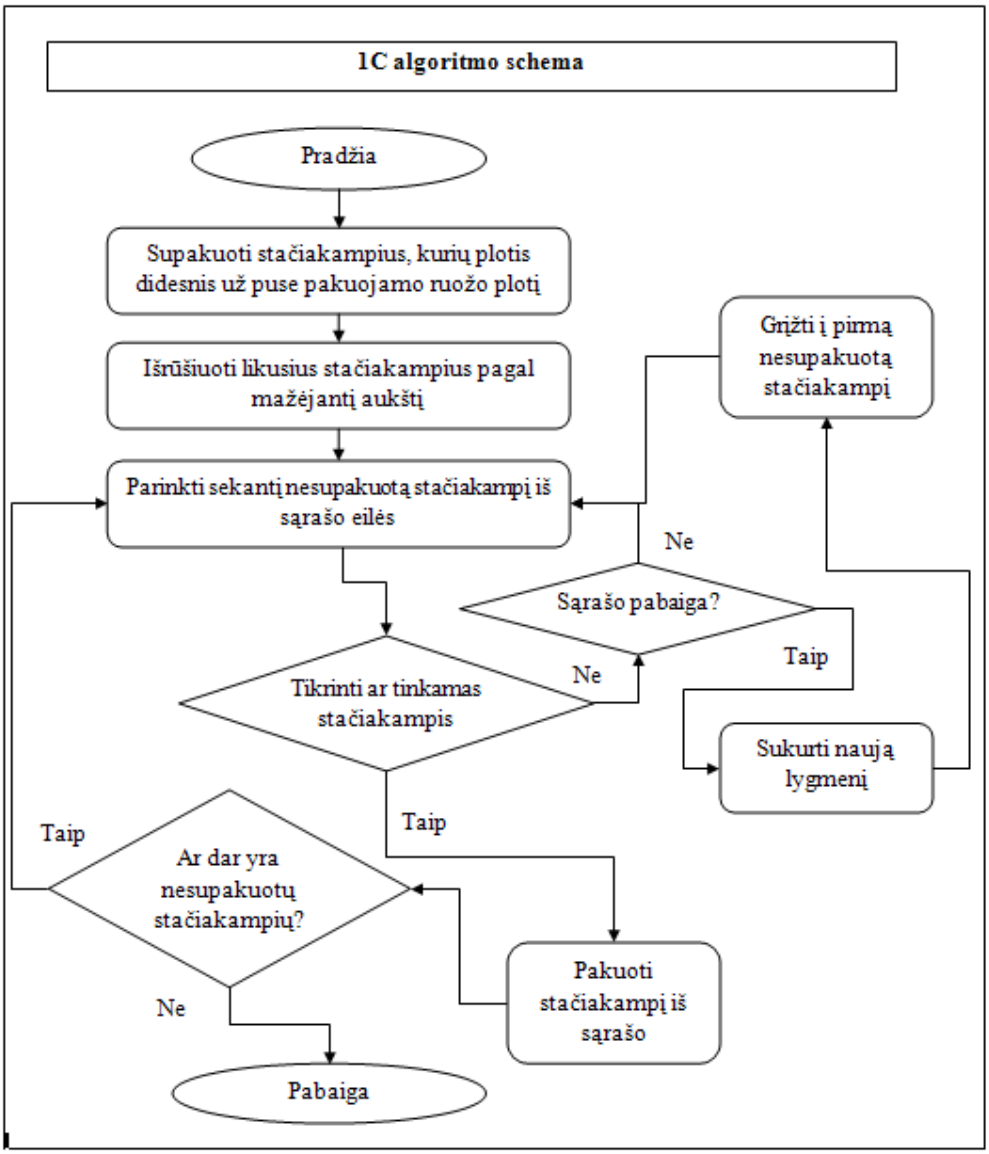



Šaltinis: sudarytas autoriaus

**11 pav. 2C algoritmo schema**

## 4.2. Vieno stulpelio euristika

Vieno stulpelio euristika veikia taip pat kaip dviejų skilčių euristika. Tačiau, užuot dalę ruožą į dvi vertikalias skiltis, tai daro vienoje skiltyje. Pirmiausiai stačiakampiai, kurių plotis yra didesnis negu pusė ruožo pločio yra sukraunami viršuje. Tada likęs objektų sąrašas, kurie turėtų būti supakuoti, išrūšiuojami pagal nedidėjantį aukštį. Tai reiškia, kad pirmas stačiakampis padėtas konkrečiame lygmenyje nustato sekančio lygmens aukštį. Jei stačiakampis padėtas einamajame lygyje ir telpa į plotį – tai viskas gerai. O jei netelpa (netinkamas), tai sukuriamas naujas lygmuo, žemiau esamo (kuris tampa einamuoju lygmeniu) ir stačiakampis dedamas jame. Taip pakavimo procesas tęsiasi iš kairės į dešinę ir žemyn (galima vertikaliai aukštyn) pagal lygius kol nebelieka pakuojamų stačiakampių. Jau suformuoti lygmenys, esantys žemiau esamo (arba aukščiau), niekada neperžiūrimi.



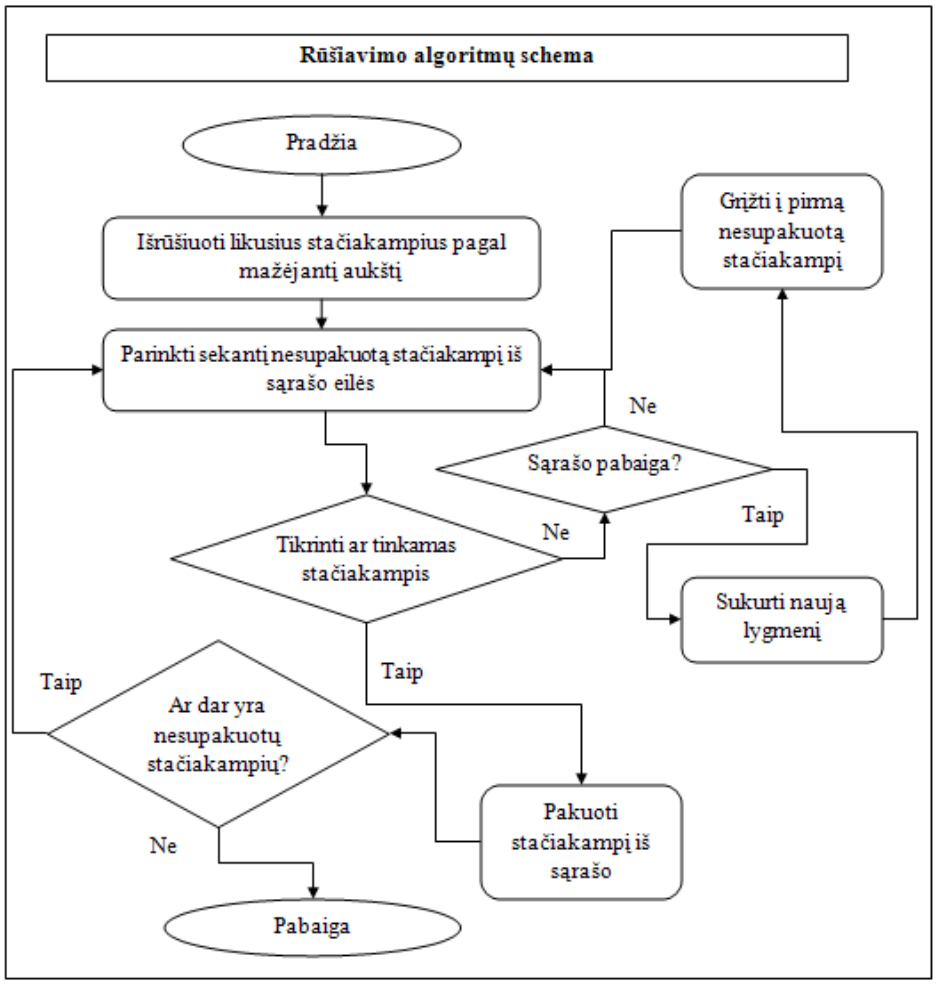

Šaltinis: sudarytas autoriaus

**12 pav. 1C algoritmo schema**

### 4.3. Rūšiavimo euristikos

Rūšiavimo euristika - tai papildymas vieno stulpelio euristikos. Norint pagerinti jau gautus rezultatus – bandomą įvairiais būdais išrūšiuoti pjaustinius. Toliau stačiakampiai pakuojami pradedant nuo viršaus iš kairės pusės dedami į dešinę tol, kol telpą per plotį. Kai nebetelpa, tai ieškomas sekantis stačiakampis, kuris dar įsitektų į likusį juostos plotį. Kai nebetelpa, tai kuriamas naujas lygmuo ir pjaustiniai dedami jame. Taip pakavimas vyksta kol išdėliojami visi stačiakampiai.

Rūšiavimo pagal aukštį euristika gana gerai užpildo tuščias likusias erdves palyginus su prieš tai buvusiais metodais, bet vis tiek dar negarantuoja optimalaus. Rūšiavimas pagal plotį išdėsto stačiakampius priešingai negu horizontalus ruožo judėjimas. Kai išrūšiuojama pagal plotą, tai pakuojama nuo didžiausio, bet tokie stačiakampiai kuria tuščius plotus, kurie vėliau sunkiai visiškai prisipildo, todėl dar pradžioje mažesniais stačiakampiais bandomą tokias erdves užpildyti. Rūšiavimas pagal kvadratingumą išrūšiuoja stačiakampius į eilę taip, kad pirmesni yra tie, kurie yra labiausiai panašūs į kvadratą.




Šaltinis: sudarytas autoriaus

**13 pav. Rūšiavimo algoritmų schema**

## 5. Pasiūlytų euristikų palyginimai

Tyrimui atlikti buvo sukurta speciali programa. Ji nuskaito etaloninių rinkinių duomenų failus xls formate. Realizuotos šešios euristikos – dviejų stulpelių euristika(2C), vieno stulpelio euristika(1C), rūšiavimas pagal aukštį(SH), rūšiavimas pagal plotį(SW), rūšiavimas pagal plotą(SS) ir rūšiavimas pagal stačiakampių kvadrantingumą(SSN). Eksperimentui buvo pasirinkti septynių skirtingų klasių Hopper and Turton testiniai duomenys (2002). Jie ypatingi tuo, kad yra žinomas galimas geriausias sprendinys – 200 vienetų aukštis, esant 200 vienetų pločiui ruožas. Šie duomenys yra sugeneruoti iš 200 x 200 išmatavimų kvadrato atsitiktinai parenkant tašką stačiakampyje ir darant vertikalius bei horizontalius pjovimus per tą tašką, kad sukurtų keturis naujus stačiakampius.

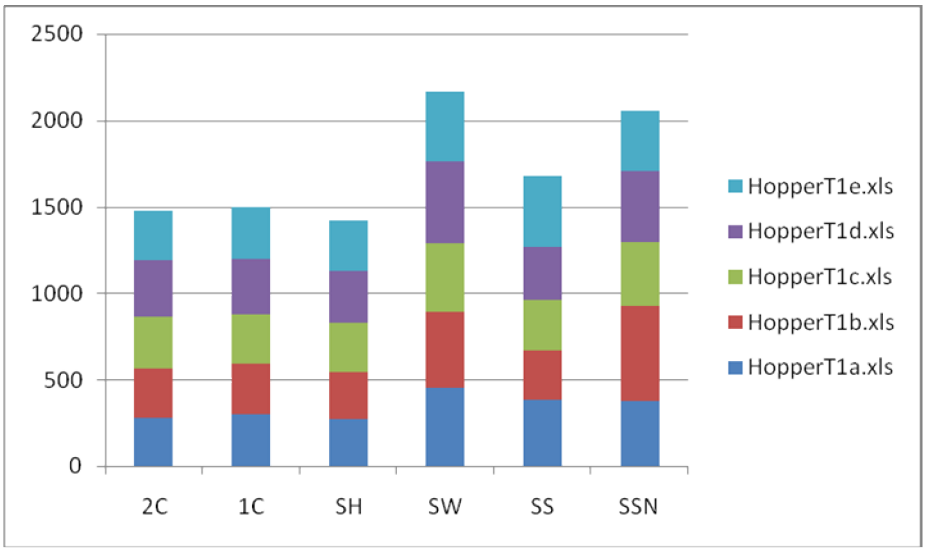
Eksperimento rezultatai, kai tiriamo ruožo plotis 200 vienetų:

Pirma duomenų klasė (17 pakuojamų stačiakampių):

**7 lentelė**

**Eksperimento rezultatai HopperT1 duomenų klasėje**

| | 2C | 1C | SH | SW | SS | SSN |
|---|---|---|---|---|---|---|
| HopperT1a.xls | 280 | 298 | 273 | 450 | 382 | 375 |
| HopperT1b.xls | 282 | 291 | 272 | 441 | 288 | 551 |
| HopperT1c.xls | 304 | 286 | 281 | 394 | 288 | 370 |
| HopperT1d.xls | 327 | 326 | 305 | 479 | 311 | 413 |
| HopperT1e.xls | 282 | 297 | 288 | 403 | 411 | 348 |



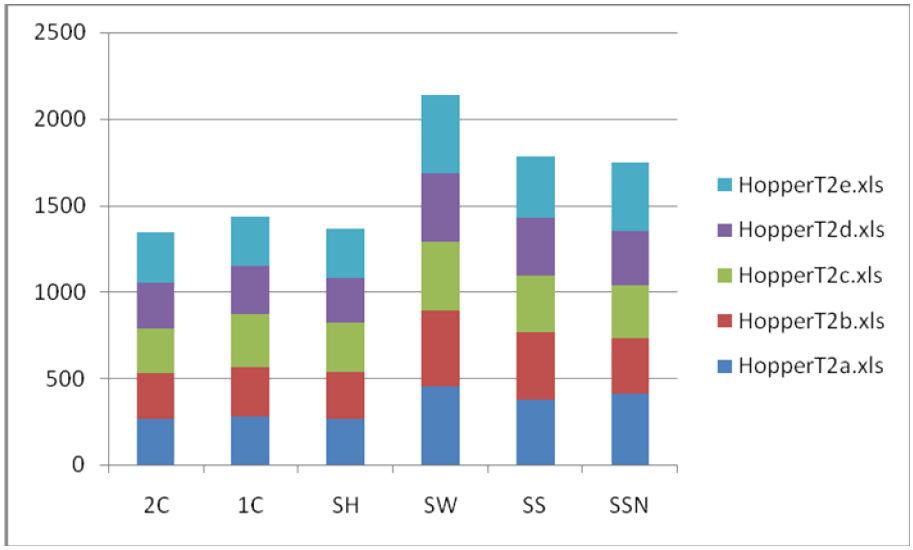

**14 pav. HopperT1 duomenų klasės rezultatų stulpelinė diagrama**

Antra duomenų klasė (25 pakuojamų stačiakampių):

**8 lentelė**

**Eksperimento rezultatai HopperT2 duomenų klasėje**

| | 2C | 1C | SH | SW | SS | SSN |
|---|---|---|---|---|---|---|
| HopperT2a.xls | 265 | 277 | 267 | 456 | 379 | 414 |
| HopperT2b.xls | 264 | 286 | 269 | 435 | 388 | 320 |
| HopperT2c.xls | 260 | 305 | 285 | 394 | 329 | 303 |
| HopperT2d.xls | 264 | 278 | 260 | 399 | 329 | 314 |
| HopperT2e.xls | 291 | 289 | 281 | 457 | 356 | 398 |



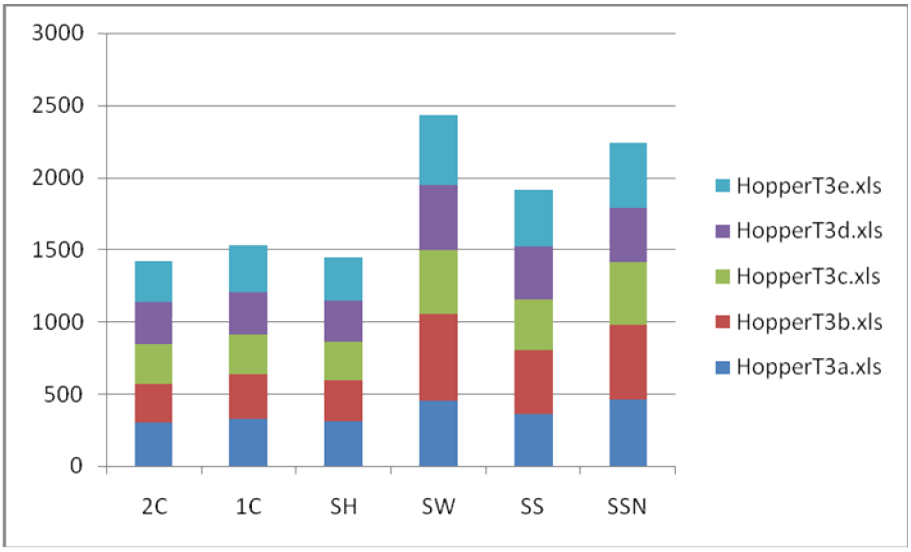

**15 pav. HopperT2 duomenų klasės rezultatų stulpelinė diagrama**

Trečia duomenų klasė (29 pakuojamų stačiakampių):

**9 lentelė**

**Eksperimento rezultatai HopperT3 duomenų klasėje**

| | 2C | 1C | SH | SW | SS | SSN |
|---|---|---|---|---|---|---|
| HopperT3a.xls | 303 | 326 | 307 | 448 | 361 | 462 |
| HopperT3b.xls | 265 | 305 | 289 | 606 | 445 | 514 |
| HopperT3c.xls | 279 | 276 | 264 | 440 | 349 | 433 |
| HopperT3d.xls | 289 | 294 | 281 | 457 | 362 | 383 |
| HopperT3e.xls | 285 | 325 | 305 | 481 | 393 | 446 |



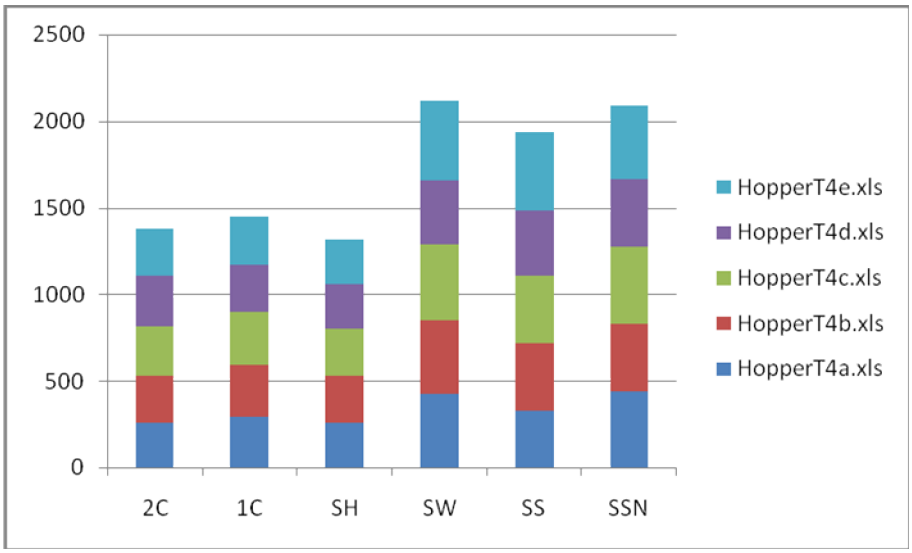

**16 pav. HopperT3 duomenų klasės rezultatų stulpelinė diagrama**

Ketvirta duomenų klasė (49 pakuojamų stačiakampių):

**10 lentelė**

**Eksperimento rezultatai HopperT4 duomenų klasėje**

| | 2C | 1C | SH | SW | SS | SSN |
|---|---|---|---|---|---|---|
| HopperT4a.xls | 260 | 289 | 257 | 423 | 328 | 438 |
| HopperT4b.xls | 267 | 304 | 272 | 430 | 390 | 394 |
| HopperT4c.xls | 290 | 305 | 270 | 433 | 392 | 444 |
| HopperT4d.xls | 288 | 270 | 259 | 374 | 376 | 386 |
| HopperT4e.xls | 274 | 278 | 258 | 456 | 447 | 425 |



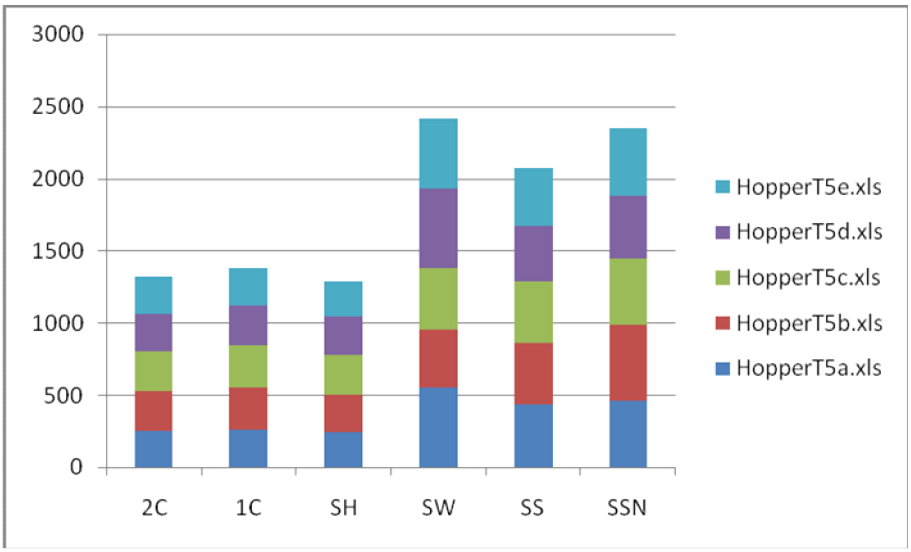

**17 pav. HopperT4 duomenų klasės rezultatų stulpelinė diagrama**

Penkta duomenų klasė (73 pakuojamų stačiakampių):

**11 lentelė**

**Eksperimento rezultatai HopperT5 duomenų klasėje**

| | 2C | 1C | SH | SW | SS | SSN |
|---|---|---|---|---|---|---|
| HopperT5a.xls | 252 | 255 | 245 | 548 | 433 | 460 |
| HopperT5b.xls | 273 | 298 | 257 | 404 | 429 | 522 |
| HopperT5c.xls | 275 | 290 | 275 | 429 | 429 | 464 |
| HopperT5d.xls | 262 | 276 | 267 | 551 | 377 | 438 |
| HopperT5e.xls | 255 | 257 | 247 | 480 | 401 | 462 |



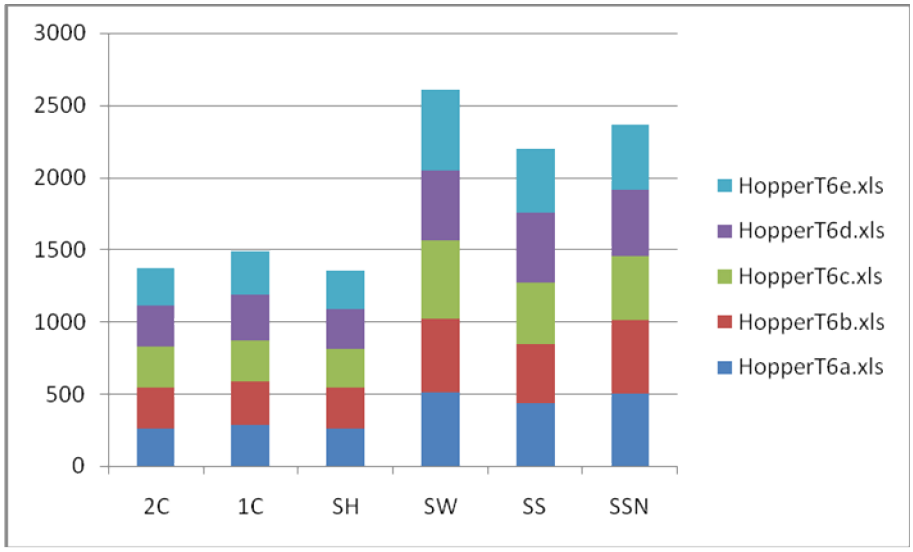

**18 pav. HopperT5 duomenų klasės rezultatų stulpelinė diagrama**

Šešta duomenų klasė (97 pakuojamų stačiakampių):

**12 lentelė**

**Eksperimento rezultatai HopperT6 duomenų klasėje**

| | 2C | 1C | SH | SW | SS | SSN |
|---|---|---|---|---|---|---|
| HopperT6a.xls | 257 | 288 | 262 | 513 | 438 | 503 |
| HopperT6b.xls | 288 | 294 | 280 | 504 | 403 | 505 |
| HopperT6c.xls | 284 | 284 | 266 | 543 | 429 | 446 |
| HopperT6d.xls | 279 | 319 | 277 | 485 | 489 | 459 |
| HopperT6e.xls | 266 | 300 | 268 | 563 | 441 | 452 |



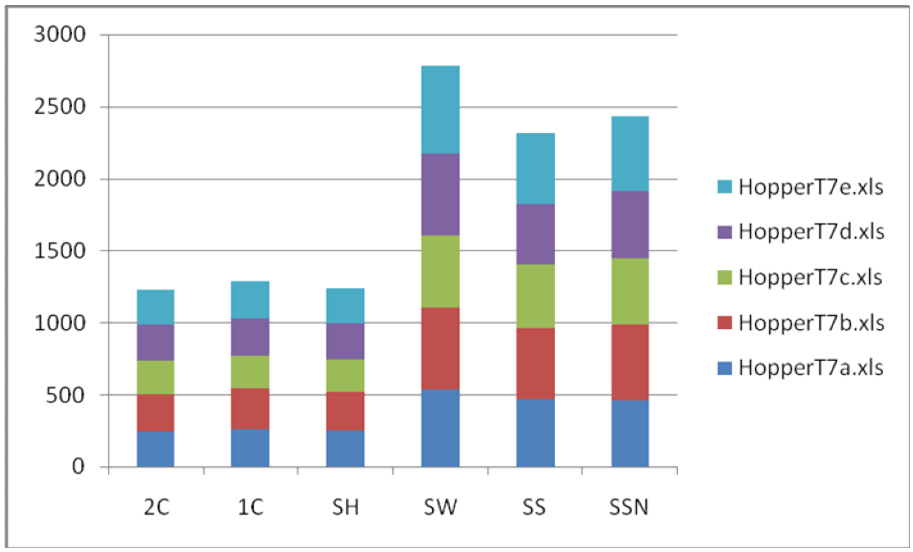

**19 pav. HopperT6 duomenų klasės rezultatų stulpelinė diagrama**

Septinta duomenų klasė (199 pakuojamų stačiakampių):

**13 lentelė**

**Eksperimento rezultatai HopperT7 duomenų klasėje**

| | 2C | 1C | SH | SW | SS | SSN |
|---|---|---|---|---|---|---|
| HopperT7a.xls | 246 | 262 | 249 | 535 | 466 | 458 |
| HopperT7b.xls | 256 | 278 | 266 | 565 | 496 | 529 |
| HopperT7c.xls | 237 | 232 | 229 | 505 | 440 | 462 |
| HopperT7d.xls | 250 | 260 | 250 | 565 | 420 | 462 |
| HopperT7e.xls | 242 | 256 | 241 | 614 | 490 | 520 |




**20 pav. HopperT7 duomenų klasės rezultatų stulpelinė diagrama**

Tyrimo rezultatų stulpelinėse diagramose matosi, kad pirmos trys euristikos pateikia žymiai geresnius rezultatus negu likusios. Ryškus autsaideris yra SW algoritmas, bet tai negarantuoja, kad atlikus testą su kitais duomenimis jis nepateiktų geresnių rezultatų negu kitos euristikos.

Buvo atliktas F-testas, kuris nusako ar gautų rezultatų imties dispersijos nėra reikšmingai skirtingos.

Gauti F-testo vidutinių aukščių rezultatai pagal duomenų klases:

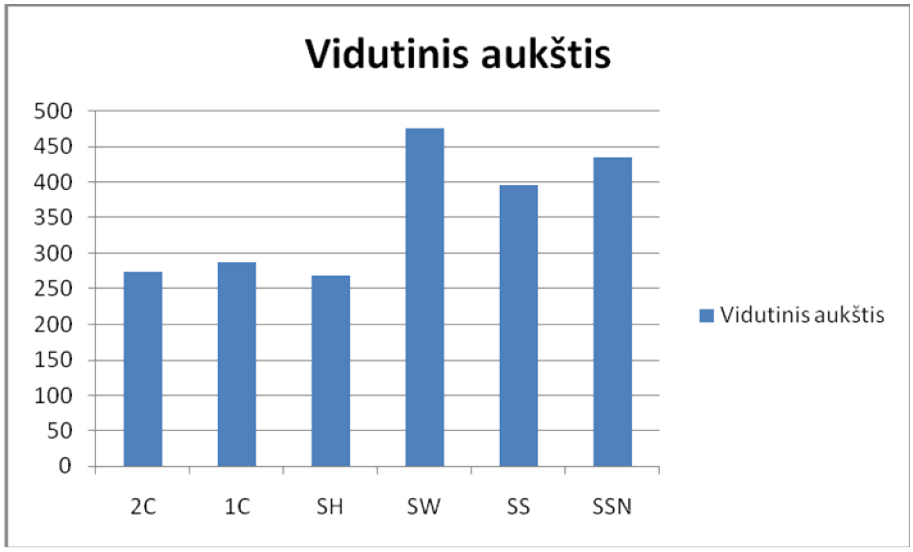
**14 lentelė**

**F-testo rezultatai**

| | 2C | 1C | SH | SW | SS | SSN | $F_{value}$ | $F_{critical}$ |
|---|---|---|---|---|---|---|---|---|
| HopperT1 | 295 | 300 | 284 | 433 | 336 | 411 | 10.34403 | 2.62 |
| HopperT2 | 269 | 287 | 272 | 428 | 356 | 350 | 24.66491 | 2.62 |
| HopperT3 | 284 | 305 | 289 | 486 | 382 | 448 | 24.00109 | 2.62 |
| HopperT4 | 276 | 289 | 263 | 423 | 387 | 417 | 42.95122 | 2.62 |
| HopperT5 | 263 | 275 | 258 | 482 | 414 | 469 | 50.65837 | 2.62 |
| HopperT6 | 275 | 297 | 271 | 522 | 440 | 473 | 117.2607 | 2.62 |
| HopperT7 | 246 | 258 | 247 | 557 | 462 | 486 | 134.9114 | 2.62 |

Dispersinės analizės rezultatai parodo, kad su visomis duomenų klasėmis yra reikšmingas skirtumas tarp gautų ruožo aukščių, nes visuomet apskaičiuota F reikšmė yra daugiau už teorinę F reikšmę ($F_{value} > F_{critical}$) su pasirinktu reikšmingumo lygmeniu $\alpha$=0.05. Taigi, skirtumas tarp gautų ruožo aukščių yra reikšmingas 5% reikšmės lygmenyje.

Taip pat gavome kiekvienos euristikos vidutinių aukščių stulpelinę diagramą (6.16 paveikslas).




**21 pav. Vidutinių aukščių stulpelinė diagrama**

Šiame grafike ties 200 vienetų aukštyje yra geriausias idealizuotas galimas sprendinys. Pirmos trys euristikos yra netoli jo. Kiti trys algoritmai apytiksliai 2 kartus viršija įsivaizduojama optimalų aukštį. Šiuo atveju – tai nėra geros euristikos.

Taip pat buvo atliktas chi-kvadrato testas tam, kad įvertintume ar yra reikšmingas statistinis skirtumas tarp dažnių, kurie gauti kiekvieno algoritmo kaip geriausias rezultatas.

Chi-kvadrato testo duomenys ir rezultatai:

**15 lentelė**

**Chi-kvadrato testo rezultatai**

| | 2C | 1C | SH | SW | SS | SSN | $\chi^2_{df}$ | $\chi_{critical}(0.05)$ |
|---|---|---|---|---|---|---|---|---|
| Kiek kartų geriausias | 15 | 0 | 22 | 0 | 0 | 0 | 91.13964 | 11.07 |

Kadangi apskaičiuotas chi-kvadratas yra didesnis už teorinį chi ($\chi^2_{df} > \chi_{critical}$) su pasirinktu reikšmingumo lygmeniu $\alpha=0.05$, tai reiškia, kad yra reikšmingas skirtumas tarp dažnių.


**22 pav. Kiek kartų ir kuris algoritmas parodė geriausią rezultatą**

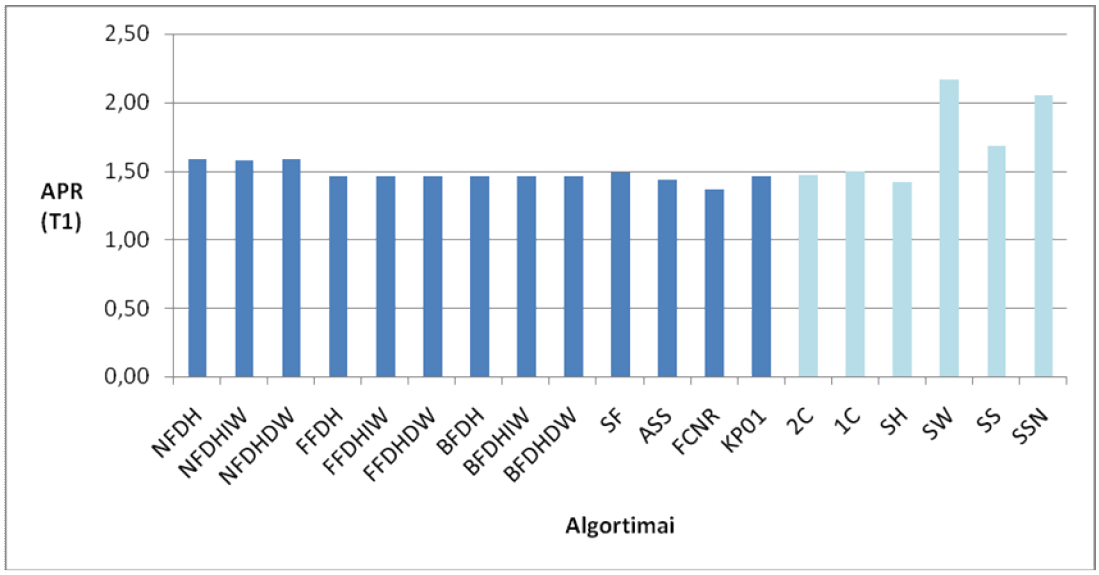
Iš chi-kvadrato testo sprendžiame, kad SH algoritmas yra geresnis už kitus. Antroje vietoje liko 2C euristika.

Paskaičiuoti pateiktų euristinių algoritmų vidutiniai efektyvumo lygiai *(Average Performance ratios)* APR(A) gautų su Hopper and Turton (2002) testiniais duomenimis (6.19 lentelė). Taip pat pateiktos pasiūlytų bei apžvelgtų 4 skyriuje euristikų rezultatų stulpelinės diagramos (23-29 paveiksliukai).

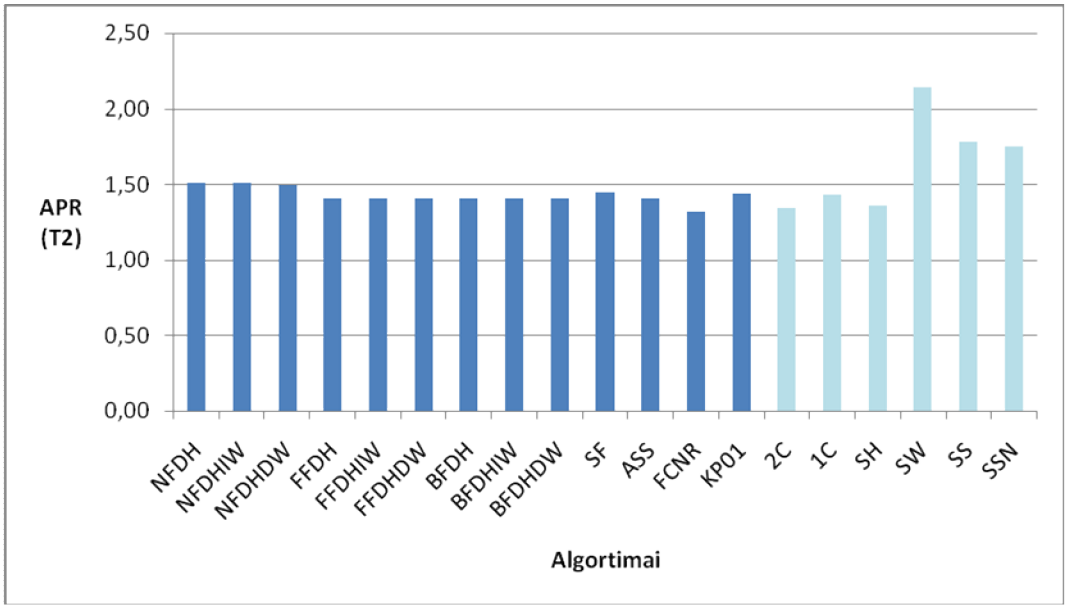
**16 lentelė**

**Algoritmų vidutinis efektyvumo lygis**

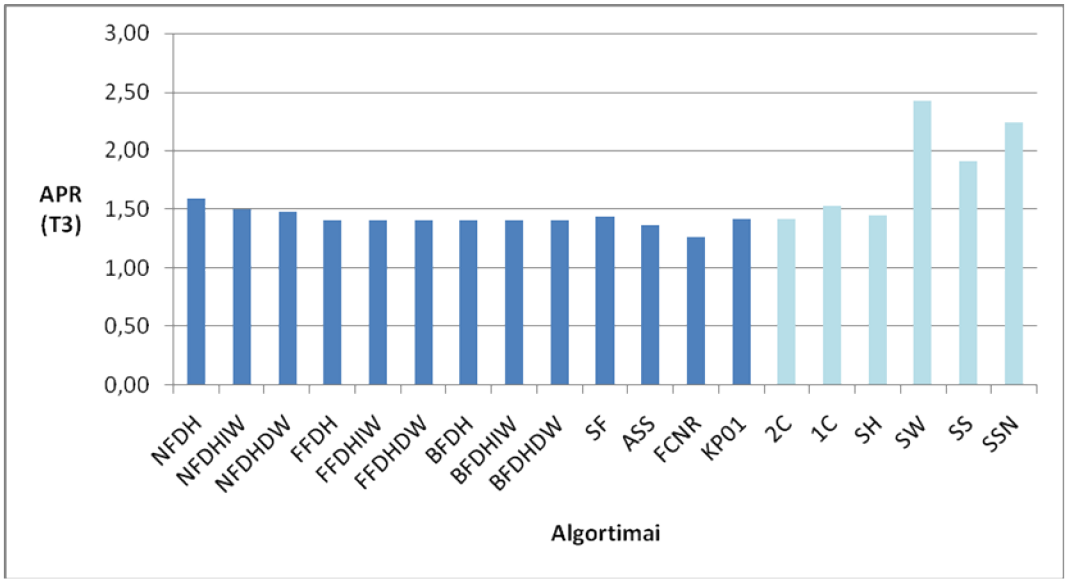
| Duomenų rinkinys | Rinkinių skaičius | 2C | 1C | SH | SW | SS | SSN |
|---|---|---|---|---|---|---|---|
| | | PR | PR | PR | PR | PR | PR |
| T1 | 5 | 1,475 | 1,5 | 1,42 | 2,165 | 1,68 | 2,055 |
| T2 | 5 | 1,345 | 1,435 | 1,36 | 2,14 | 1,78 | 1,75 |
| T3 | 5 | 1,42 | 1,525 | 1,445 | 2,43 | 1,91 | 2,24 |
| T4 | 5 | 1,38 | 1,445 | 1,315 | 2,115 | 1,935 | 2,085 |
| T5 | 5 | 1,315 | 1,375 | 1,29 | 2,41 | 2,07 | 2,345 |
| T6 | 5 | 1,375 | 1,485 | 1,355 | 2,61 | 2,2 | 2,365 |
| T7 | 5 | 1,23 | 1,29 | 1,235 | 2,785 | 2,31 | 2,43 |



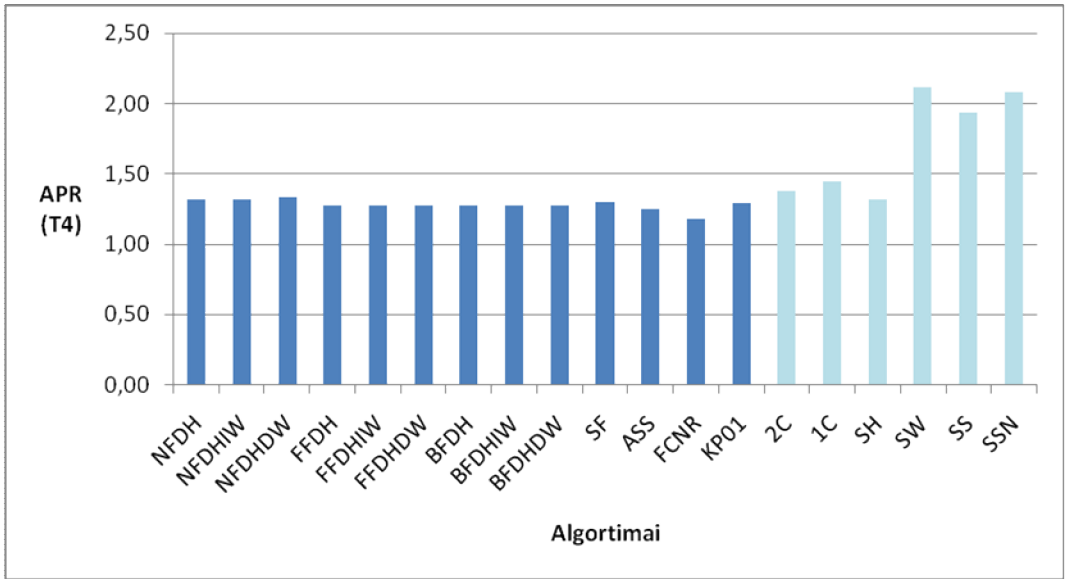

**23 pav. Algoritmų vidutinis efektyvumo lygis T1 duomenų klasėje.**



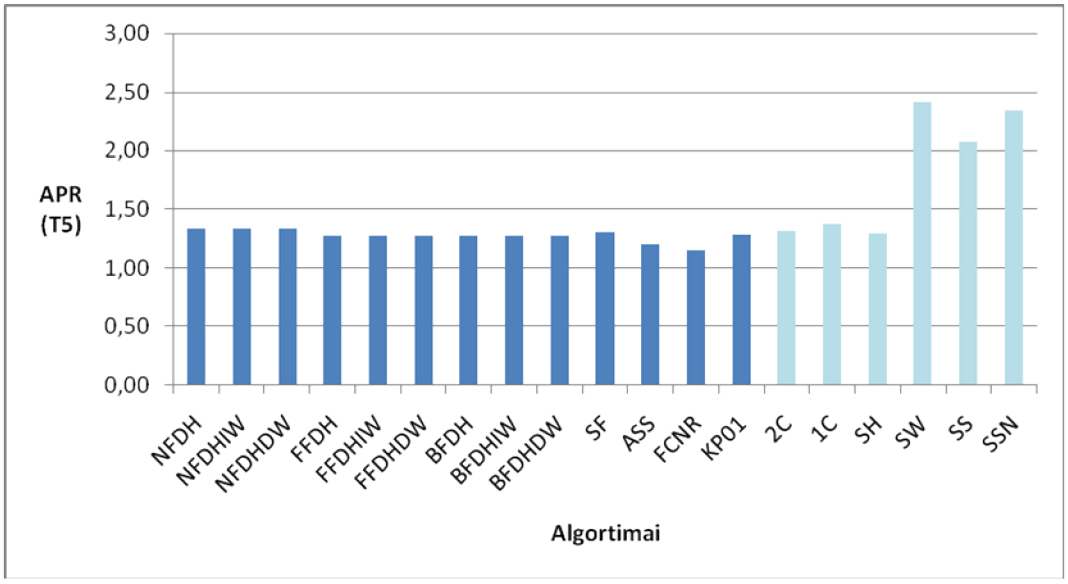

**24 pav. Algoritmų vidutinis efektyvumo lygis T2 duomenų klasėje.**



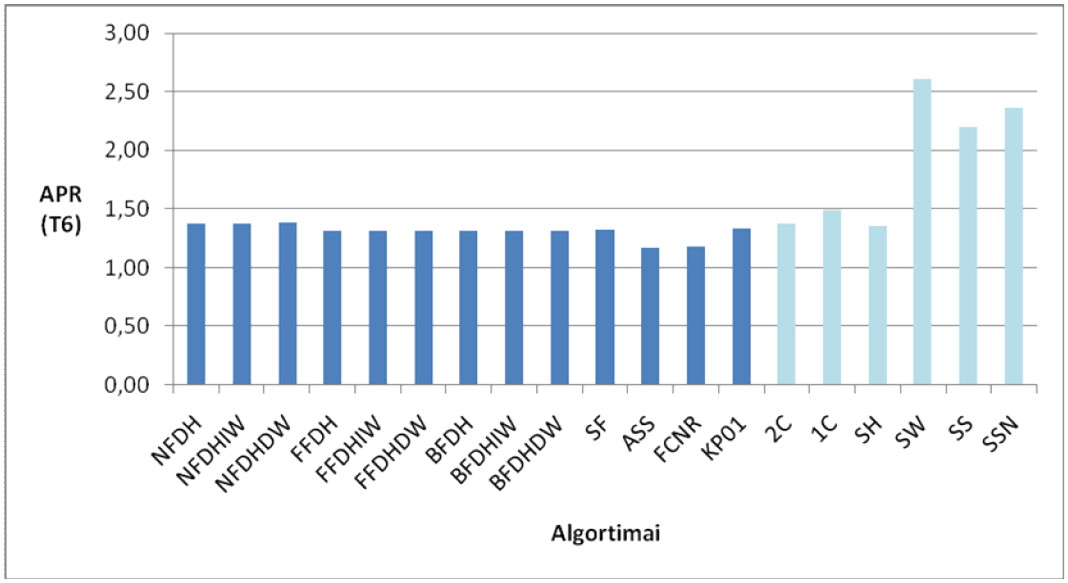

**25 pav. Algoritmų vidutinis efektyvumo lygis T3 duomenų klasėje.**



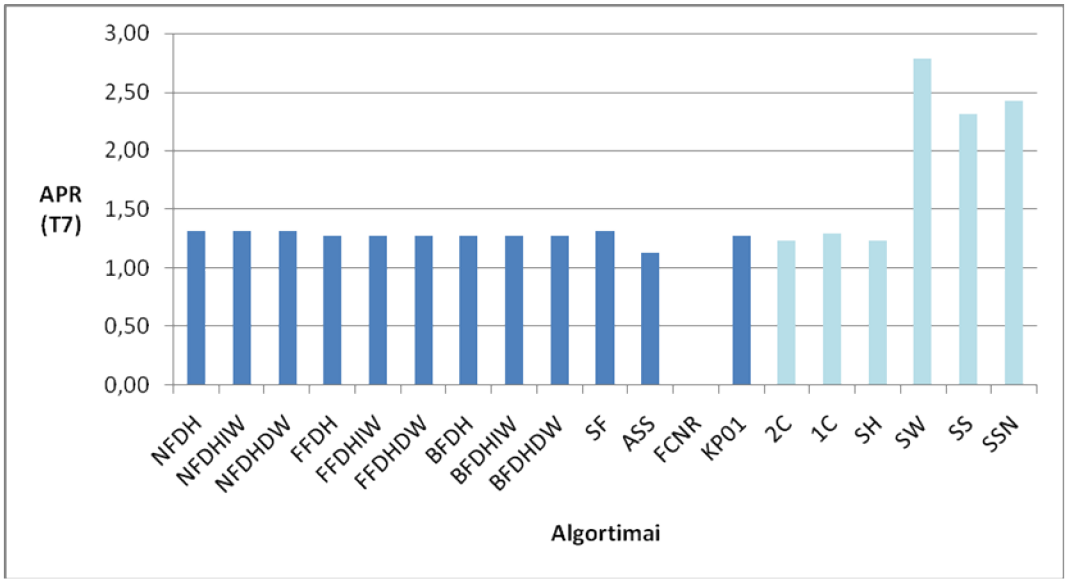

**26 pav. Algoritmų vidutinis efektyvumo lygis T4 duomenų klasėje.**




**27 pav. Algoritmų vidutinis efektyvumo lygis T5 duomenų klasėje.**




**28 pav. Algoritmų vidutinis efektyvumo lygis T6 duomenų klasėje.**




**29 pav. Algoritmų vidutinis efektyvumo lygis T7 duomenų klasėje.**

# IŠVADOS IR REKOMENDACIJOS

- Pasiūlytų euristikų veikimo laikas yra trumpesnis nei aprašytų.
- Dispersinės analizės rezultatai parodė, kad yra reikšmingas skirtumas tarp vidutinių ruožo aukščių ir dažnių, kurie gauti kiekvieno pasiūlyto metodo kaip geriausias rezultatas, 5% reikšmės lygmenyje.
- Dažniausiai naudojamų (aprašytų 3 skyriuje) ir pasiūlytų euristikų rezultatai buvo palyginti pagal vidutinį efektyvumo lygį. Algoritmai SW, SS ir SSN eksperimento metu neparodė gerų rezultatų, nes pagal efektyvumą jie atsiliko nuo visų kitų metodų. 2C, 1C ir SW euristikos pateikė artimus sprendimus dažniausiai naudojamoms euristikoms.
- Didžiausia problema išliko nepanaudotų plotų supjaustymas, todėl šioje vietoje reikėtų pratęsti tyrimus.

## LITERATŪRA


- [1] Hopper E., Turton B.C.H. An empirical investigation of meta-heuristic and heuristic algorithms for 2D packing problem // European Journal of Operational Research – 2001, Nr. 128, p. 34 – 57.
- [2] Mockus J. A set of examples of global and discrete optimization 2, p. 43.
- [3] Diskretusis optimizavimas, prieiga internete <http://mockus.org/optimum/>, peržiūros data 2007-06-21.
- [4] N. Ntene, J.H. van Vuuren A survey and comparison of level heuristics for the 2D oriented strip packing problem // Department of Applied Mathematics, University of Stellenbosch, Private Bag X1, Matieland, 7602, Republic of South Africa, p. 3-20.
- [5] 2D Strip Packing Problem (SPP), prieiga internete <http://dip.sun.ac.za/~vuuren/repositories/levelpaper/spp%5B1%5D.htm>, peržiūros data 2008-05-02.
- [6] Exact algorithms for the guillotine strip cutting/packing problem, prieiga internete <http://www.citeulike.org/tag/guillotine>, peržiūros data 2008-05-02.
- [7] The Two Dimensional Cutting Stock Problem, prieiga internete <http://www.informatik.uni-osnabrueck.de/papers_html/or_94/cutpaper.html>, peržiūros data (2005-05-03)
- [8] J. Mockus Theory of games and markets with examples, p. 71-72.
- [9] Statistics calculators, prieiga internete <http://www.danielsoper.com/>, peržiūros data 2008-05-11.
- [10] Andrea Lodi, Silvano Martello, Daniele Vigo Approximation algorithms for the oriented two-dimensional bin packing problem // European Journal of Operational Research 112 (1999) 158-166, 1997m., p. 1-9.
- [11] Christoph Nitsche, Guntram Scheithauer *, Johannes Terno Tighter relaxations for the cutting stock problem // European Journal of Operational Research 112 (1999) 654-663, 1997m., p. 1-10.
- [12] Francesco Conti a, Federico Malucelli b, Sara Nicoloso c, Bruno Simeone On a 2-dimensional equipartition problem // European Journal of Operational Research 113 (1999) 215-231 1997m., p. 1-17.
- [13] Chi-square test, prieiga internete <http://en.wikipedia.org/wiki/Chisquare_test#Chisquare_test_for_contingency_table_example>, peržiūros data 2008-05-02.

## SANTRAUKA ANGLŲ KALBA

Ragaišis, Vaidotas. (2008) Solution and analysis of two dimensions guillotinable cut problem. Master of Science graduation paper. Kaunas university of technology, department of informatics.

# SUMMARY

The theme of the Master of Science degree paper is “Solution and analysis of two dimensions guillotinable cut problem”. The main goal in this paper is solution and analysis of two dimensions guillotinable cut problem.

During the period of implementation we implemented a number of level heuristics from the literature and proposed possible improvements to some of these algorithms. 6 algorithms were compared in terms of their solution qualities and their ability to obtain the smallest strip height. The results of the analyses of variance indicate that statistically, there is difference between the mean strip heights obtained by the algorithms at a 5% level of significance. The algorithms were also compared in terms of how close the strip heights obtained were to the optimal solution