KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

Monika Levickytė

KONTEINERIO UŽPILDYMO UŽDAVINIO SPRENDIMUI TAIKOMOS EURISTIKOS TYRIMAS

Magistro darbas

Vadovas
doc. dr. N. Listopadskis

KAUNAS, 2012

KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. N.Listopadskis
2012 06 02

KONTEINERIO UŽPILDYMO UŽDAVINIO SPRENDIMUI TAIKOMOS EURISTIKOS TYRIMAS

Taikomosios matematikos magistro baigiamasis darbas

Vadovas
doc. dr. N. Listopadskis
2012 06 01

Recenzentas
doc. dr. T. Blažauskas
2012 06 01

Atliko
FMMM 0 gr. stud.
M. Levickytė
2012 05 30

KAUNAS, 2012

KVALIFIKACINĖ KOMISIJA

Pirmininkas: Rimantas Rudzkis, profesorius (VU MII)

Sekretorius: Eimutis Valakevičius, docentas (KTU)

Nariai: Jonas Valantinas, profesorius (KTU)

Vytautas Janilionis, docentas (KTU)

Vidmantas Povilas Pekarskas, profesorius (KTU)

Zenonas Navickas, profesorius (KTU)

Arūnas Barauskas, dr., vice-prezidentas projektams (UAB „Baltic Amadeus“)

Levickytė M. Konteinerio užpildymo uždavinio sprendimui taikomos euristikos tyrimas: taikomosios matematikos magistro darbas / darbo vadovas doc. dr. N. Listopadskis; taikomosios matematikos katedra, fundamentaliųjų mokslų fakultetas, Kauno technologijos universitetas. – Kaunas, 2012. – 77 p.

SANTRAUKA

Šiame darbe nagrinėjamas kombinatorinio optimizavimo uždavinys yra specialus trimatis pakavimo uždavinys, itin dažnai atsirandantis logistikos ir pervežimų pramonėje. Šio uždavinio atveju yra bandoma vieną stačiakampio gretasienio formos fiksuotų dimensijų konteinerį užpildyti tam tikru duotos dėžių aibės poaibiu, panaudojant ortogonalų pakavimą. Dažniausiai šis uždavinys yra vadinamas konteinerio užpildymo problema (trumpiau CLP).

Dėl šio uždavinio kombinatorinio sudėtingumo optimalaus sprendinio suradimas yra įmanomas tik mažos apimties uždavinių atveju. Šiame darbe buvo atliktas CLP sprendimui naudojamos euristikos taikymo galimybių tyrimas, sprendžiant atsitiktinai sugeneruotus pakavimo uždavinius.

Eksperimentų rezultatai parodė, kad nagrinėjamos euristikos veikimas labai priklauso nuo apibrėžtos rangavimo taisyklės, taikomos parinkti skaičių M_3 galimų sluosnio storių ir skaičių M_2 galimų juostos storių. Darbe buvo tiriamos dvidešimt keturios skirtingos rangavimo taisyklės, kurios buvo palyginamos pagal gautą konteinerio užpildymą, pakavimo laiką bei pagal keletą kitų rodiklių. Taip pat buvo atliktas konteinerio užpildymo ir pakavimo laiko priklausomybės nuo parinktos parametrų M_3 ir M_2 kombinacijos bei nuo pakuojamų dėžių kiekio tyrimas. Euristikos veikimas taip pat buvo eksperimentiškai tiriamas ir palyginamas pagal krovinio tipą, t.y., išskiriant vienarūšį, stipriai ir silpnai įvairiarūšį krovinio asortimentą.

Atlikto tyrimo rezultatai buvo pristatyti keliose konferencijose (10-oji studentų konferencija „Taikomoji matematika“, 2012; Matematika ir matematinis modeliavimas, 2012). Taip pat šios temos pagrindu buvo išleistas straipsnis [8], o dar vienas straipsnis [7] - pateiktas spaudai.

Levickytė M. An analysis of heuristic for the container loading problem: Master‘s work in applied mathematics / supervisor dr. assoc. prof. N. Listopadskis; Department of Applied mathematics, Faculty of Fundamental Sciences, Kaunas University of Technology. – Kaunas, 2012. – 77 p.

SUMMARY

The combinatorial optimization problem considered in this paper is a special three-dimensional packing problem arising especially in the logistics and transportation industries. This problem is that of orthogonally packing a subset of some given rectangular-shaped boxes into a rectangular container of fixed dimensions. It is generally called a container loading problem (CLP).

Due to the combinatorial complexity of this problem, it seems impossible to solve it optimally for every instance. For this reason, the experimental behaviour of special heuristic on sets of randomly generated test problems has been analysed.

Computational experiments showed that the performance of heuristic algorithm strictly depends on the ranking rules, which are used to select a number M_3 of layer depths and a number M_2 of strip widths/heights. Twenty-four different ranking rules were analysed and compared by filling ratio, by packing time and by some other rates. In addition to this, the influence on filling ratio and packing time depending on various combination of parameters M_3 and M_2 and depending on the number of boxes which we are trying to pack is considered. The performance of the heuristic is also experimentally compared for homogeneous, strongly and weakly heterogeneous instances.

This work has been reported at some conferences (10th Student’s Conference of Applied Mathematics , 2012; Mathematics and Mathematical Modelling, 2012). In addition to this, one article have been published based on this topic [8], and another is going to be published [7].

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

ĮVADAS

Šiame darbe yra nagrinėjamas konteinerio užpildymo uždavinys (angl. *Container Loading Problem* (trumpiau CLP)), priskiriamas trimačio pakavimo uždavinių kategorijai. Tam tikrus šio uždavinio variantus labai dažnai sprendžiame atlikdami kasdieninės veiklos operacijas, pavyzdžiui: užpildydami kuprinę ar lagaminą daiktais, parduotuvėje kraudami prekes į vežimėlį ir pan.

Tuo tarpu pjaustymo ir pakavimo pramonėje CLP yra sprendžiamas, pavyzdžiui, pjaustant medieną ar putplastį į smulkesnes dalis, pakraunant paletes prekėmis, užpildant konteinerius kroviniu ir t.t. [11]. Logistikoje šis uždavinys taip pat labai svarbus - optimalus konteinerio erdvės panaudojimas ne tik sumažins transportavimo išlaidas, tačiau taip pat pagerins krovinio stabilumo ir atramos sąlygas bei padės išvengti prekių sugadinimo.

Literatūroje, priklausomai nuo tikslo funkcijos ir kraštinių sąlygų, yra išskiriama keletas CLP variantų. Detaliau šie variantai aptarti 1.5 poskyryje. Šiame darbe nagrinėjamas CLP variantas, kurio tikslas - vieną fiksuotų dimensijų konteinerį užpildyti tam tikru duotos dėžių aibės poaibiu taip, kad:

- bendras supakuotų dėžių tūris būtų maksimizuotas,
- būtų išlaikomi tam tikri apribojimai (pvz.: dėžių orientacijos, stabilumo, krovinio gravitacijos centro, riboto svorio ir kt.).

Išsami pakavimo uždavinių klasifikacija pateikta Dyckhoff (1990) ir Wascher (2007) darbuose [4], [13].

Paprastai pakavimo uždavinių atveju yra operuojama su diskretinio tipo kintamaisiais (sveikųjų skaičių rinkiniais, perstatymais ir pan.), todėl jie (taigi ir CLP) yra priskiriami kombinatorinio optimizavimo (angl. *Combinatorial optimization*) uždavinių klasei [1].

CLP taip pat yra priskiriamas NP sunkių (angl. *Nondeterministic polynomial hard*) uždavinių sudėtingumo klasei, nes, didėjant uždavinio apimčiai, visais atvejais tiksliai jį išspręsti per trumpą laiką tampa nebeįmanoma. Dėl šios priežasties yra ieškoma apytikslių sprendimo metodų, leidžiančių rasti sprendinius per priimtiną laiką ir su priimtinomis kompiuterio atminties sąnaudomis, tačiau tokie surasti sprendiniai yra nebūtinai optimalūs.

Šio darbo pagrindinis tikslas yra ištirti euristikos, pagrįstos sienų rentimo metodika (angl. *Wall-building approach*) ir medžio paieškos (angl. *Tree search*) algoritmu [11], taikymo galimybes. Keičiant vidines euristikos procedūras – rangavimo taisykles – yra atliekama rezultatų (gauto sprendinio ir sprendimo laiko) lyginamoji analizė. Taip pat atliekami parametrų, apibrėžiančių paieškos variantų skaičių, pakuojamų dėžių kiekį bei krovinio tipą (vienarūšis, stipriai ir silpnai įvairiarūšis), tyrimai.

Bendrojoje dalyje skaitytojas supažindinamas su pagrindiniais kombinatorinio optimizavimo uždavinių ir sudėtingumo klasių teorijos elementais, pakavimo uždavinių struktūra ir klasifikacija. Taip pat pateikiama CLP sprendimui taikomų metodų apžvalga, apibūdinami nagrinėjamos euristikos veikimo principai bei realizacijos ypatybės, aptariama tyrimo atlikimo metodika. Tiriamojoje dalyje pateikti euristikos analizės rezultatai. Po to pateikiamas programos aprašymas bei jos taikymo pavyzdžiai. Galiausiai diskusijos skyrelyje aptariami gauti rezultatai ir suformuluojamos išvados.

1. BENDROJI DALIS

1.1. KOMBINATORINIO OPTIMIZAVIMO UŽDAVINIO SAMPRATA

Spręsdami optimizavimo uždavinį, siekiame rasti geriausią nepriklausomų dydžių konfigūraciją (sprendinį). Priklausomai nuo sprendinių tipo yra skiriamos dvi optimizavimo uždavinių klasės. Pirmajai klasei yra priskiriami tie optimizavimo uždaviniai, kurių sprendiniai yra užkoduojami naudojant realaus tipo kintamuosius, o antrajai - diskretinio tipo kintamuosius (pvz.: sveikuosius skaičius, sveikųjų skaičių rinkinius, perstatymus, poaibius, grafus ir pan.). Antrosios klasės uždaviniai ir yra vadinami kombinatorinio optimizavimo uždaviniais (angl. *Combinatorial optimization problems*). Kintamųjų reikšmių aibė šiuo atveju yra baigtinė ar bent jau suskaičiuojama.

Remiantis Blum ir Roli (2003) [1], kombinatorinio optimizavimo uždavinys gali būti apibrėžiamas rinkiniu $P = (S, f)$, kai duota:

- $X = \{x_1, \ldots, x_n\}$ - kintamųjų aibė;
- $D_1, \ldots, D_n$ - kintamųjų apibrėžimo sritys;
- įvairūs apribojimai;
- tikslo funkcija $f : D_1 \times \ldots \times D_n \to R$, kurios ekstremumo (priklausomai nuo konkretaus sprendžiamo uždavinio – minimumo arba maksimumo) ieškome.

Tuomet leistinųjų sprendinių aibė yra:

$$S = \left\{ s = \{(x_1, v_1), \ldots, (x_n, v_n)\} \mid v_i \in D_i, \; s \text{ tenkina visus apribojimus} \right\} \qquad \textbf{(1.1)}$$

Norint išspręsti kombinatorinio optimizavimo uždavinį, reikia surasti tokį sprendinį $s^* \in S$, su kuriuo tikslo funkcijos reikšmė būtų atitinkamai minimali arba maksimali, t.y. $f(s^*) \le f(s)$ arba $f(s^*) \ge f(s)$, kai $\forall s \in S$.

Keletas kombinatorinio optimizavimo uždavinių pavyzdžių: garsusis keliaujančio pirklio uždavinys (angl. *Travelling salesman problem*), kvadratinio paskirstymo uždavinys (angl. *Quadratic assignment problem*), grafo dalijimo uždavinys (angl. *Graph partitioning problem*) ir kt. Šiame darbe nagrinėjamas konteinerio užpildymo uždavinys (CLP) taip pat priskiriamas kombinatorinio optimizavimo uždavinių klasei.

1.2. UŽDAVINIŲ SUDĖTINGUMO KLASĖS

Sudėtingumo klasė yra aibė uždavinių, kuriuos sieja panašus sudėtingumas. Skiriamos P, NP, NP sunkių ir NP pilnųjų uždavinių sudėtingumo klasės.

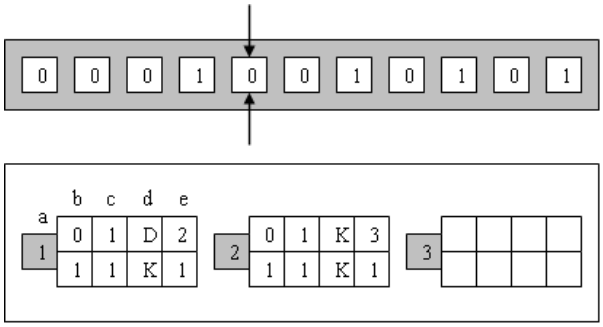
Sudėtingumo klasei P (angl. *Deterministic polynomial time*) priskiriami uždaviniai, kuriuos galima išspręsti deterministine Tiuringo mašina (angl. *Turing machine*) per polinominį laiką $m(n)$. NP (angl. *Nondeterministic polynomial time*) – tai sudėtingumo klasė, jungianti uždavinius, kuriuos galima

išspręsti nedeterministine Tiuringo mašina per polinominį laiką $m(n)$. Čia $m(n) = O(n^k)$, kur k - konstanta, priklausanti nuo konkretaus uždavinio, kurio dydis yra n.

Tiuringo mašina – tai abstraktus kompiuterio vykdymo modelis, sukurtas siekiant matematiškai apibrėžti algoritmus [3]. Tiuringo mašiną sudaro (žr. 1.1 pav.):

- juosta, padalinta į langelius, kuriuose gali būti vienas iš naudojamos abėcėlės simbolių;
- galvutė, kuri skaito ir rašo į langelį, galinti judėti į abi puses (*K* – į kairę, *D* – į dešinę);
- būsenų registras, saugantis automato būseną (būsenų skaičius yra baigtinis, pradinė būsena visada apibrėžta);
- veiksmų lentelė, nusakanti, kokį simbolį rašyti (c), į kurią pusę per vieną langelį pajudėti (d), taip pat kokia bus nauja būsena (e), priklausomai nuo esamos būsenos (a) ir perskaitytos langelio rekšmės (b). Jei veiksmų lentelėje nėra aprašyto veiksmo dabartinei būsenai ir langelio reikšmei, mašina baigia darbą (pvz., 1.1 pav. 3 būsena).

Kai kiekvienai perskaityto simbolio ir būsenos porai yra pateikiama ne daugiau kaip viena reikšmė veiksmų lentelėje, tuomet Tiuringo mašina vadinama deterministine, priešingu atveju – nedeterministine. Tiuringo mašinos iliustracija pateikiama 1.1 paveiksle.



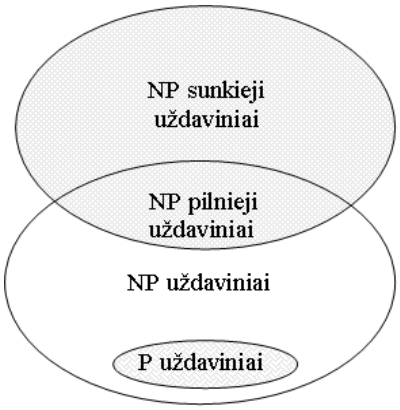
1.1 pav. Tiuringo mašinos iliustracija

Tarp NP sudėtingumo uždavinių yra apibrėžiamas NP pilnųjų (angl. *NP-complete*) uždavinių poaibis. Jam priklauso sudėtingiausi uždaviniai, kurie parenkami taip, kad jei pavyktų sukurti kurio nors šios klasės uždavinio polinominio sudėtingumo sprendimo algoritmą, tai ir bet kurį kitą NP klasės uždavinį galėtume išspręsti, atlikę polinominį skaičių veiksmų.

Uždavinių sudėtingumo teorijoje dažnai naudojama dar viena sudėtingumo klasė – NP sunkių uždavinių (angl. *NP-hard*). Sakome, kad uždavinys A priklauso NP sunkių uždavinių klasei, jei egzistuoja toks uždavinys B, priklausantis NP pilnųjų uždavinių klasei, kurį polinominio sudėtingumo algoritmu galime transformuoti į A. Čia uždavinio B transformavimas į uždavinį A reiškia, kad, žinodami A sprendimo polinominio sudėtingumo algoritmą, galime sudaryti ir uždavinio B polinominio sudėtingumo algoritmą. Taigi NP sunkieji uždaviniai yra ne mažiau sudėtingi už NP pilnuosius.

Pastebėsime, kad NP sunkiesiems uždaviniams nėra reikalavimo, kad jie būtų iš NP klasės. NP sunkieji uždaviniai, priklausantys NP klasei, priskiriami NP pilnųjų uždavinių klasei.

1.2 paveiksle pavaizduota diagrama, išreiškianti daugelio šios srities specialistų požiūrį į P, NP, NP pilnųjų ir NP sunkių uždavinių klasių tarpusavio ryšį [12]. Dar niekam nepavyko sukurti deterministinio polinominio laiko algoritmo, kuris visais atvejais optimaliai spręstų bet kurį uždavinį iš NP klasės, todėl labai tikėtina, kad hipotezė $P \neq NP$ yra teisinga, t.y. $P \subset NP$. Vis dėlto šis faktas dar neįrodytas, todėl laikoma, kad $P \subseteq NP$.



1.2 pav. Sudėtingumo klasių diagrama

Šiame darbe nagrinėjamas trimatis konteinerio užpildymo uždavinys yra priskiriamas NP sunkių uždavinių klasei.

1.3. APIBENDRINTA PAKAVIMO UŽDAVINIŲ STRUKTŪRA

Pakavimo (kartu ir pjaustymo) uždavinių struktūrą galima apibendrinti tokiu būdu: duota tam tikra aibė didelių objektų (pvz., konteinerių) ir tam tikra aibė mažų objektų (pvz., dėžių). Šie objektai yra nusakomi viena, dviem, trimis arba didesniu skaičiumi n geometrinio matavimo dimensijų. Reikia paimti tik dalį arba visus mažus objektus, sugrupuoti juos į vieną arba daugiau poaibių ir kiekvieną sudarytą poaibį priskirti vienam iš didelių objektų taip, kad būtų tenkinamos toliau išvardintos sąlygos. Kiekvieno mažų objektų poaibio elementai turi būti išdėstyti atitinkamame dideliame objekte taip, kad:

- maži objektai būtų visiškai įtalpinti didelio objekto viduje;
- maži objektai nepersidengtų;
- duota vienmatė arba daugiamatė tikslo funkcija būtų optimizuota.

Uždavinio sprendinys priklauso nuo to, ar yra panaudojami visi ar tik dalis didelių objektų bei visi ar tik dalis mažų objektų. Formaliai pakavimo uždavinį galima suskaidyti į penkis smulkesnius uždavinius [13]:

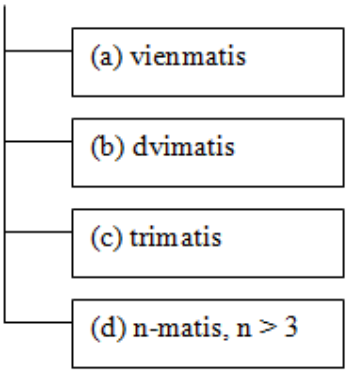
- didelių objektų išrinkimas iš visos duotos aibės;
- mažų objektų išrinkimas iš visos duotos aibės;

- atrinktų mažų objektų grupavimas;
- sudarytų mažų objektų poaibių priskyrimas atrinktiems dideliems objektams;
- išdėstymo uždavinys, kai maži objektai, atsižvelgiant į jų įvairovę ir savybes, išdėstomi dideliame objekte taip, kad būtų tenkinamos anksčiau paminėtos geometrinės sąlygos.

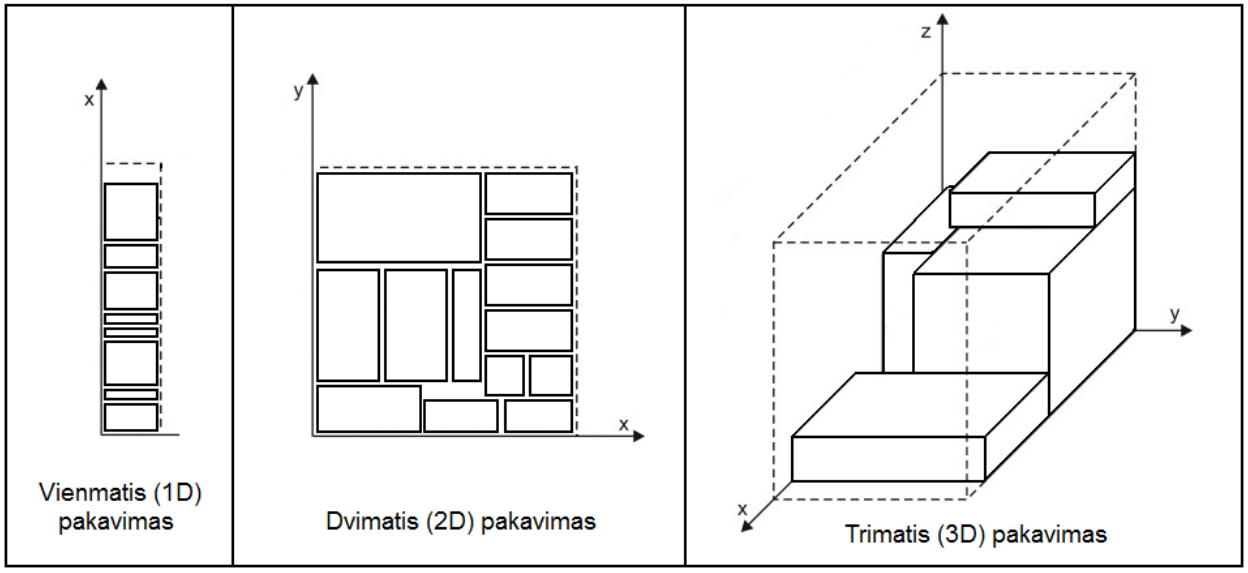
1.4. PAKAVIMO UŽDAVINIŲ KLASIFIKACIJA

Pagrindinės pakavimo uždavinių klasifikacijos idėjos buvo pasiūlytos autoriaus Dyckhoff (1990) [4]. Modifikuotą klasifikaciją, leidžiančią gauti tikslesnį pakavimo uždavinių skaidymą į homogenines klases, pateikė Wascher ir kt. (2007) [13]. Toliau pateikiami keturi svarbiausi klasifikavimo kriterijai:

1. pagal <u>dimensijų skaičių</u>:

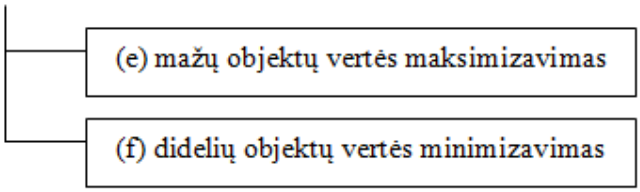


Dimensijų skaičiaus kriterijus nusako minimalų skaičių (1, 2, 3, $n > 3$) geometrinio matavimo dimensijų, reikalingų visiškai apibūdinti pakavimo struktūrą. 1.3 paveiksle pateikta vienmačio, dvimačio ir trimačio pakavimų iliustracija. Šiame darbe nagrinėjamas konteinerio užpildymo uždavinys atitinka trimačio (3D) pakavimo atvejį.



1.3 pav. Vienmačio, dvimačio ir trimačio pakavimo uždavinių iliustracijos

2. pagal <u>užduoties tipą</u>:

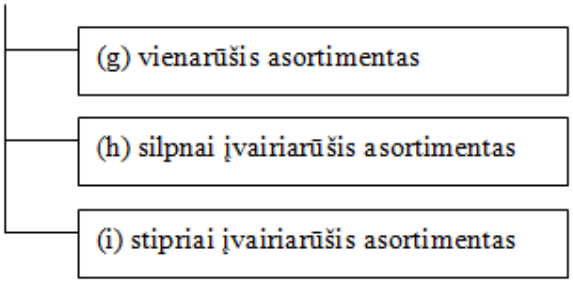


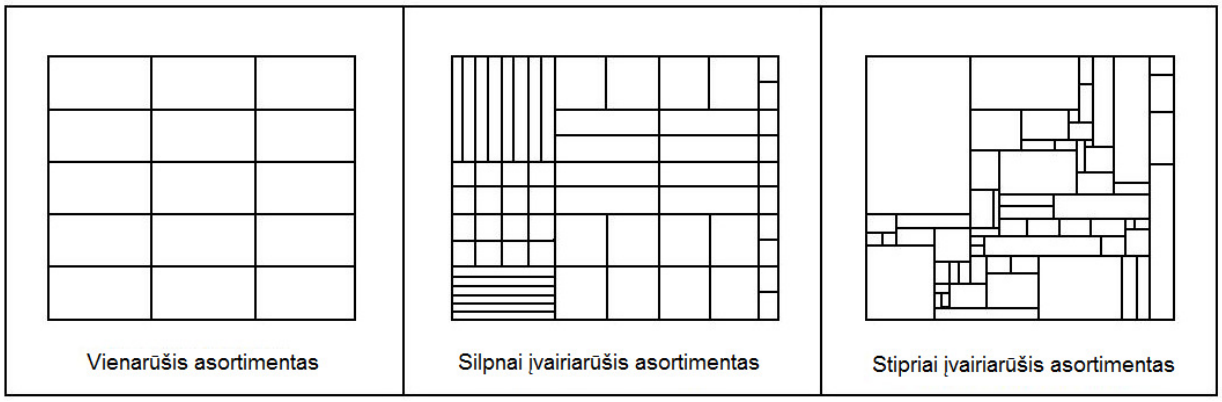
Tiek (e), tiek (f) atvejais mažų objektų aibė yra priskiriama duotai didelių objektų aibei. (e) atveju didelių objektų aibės elementai nebūtinai tinka visiems mažiems objektams, todėl dalis mažų objektų gali likti ir nepanaudoti. Tuo tarpu dideli objektai – priešingai - privalo būti panaudoti visi, priskiriant jiems mažų objektų aibės dalį (poaibį) su maksimalia „verte“. Taigi šiuo atveju nėra sprendžiamas 1.3 poskyryje paminėtas didelių objektų išrinkimo uždavinys. Tuo tarpu (f) atveju, t.y. siekiant minimizuoti didelių objektų „vertę“, didelių objektų aibės elementai privalo tikti visiems mažiems objektams. Šiuo atveju visi maži objektai turi būti panaudoti, kitaip sakant, nėra sprendžiamas 1.3 poskyryje paminėtas mažų objektų išrinkimo uždavinys. Taigi šiuo atveju visi maži objektai yra susiejami su didelių objektų aibės dalimi (poaibiu), turinčiu mažiausią „vertę“.

Verta pastebėti, kad sąvoka „vertė“ yra vartojama apibendrintai ir, sprendžiant konkretų uždavinį, yra apibrėžiama tiksliau. Dažnai objektų vertė gali būti tiesiogiai proporcinga jų dydžiui, t.y.: ilgiui (vienmačiu atveju), plotui (dvimačiu atveju) ar tūriui (trimačiu atveju).

Šiame darbe nagrinėjamas konteinerio užpildymo uždavinys atitinka (e) užduoties tipą, t.y. mažų objektų vertės maksimizavimą.

3. pagal <u>mažų objektų įvairovę</u>:

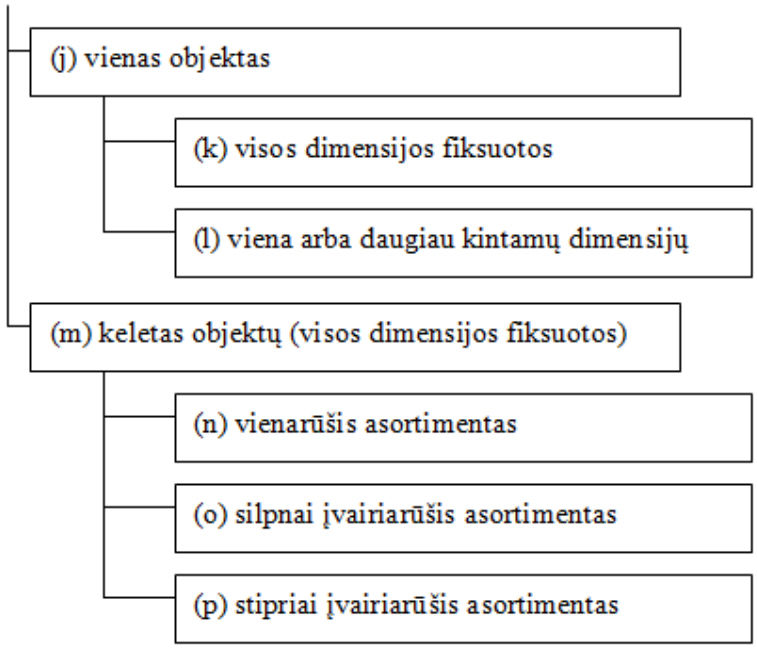




1.4 pav. Vienarūšio, silpnai ir stipriai įvairiarūšio asortimento iliustracija

Taigi (g) atveju visi maži objektai yra tos pačios formos ir dydžio visų dimensijų atžvilgiu (žr. 1.4 pav. 1 dalį). Tuo tarpu (h) atveju visi maži objektai yra keleto skirtingų dydžių ir gali būti sugrupuoti į keletą klasių, kuriose visi elementai būtų tų pačių formų ir dydžių (žr. 1.4 pav. 2 dalį). Atveju (i) turime labai mažai arba visai neturime vienodos formos ir dydžių mažų objektų (žr. 1.4 pav. 3 dalį). Šiame darbe, tiriant pasirinktos euristikos (žr. 1.8 poskyrį) veikimą, bus nagrinėjami visi trys mažų objektų asortimento variantai, t.y.: (g), (h) ir (i).

4. pagal <u>didelių objektų įvairovę</u>:



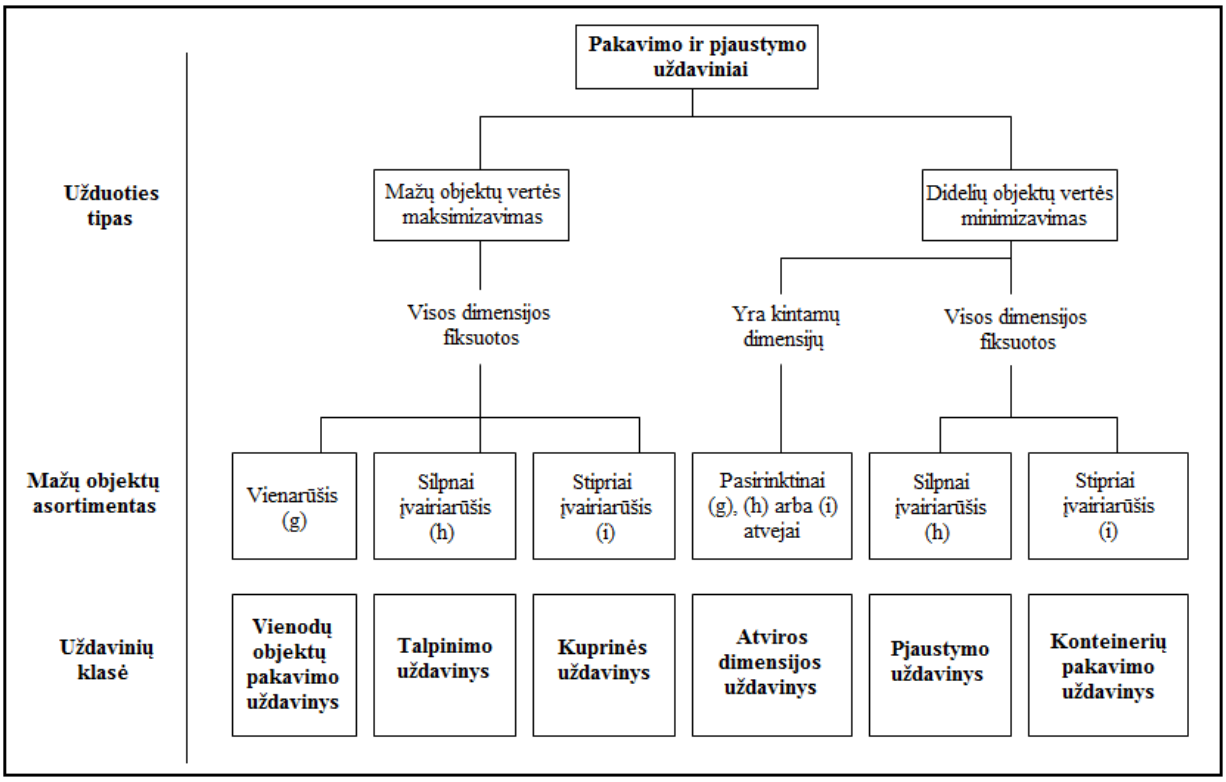
Taigi (j) atveju didelių objektų aibė yra sudaryta iš vieno elemento, kurio praplėtimas gali būti fiksuotas visoms dimensijoms (atvejis (k)), arba gali būti kintamas viena arba daugiau dimensijų kryptimis (atvejis (l)). Atsižvelgus į literatūroje nagrinėjamus uždavinius, keleto didelių objektų atveju nėra poreikio juos skirstyti į fiksuotų ir kintamų dimensijų uždavinius, todėl šiuo atveju nagrinėjamas tik fiksuotų dimensijų atvejis (m). Minėta kategorija išskaidoma analogiškai, kaip ir mažų objektų atveju (vienarūšis, silpnai ir stipriai įvairiarūšis asortimentas).

Šiame darbe nagrinėjamo konteinerio užpildymo uždavinio atveju yra duotas vienas didelis objektas, kurio visos dimensijos yra fiksuotos. Tai atitinka didelių objektų įvairovės atvejį (k).

Dažniausiai pakavimo ir pjaustymo uždaviniai yra klasifikuojami tik pagal du iš minėtų kriterijų, t.y. pagal užduoties tipą ir mažų objektų įvairovę. Taip pat atsižvelgiama į tai, ar visos didelių objektų dimensijos yra fiksuotos, ar dalis jų – kintamos [13]. Tokio klasifikavimo schema pateikta 1.5 paveiksle. Pavyzdžiui, šiame darbe tiriamas konteinerio užpildymo uždavinys, kai pakuojamas stipriai įvairiarūšis krovinys, būtų priskiriamas Kuprinės uždavinių klasei.

Literatūroje dar galima rasti klasifikavimą pagal papildomus kriterijus, leidžiančius susiaurinti uždavinio sritį. Pavyzdžiui, gali būti atsižvelgiama į pakuojamų mažų objektų formą - taisyklingą (pvz.:

stačiakampiai, skrituliai, cilindrai, rutuliai ir pan.) arba netaisyklingą. Taip pat gali būti išskiriamas ortogonalus (kai pakavimas atliekamas lygiagrečiai didelio objekto kraštui) arba neortogonalus pakavimas ir t.t.



1.5 pav. Pagrindiniai pakavimo ir pjaustymo uždavinių tipai

1.5. KONTEINERIO UŽPILDYMO PROBLEMOS (CLP) VARIANTAI

Priklausomai nuo tikslo funkcijos ir kraštinių apribojimų, yra skiriami tokie konteinerio užpildymo uždavinio (angl. *Container loading problem (CLP)*) variantai [11]:

- **Juostos pakavimo CLP** (angl. *Strip packing*). Šiuo atveju duotas vienas konteineris, turintis fiksuotą aukštį ir plotį, tačiau kintamą ilgį. Uždavinio tikslas – supakuoti visas dėžes taip, kad konteinerio ilgis būtų minimizuotas.
- **Kuprinės užpildymo CLP** (angl. *Knapsack loading*). Šiuo atveju kiekviena dėžė be pagrindinių išmatavimų (ilgio, pločio ir aukščio) dar turi papildomą parametrą – vertę. Uždavinio tikslas – iš duotos dėžių aibės vieną konteinerį užpildyti tokiu dėžių poaibiu, kad būtų maksimizuota bendra supakuotų dėžių vertė.
- **Kelių vienodų konteinerių CLP** (angl. *Bin-packing*). Šiuo atveju duoti keli vienodų išmatavimų konteineriai. Uždavinio tikslas – visas dėžes supakuoti į kiek įmanoma mažesnį skaičių konteinerių. Kitaip sakant, reikia minimizuoti panaudotų konteinerių skaičių.

- **Kelių skirtingų konteinerių CLP** (angl. *Multi-container loading*). Šis variantas yra panašus į ką tik aptartą kelių vienodų konteinerių CLP, tačiau šiuo atveju konteinerių išmatavimai gali skirtis. Uždavinio tikslas – pasirinkti tokį konteinerių aibės poaibį, kad būtų minimizuojami kaštai.

1.6. PASIRINKTO CLP VARIANTO FORMULUOTĖ

Šiame darbe nagrinėsime trimatį vieno konteinerio užpildymo uždavinį, kuris atitinka kuprinės užpildymo variantą, aptartą 1.5 poskyryje. Taigi šiuo atveju reikia vieną fiksuotų dimensijų konteinerį (didelį objektą) užpildyti tam tikru duotos dėžių (mažų objektų) aibės poaibiu. Šis poaibis turi būti parinktas taip, kad bendra supakuotų daiktų vertė būtų maksimizuota, tačiau taip pat būtų išlaikomi tam tikri apribojimai.

Modelyje darysime tokias prielaidas:

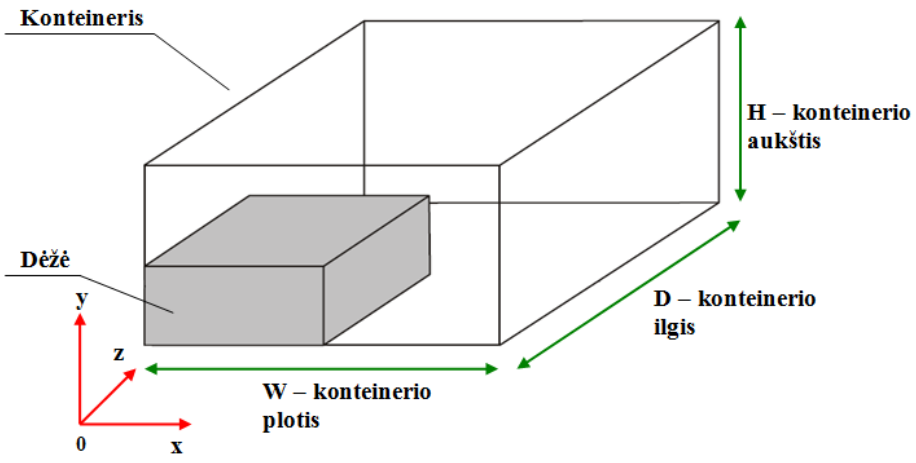
- tiek konteineris, tiek pakuojamos dėžės yra stačiakampio gretasienio formos;
- visos dėžės yra dėliojamos lygiagrečiai konteinerio sienoms, t.y. turime ortogonalų pakavimą;

Toliau darbe naudojami tokie žymėjimai: konteinerio plotis W, aukštis H ir ilgis D (žr. 1.6 pav.). Tuomet užpildomo konteinerio tūris V yra apskaičiuojamas pagal (1.2) formulę.

$$V = W \cdot H \cdot D \tag{1.2}$$

Tuo tarpu turimų dėžių indeksų aibė yra $N = \{1, 2, ..., n\}$, o j-toji dėžė ($j \in N$) yra apibūdinama keturiais parametrais, t.y.: pločiu w_j, aukščiu h_j, ilgiu d_j ir verte v_j. Tarsime, kad dėžės vertė reiškia jos tūrį (žr. (1.3) formulę).

$$v_j = w_j h_j d_j \tag{1.3}$$



1.6 pav. Užpildomo konteinerio iliustracija

Šio uždavinio atveju yra siekiama maksimizuoti supakuotų dėžių bendrą tūrį, arba kitaip, minimizuoti nepanaudotos konteinerio erdvės dalį. Tarkime, kad $\hat{N} \subset N$ yra supakuotų dėžių indeksų

aibė. Tuomet sprendžiamo pakavimo uždavinio tikslo funkcija, kurią siekiama maksimizuoti, yra apibrėžiama (1.4) formule.

$$f = \frac{\sum_{j \in \hat{N}} w_j h_j d_j}{W \cdot H \cdot D} \quad \textbf{(1.4)}$$

Dėžės turi būti supakuotos taip, kad visiškai tilptų konteinerio viduje ir tarpusavyje nepersidengtų. Literatūroje dar galima rasti įvairių papildomų apribojimų, keliamų krovinio išdėstymui konteineryje [2], pavyzdžiui:

- **(C1) – dėžių orientacija.** Gali būti ribojamas dėžių pasukimas tam tikromis ortogonaliomis kryptimis.
- **(C2) – ribota dėžės apkrova iš viršaus.** Draudžiama ant tam tikrų dėžių viršaus krauti kitas dėžes (siekiant išvengti jų deformacijos ar pan.).
- **(C3) – ribotas bendras krovinio svoris.** Reikalaujama, kad bendras konteinerio svoris neviršytų nustatyto svorio limito.
- **(C4) – dėžių stabilumas.** Reikalaujama, kad dėžių stabilumas nebūtų mažesnis už nustatytą ribą. Supakuotos dėžės stabilumas yra apskaičiuojamas kaip santykis tarp jos apatinės pagrindo plokštumos, esančios sąlytyje su apačioje esančia dėže, ir jos visos apatinės pagrindo plokštumos.
- **(C5) – krovinio gravitacijos centras.** Reikalaujama, kad gravitacijos centras būtų kiek galima arčiau konteinerio grindų plokštumos geometrinio vidurio taško.
- **(C6) – vienodų dėžių grupavimas.** Reikalaujama, kad vienodos dėžės būtų pakuojamos kiek įmanoma arčiau viena kitos (siekiant sumažinti sukrovimo, krovinio patikrinimo ir iškrovimo laiko sąnaudas ar pan.).
- **(C7) – dėžių pakavimo prioritetai.** Reikalaujama, kad dėžės, turinčios didesnį prioriteto laipsnį, būtų pakuojamos anksčiau už kitas.
- **(C8) – kelių iškrovimo taškų situacijos.** Reikalaujama sugrupuoti pakuojamas dėžes, turinčias tą patį pristatymo tikslą.
- **(C9) – kelių rūšių dėžių atskyrimas konteineryje.** Jei, pavyzdžiui, tame pačiame konteineryje turi būti gabenami maisto produktai ir cheminės medžiagos, tai reikalaujama, kad dėžių išdėstymas konteineryje būtų toks, kad būtų išvengta šių medžiagų kontakto.
- ir kt.

Sprendžiant praktinius uždavinius, dažniausiai yra įtraukiami tik dėžių orientacijos (C1) ir stabilumo (C4) apribojimai. Tačiau dėžės gali būti sėkmingai pakuojamos ir neatsižvelgiant į stabilumo reikalavimą, nes praktiškai visuomet galima tarpus tarp dėžių užpildyti specialiomis medžiagomis.

Šiame darbe tarsime, kad dėžės gali būti pasukamos visomis ortogonaliomis kryptimis, ir nėra atsižvelgiama į dėžių stabilumo reikalavimą. Taip pat, nemažindami bendrumo, tarsime, kad tiek konteinerio, tiek visų dėžių išmatavimai yra teigiami sveikieji skaičiai. Modelyje bus išskiriami trys 1.4 poskyryje aptarti krovinio (pakuojamų dėžių aibės) tipai, t.y.: vienarūšis (angl. *homogeneous*), silpnai įvairiarūšis (angl. *weakly heterogeneous*) ir stipriai įvairiarūšis (angl. *strongly heterogeneous*).

1.7. SPRENDIMUI TAIKOMŲ METODŲ APŽVALGA

1.7.1. PAGRINDINĖS METODŲ KLASĖS

Optimizavimo uždavinius sprendžiantys algoritmai gali būti skirstomi į tikslius ir apytikslius. Taip pat vartojami terminai „euristinis“ (angl. *heuristic*), „metaeuristinis“ (angl. *metaheuristic*) ir „hibridinis“ (angl. *hybrid*).

Euristiniu algoritmu priimta laikyti tokį optimizavimo uždavinių sprendimo metodą, kuriuo siekiama surasti aukštos kokybės, bet nebūtinai optimalų sprendinį per priimtiną skaičiavimo laiką. Šie algoritmai yra pritaikyti specifinių uždavinių sprendimui. Terminas „metaeuristinis“ kilęs iš graikiško žodžio *heuriskein*, kuris reiškia „ieškoti“, ir priedėlio *meta*, reiškiančio „virš“, „aukštesniame lygyje“. Metaeuristiniai algoritmai (kaip ir euristiniai) negarantuoja gautų sprendinių optimalumo, o surasti sprendiniai paprastai yra tik lokaliai optimalūs duotos aplinkos atžvilgiu. Tuo euristiniai ir metaeuristiniai metodai skiriasi tiek nuo tiksliųjų algoritmų (angl. *Exact algorithms*), kurie garantuoja optimalaus sprendinio suradimą, tiek nuo aproksimacinių algoritmų (angl. *Approximate algorithms*), kurie užtikrina, kad gauto sprendinio kokybė skirsis nuo optimalaus ne daugiau kaip iš anksto fiksuota paklaida $\varepsilon > 0$ [9]. Metaeuristikose sprendinio paieškos procesą dažnai reguliuoja žemesnio lygio euristikos. Hibridiniuose algoritmuose yra apjungiami keli skirtingi metodai. Taip yra daroma todėl, kad labai dažnai apjungtų metodų kombinacijos naudojimas leidžia kompensuoti šių pavienių algoritmų trūkumus bei sustiprinti jų privalumus sprendžiant konkrečius uždavinius.

1.7.2. CLP SPRENDIMUI TAIKOMI METODAI

Konteinerio užpildymo uždavinys (CLP) yra priskiriamas NP sunkių uždavinių klasei. Dar nėra žinomas polinominio laiko algoritmas, visais atvejais tiksliai sprendžiantis šio tipo uždavinius. Būtent todėl, ieškant efektyviausio metodo, buvo sukurta nemažai skirtingų šių uždavinių sprendimo algoritmų ir jų hibridinių junginių.

Tiksliai konteinerio užpildymo uždavinį galima išspręsti taikant mišrų sveikaskaitį programavimą (angl. *Mixed Integer Programming (MIP)*) arba netiesinį programavimą (angl. *Nonlinear*

Programming (NLP)) [2]. Tačiau šie algoritmai gali būti taikomi tik nedidelės apimties uždavinių atveju.

Euristiniai metodai, taikomi CLP sprendimui, gali būti suskirstyti į tris pogrupius, atsižvelgiant į juose taikomą metodiką [5]:

1. **Tradicinės euristikos**. Šiam pogrupiui priskiriami konstravimo metodai (pvz., godžioji euristika (angl. *Greedy*)) ir sprendinio pagerinimo metodai (pvz., vietinės paieškos (angl. *Local search*)). Tradicines euristikas plėtojo Bischoff ir kt. (1995), Bischoff ir Ratcliff (1995) bei Lim ir kt. (2003).
2. **Metaeuristikos**. Genetinius algoritmus (angl. *Genetic algorithm (GA)*) nagrinėjo Hemminki (1994), Gehring ir Bortfeldt (1997, 2001, 2002). Gehring ir Bortfeldt taip pat pristatė lygiagretaus skaičiavimo GA (angl. *Parallel GA*) (2002) bei hibridinį GA (2001). Hibridinį GA, pagrįstą skruzdėlių kolonijos optimizavimo (angl. *Ant Colony Optimization*) idėjomis, pasiūlė Liang ir kt. (2007). Tabu paieškos (angl. *Tabu search (TS)*) algoritmus nagrinėjo Sixt (1996), Bortfeldt ir kt. (2003). Gehring ir Bortfeldt (2003) taip pat pasiūlė lygiagretaus skaičiavimo TS algoritmą (angl. *Parallel TS*), o Liu ir kt. (2011) – hibridinį TS. Modeliuojamo atkaitinimo (angl. *Simulated annealing (SA)*) algoritmai buvo nagrinėti autorių Sixt (1996) bei Mack ir kt. (2004), o hibridinis SA – Peng ir Zhang (2009). Moura ir Oliveira (2005) bei Parreno ir kt. (2007) pristatė GRASP (iš angl. *Greedy Randomized Adaptive Search Procedure*) algoritmą. Bičių kolonijos algoritmą (angl. *Bee-colony*) pristatė Dereli ir Das (2011), kintamos kaiminystės metodą (angl. *Variable neighborhood search (VNS)*) – Parreno ir kt. (2010).
3. **Medžio paieškos metodai**. Nupjauto medžio arba grafo paieškos metodai taip pat buvo sėkmingai taikomi CLP sprendimui. Pavyzdžiui, Morabito ir Arenales (1994) pasiūlyta IR/AR grafo paieška (angl. *AND/OR graph search*), taip pat autorių Eley (2002), Hifi (2002), Terno ir kt. (2000), Pisinger (2002), Fanslau ir Bortfeldt (2009) darbai.

Dažniausiai konteinerio užpildymui taikomos tokios strategijos:

- **Sienų rentimo metodas.** Konteineris yra užpildomas vertikaliais sluoksniais („sienomis“). Tokia metodika buvo taikyta autorių George ir Robinson (1980), Bischoff ir Marriot (1990), Loh ir Nee (1992), Gehring ir kt. (1990), Chien ir Wu (1998), Bortfeldt ir Gehring (2001), Pisinger (2002), Cecilio ir Morabito (2004) bei Moura ir Oliveira (2005).

- **Bokštų (stulpelių) statymo metodas.** Šiuo atveju dėžės pirmiausia yra supakuojamos į tam tikrus bokštus (stulpelius), kurie po to yra dėliojami ant konteinerio grindų plokštumos. Ši metodika buvo taikoma Haessler ir Talbot (1990), Bischoff ir Ratcliff (1995) bei genetiniame autorių Gehring ir Bortfeldt (1997) algoritmuose.
- **Horizontalių sluoksnių konstravimo metodas.** Konteineris yra pildomas iš apačios į viršų, formuojant horizontalius sluoksnius. Šią metodiką savo algoritmuose taikė Bischoff ir Ratcliff (1995, 1998), Terno ir kt. (2000) bei Lim ir Zhang (2005).
- **Blokų konstravimo metodas.** Šiuo atveju konteineris yra užpildomas tam tikrais blokais, kurie paprastai yra sudaromi iš vienodo tipo ir vienodos erdvinės orientacijos dėžių. Šios strategijos taikymo pavyzdžiai yra Bortfeldt ir Gehring (1998), Eley (2002) medžio paieškos, Bortfeldt ir kt. (2003) tabu paieškos algoritmai bei Mack ir kt. (2004) modeliuojamo atkaitinimo ir tabu paieškos hibridinis algoritmas.
- **Giljotininio pjūvio metodas.** Šios strategijos atveju, naudojant giljotininius pjūvius, konteineris yra padalinamas į mažesnes dalis. Pavyzdžiui, šia strategija yra pagrįstas autorių Morabito ir Arenales (1994) grafo paieškos metodas.

Kadangi pastaruoju metu trimatis konteinerio užpildymo uždavinys sulaukia vis daugiau tyrėjų dėmesio, tai be išvardintų metodikų literatūroje dar galima rasti ir kitokių metodų variacijų. Pavyzdžiui, „Daugiakrypčio statymo - augimo metodas“ (angl. *Multi-Directional Building-Growing Approach*), kuris skiriasi nuo tradicinės sienų statymo ar sluoksnių konstravimo metodikos tuo, kad „statinio augimas“ yra leidžiamas įvairiomis konteinerio kryptimis, t.y. baziniu pagrindu imant bet kurią konteinerio sieną.

Dauguma paminėtų autorių atsižvelgė į dėžių orientacijos (C1) ir stabilumo apribojimus (C4) (žr. 1.6 poskyrį). Tuo tarpu krovinio gravitacijos centro apribojimą (C5) vertino Wodziak ir Fadel (1996), Davies ir Bischoff (1999), Bortfeldt ir Gehring (2001) bei Eley (2002). Riboto bendro krovinio svorio apribojimą (C3) vertino Terno ir kt. (2000), Bortfeldt ir Gehring (2001), Eley (2002), Mack ir kt. (2004). Ribotos apkrovos iš viršaus (C2) apribojimą vertino Bischoff (2006).

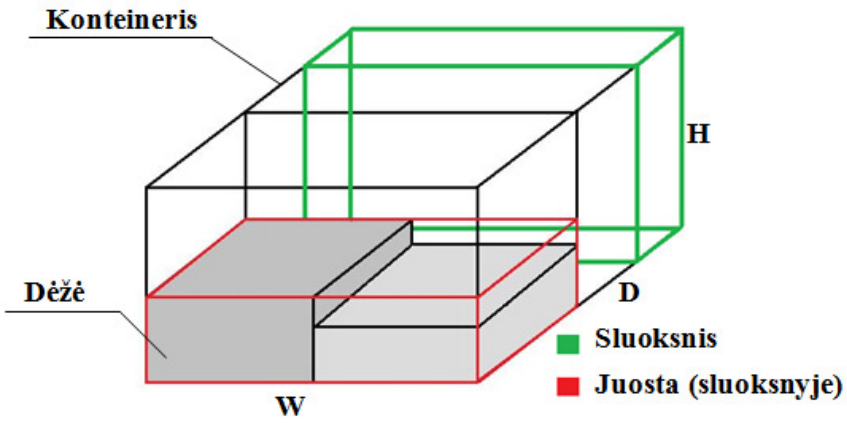
Iš lietuvių autorių galima būtų paminėti Juraitį ir kt. (2006). Jų straipsnyje [6], panaudojant genetinį algoritmą, buvo ieškoma mišinio, sudaryto iš tam tikrų godžiųjų euristikų ir Monte-Carlo paieškos, proporcijų, kurios leistų pasiekti efektyvesnių rezultatų.

Galima pabrėžti, kad konteinerio užpildymo uždavinio sprendimui taikomų algoritmų kūrimas turi gana ryškų komercinį atspalvį. Dėl šios priežasties prieiga prie įvairių algoritmų programinių kodų yra labai apribota. Šio darbo tyrimui buvo pasirinkta euristika, kurios programinis kodas yra viešai prieinamas literatūros šaltinyje [14] ir gali būti laisvai naudojamas akademiniais tikslais.

1.8. NAGRINĖJAMOS EURISTIKOS APRAŠYMAS

1.8.1. PAGRINDINIAI VEIKIMO PRINCIPAI

Pasirinkta euristika yra priskiriama medžio paieškos (angl. *Tree Search*) metodų klasei ir yra paremta sienų statymo (angl. *Wall-building*) metodika (žr. 1.7.2 poskyrį) [11]. Šiuo atveju pakuojama erdvė yra suskaidoma į tam tikrą skaičių vertikalių arba horizontalių sluoksnių (sienų), kurių kiekvienas vėl yra suskaidomas į tam tikrą skaičių stačiakampio gretasienio formos vertikalių arba horizontalių juostų. 1.7 paveiksle yra pateikta konteinerio erdvės skaidymo į vertikalius sluoksnius ir sluoksnio skaidymo į horizontalias juostas pavyzdžio iliustracija.



1.7 pav. Konteinerio erdvės skaidymo iliustracija

Tiek sluoksnio, tiek kiekvienos juostos storis yra parenkamas šakų ir ribų (angl. *Branch and Bound*) algoritmu, kai kiekvienoje viršūnėje yra nagrinėjamas tik tam tikras poaibis šakų. Tokiu būdu, taikant specialias rangavimo taisykles, yra parenkamas skaičius M_3 galimų sluoksnių storių ir skaičius M_2 galimų juostų storių. Turint fiksuotus sluoksnio ir juostos storius, nagrinėjama juosta yra užpildoma sprendžiant binarinį (0-1) kuprinės uždavinį. Taigi šios euristikos atveju pradinis trimatis pakavimo uždavinys yra transformuojamas į gerokai paprastesnį vienmatį pakavimo uždavinį. Galima paminėti, kad užpildant sluoksnius juostos gali būti orientuotos tiek vertikaliai, tiek horizontaliai.

Kiekvienam sluoksniui iš sluoksnio storių $d'_1, .d'_2, .., d'_{M_3}$ yra parenkamas tas, kuriam gaunamas geriausias tame žingsnyje pakuojamos konteinerio erdvės užpildymas. Tuomet yra pereinama prie kito sluoksnio, ir vėl viskas kartojama, kol galiausiai užpildomi visi konteinerio sluoksniai. Detalesnis sluoksnių ir juostų pildymo algoritmo esminių punktų aprašymas pateiktas 1.8.2-1.8.5 poskyriuose. Trumpumo dėlei sluoksnio storio parinkimo ir sluoksnio pildymo procedūroms įvedami pažymėjimai atitinkamai „SSP“ ir „SP“.

1.8.2. PROCEDŪROS „SSP“ APIBENDRINTA VEIKIMO SCHEMA

Procedūros `Sluoksnio_storio_parinkimas(`d`,`V`,`N'`)` (trumpiau SSP) schema, užrašyta pseudoprogramavimo kalba, pateikta 1.1 lentelėje. Čia perduodami parametrai yra:

- N' – pakuojamų dėžių poaibis;
- d – dar nepanaudotos konteinerio dalies ilgis;
- V – praeitame žingsnyje supakuotų daiktų tūris.

1.1 lentelė

Procedūros `Sluoksnio_storio_parinkimas()` veikimo schema

Algoritmas **Sluoksnio_storio_parinkimas**(d, V, N')			
	Jeigu $V > V^*$, **tai** išsaugoti sprendinį $X^* \leftarrow X$ ir nustatyti $V^* \leftarrow V$;		
	Jeigu $d < \min_{j \in N'} \min\{w_j, h_j, d_j\}$, **tai** grąžinti sprendinį;		
	Atrinkti M_3 sluoksnio storių $\{d'_1, d'_2, ..., d'_{M_3}\}$, turinčių didžiausius rangus;		
	Vykdyti kiekvienam sluoksnio storiui $d' \in \{d'_1, d'_2, ..., d'_{M_3}\}$		
		Pradėti	
			Suporuoti dėžes taip, kad bendras ilgis d_k būtų kuo artimesnis d';
			Nustatyti $U^* \leftarrow 0$;
			Vykdyti **Sluoksnio_pildymas**(W, H, d', 0, N');
			Tegu L – supakuotų dėžių poaibis, o U^* – jų tūris;
			Vykdyti **Sluoksnio_storio_parinkimas**($d - d'$, $V + U^*$, $N' \setminus L$);
		Baigti	

Rekursinėje procedūroje `Sluoksnio_storio_parinkimas()`, remiantis tam tikromis rangavimo taisyklėmis, yra atrenkamas skaičius M_3 skirtingų sluoksnio storių $\{d'_1, d'_2, ..., d'_{M_3}\}$. Tuomet kiekvienam sluoksnio storiui d', jei įmanoma, yra bandoma suporuoti dėžes po dvi, kad būtų gauti kiek įmanoma tinkamesnių išmatavimų dėžių junginiai (žr. į 1.8.5 poskyrį).

Toliau yra kreipiamasi į procedūrą `Sluoksnio_pildymas(`W`,`H`,`d'`, 0,`N'`)` (žr. į 1.8.3 poskyrį), kurios trys pirmieji parametrai nurodo, kad bus užpildomas sluoksnis su dimensijomis

(išmatavimais) $W \times H \times d'$. Šiame sluoksnyje supakuotų dėžių poaibis L yra pašalinamas iš tolimesnio uždavinio, ir į procedūrą `Sluoksnio_storio_parinkimas()` yra kreipiamasi rekursiškai, jai perduodant dar nesupakuotų daiktų poaibį $N' \setminus L$, dar nepanaudotos konteinerio dalies ilgį $d - d'$ ir praeituose žingsniuose supakuotų daiktų tūrį $V + U^*$. Procedūroje yra grįžtama atgal, jei $d < \min_{j \in N'} \min\{w_j, h_j, d_j\}$, t.y. jei nebėra laisvos erdvės naujam sluoksniui sudaryti.

Procedūra `Sluoksnio_storio_parinkimas()` naudoja du globalius kintamuosius, t.y:

- X^* – einamasis „geriausias“ sprendinys (aibė supakuotų dėžių ir jų pozicijų);
- V^* – supakuotų daiktų tūris, atitinkantis sprendinį X^*;

Pirmą kartą kreipiantis į procedūrą `Sluoksnio_storio_parinkimas()`, globalaus kintamojo V^* reikšmė turi būti priskirta $V^* = 0$, t.y. kreipinys turi atrodyti taip: `Sluoksnio_storio_parinkimas(`D`, 0, `N`)`. Čia D - konteinerio ilgis, N - pradinė dėžių aibė. Įvykdžius visas iteracijas, X^* grąžins euristinį sprendžiamo pakavimo uždavinio sprendinį ir supakuotų daiktų tūrį V^* (tikslo funkcijos reikšmę).

1.8.3. PROCEDŪROS „SP“ APIBENDRINTA VEIKIMO SCHEMA

Rekursinės procedūros `Sluoksnio_pildymas(`$\underline{w}$`, `$\underline{h}$`, `d'`, `U`, `N''`)` (toliau SP) schema, užrašyta pseudoprogramavimo kalba, pateikta 1.2 lentelėje. Čia perduodami parametrai yra:

- $\underline{w} \times \underline{h} \times d'$ - užpildomo sluoksnio dimensijos (išmatavimai);
- N'' – pakuojamų dėžių poaibis;
- U – šiame sluoksnyje jau supakuotų dėžių tūris;

Šioje procedūroje sluoksnis yra užpildomas vertikaliomis arba horizontaliomis juostomis. Prieš užpildant juostą, poaibiui N'' priklausančios dėžės yra pasukamos taip, kad išnaudotų kuo mažesnę dalį atitinkamai aukščio ar pločio dimensijų atžvilgiu.

Procedūra `Sluoksnio_pildymas()` naudoja du globalius kintamuosius, t.y:

- L – einamasis „geriausias“ sprendinys sluoksnyje (aibė supakuotų dėžių ir jų pozicijų);
- U^* – einamajame sluoksnyje supakuotų daiktų tūris, atitinkantis sprendinį L;

Siekiant pagreitinti procedūros `Sluoksnio_pildymas()` veikimą, yra skaičiuojama viršutinė riba (žr. (1.5) formulę).

$$U_b = U + \underline{w} \cdot \underline{h} \cdot d' \qquad \textbf{(1.5)}$$

Jei apskaičiuota riba rodo, kad, einant toliau ta šaka, nėra galimybės gauti geresnio užpildymo už dabartinį geriausią sluoksnio užpildymą U^*, tuomet yra atliekamas grįžimo atgal žingsnis.

1.2 lentelė

Procedūros `Sluoksnio_pildymas()` veikimo schema

Procedūra **Sluoksnio_pildymas**($\underline{w}$, $\underline{h}$, d' , U , N'')	
	Jeigu $U > U^*$, **tai** išsaugoti sprendinį aibėje L ir nustatyti $U^* \leftarrow U$;
	Priskirti $l \leftarrow \min_{j \in N''} \min\{w_j, h_j, d_j\}$;
	Jeigu $(\underline{w} < l)$ arba $(\underline{h} < l)$, **tai** grąžinti sprendinį;
Vertikalios juostos pakavimas	Atrinkti M_2 juostos storių $\{w'_1, w'_2, ..., w'_{M_2}\}$, turinčių didžiausius rangus;
	Vykdyti kiekvienam juostos storiui $w' \in \{w'_1, w'_2, ..., w'_{M_2}\}$
	Pradėti
	Pasukti dėžes ir išspręsti binarinį (0–1) kuprinės uždavinį su talpa $\underline{h}$;
	Tegu K – supakuotų dėžių poaibis;
	Vykdyti **Sluoksnio_pildymas**($\underline{w} - w'$, $\underline{h}$, d' , $U + \sum_{j \in K} v_j$, $N'' \setminus K$);
	Baigti
Horizontalios juostos pakavimas	Atrinkti M_2 juostos storių $\{h'_1, h'_2, ..., h'_{M_2}\}$, turinčių didžiausius rangus;
	Vykdyti kiekvienam juostos storiui $h' \in \{h'_1, h'_2, ..., h'_{M_2}\}$
	Pradėti
	Pasukti dėžes ir išspręsti binarinį (0–1) kuprinės uždavinį su talpa $\underline{w}$;
	Tegu K – supakuotų dėžių poaibis;
	Vykdyti **Sluoksnio_pildymas**($\underline{w}$, $\underline{h} - h'$, d' , $U + \sum_{j \in K} v_j$, $N'' \setminus K$);
	Baigti

1.8.4. JUOSTOS UŽPILDYMO PROCEDŪRA

Tiek horizontalių, tiek vertikalių juostų užpildymas yra analogiškas, todėl detaliau aprašomas tik vertikalios juostos užpildymo atvejis.

Užpildant vertikalią pločio w' ir ilgio d' juostą, yra sprendžiamas binarinis (0-1) kuprinės uždavinys, kurio sprendimui pasirinktas efektyvus `minknap` algoritmas [10]. Pirmiausia kiekviena j-toji dėžė yra pasukama viena iš galimų šešių padėčių taip, kad $w_j \le w'$, $d_j \le d'$ ir h_j būtų minimizuotas. Jeigu neįmanoma j-tosios dėžės pasukti taip, kad jos siena tilptų plokštumos dalyje $w' \times d'$, tuomet ši dėžė yra priskiriama „netinkamų“ dėžių aibei D. Priešingu atveju, t.y. jeigu dėžė yra „tinkama“ ($j \in N \setminus D$), tai įvedami pažymėjimai (1.6) ir (1.7).

$$a_j = h_j \tag{1.6}$$

$$c_j = v_j = w_j h_j d_j \tag{1.7}$$

Tuomet juostos užpildymo uždavinys formuluojamas taip: reikia maksimizuoti vertę (1.8), išlaikant apribojimą (1.9), kur $x_j \in \{0, 1\}$, $j \in N \setminus D$.

$$Z = \sum_{j \in N \setminus D} c_j x_j \tag{1.8}$$

$$\sum_{j \in N \setminus D} a_j x_j \le b \tag{1.9}$$

Čia $N \setminus D$ - tai „tinkamų“ dėžių aibė, o b - tai procedūros `Sluoksnio_pildymas()` parametras $\underline{h}$.

1.8.5. DĖŽIŲ SUPORAVIMAS

Kiekvienam sluoksnio storiui d', jei įmanoma, yra bandoma suporuoti dėžes po dvi, kad būtų gauti kiek įmanoma tinkamesnių išmatavimų dėžių junginiai.

Tarkime, kad j-toji dėžė gali būti pasukta taip, kad jos ilgio parametras d_j būtų kiek įmanoma didesnis, tačiau tenkinantis apribojimą $d_j \le d'$. Tuomet matas, nusakantis, kaip „gerai“ j-toji dėžė užpildo dimensiją d', yra apibrėžiamas (1.10) formule.

$$\alpha(j) = \frac{v_j}{w_j h_j d'} = \frac{d_j}{d'} \tag{1.10}$$

Norint suporuoti dėžes su indeksais i ir j ($i \ne j$), yra nagrinėjami visi galimi abiejų dėžių pasukimo variantai, kuriems tenkinama sąlyga (1.11). Kiekvieną kartą yra skaičiuojamas užpildymo santykis, nusakomas formule (1.12).

$$d_i + d_j \le d' \tag{1.11}$$

$$\beta(i, j) = \frac{v_i + v_j}{d' \cdot \max\{w_i, w_j\} \cdot \max\{h_i, h_j\}} \tag{1.12}$$

Jeigu $\beta(i, j) \le \alpha(j)$ visoms dėžėms i ir visiems galimiems pasukimams, tai tuomet j-toji dėžė yra paliekama viena, t.y. neporuojama su jokia kita dėže. Priešingu atveju yra parenkama i-toji dėžė su

atitinkamu pasukimu, kuriems yra gaunama didžiausia $\beta(i, j)$ reikšmė, ir yra sudaroma nauja dėžė k su atitinkamomis dimensijomis (1.13), (1.14) ir (1.15).

$$w_k = \max\{w_i, w_j\} \quad \textbf{(1.13)}$$

$$h_k = \max\{h_i, h_j\} \quad \textbf{(1.14)}$$

$$d_k = d' \quad \textbf{(1.15)}$$

Pradinės dėžės su indeksais i ir j yra laikinai pašalinamos iš uždavinio, siekiant išvengti jų dvigubo panaudojimo užpildant sluoksnius.

1.9. EURISTIKOS TYRIMO ATLIKIMO METODIKA

Pasirinktos euristikos tyrimas bus atliekamas sprendžiant įvairius konteinerio užpildymo uždavinius su programos vykdymo metu sugeneruotais duomenimis. Konkretaus tyrimo atveju dalis tam tikrų sprendimo algoritmo ir generavimo parametrų bus fiksuojama, o likusi dalis – keičiama. Šių parametrų žymėjimai ir aprašymai pateikti 1.3 lentelėje.

1.3 lentelė

Sprendimo algoritmo ir generavimo parametrų aprašymai

Nr.	Žymėjimas	Aprašymas
1	M_3	Medžio paieškos variantų trimačiame (3D) pakavime parametras. Kiekviename algoritmo žingsnyje nagrinėjamas skaičius M_3 galimų sluoksnių storių.
2	M_2	Medžio paieškos variantų dvimačiame (2D) pakavime parametras. Kiekviename algoritmo žingsnyje nagrinėjamas skaičius M_2 galimų juostų storių.
3	W	Konteinerio plotis
4	H	Konteinerio aukštis
5	D	Koteinerio ilgis
6	a	Dėžių išmatavimai generuojami pagal tolygųjį diskretųjį intervale $[a, b]$ skirstinį.
7	b	
8	k	Krovinio tipo parametras. Kai $k = 0$ - stipriai įvairiarūšis; kai $k = 1$ - vienarūšis; kai $k = 2, 3, ...$ - tai silpnai įvairiarūšis krovinys. Čia $k << n$, kur n - sugeneruotų dėžių skaičius.
9	T	Dėžės generuojamos tol, kol jų bendras tūris viršija T % konteinerio tūrio.
10	t_{LIMIT}	Nustatytas t_{LIMIT} sekundžių laiko limitas, kurį viršijus skaičiavimai yra nutraukiami.
11	R	Generuojamų pakavimo uždavinių (realizacijų) skaičius.
12	RT	Rangavimo taisyklė, taikoma sluoksnių ir juostų storiams parinkti. $RT := RTi$, jei pasirenkama i-toji rangavimo taisyklė ($i = \overline{1, 24}$).

Tirdami algoritmo veikimą, stebėsime šių rodiklių reikšmes:

- t - sprendimo laikas (sekundėmis), apskaičiuojamas kaip algoritmo vykdymo pabaigoje (t_{pab}) ir pradžioje (t_0) užfiksuotų laiko momentų skirtumas (žr. formulę (1.16)).

$$t = t_{pab} - t_0 \tag{1.16}$$

- U - gautas konteinerio užpildymas, išreikštas procentais (žr. formulę (1.17)).

$$U = \frac{\sum_{k \in \hat{N}} w_k h_k d_k}{W \cdot H \cdot D} \tag{1.17}$$

Čia $\hat{N}$ - supakuotų dėžių indeksų aibė.

Tarkime, kad turime eksperimentų rezultatus: $U_1, U_2, ..., U_R$ ir $t_1, t_2, ..., t_R$. Pagrindinių tiriamų skaitinių charakteristikų reikšmės yra apskaičiuojamos pagal (1.18) – (1.25) pateiktas formules.

1. $U_{\min}$ - minimalus konteinerio užpildymas (%):

$$U_{\min} = \min_{i=1,R} U_i \tag{1.18}$$

2. U_{vid} - vidutinis konteinerio užpildymas (%):

$$U_{vid} = \frac{1}{R} \sum_{i=1}^{R} U_i \tag{1.19}$$

3. $U_{\max}$ - maksimalus konteinerio užpildymas (%):

$$U_{\max} = \max_{i=1,R} U_i \tag{1.20}$$

4. σ_U - konteinerio užpildymo standartinis nuokrypis:

$$\sigma_U = \sqrt{\frac{1}{R-1} \sum_{i=1}^{R} (U_i - U_{vid})^2} \tag{1.21}$$

5. $t_{\min}$ - minimalus vieno pakavimo uždavinio sprendimo laikas (s):

$$t_{\min} = \min_{i=1,R} t_i \tag{1.22}$$

6. t_{vid} - vidutinis vieno pakavimo uždavinio sprendimo laikas (s):

$$t_{vid} = \frac{1}{R} \sum_{i=1}^{R} t_i \tag{1.23}$$

7. $t_{\max}$ - maksimalus vieno pakavimo uždavinio sprendimo laikas (s):

$$t_{\max} = \max_{i=1,R} t_i \tag{1.24}$$

8. σ_t - uždavinio sprendimo laiko standartinis nuokrypis:

$$\sigma_t = \sqrt{\frac{1}{R-1}\sum_{i=1}^{R}\left(t_i - t_{vid}\right)^2} \qquad \textbf{(1.25)}$$

Papildomi rodikliai, kurių prireiks atliekant kai kuriuos tyrimus, bus pateikti tiriamojoje dalyje.

Tyrimui naudojamos kompiuterinės technikos parametrai: procesoriaus tipas – Intel Core2 Duo T7250 (2.00 GHz), operacinė atmintis (RAM) – 2.00 GB, operacinės sistemos tipas – 32-bit.

2. TIRIAMOJI DALIS IR REZULTATAI

2.1. RANGAVIMO TAISYKLIŲ APIBRĖŽIMAI

Pasirinktos euristikos veikimą iš pradžių tirsime nagrinėdami tam tikras dažnumo funkcijų (trumpiau DF) bei prioriteto taisyklių (trumpiau PT) kombinacijas. Pasirinkta DF ir PT kombinacija nusakys atitinkamą sluoksnių ir juostų storių parinkimo rangavimo taisyklę (trumpiau RT).

2.1 lentelė

Rangavimo taisyklių pažymėjimai

Rangavimo taisyklė	Sluoksnio storio parinkimas					Juostos storio parinkimas			
	Dažnumo funkcija		Prioriteto taisyklė			Dažnumo funkcija		Prioriteto taisyklė	
	DF1	DF2	PT1	PT2	PT3	DF1	DF2	PT1	PT2
RT1	√		√			√		√	
RT2	√		√			√			√
RT3	√		√				√	√	
RT4	√		√				√		√
RT5	√			√		√		√	
RT6	√			√		√			√
RT7	√			√			√	√	
RT8	√			√			√		√
RT9	√				√	√		√	
RT10	√				√	√			√
RT11	√				√		√	√	
RT12	√				√		√		√
RT13		√	√			√		√	
RT14		√	√			√			√
RT15		√	√				√	√	
RT16		√	√				√		√
RT17		√		√		√		√	
RT18		√		√		√			√
RT19		√		√			√	√	
RT20		√		√			√		√
RT21		√			√	√		√	
RT22		√			√	√			√
RT23		√			√		√	√	
RT24		√			√		√		√

2.2 lentelė

Dažnumo funkcijų ir prioriteto taisyklių apibrėžimai

	Žymėjimas	Aprašymas
Dažnumo funkcija	DF1	Duotai reikšmei $k = \alpha, ..., \beta$ dažnumo funkcija $$f(k) = \sum_{i=1}^{n} 1_{\{w_i = k \vee h_i = k \vee d_i = k\}} \quad \textbf{(2.1)}$$ grąžina skaičių kartų, kiek reikšmė k pasikartojo tarp tame algoritmo žingsnyje pakuojamų dėžių dimensijų kaip bet kuri dėžės dimensija, t.y. kaip w_i, h_i arba d_i.
	DF2	Duotai reikšmei $k = \alpha, ..., \beta$ dažnumo funkcija $$f(k) = \sum_{i=1}^{n} 1_{\{\max\{w_i, h_i, d_i\} = k\}} \quad \textbf{(2.2)}$$ grąžina skaičių kartų, kiek reikšmė k pasikartojo tarp tame algoritmo žingsnyje pakuojamų dėžių dimensijų kaip maksimali dėžės dimensija, t.y. kaip $\max\{w_i, h_i, d_i\}$.
Prioriteto taisyklė	PT1	Pasirenkamos pačios didžiausios dimensijos k, kurioms dažnumo funkcija tenkina nelygybę $f(k) > 0$.
	PT2	Pasirenkamos didžiausios dimensijos su pseudo-didėjančia dažnumo funkcijos reikšme: pirmiausia yra pasirenkama didžiausia dimensija k_1, kuriai $f(k_1) \geq 1$. Tuomet pasirenkama didžiausia dimensija k_2, kuriai $f(k_2) \geq 2$ ir t.t. Taigi i-tuoju žingsniu pasirenkama didžiausia dimensija k_i, kuriai $f(k_i) \geq i$.
	PT3	Pasirenkamos dažniausiai pasitaikiusios dimensijos k, t.y. dimensijos su didžiausiomis dažnumo funkcijos $f(k)$ reikšmėmis.

Pastaba. Formulės (2.1) ir (2.2) apibrėžia dažnumo funkcijas, taikomas sluoksnio storio parinkimui, todėl paieška vykdoma tarp visų dėžės dimensijų w_i, h_i ir d_i. Tuo tarpu, parenkant juostų storius, sluoksnio storis d_i jau yra fiksuotas, ir dėžės yra pasuktos taip, kad tilptų šiame sluoksnyje. Dėl šios priežasties, skaičiuojant atitinkamas dažnumo funkcijos reikšmes, paieška yra vykdoma tik tarp likusių dimensijų, t.y. tarp w_i ir h_i.

Raidėmis α ir β yra žymimos atitinkamai mažiausia ir didžiausia tam tikrame algoritmo žingsnyje dėžių dimensijos, atsižvelgus į tai, kad tiek sluoksnis, tiek juosta turi tilpti konteineryje.

2.2. RANGAVIMO TAISYKLIŲ PALYGINIMAS

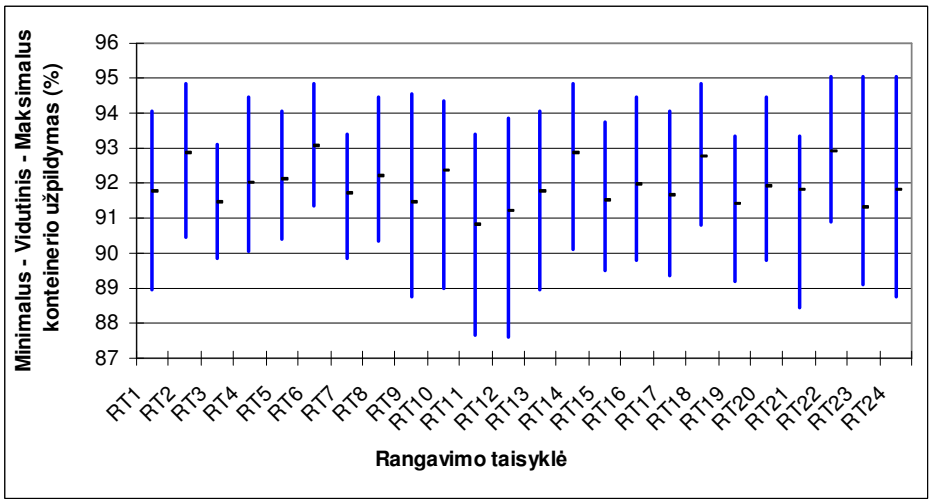
Rangavimo taisyklių, apibrėžtų 2.1 lentelėje (žr. 2.1 poskyrį), palyginimą atlikome fiksavę 2.3 lentelėje pateiktas sprendimo algoritmo ir generavimo parametrų reikšmes. Atliekant šį tyrimą buvo keičiama tik rangavimo taisyklė, t.y. RTi (arba vaizdžiau RT_i), kai $i = \overline{1, 24}$.

2.3 lentelė

Parametrų reikšmės, rangavimo taisyklių tyrimas

Parametras	RT	M_3	M_2	W	H	D	a	b	k	T	t_{LIMIT}	R
Reikšmė	Keičiama	4	8	100	100	100	1	50	0	120	120	100

1.9 poskyryje aptartų ir eksperimento metu gautų skaitinių charakteristikų reikšmių lentelė pateikta prieduose (žr. 1 priedo 1 lentelę). Grafinė gautų eksperimento rezultatų iliustracija pateikta 2.1 - 2.5 paveiksluose.

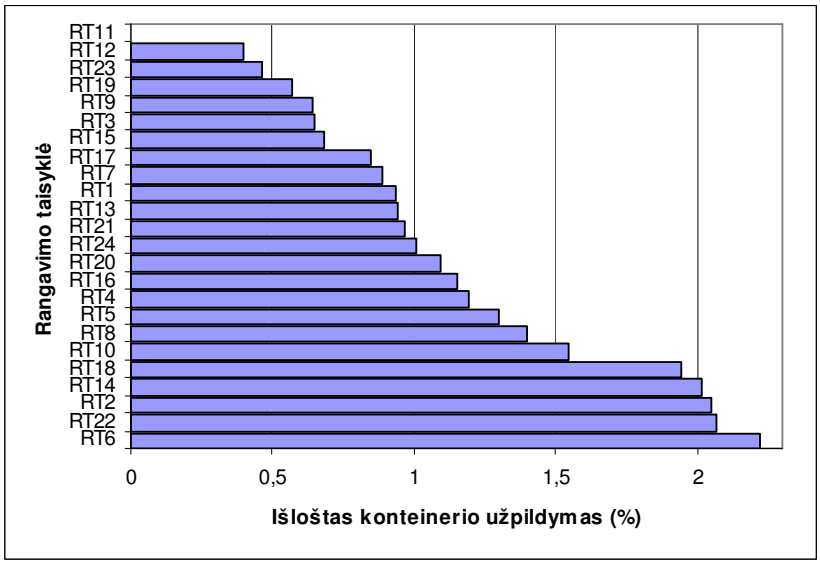


2.1 pav. Rangavimo taisyklių palyginimas pagal U_{min}, U_{vid} ir U_{max}

Didžiausias minimalus konteinerio užpildymas (U_{min}) gautas rangavimo taisyklei RT6, o mažiausias U_{min} – rangavimo taisyklei RT12. Skaitinės charakteristikos U_{min} kitimo intervalas šiuo atveju yra [87,59; 91,33]. Didžiausias vidutinis konteinerio užpildymas (U_{vid}) gautas rangavimo taisyklei RT6, o mažiausias U_{vid} – rangavimo taisyklei RT11. Skaitinės charakteristikos U_{vid} kitimo intervalas šiuo atveju yra [90,82; 93,03]. Didžiausias maksimalus konteinerio užpildymas (U_{max}) gautas rangavimo taisyklei RT22, o mažiausias U_{max} – rangavimo taisyklei RT3. Skaitinės charakteristikos U_{max} kitimo intervalas šiuo atveju yra [93,1; 95,07]. Konteinerio užpildymo standartinis nuokrypis σ_U, perbėgant visas rangavimo taisykles, įgijo reikšmes iš intervalo [0,75; 1,13].

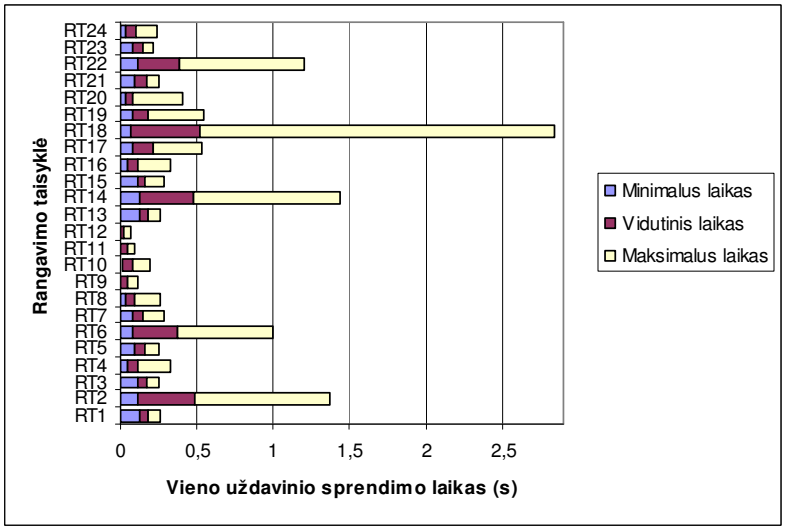
Išloštas konteinerio užpildymas (%) mažiausią U_{vid} duodančios RT (šiuo atveju - RT11) atžvilgiu pateiktas 2.2 paveiksle. Vadinasi, vietoj RT11 pasirinkę rangavimo taisykles RT6, RT22, RT2 arba RT14, galėtume gauti daugiau nei dviem procentiniais punktais didesnį vidutinį konteinerio užpildymą.

Maksimalus išloštas užpildymas gaunamas rangavimo taisyklei RT6 ir yra apytiksliai lygus 2,22%. Tuo tarpu rangavimo taisyklių RT12 ir RT23 atvejais išlošis nesiekia net 0,5%.



2.2 pav. Išloštas konteinerio užpildymas (%) mažiausią U_{vid} duodančios RT atžvilgiu

Taigi, remiantis išlošto konteinerio užpildymo reikšme, visas rangavimo taisykles galima būtų suskirstyti į kelis segmentus, pavyzdžiui: išlošis iki 1%; nuo 1% iki 2%; daugiau nei 2%.



2.3 pav. Rangavimo taisyklių palyginimas pagal t_{min}, t_{vid} ir t_{max}

Didžiausias minimalus sprendimo laikas (t_{min}) gautas rangavimo taisyklei RT1 ir yra lygus 0,125 s. Tuo tarpu didžiausi vidutinis ir maksimalus sprendimo laikai (atitinkamai t_{vid} ir t_{max}) gauti rangavimo taisyklei RT18, o mažiausi t_{vid} ir t_{max} – rangavimo taisyklei RT12. Skaitinės charakteristikos t_{vid} kitimo intervalas šiuo atveju yra [0,028; 0,522], o t_{max} atveju - [0,063; 2,839]. Kitaip sakant, eksperimento metu gautos mažiausia ir didžiausia t_{vid} reikšmės skiriasi beveik 19 kartų, o mažiausia ir didžiausia t_{max} reikšmės - beveik 45 kartus. Sprendimo laiko standartinis nuokrypis σ_t, perbėgant visas rangavimo taisykles, įgijo reikšmes iš intervalo [0,013; 0,473].

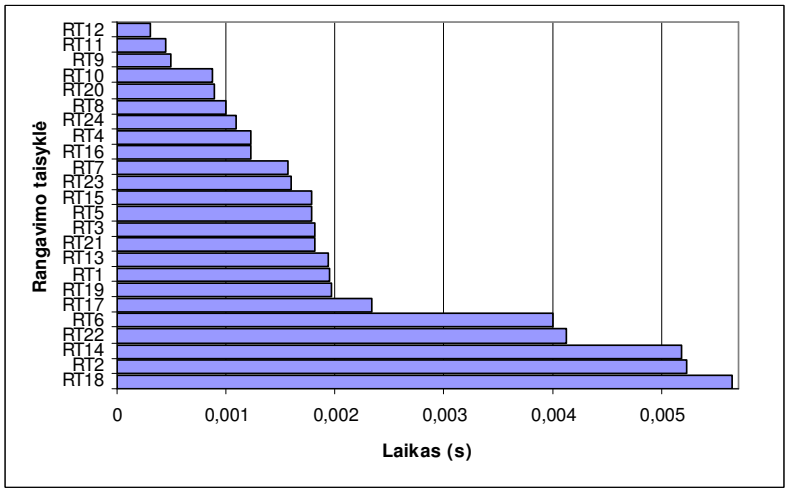
Pralošto laiko (sekundėmis) mažiausią t_{vid} gaunančios RT atžvilgiu (šiuo atveju - RT12) diagrama pateikta prieduose (žr. 1 priedo 1 pav.). Šiuo atveju didžiausią t_{vid} dalį pralošiame RT18 atveju, t.y. gauname beveik 0,5 sekundėmis ilgesnį vidutinį sprendimo laiką nei RT12 atveju.

Norint rangavimo taisykles palyginti atsižvelgus iškart į abu rodiklius, t.y. U_{vid} ir t_{vid}, vienas iš būdų – skaičiuoti santykinį dydį (2.3) arba jam atvirkštinį dydį - (2.4).

$$t_{proc} = \frac{t_{vid}}{U_{vid}} \quad \textbf{(2.3)}$$

$$U_{sek} = \frac{U_{vid}}{t_{vid}} \quad \textbf{(2.4)}$$

Čia t_{proc} reikšmė – tai laikas, kurio reikia konteinerio 1% užpildymui, o U_{sek} – per sekundę užpildomų procentų kiekis. Rodiklio t_{proc} įgytos reikšmės priklausomai nuo pasirinktos rangavimo taisyklės pavaizduotos 2.4 paveiksle, tuo tarpu U_{sek} – pateiktos prieduose (žr. 1 priedo 2 pav.)



2.4 pav. Laikas (s), kurio reikia konteinerio 1% užpildymui

Ilgiausią laiką, kurio reikia konteinerio tūrio vieno procento užpildymui, gavome su RT18. Šiuo atveju jis apytiksliai lygus 0,006 s. Apie 0,004 s. ir didesnius laikus taip pat gavome RT6, RT22, RT14 ir RT2 atvejais. Taigi, remiantis t_{proc} reikšmėmis, visas rangavimo taisykles galima būtų suskirstyti į keletą žymesnių segmentų, pavyzdžiui: kai laikas iki 0,0025 s.; kai laikas nuo 0,0025 s.

Prieduose taip pat yra pateikiama diagrama, vaizduojanti, kiek kartų laikas, kurio reikia konteinerio 1% užpildymui, gaunamas taikant pasirinktą RT_i, yra didesnis už bazinės RT (žr. 1 priedo 3 pav.). Bazine RT šiuo atveju yra laikoma ta rangavimo taisyklė, kuriai rodiklio t_{proc} reikšmė yra mažiausia, t.y. RT12.

Dar vienas būdas rangavimo taisykles palyginti atsižvelgiant iškart į abu rodiklius U_{vid} ir t_{vid} – tai sąlyginio santykinio dydžio skaičiavimas. Bendruoju atveju jis gali būti apskaičiuojamas pagal (2.5) formulę, kur RT_B – pagal tam tikrus kriterijus pasirinkta bazinė rangavimo taisyklė.

$$S_i(RT_B) = \frac{t_{vid}(RT_i) - t_{vid}(RT_B)}{U_{vid}(RT_i) - U_{vid}(RT_B)}, \quad i = \overline{1, 24} \; ir \; i \neq B \qquad \textbf{(2.5)}$$

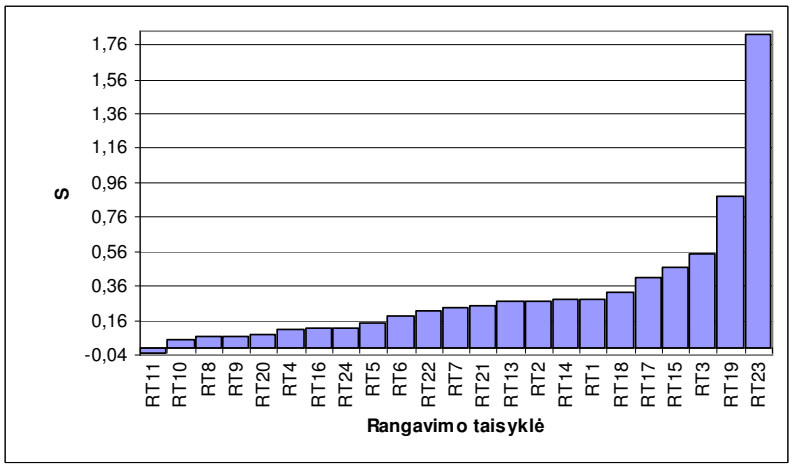
Toliau bazine laikysime RT12, t.y. tą rangavimo taisyklę, kuriai gaunama mažiausia rodiklio t_{vid} reikšmė. Taigi šiuo atveju (2.5) rodiklis turės (2.6) išraišką.

$$S_i = \frac{t_{vid}(RT_i) - \min\limits_{1 \le k \le 24}(t_{vid}(RT_k))}{U_{vid}(RT_i) - U_{vid}\left(RT_{\arg\min\limits_{1 \le k \le 24}(t_{vid}(RT_k))}\right)}, \quad i = \overline{1, 24} \; ir \; i \neq 12 \qquad \textbf{(2.6)}$$

Pastebėsime, kad (2.5) ir (2.6) rodiklius korektiškai apskaičiuoti galime tik tais atvejais, kai $U_{vid}(RT_i) \neq U_{vid}(RT_B)$, t.y. ragavimo taisyklėms RT_i ir RT_B $(i \neq B)$ gaunami konteinerio užpildymai nėra lygūs. Šio tyrimo metu atliktų eksperimentų rezultatai minėtą sąlygą tenkino, todėl (2.6) rodiklius apskaičiuoti galėjome.

Nesunku pastebėti, kad, taip apibrėžus, (2.6) formulės skaitiklis bus visada neneigiamas, t.y. RT12 bus visuomet ne blogesnė už RT_i sprendimo laiko atžvilgiu. Tokiu būdu teigiama rodiklio S_i reikšmė parodo, jog $U_{vid}(RT_i) > U_{vid}(RT_{12})$, arba kitaip: RT_i yra geresnė už RT12 gauto konteinerio užpildymo atžvilgiu. Atitinkamai neigiama rodiklio S_i reikšmė parodo, jog $U_{vid}(RT_i) < U_{vid}(RT_{12})$, arba kitaip: RT_i yra blogesnė už RT12 gauto konteinerio užpildymo atžvilgiu.

Vadinasi, jei $S_i < 0$, tai su RT_i buvo gautas ne tik blogesnis konteinerio užpildymas lyginant su RT12, bet dar ir sugaišta daugiau skaičiavimų laiko. Tai esminis neefektyvumo rodiklis, leidžiantis atitinkamą rangavimo taisyklę RT_i eliminuoti iš tolimesnio tyrimo. Pavyzdžiui, šio tyrimo metu neigiamą S_i reikšmę gavome rangavimo taisyklei RT11 (žr. 2.5 pav.).



2.5 pav. Laikas (s), kurio reikia papildomo 1% užpildymui (kai S > 0)

Tuo tarpu jei $S_i > 0$, tai rodiklį S_i galima traktuoti kaip laiką, kurio reikia papildomo 1% užpildymui bazinės rangavimo taisyklės atžvilgiu. Rangavimo taisyklė yra tuo geresnė, kuo šis laikas yra mažesnis. Mažiausią minėtą laiką gavome rangavimo taisyklėms RT10, RT8, RT9 ir RT20, didžiausią –

RT23. Skaitinė charakteristika S_i, perbėgant visas rangavimo taisykles, įgijo teigiamas reikšmes iš intervalo [0,045; 1,825], t.y. mažiausia ir didžiausia teigiama reikšmė skyrėsi beveik 40 kartų.

Tolimesniems tyrimams buvo pasirinkta ta rangavimo taisyklė, kuriai buvo gautas didžiausias vidutinis konteinerio užpildymas (U_{vid}), t.y. RT6.

2.3. PAIEŠKOS VARIANTŲ 2D IR 3D PAKAVIME TYRIMAS

Parametrų M_2 ir M_3 tyrimo pirmąją dalį atlikome fiksavę 2.4 lentelėje pateiktas sprendimo algoritmo ir generavimo parametrų reikšmes.

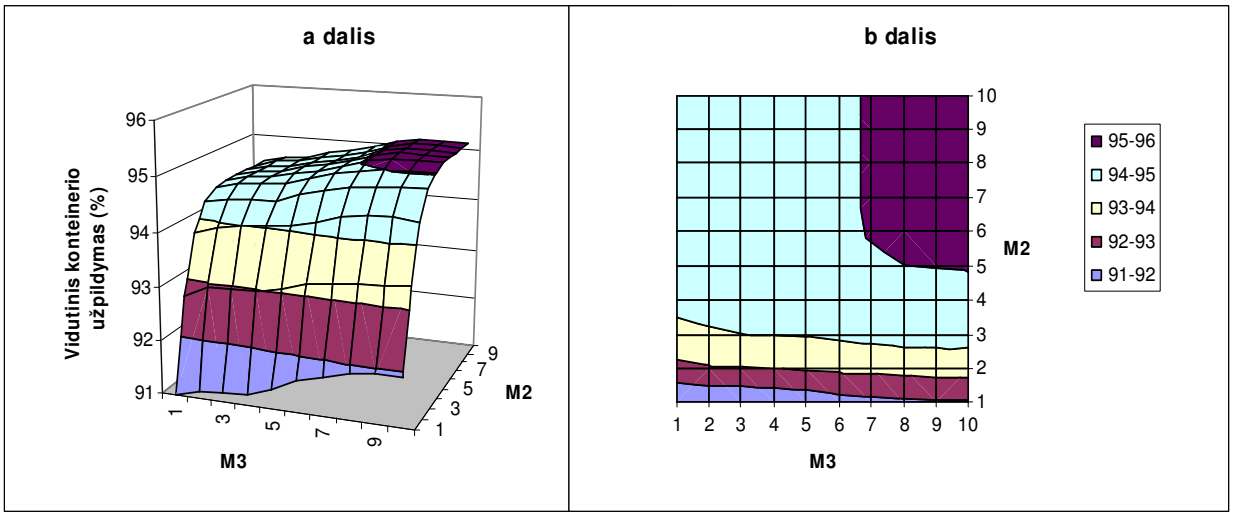
2.4 lentelė

Parametrų reikšmės, M_2 ir M_3 tyrimas

Parametras	RT	M_3	M_2	W	H	D	a	b	k	T	t_{LIMIT}	R
Reikšmė	RT6	Keičiama	Keičiama	100	100	100	1	50	0	150	220	100

Atlikdami eksperimentus, pastebėjome, kad, net ir nežymiai padidinus paieškos lygio parametrų M_2 ir M_3 reikšmes, skaičiavimų laikas gerokai pailgėdavo. Todėl, atsižvelgę į šį faktą bei siekį, kad, atliekant eksperimentą, būtų spėta išspręsti $R = 100$ pakavimo uždavinių, šiuo atveju pasirinkome didesnį laiko limitą $t_{LIMIT} = 220\,s$.

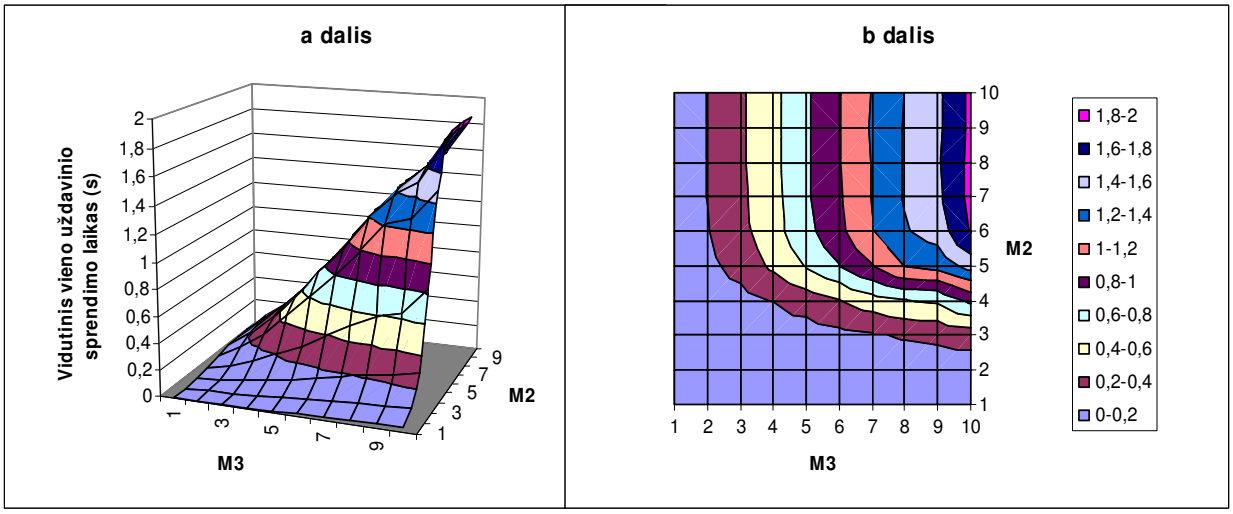
Šio eksperimento metu gautų rodiklių U_{vid} ir t_{vid} reikšmių lentelės pateiktos prieduose (žr. 2 priedo 2 ir 3 lenteles). Gauti eksperimento rezultatai taip pat pavaizduoti grafiškai 2.6 ir 2.7 paveiksluose. Abiejų paveikslų a dalyse pateiktos trimačių paviršių iliustracijos, o b dalyse – šių paviršių projekcijos į plokštumą (M_2, M_3) spalvinės iliustracijos.



2.6 pav. Vidutinis konteinerio užpildymas esant skirtingoms M_2 ir M_3 kombinacijoms

Skaitinės charakteristikos U_{vid} reikšmės, perbėgant įvairias parametrų M_2 ir M_3 kombinacijas, kito nuo 91,004% (kai $M_2 = 1$ ir $M_3 = 1$) iki 95,129% (kai $M_2 = 7$ ir $M_3 = 8$), t.y. kito net keturiais procentiniais punktais. Galima pastebėti, kad atlikto tyrimo atveju M_2 ir M_3 reikšmių padidinimas ne

visuomet garantavo geresnį konteinerio užpildymą U_{vid}. Pavyzdžiui, kai $M_2 = 7$ ir $M_3 = 8$, tai $U_{vid} \approx 95{,}129\%$; kai $M_2 = 10$ ir $M_3 = 10$, tai $U_{vid} \approx 95{,}101\%$.



2.7 pav. Vidutinis vieno uždavinio sprendimo laikas esant skirtingoms M_2 ir M_3 kombinacijoms

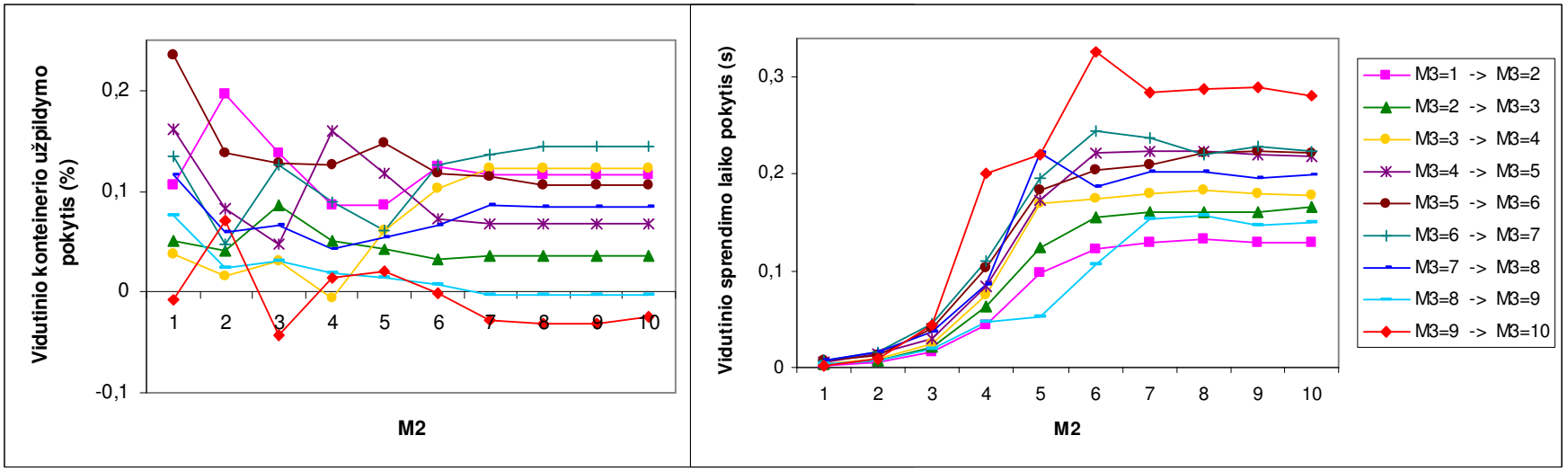
Skaitinės charakteristikos t_{vid} reikšmės, perbėgant įvairias parametrų M_2 ir M_3 kombinacijas, įgijo reikšmes iš intervalo [0,00125; 1,85641]. Taigi, keičiant M_2 ir M_3 reikšmes mūsų pasirinktose ribose, intervalo kraštinių reikšmių atžvilgiu t_{vid} padidėjo net 1485 kartus.

Taip pat galima nagrinėti U_{vid} ir t_{vid} pokyčius, gaunamus padidinus atitinkamai parametro M_3 arba parametro M_2 reikšmę vienetu. Toliau žymėsime: $\Delta_{M3}U_{vid}$ ir $\Delta_{M3}t_{vid}$ - pokyčius, atitinkančius M_3 reikšmės keitimą (žr. 2.8 pav. ir 2 priedo 4-5 lenteles); $\Delta_{M2}U_{vid}$ ir $\Delta_{M2}t_{vid}$ - pokyčius, atitinkančius M_2 reikšmės keitimą (žr. 2.9 pav. ir 2 priedo 6-7 lenteles).

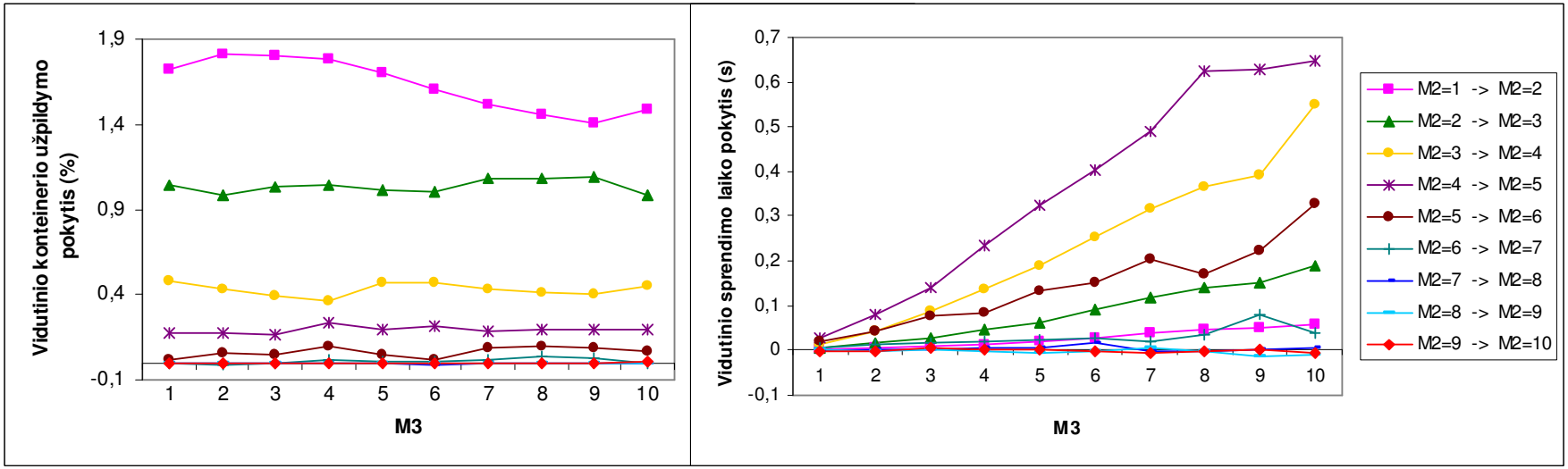
Iš 2.9 paveikslo pastebime, kad, vos tik pradėjus didinti M_2 reikšmę (pavyzdžiui, nuo 1 iki 2), nepriklausomai nuo fiksuotos parametro M_3 reikšmės pokytis $\Delta_{M2}U_{vid}$ buvo didžiausias. Perbėgant visas M_3 reikšmes, $\Delta_{M2}U_{vid}$ šiuo atveju kito nuo 1,4059% iki 1,8102%. Tuo tarpu toliau didinant M_2 reikšmę, konteinerio užpildymo išlošio dydis ($\Delta_{M2}U_{vid}$) palaipsniui mažėjo. Tam tikrais atvejais gautas pokytis buvo lygus 0 arba neigiamas, t.y., padidinę M_2 reikšmę vienetu, geresnio konteinerio užpildymo negavome.

Tuo tarpu pokytis $\Delta_{M2}t_{vid}$, didinant M_2 reikšmę, tik iš pradžių didėjo. Pavyzdžiui, didžiausias pokytis, perbėgant visas M_3 reikšmes, buvo gaunamas, kai M_2 reikšmė keičiama nuo 4 iki 5 (šiuo atveju $\Delta_{M2}t_{vid}$ kito nuo 0,02839 s. iki 0,6485 s.). Toliau didinant M_2 reikšmę, $\Delta_{M2}t_{vid}$ pradėjo mažėti, ir galiausiai vidutinė vieno uždavinio sprendimo trukmė stabilizavosi.

Didinant M_3 reikšmę, kryptingų $\Delta_{M3}U_{vid}$ ir $\Delta_{M3}t_{vid}$ kitimo tendencijų nepastebime (žr. 2.8 pav.)

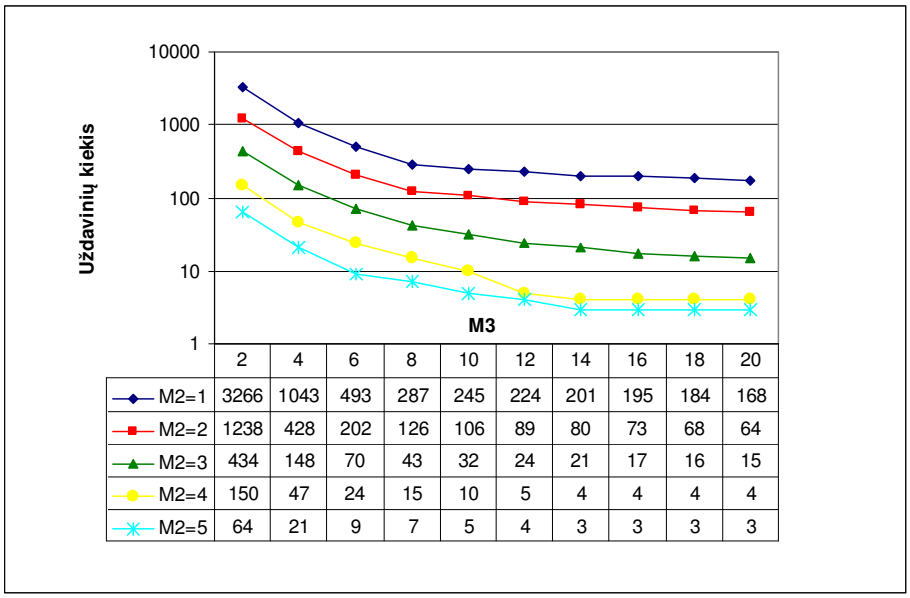


2.8 pav. U_{vid} ir t_{vid} pokyčiai, padidinus parametro M_3 reikšmę vienetu



2.9 pav. U_{vid} ir t_{vid} pokyčiai, padidinus parametro M_2 reikšmę vienetu

2.10 paveiksle pateikti išspręstų pakavimo uždavinių kiekio priklausomybės nuo pasirinktos parametrų M_2 ir M_3 reikšmių kombinacijos rezultatai. Atliekant šią tyrimo dalį, buvo fiksuojamos tos pačios 2.4 lentelėje pateiktų parametrų reikšmės, išskyrus pasirinktą kitą laiko limitą $t_{LIMIT} = 10\ s.$ ir sprendžiamų pakavimo uždavinių skaičių R, kuris šiuo atveju išvis nėra fiksuojamas. Norint vaizdžiau pateikti rezultatus, grafiko ordinačių ašyje buvo atidėta logaritminė skalė (žr. 2.10 pav.).



	2	4	6	8	10	12	14	16	18	20
M2=1	3266	1043	493	287	245	224	201	195	184	168
M2=2	1238	428	202	126	106	89	80	73	68	64
M2=3	434	148	70	43	32	24	21	17	16	15
M2=4	150	47	24	15	10	5	4	4	4	4
M2=5	64	21	9	7	5	4	3	3	3	3

2.10 pav. Per 10 sekundžių išspręstų pakavimo uždavinių kiekis

Parametro M_3 reikšmę padidinus nuo 2 iki 20, t.y. 10 kartų, išspręstų pakavimo uždavinių kiekis sumažėdavo vidutiniškai 25,31 karto. Tuo tarpu parametro M_2 reikšmę padidinus nuo 1 iki 5, t.y. 5 kartus, išspręstų pakavimo uždavinių kiekis sumažėdavo vidutiniškai 55 kartus.

2.4. PAKUOJAMŲ DĖŽIŲ KIEKIO ĮTAKOS TYRIMAS

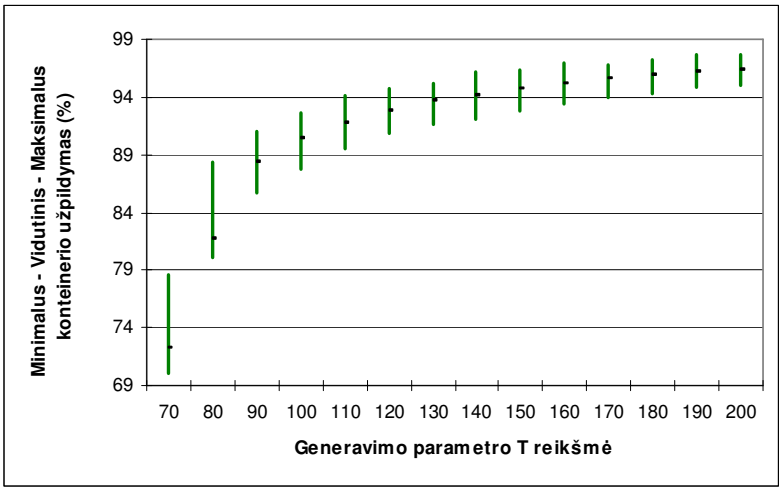
Atliekant pakuojamų dėžių kiekio įtakos tyrimą, fiksavome 2.5 lentelėje pateiktas sprendimo algoritmo ir generavimo parametrų reikšmes. Šiuo atveju buvo keičiamas tik parametras T. Priminsime, kad pagal naudojamą generavimo modelį kiekvienam pakavimo uždaviniui dėžės yra generuojamos tol, kol jų bendras tūris viršija T % konteinerio tūrio.

2.5 lentelė

Parametrų reikšmės, pakuojamų dėžių kiekio tyrimas

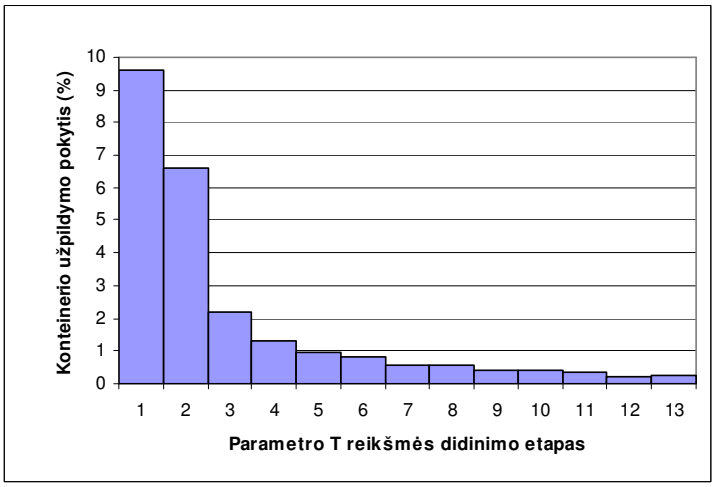
Parametras	RT	M_3	M_2	W	H	D	a	b	k	T	t_{LIMIT}	R
Reikšmė	RT6	4	8	100	100	100	1	50	0	Keičiama	120	100

Eksperimento metu gautų skaitinių charakteristikų reikšmių lentelė pateikta prieduose (žr. 3 priedo 8 lentelę). Grafinė gautų konteinerio užpildymo rezultatų iliustracija pateikta 2.11 paveiksle, o sprendimo laiko – 3 priedo 4 paveiksle.



2.11 pav. Gautos U_{min}, U_{vid} ir U_{max} reikšmės priklausomai nuo T reikšmės

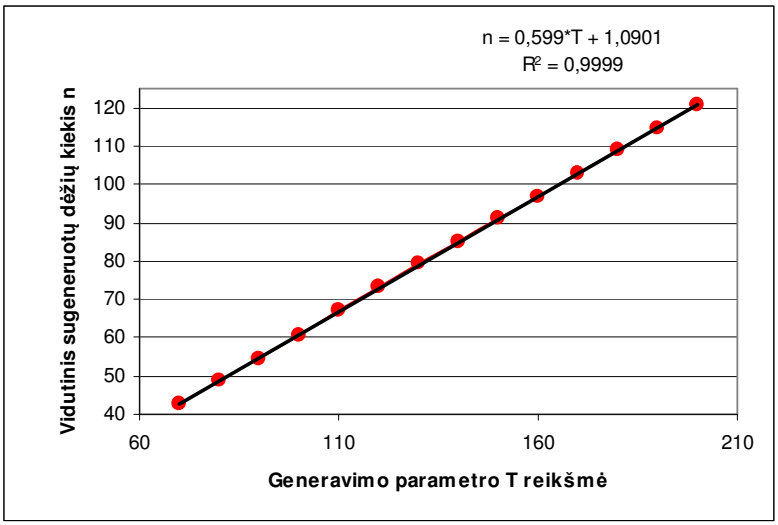
Iš 2.11 paveikslo matome, kad kuo daugiau dėžių (konteinerio tūrio prasme) yra pakuojama, tuo geresnių konteinerio užpildymo rezultatų galime pasiekti. Vis dėlto, nuosekliai didinant parametro T reikšmę (pvz.: nuo 70% iki 80% (1 etapas); nuo 80% iki 90% (2 etapas) ir t.t.), konteinerio užpildymo išlošio dydis palaipsniui mažėja (žr. 2.12 pav.). Keičiant T reikšmę mūsų pasirinktose ribose, užpildymas U_{vid} padidėjo nuo 70,01% iki 96,41%, tuo tarpu užpildymo pokytis sumažėjo nuo 9,57% iki 0,26%.



2.12 pav. U_{vid} pokyčio kitimo tendencija didinant parametrą T

Tiriant pakavimo uždavinių sprendimo laiką, yra tikslinga nagrinėti ne tik jo priklausomybę generavimo parametro T atžvilgiu, bet ir vidutinio sugeneruotų dėžių skaičiaus n atžvilgiu. Dėl šios priežasties pirmiausia įvertiname dydžių T ir n tarpusavio priklausomybę, atitinkančią fiksuotas likusių generavimo parametrų, t.y.: W, H, D, a, b ir k, reikšmes (žr. 2.13 pav.).

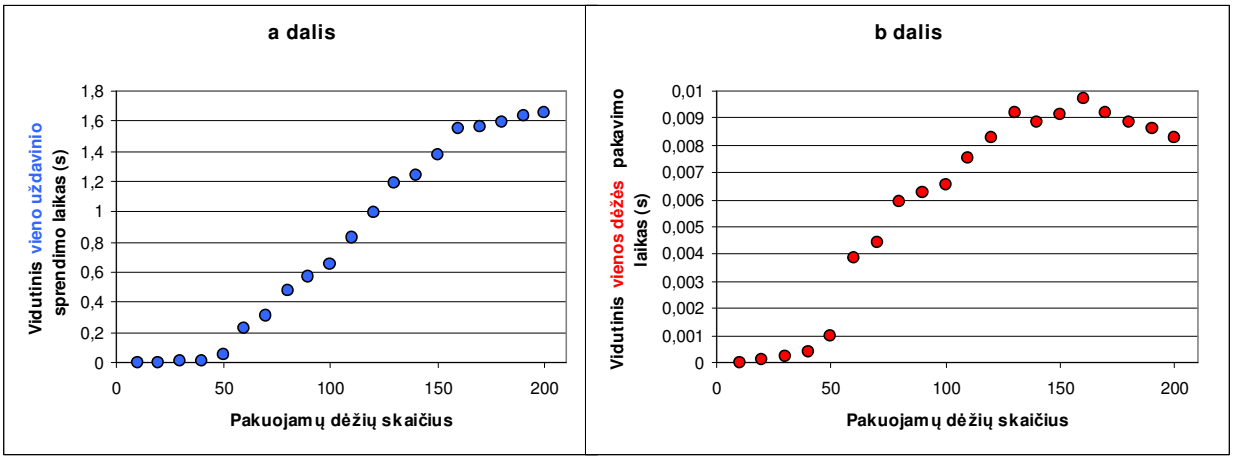
Iš 2.13 paveikslo pastebime, kad praktiškai gauta tiesinė priklausomybė. Taigi, pasinaudodami gauta tiesinio trendo lygtimi ir jai atvirkštine, galime nesunkiai pereiti nuo dydžio T prie dydžio n ir atvirkščiai.



2.13 pav. Generavimo parametro T ir sugeneruotų dėžių kiekio n priklausomybė

Vidutinis vieno uždavinio sprendimo laikas (t_{vid}) pakuojamų dėžių skaičiaus n atžvilgiu atidėtas 2.14 paveikslo 1-oje dalyje. Tuo tarpu 2-oje dalyje yra atidėtas vidutinis vienos dėžės pakavimo laikas (trumpiau $t_{vid/n}$), apskaičiuojamas pagal (2.7) formulę.

$$t_{vid/n} = \frac{t_{vid}}{n} \tag{2.7}$$



2.14 pav. Vidutinis sprendimo laikas (sekundėmis)

Nuosekliai didinant pakuojamų dėžių skaičių n nuo 10 iki 200 žingsniu 10, vidutinis vieno uždavinio sprendimo laikas pailgėjo nuo $1{,}6 \cdot 10^{-4}$ s. (kai $n = 10$) iki 1,65221 s. (kai $n = 200$). Atitinkamai vidutinis vienos dėžės pakavimo laikas padidėjo nuo $1{,}6 \cdot 10^{-5}$ s. (kai $n = 10$) iki $8{,}26 \cdot 10^{-3}$ s. (kai $n = 200$). Vadinasi, dėžių skaičiui padidėjus 20 kartų, vienos dėžės pakavimo laikas pailgėjo net 516 kartų.

2.5 KROVINIO TIPO ĮTAKOS TYRIMAS

Tirsime tokius krovinio (pakuojamų dėžių aibės) tipus:

1. Stipriai įvairiarūšis krovinys, sudarytas iš labai daug skirtingų rūšių dėžių. Pasirenkame dėžių tipų skaičiaus parametrą $k = 0$.
2. Silpnai įvairiarūšis krovinys, sudarytas iš keletos skirtingų rūšių dėžių. Pasirenkame dėžių tipų skaičiaus parametrą $k = 10$.
3. Vienarūšis krovinys, sudarytas iš vienos rūšies dėžių. Pasirenkame dėžių tipų skaičiaus parametrą $k = 1$.

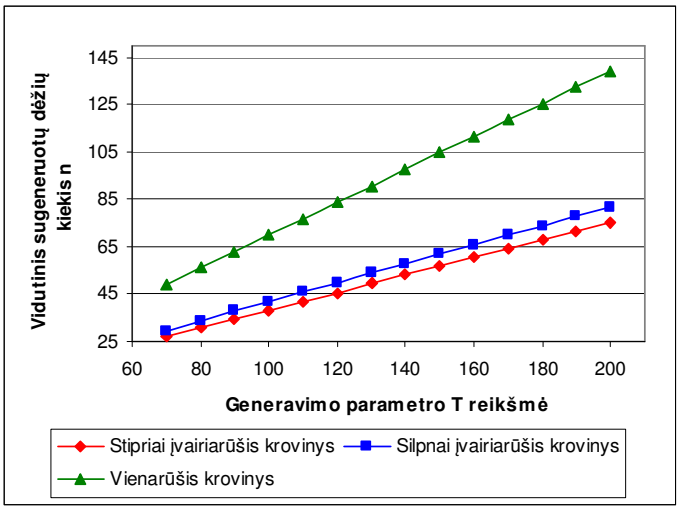
Atliekant krovinio tipo įtakos tyrimą, fiksavome 2.6 lentelėje pateiktas sprendimo algoritmo ir generavimo parametrų reikšmes. Šiuo atveju buvo keičiamas parametras k.

2.6 lentelė

Parametrų reikšmės, krovinio tipo tyrimas

Parametras	RT	M_3	M_2	W	H	D	a	b	k	T	t_{LIMIT}	R
Reikšmė	RT6	4	8	100	100	100	10	50	Keičiama	90	3600	100

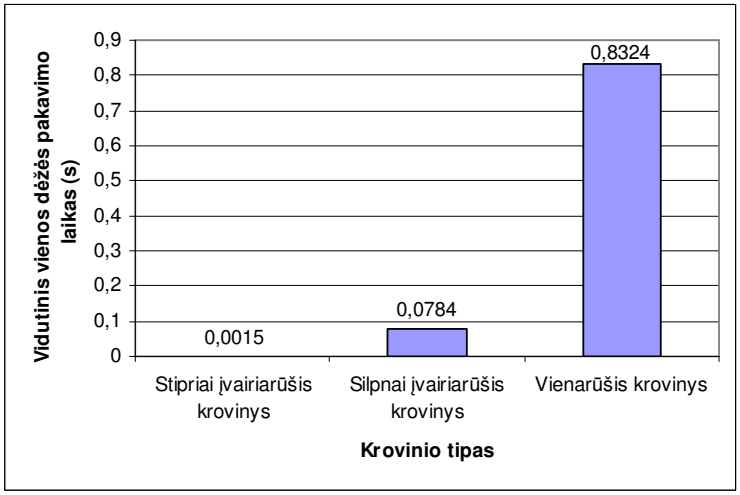
Prieš tiriant pačios euristikos veikimą, buvo atliktas vidutinio sugeneruotų dėžių kiekio n ir generavimo parametro T priklausomybės tyrimas, analogiškas aptartam 2.4 poskyryje. Šiuo atveju tyrimas buvo atliktas atskirai kiekvienam krovinio tipui (stipriai, silpnai įvairiarūšis ir vienarūšis). Šio tyrimo tikslas – palyginti pagal 2.6 lentelės parametrus generuojamus trijų tipų (atitinkamai $k = 0$, $k = 10$ ir $k = 1$) pakavimo uždavinius kiekybiniu požiūriu. Šios eksperimento dalies metu gauti rezultatai pateikti 4 priedo 9 lentelėje bei 2.15 paveiksle.



2.15 pav. Vidutinis sugeneruotų dėžių kiekis esant skirtingų tipų kroviniams

Pastebime, kad, generuojant stipriai ir silpnai įvairiarūšio krovinio duomenis, vidutinis sugeneruotų dėžių kiekis n yra gerokai mažesnis nei vienarūšio krovinio atveju, nepriklausomai nuo T

reikšmės. Dėl šios priežasties skirtingo krovinio tipo pakavimo uždavinių, gautų fiksavus tą pačią parametro T reikšmę, rodiklio t_{vid} tiesioginis palyginimas nėra tikslingas. Kur kas tikslingiau palyginimą atlikti pagal vidutinį vienos dėžės pakavimo laiką, t.y. $t_{vid/n}$, apibrėžtą 2.4 poskyryje. Vienos dėžės pakavimo laiko ($t_{vid/n}$) grafinė iliustracija pateikta 2.16 diagramoje.



2.16 pav. Vidutinis vienos dėžės pakavimo laikas (s) esant skirtingų tipų kroviniams

Pastebime, kad $t_{vid/n}$ reikšmė vienarūšio krovinio atveju buvo beveik 11 kartų didesnė nei silpnai įvairiarūšio krovinio atveju ir beveik 555 kartus didesnė nei stipriai įvairiarūšio krovinio atveju. Kiti eksperimento rezultatai, gauti fiksavus 2.6 lentelėje nurodytas parametrų reikšmes, pateikti 2.7 lentelėje. Čia r – tai vidutinis nesupakuotų dėžių skaičius.

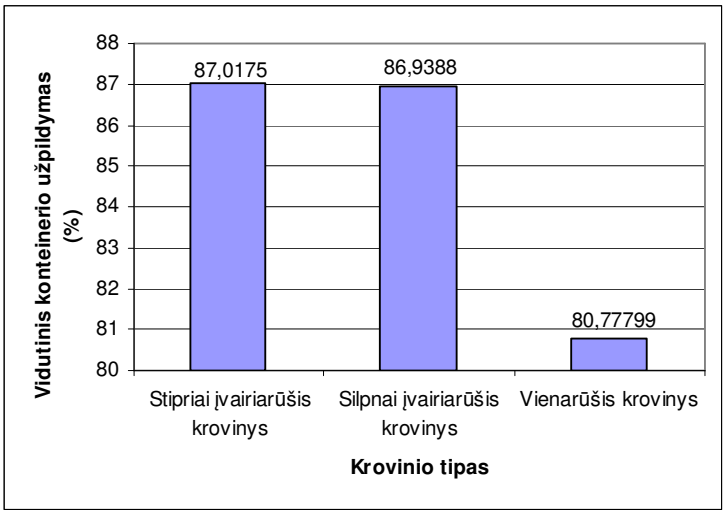
2.7 lentelė

Pakavimo rodiklių palyginimas pagal krovinio tipą

Rodiklis	Krovinio tipas		
	Stipriai įvairiarūšis krovinys	Silpnai įvairiarūšis krovinys	Vienarūšis krovinys
	k = 0	k = 10	k = 1
n	34,64	37,84	62,95
r	3,6	2,67	4,06
U_{vid}	87,0175	86,9388	80,77799
$U_{\min}$	83,4557	72,5407	46,956
$U_{\max}$	90,1419	91,4476	92,16
σ_U	1,21472	2,84295	10,05202
t_{vid}	0,05305	2,96823	52,3976
$t_{\min}$	0,015	0,031	0
$t_{\max}$	0,265	205,968	2605,48
σ_t	0,0384073	20,5607	318,6533

2.17 paveiksle pavaizduotas gautas konteinerio užpildymas U_{vid}, kai buvo sprendžiami pakavimo uždaviniai su skirtingų tipų kroviniais. Šiuo atveju kiekvienam uždaviniui dėžės buvo generuojamos tol,

kol jų bendras tūris viršijo $T = 90\%$ konteinerio tūrio. Tokia parametro T reikšmė buvo pasirinkta tikslingai. Pirmiausia dėl to, kad didesnės T reikšmės, t.y. $T = 100\%$ ar daugiau, parinkimas praktiškai nulemdavo nebepriimtiną uždavinio sprendimo laiko šuolį vienarūšio krovinio atveju. Iš kitos pusės, parinkus mažesnę T reikšmę, t.y. $T = 80\%$ ar mažiau, sprendžiant uždavinius atsirasdavo atvejų, kai visiškai visos dėžės būdavo supakuojamos į konteinerį. Minėtais atvejais gautas didesnis konteinerio užpildymas U_{vid} nebūtinai rodytų geresnius pakavimo rezultatus. Siekiant išvengti šios dviprasmybės, buvo pasirinktas $T = 90\%$, su kuriuo vienoje realizacijoje likdavo vidutiniškai nuo 2,67 iki 4,06 nesupakuotų dėžių.



2.17 pav. Vidutinis konteinerio užpildymas esant skirtingų tipų kroviniams

Pastebime, kad stipriai ir silpnai įvairiarūšio krovinio atvejais gautas labai panašus konteinerio užpildymas (skirtumas tarp jų mažesnis nei 0,079%). Tuo tarpu vienarūšio krovinio atveju gautas vidutinis konteinerio užpildymas buvo net 6,24% mažesnis nei stipriai įvairiarūšio krovinio atveju.

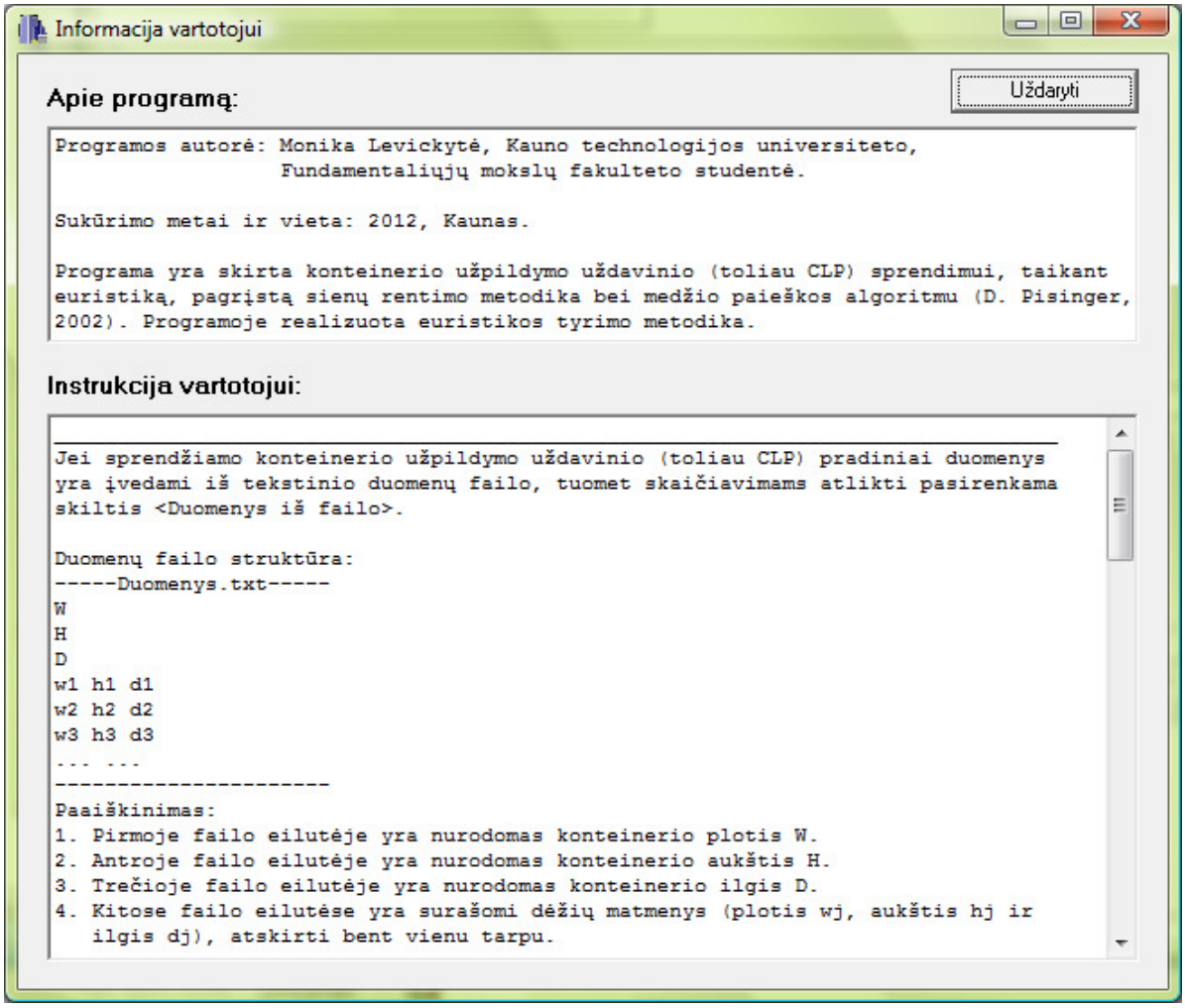
3. PROGRAMINĖ REALIZACIJA IR INSTRUKCIJA VARTOTOJUI

3.1. BENDRIEJI NURODYMAI

Programa parašyta *C++* programavimo kalba *C++ Builder 5* aplinkoje. Tokia programavimo kalba ir terpė buvo pasirinktos dėl kelių priežasčių. Pirmiausia dėl to, kad *C++ Builder* aplinka labai patogi vartotojo sąsajos kūrimui: nemažas programos komponentų sąrašas, galimybė keisti jų savybes ir užprogramuoti jų reakcijas į įvykius, kurie gali įvykti programos vykdymo metu. Be to, atliekant euristikos tyrimą, buvo operuojama su dideliais duomenų kiekiais. Dėl šios priežasties buvo naudojamas ne statinis, o dinaminis duomenų masyvas. C++ programavimo kalba labai patogi dirbti su dinamine atmintimi: jos išskyrimu, tvarkymu bei grąžinimu. Programos kodo fragmentai pateikiami 6 priede.

Naudotos kompiuterinės technikos parametrai: procesoriaus tipas – Intel Core2 Duo T7250 (2.00 GHz), operacinė atmintis (RAM) – 2.00 GB, operacinė sistema – Windows Vista Home Premium, operacinės sistemos tipas – 32-bit.

Pagrindinė programa, duomenų, rezultatų ir kiti svarbūs failai saugomi kataloge *Magistro darbas*. Programa paleidžiama spragtelėjus vykdomąjį failą *Project1.exe*. Paleidus jį, vartotojas gali pasirinkti vieną iš trijų puslapių: *Generuoti duomenys*, *Duomenys iš failo* ir *Testavimas*. Rekomenduojama pirmiausia perskaityti informaciją vartototojui, kuri yra pasiekiama viršutiniame meniu spragtelėjus ant skilties *Informacija*. Atsidariusiame lange yra pateikiama informacija apie programos autorių, sprendžiamą uždavinį bei detali instrukcija, kaip tinkamai naudotis programa (žr. 3.1 pav.).

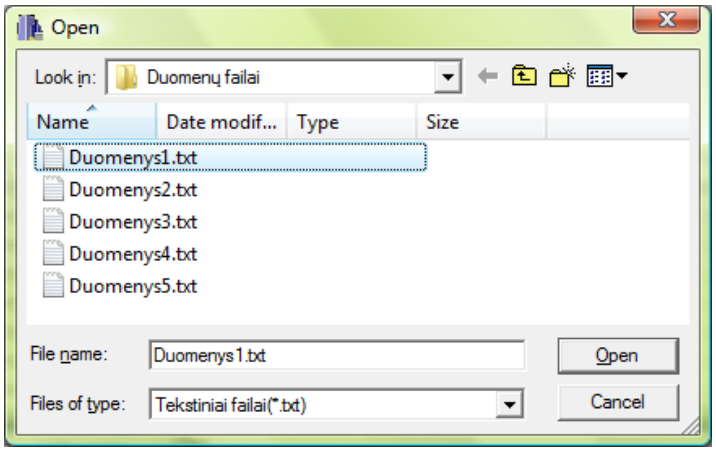


3.1 pav. Informacijos vartotojui langas

Visa instrukcija taip pat pateikiama ir faile *Instrukcija.txt*, esančiame pagrindiniame kataloge *Magistro darbas* (žr. 5 priedą).

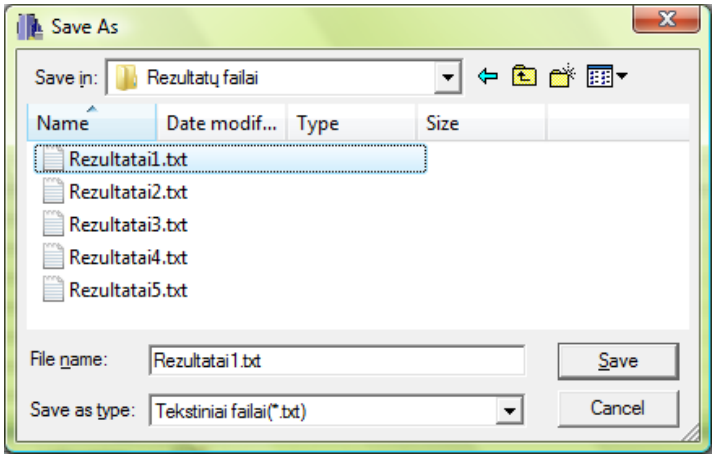
Duomenų ir rezultatų failų parinkimui panaudoti komponentai *OpenDialog* (žr. 3.2 pav.) ir *SaveDialog* (žr. 3.3 pav). Duomenų failas turi būti parenkamas tik tuomet, kai sprendžiamam konteinerio užpildymo uždaviniui pradiniai duomenys turi būti nuskaitomi iš tekstinio duomenų failo,

t.y. kai pasirenkama skiltis *Duomenys iš failo*. Duomenų failo struktūra ir jos paaiškinimas pateikiami faile *Instrukcija.txt* (žr. 5 priedą), *Informacija vartotojui* lange (žr. 3.1 pav.) bei detaliau aptarti 3.3 poskyryje. Už tinkamą duomenų failo sudarymą yra atsakingas vartotojas.



3.2 pav. Duomenų failo parinkimo langas

Rezultatų failas turi būti parenkamas tais atvejais, kai pasirenkamos *Duomenys iš failo* arba *Generuoti duomenys* skiltys. Parinktas rezultatų failas iškart yra išvalomas. Tą reikėtų atsiminti, nurodant rezultatų failą, kuriame jau yra saugoma svarbi informacija. Tokiu būdu bus išvengta bereikalingo duomenų sugadinimo. Kol rezultatų failas neparinktas, neleidžiama pradėti skaičiavimų.



3.3 pav. Rezultatų failo parinkimo langas

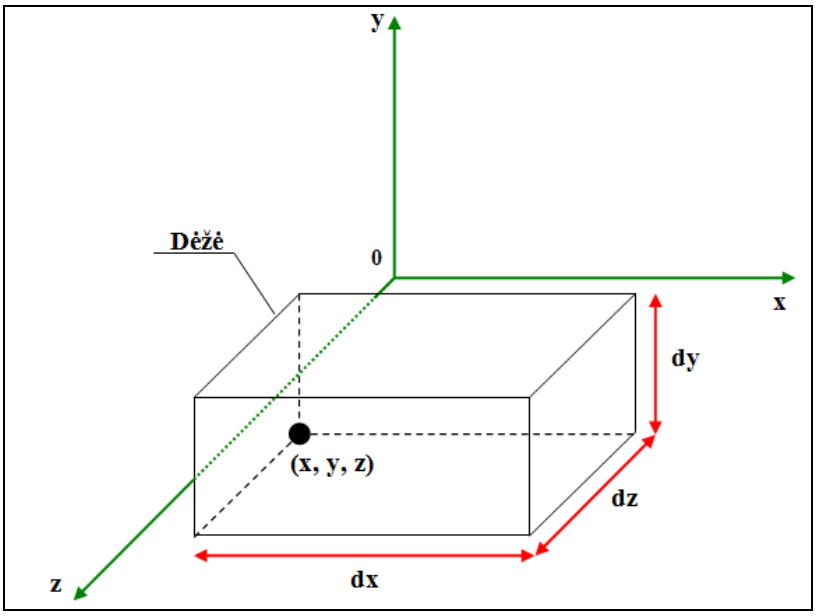
Testavimo skilties atveju gauti rezultatai yra saugomi pagal nutylėjimą parinktame faile *TestRez.txt*, kuris, jei dar nebuvo sukurtas, yra automatiškai sukuriamas pagrindinės programos kataloge programos vykdymo metu. Šiuo atveju pačiam vartotojui failo parinkti nereikia.

Nepriklausomai nuo pasirinkto puslapio (*Generuoti duomenys*, *Duomenys iš failo* ar *Testavimas*) turi būti fiksuojami euristikos parametrai, t.y. sluoksnio ir juostos storio parinkimui taikomos dažnumo funkcijos (DF) ir prioriteto taisyklės (PT). Pagal nutylėjimą parinkta rangavimo taisyklę RT6 atitinkanti DF ir PT kombinacija. Parametrai DF ir PT detaliau buvo aptarti 2.1 poskyryje. Tuo tarpu euristikos

parametrai, apibrėžiantys paieškos variantų atitinkamai 2D ir 3D pakavime skaičių, gali būti keičiami tik programiniu būdu. Pagal nutylėjimą laisvai parinktos jų reikšmės: $M_3 = 4$ ir $M_2 = 8$.

Pasirinkus *Duomenys iš failo* arba *Generuoti duomenys* puslapius, pagrindiniai CLP uždavinio rezultatai pateikiami lentelės pavidalu. Šioje lentelėje yra pateikiamas visų pakuojamų dėžių sąrašas. Paskutiniame kiekvienos eilutės stulpelyje yra pateikiama loginio kintamojo reikšmė 0 arba 1. Lentelės j-tosios eilutės ($j = \overline{1, n}$, n - sugeneruotų dėžių skaičius) paskutiniame stulpelyje esanti reikšmė 1 reiškia, kad j-toji dėžė yra pakuojama į konteinerį, o reikšmė 0 – kad j-toji dėžė lieka nesupakuota.

Stulpeliuose dx, dy ir dz yra pateikiami atitinkamai dėžių plotis, aukštis ir ilgis. Verta pastebėti, kad šiuo atveju dėžės jau yra pasuktos į reikiamą padėtį, todėl dydžių dx, dy ir dz reikšmės, lyginant su pradinėmis tos dėžės parametrų reikšmėmis w_j, h_j ir d_j, gali būti sukeistos vietomis. Paprastumo dėlei toliau yra grįžtama prie pradinių žymėjimų, t.y. reikšmės dx, dy ir dz yra interpretuojamos atitinkamai kaip w_j, h_j ir d_j.



3.4 pav. Pakuojamos dėžės padėties iliustracija

Lentelės stulpeliuose x, y ir z yra pateikiamos kiekvienos dėžės pakavimo pozicijos. Laikome, kad koordinačių pradžios taškas yra kairysis apatinis galinis konteinerio kampas, t.y. jame $x = 0$, $y = 0$ ir $z = 0$. Tuomet x_j, y_j ir z_j nusakys j-tosios dėžės kairiojo apatinio galinio kampo koordinates, t.y. (x_j, y_j, z_j), kur $j = \overline{1, n}$ (žr. 3.4 pav.). Jeigu j-toji dėžė nėra pakuojama, tuomet jai $x_j = 0$, $y_j = 0$ ir $z_j = 0$.

Suradus uždavinio sprendinį, yra tikrinama, ar tas sprendinys yra korektiškas, t.y.: ar pakuojamos dėžės nepersidengia; ar dėžės nėra padubliuotos; ar jos nepatenka už konteinerio ribų ir pan.

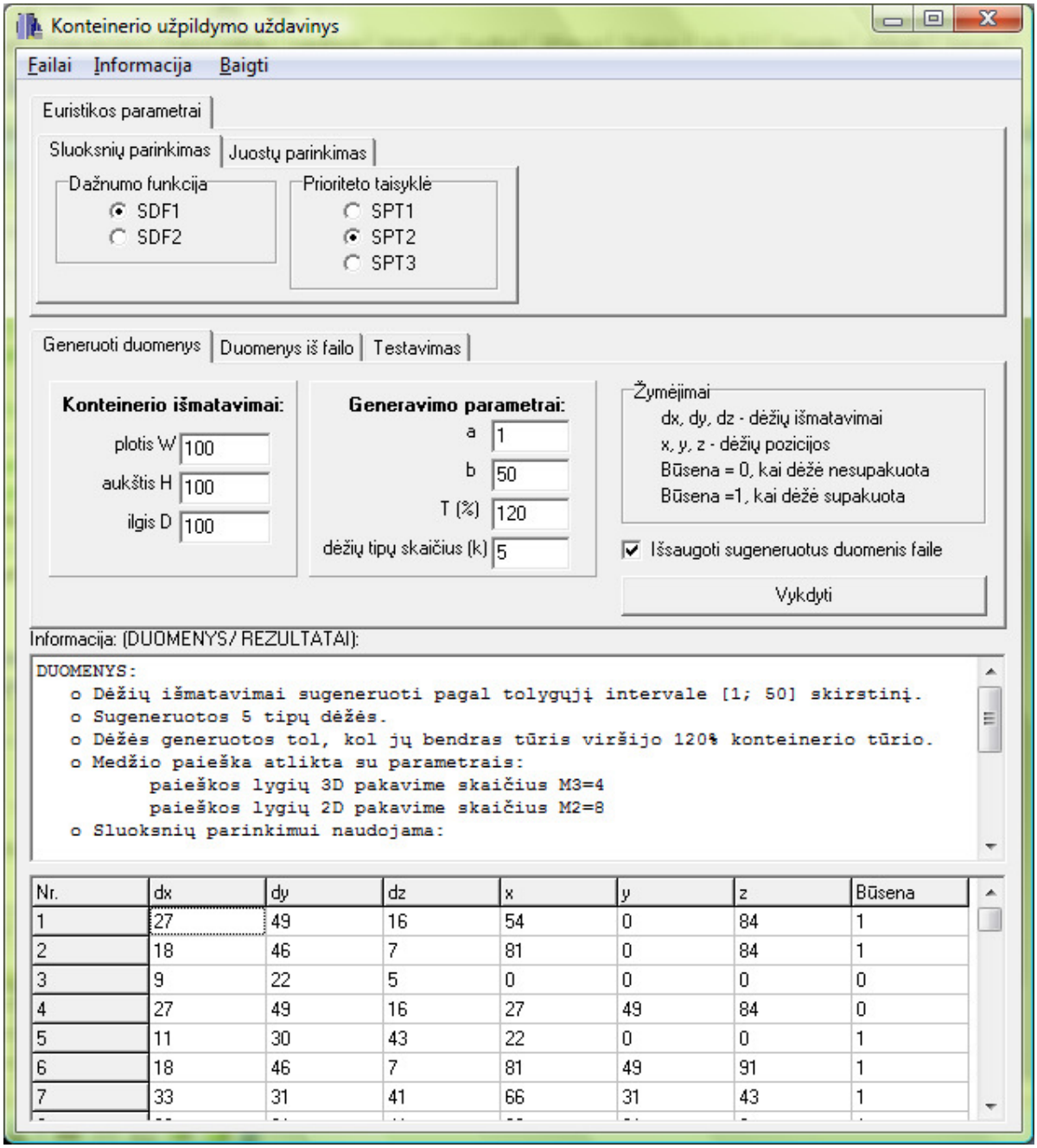
Pavyzdžiui, i-toji ir j-toji dėžės nepersidengia, jei teisingas toks sąryšis:

$$x_i + w_i \le x_j \vee x_j + w_j \le x_i \vee y_i + h_i \le y_j \vee y_j + h_j \le y_i \vee z_i + d_i \le z_j \vee z_j + d_j \le z_i$$

Čia $i \in \hat{N}$ ir $j \in \hat{N}$, kur $\hat{N} \subset N$ yra supakuotų dėžių indeksų aibė.

3.2. DUOMENŲ GENERAVIMO ATVEJIS

Programos langas puslapio *Generuoti duomenys* atveju pavaizduotas 3.5 paveiksle. Šiuo atveju įvedami pradiniai duomenys – tai konteinerio matmenys (W, H ir D) ir generavimo parametrai (a, b, T ir k). Konteinerio ir generavimo parametrų reikšmės detaliau buvo aptartos 1.9 poskyryje.



3.5 pav. Programos langas, kai duomenys yra generuojami

Visa informacija apie pasirinktas pradinių duomenų ir generavimo parametrų reikšmes bei gautą pakavimo uždavinio sprendinį, įvykdžius programos skaičiavimus, yra pateikiama programos rezultatų lange ir išsaugoma vartotojo nurodytame rezultatų faile. Toliau pateikiamas spręsto pavyzdžio rezultatų failo turinys.

Failo `Rezultatai.txt` turinys:

```
DUOMENYS:
   o Dėžių išmatavimai sugeneruoti pagal tolygųjį intervale [1; 50] skirstinį.
   o Sugeneruotos 5 tipų dėžės.
   o Dėžės generuotos tol, kol jų bendras tūris viršijo 120% konteinerio tūrio.
   o Medžio paieška atlikta su parametrais:
         paieškos lygių 3D pakavime skaičius M3=4
         paieškos lygių 2D pakavime skaičius M2=8
   o Sluoksnių parinkimui naudojama:
         dažnumo funkcija - SDF1, prioriteto taisyklė - SPT2
   o Juostų parinkimui naudojama:
         dažnumo funkcija - JDF1, prioriteto taisyklė - JPT2
REZULTATAI:
   o Sugeneruotų dėžių skaičius: 71
   o Gautas konteinerio užpildymas: 95.4669 %
   o Nesupakuotų dėžių skaičius: 25
   o Algoritmo veikimo laikas: 11.95 s
 +----+--------------------+-----------------+-------------------+
 |    |  Dėžių išmatavimai | Dėžių pozicijos | Dėžė:             |
 | Nr.|--------------------|-----------------|   0 - supakuota   |
 |    |  dx  |  dy  |  dz  |  x  |  y  |  z  |   1 - nesupakuota |
 +----+------+------+------+-----+-----+-----+-------------------+
 |   1|    27|    49|    16|   54|    0|   84|                  1|
 |   2|    18|    46|     7|   81|    0|   84|                  1|
 |   3|     9|    22|     5|    0|    0|    0|                  0|
 |   4|    27|    49|    16|   27|   49|   84|                  0|
 |   5|    11|    30|    43|   22|    0|    0|                  1|
 | ...|   ...|   ...|   ...|  ...|  ...|  ...|                ...|
 |  67|    27|    49|    16|   46|   49|   84|                  0|
 |  68|    27|    49|    16|   27|    0|   84|                  1|
 |  69|    27|    49|    16|    0|    0|    0|                  0|
 |  70|    33|    31|    41|   66|   30|    0|                  1|
 |  71|    46|     7|    18|   46|   93|   43|                  1|
 +----+------+------+------+-----+-----+-----+-------------------+
   o Konteinerio tūris (W*H*D): 1000000
   o Užpildytas tūris:  954669
```

3.3. DUOMENŲ ĮVEDIMO IŠ FAILO ATVEJIS

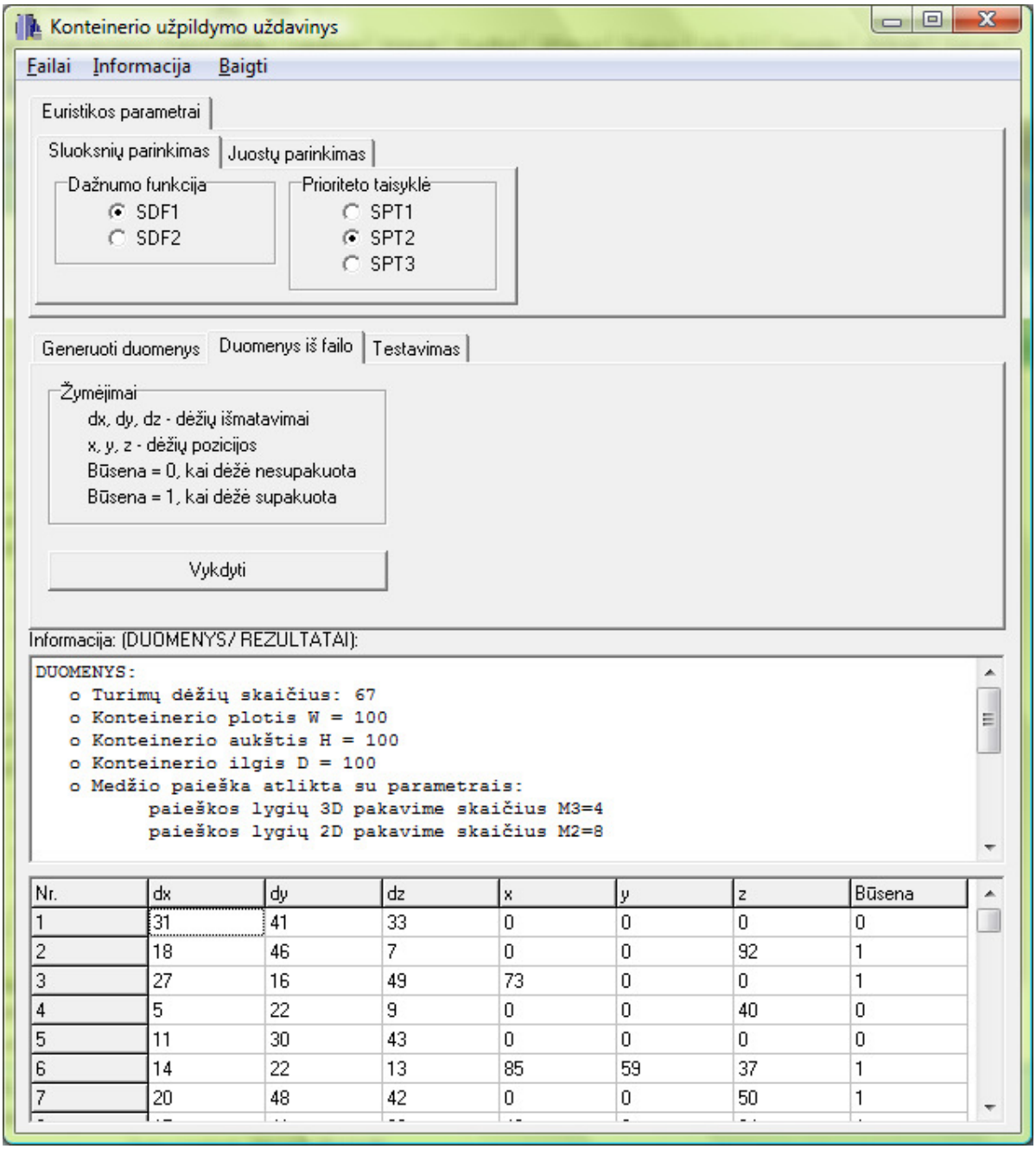
Norint skaičiavimus atlikti su vartotojo įvedamais duomenimis, reikia, kad duomenų failas turėtų tokią struktūrą:

`FailoVardas.txt`

```
W
H
D
w1 h1 d1
w2 h2 d2
w3 h3 d3
w4 h4 d4
...
```

Čia W - konteinerio plotis, H - konteinerio aukštis ir D - konteinerio ilgis. Toliau faile pateikiami dėžių matmenys, t.y.: j-tosios dėžės plotis w_j, aukštis h_j ir ilgis d_j ($j = 1, 2, \ldots$).

Programos langas puslapio *Duomenys iš failo* atveju pavaizduotas 3.6 paveiksle.



3.6 pav. Programos langas, kai duomenys įvedami iš failo

Toliau pateikiamas spręsto pavyzdžio rezultatų failo turinys.

Failo `Rezultatai.txt` turinys:

```
DUOMENYS:
   o Turimų dėžių skaičius: 67
   o Konteinerio plotis W = 100
   o Konteinerio aukštis H = 100
   o Konteinerio ilgis D = 100
```

```
   o Medžio paieška atlikta su parametrais:
         paieškos lygių 3D pakavime skaičius M3=4
         paieškos lygių 2D pakavime skaičius M2=8
   o Sluoksnių parinkimui naudojama:
         dažnumo funkcija - SDF1, prioriteto taisyklė - SPT2
   o Juostų parinkimui naudojama:
         dažnumo funkcija - JDF1, prioriteto taisyklė - JPT2
REZULTATAI:
   o Gautas konteinerio užpildymas: 92.1929 %
   o Nesupakuotų dėžių skaičius: 22
   o Algoritmo veikimo laikas: 0.219 s
```

	Dėžių išmatavimai			Dėžių pozicijos			Dėžė:
Nr.	dx	dy	dz	x	y	z	0 - supakuota 1 - nesupakuota
1	31	41	33	0	0	0	0
2	18	46	7	0	0	92	1
3	27	16	49	73	0	0	1
4	5	22	9	0	0	40	0
5	11	30	43	0	0	0	0
...	...	...	...	...	...	...	...
62	27	10	13	68	89	35	1
63	30	14	18	43	0	32	1
64	24	11	47	44	89	0	1
65	29	28	3	0	63	47	1
66	36	14	47	43	48	50	1
67	26	45	48	20	0	50	1

```
   o Konteinerio tūris (W*H*D): 1000000
   o Užpildytas tūris:  921929
```

3.4. TESTŲ ATLIKIMO GALIMYBĖ

Norint pagreitinti testų atlikimo eigą, buvo realizuota skiltis *Testavimas*. Joje vartotojas gali be pasirinktų konteinerio išmatavimų (W, H ir D) bei generavimo parametrų (a, b, T ir k) papildomai įvesti norimą sprendžiamų pakavimo uždavinių (realizacijų) skaičių R. Taip pat realizuota galimybė apriboti suminį sprendimo laiką, nustatant pasirinktą laiko limitą (sekundėmis) t_{LIMIT}. Apriboti skaičiavimo laiką labai svarbu sprendžiant labai didelės apimties pakavimo uždavinius.

Programos lango vaizdas pateiktas 3.7 paveiksle. Šiuo atveju rezultatų failo *TestRez.txt* turinys galėtų atrodyti taip:

Failo `TestRez.txt` turinys:

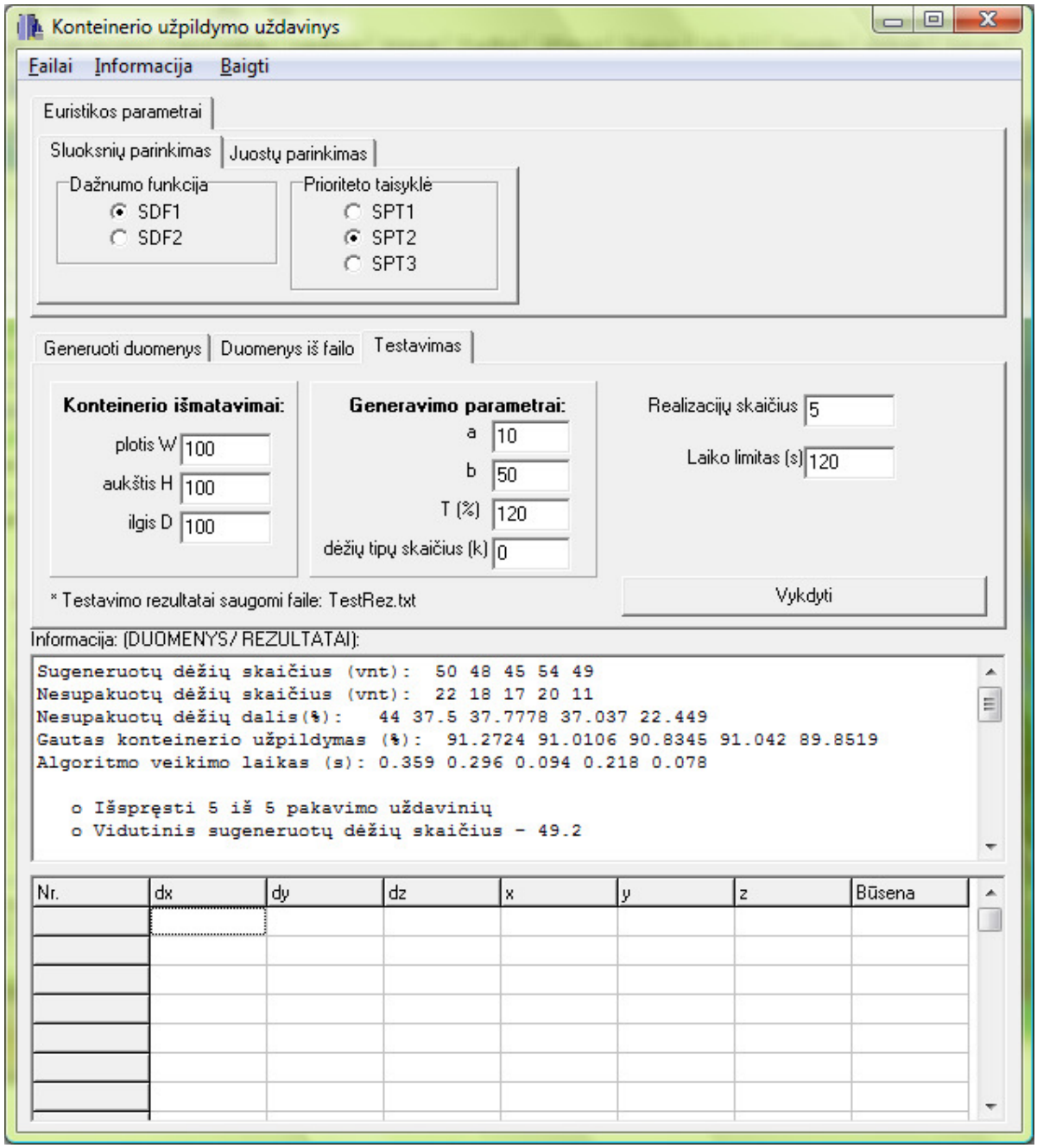
```
Sugeneruotų dėžių skaičius (vnt):      50 48 45 54 49
Nesupakuotų dėžių skaičius (vnt):      22 18 17 20 11
Nesupakuotų dėžių dalis(%):            44 37.5 37.7778 37.037 22.449
Gautas konteinerio užpildymas (%):     91.2724 91.0106 90.8345 91.042 89.8519
     Algoritmo veikimo laikas (s):          0.359 0.296 0.094 0.218 0.078

   o Išspręsti 5 iš 5 pakavimo uždavinių
   o Vidutinis sugeneruotų dėžių skaičius - 49.2
   o Vidutinis nesupakuotų dėžių skaičius - 17.6

KONTEINERIO UŽPILDYMO (%) REZULTATAI:
   o Vidutinis konteinerio užpildymas - 90.8023
   o Minimalus konteinerio užpildymas - 89.8519
   o Maksimalus konteinerio užpildymas - 91.2724
   o Konteinerio užpildymo standartinis nuokrypis (stdev) - 0.553655
```

```
ALGORITMO VEIKIMO LAIKO (s) REZULTATAI:
   o Vidutinis vienos realizacijos sprendimo laikas- 0.209
   o Minimalus vienos realizacijos sprendimo laikas- 0.078
   o Maksimalus vienos realizacijos sprendimo laikas - 0.359
   o Vykdymo laiko standartinis nuokrypis (stdev) - 0.12302

     ALGORITMO PARAMETRAI:
        o M3=4, M2=8
        o RT={SDF1,SPT2,JDF1,JPT2}
        o {W,H,D}={100,100,100}
        o a=10, b=50, T=120%, k=0
        o Laiko limitas: 120s
        o Realizacijų skaičius: 5
```



3.7 pav. Programos langas, skirtas algoritmo tyrimui atlikti

4. DISKUSIJA

Iš atliktų eksperimentų rezultatų pastebėjome, kad nagrinėjamos euristikos veikimas labai priklauso nuo apibrėžtos rangavimo taisyklės, taikomos sluoksnio ir juostos storių parinkimui. Pavyzdžiui, optimali užpildymo U_{min} ir U_{vid} atžvilgiu buvo rangavimo taisyklė RT6, o U_{max} atžvilgiu – RT22. Taigi, jei, sprendžiant konteinerio užpildymo uždavinį (CLP), yra orientuojamasi į kuo efektyvesnį konteinerio erdvės panaudojimą, euristikoje galėtų būti parinkta, pavyzdžiui, rangavimo taisyklė RT6. Taip pat buvo pastebėta, kad geriausia laiko atžvilgiu buvo rangavimo taisyklė RT12. Jos atveju skaičiavimai buvo atliekami beveik 13 kartų greičiau nei geriausią užpildymą duodančios rangavimo taisyklės RT6 atveju. Vadinasi, jei, sprendžiant CLP, yra orientuojamasi į kuo mažesnes sprendimo laiko sąnaudas, euristikoje turėtų būti parinkta rangavimo taisyklė RT12.

Sprendžiant santykio tarp sprendimo laiko ir konteinerio užpildymo problemą, buvo bandoma skaičiuoti santykinius dydžius, tokius kaip: laikas, kurio reikia konteinerio 1% užpildymui; laikas, kurio reikia papildomo 1% užpildymui (RT12 atžvilgiu) ir kt. Tačiau gauti skaičiavimų rezultatai parodė, kad kai kurių santykinių dydžių skaičiavimas nedavė akivaizdesnių rangavimo taisyklių diferencijavimo rezultatų, lyginant su tais, kurie buvo gauti jas diferencijuojant tik pagal vieną iš rodiklių, t.y. tik pagal sprendimo laiką arba tik pagal konteinerio užpildymą. Vadinasi, norint gauti tinkamesnį rangavimo taisyklių diferencijavimą, vienu metu įvertinant keletą rodiklių, galbūt reikėtų sukurti kitą metodiką. Pavyzdžiui, minėti rodikliai galėtų būti vertinami su tam tikrais svoriais, tačiau šiems svoriams nustatyti jau reikėtų papildomų ekspertinių duomenų.

Tiriant euristikoje taikomos medžio paieškos parametrų M_2 ir M_3 įtaką rezultatams, galima pastebėti, kad tiek uždavinio sprendimo laikas, tiek sprendinio kokybė labai priklauso nuo jų parinkimo. Taigi šių parametrų reikšmės turėtų būti parenkamos atsižvelgiant į vartotojo konteinerio užpildymo lūkesčius bei jam priimtinas sprendimo laiko sąnaudas. Be to, buvo pastebėta, kad didinant parametro M_2 reikšmę, konteinerio užpildymo išlošis tendencingai mažėja, todėl šio parametro reikšmė taip pat gali būti nustatoma ir fiksavus užpildymo pokyčio reikšmingumo lygmenį.

Rezultatai taip pat parodė, kad kuo didesnis bendras pakuojamų dėžių tūris konteinerio tūrio atžvilgiu, tuo geresnių konteinerio užpildymo rezultatų galime tikėtis. Taip yra dėl to, kad papildomos dėžės sprendimo algoritmui suteikia daugiau pasirinkimo laisvės atliekant juostos užpildymą.

Taip pat buvo pastebėta, kad geriausi vidutiniai konteinerio užpildymo ir sprendimo laiko rezultatai yra gaunami stipriai ir silpnai įvairiarūšio krovinio atveju, o blogiausi – vienarūšio krovinio atveju. Pirmaisiais atvejais platesnė dėžių dimensijų įvairovė gerokai palengvina dėžių grupavimo į tinkamus sluoksnius ir juostas procesą. Tuo tarpu vienarūšio krovinio atveju užpildymo rezultatai labai priklauso nuo konkretaus uždavinio duomenų. Pavyzdžiui, pakuojant labai didelių išmatavimų (konteinerio matmenų atžvilgiu) vienarūšes dėžes, labai geras konteinerio erdvės užpildymas dažniausiai nėra gaunamas.

IŠVADOS

1. Geriausias vidutinis konteinerio erdvės užpildymas yra gaunamas euristikoje taikant rangavimo taisyklę RT6, o blogiausias – RT11.

2. Mažiausios vidutinės uždavinio sprendimo laiko sąnaudos yra gaunamos euristikoje taikant rangavimo taisyklę RT12, o didžiausios – RT18.

3. Medžio paieškos parametrų M_2 ir M_3 reikšmės turi būti parenkamos, atsižvelgiant į konteinerio užpildymo lūkesčius bei priimtinas sprendimo laiko sąnaudas. Be to, didinant parametro M_2 reikšmę, konteinerio užpildymo išlošis tendencingai mažėja, todėl šio parametro reikšmė taip pat gali būti nustatoma ir fiksavus užpildymo pokyčio reikšmingumo lygmenį.

4. Kuo didesnis pakuojamų dėžių bendro tūrio ir konteinerio tūrio santykis, tuo geresnių konteinerio užpildymo rezultatų galime pasiekti.

5. Geriausi vidutiniai konteinerio užpildymo ir vienos dėžės pakavimo laiko rezultatai yra gaunami stipriai įvairiarūšio krovinio atveju, o blogiausi – vienarūšio krovinio atveju.

6. Vienarūšio krovinio atveju gaunami pakavimo rezultatai labai priklauso nuo konkretaus uždavinio duomenų. Tiek konteinerio užpildymo, tiek pakavimo laiko sklaida šiuo atveju yra gerokai didesnė nei kitų tipų krovinio atvejais.

REKOMENDACIJOS

Darbe buvo išnagrinėta euristika, paremta sienų rentimo metodika ir medžio paieškos algoritmu (Pisinger, 2002). Tuo tarpu konteinerio užpildymo uždavinio (CLP) sprendimui yra taip pat taikoma ir nemažai kitų algoritmų, pavyzdžiui: tabu paieškos, genetinis, modeliuojamo atkaitinimo, skruzdžių kolonijos ir kt. Šių algoritmų detalesnis tyrimas galėtų būti tęstinė šio darbo dalis.

Šiame darbe suformuluotame CLP modelyje buvo daromos prielaidos, kad tiek konteineris, tiek pakuojamos dėžės yra stačiakampio gretasienio formos. Šios prielaidos gerokai supaprastino sprendžiamą uždavinį. Vis dėlto, norint, kad modelis tiksliau atkartotų realaus pasaulio pakavimo uždavinius, būtų pravartu panagrinėti ir kitokių formų objektų pakavimo galimybes.

Pasirinktoje euristikoje nebuvo atsižvelgiama į papildomus apribojimus, tokius kaip: dėžių orientacija, ribotas krovinio svoris, krovinio gravitacijos centras, dėžių stabilumas ir kt. Šių apribojimų įtraukimo į modelį galimybių tyrimas taip pat galėtų būti tęstinė šio darbo dalis. Nagrinėjamoje euristikoje realizavus minėtus apribojimus, galima būtų tirti jų įtraukimo į modelį įtaką pakavimo rezultatams (sprendinio kokybei, pakavimo laikui ir pan.).

Tolesnis šio darbo tyrimas taip pat galėtų būti ir naujos rangavimo taisyklių palyginimo metodikos kūrimas. Pavyzdžiui, atlikus tyrimą, paaiškėjo, kad vienos rangavimo taisyklės yra geresnės minimalaus užpildymo (U_{min}) atžvilgiu, kitos – maksimalaus užpildymo (U_{max}), trečios – vidutinio užpildymo (U_{vid}) prasme ir pan. Todėl, norint išrinkti vieną optimaliai tinkamą rangavimo taisyklę, reikia sukurti metodiką, leidžiančią rangavimo taisykles palyginti iškart pagal visus rodiklius. Pavyzdžiui, minėti rodikliai (U_{vid}, U_{max} ir kt.) galėtų būti vertinami su tam tikrais svoriais, kurių nustatymui reikėtų remtis ekspertiniais duomenimis.

PADĖKOS

Nuoširdžiai dėkoju darbo vadovui doc. dr. Narimantui Listopadskiui už vadovavimą magistriniam darbui. Dėkoju už konsultacijoms skirtą laiką, visapusę pagalbą bei vertingus patarimus.

LITERATŪRA

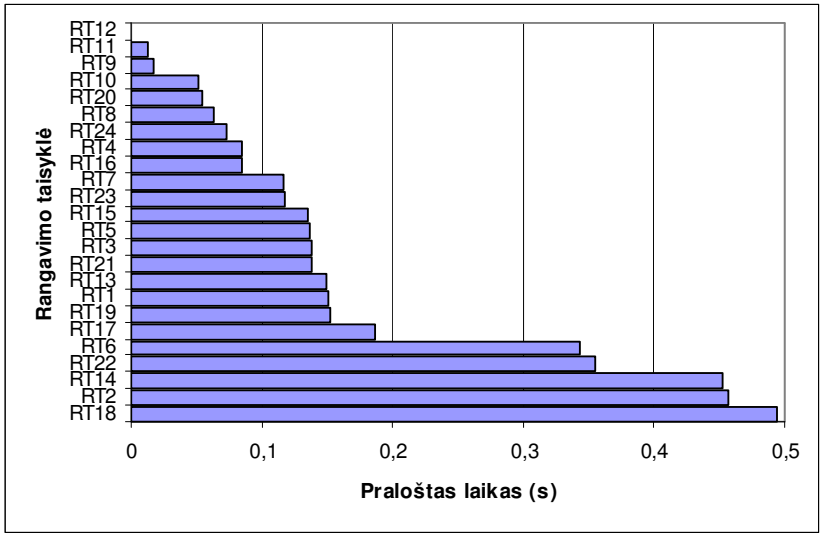
1. Blum, C., A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison / *ACM Computing Surveys*, Vol. 35(3), 2003, p. 268-308.
2. Caceres, R. G., C. V. Mejia, J. C. Villalobos. Integral Optimization of the Container Loading Problem / *Stochastic Optimization – Seeing the Optimal for the Uncertain*, 2011, p. 226-254.
3. Copeland, B. J. What is a Turing machine? / Turing archive catalogue. http://www.alanturing.net/turing_archive/pages/Reference%20Articles/What%20is%20a%20Turing%20Machine.html, 15/02/2012.
4. Dyckhoff, H. A typology of Cutting and Packing Problems / *European Journal of Operational Research*, Vol. 44, 1990, p. 145-159.
5. Fanslau, T., A. Bortfeldt, *A Tree Search Algorithm for Solving the Container Loading Problem*, Informs Journal on Computing, No.2 (22), 222-235, 2010.
6. Juraitis, M., T. Stonys, A. Starinskas, D. Jankauskas, D. Rubliauskas. *A* Randomized Heuristics for the Container Loading Problem: Further Investigations / *Information Technology and Control*, Vol. 35 , 2006, p. 7-12.
7. Levickytė, M., N. Listopadskis, An analysis of heuristic for the container loading problem / *Matematika ir matematinis modeliavimas* / Kauno technologijos universitetas, 2012. Straipsnis pateiktas spaudai.
8. Levickytė, M., N. Listopadskis, Konteinerio užpildymo uždavinio sprendimo algoritmų tyrimas / *Taikomoji matematika*: X studentų konferencijos pranešimų medžiaga / Kauno technologijos universitetas. ISBN 978-609-02-0308-8, Kaunas: Technologija, 2012, p. 19-20.
9. Misevičius, A., J. Blonskis, V. Bukšnaitis. Kombinatorinis optimizavimas ir metaeuristiniai metodai: teoriniai aspektai / Informacijos mokslai, Vol. 42-43, 2007, p. 213-219.
10. Pisinger, D. A minimal algorithm for the 0-1 Knapsack Problem / *European Journal of Operational Research*, Vol. 45, 1997, p. 758-767.
11. Pisinger, D. Heuristics for the container loading problem / *European Journal of Operational Research*, Vol. 141, 2002, p. 382-392.
12. Tovey, C. A. Tutorial on Computational Complexity / *Interfaces*, Vol. 32(3), 2002, p. 30-61.
13. Wascher, G., H. Haubner, H. Schumann. An Improved Typology of Cutting and Packing Problems / *European Journal of Operational Research*, Vol. 183, 2007, p. 1109-1130.
14. http://www.diku.dk/hjemmesider/ansatte/pisinger/codes.html

1 PRIEDAS. RANGAVIMO TAISYKLIŲ PALYGINIMAS

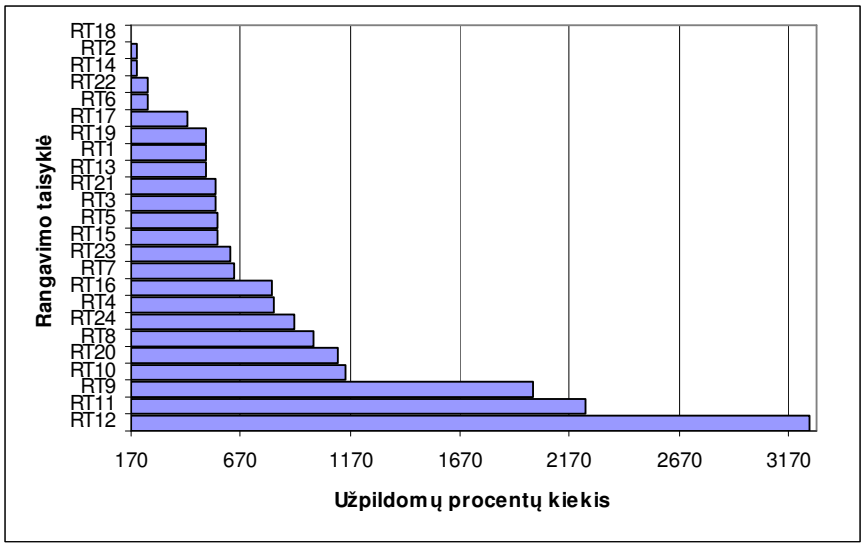
1 lentelė

Pakavimo rodiklių reikšmės priklausomai nuo pasirinktos rangavimo taisyklės

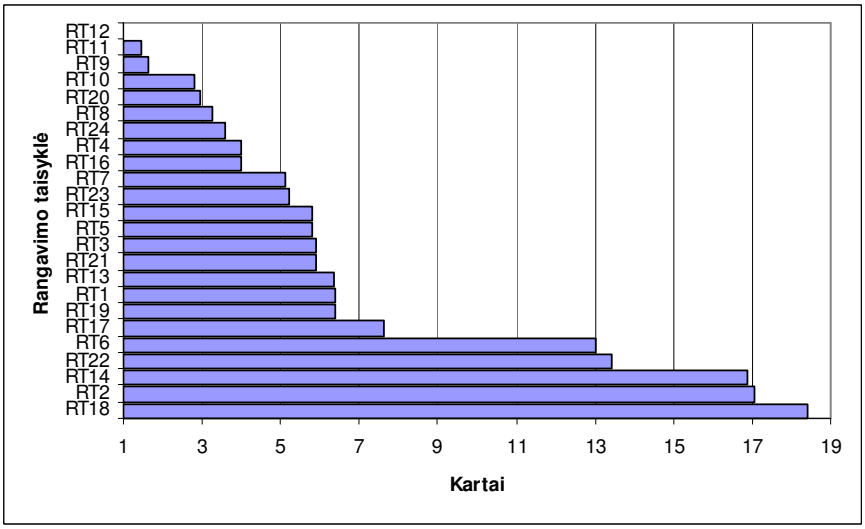
	$U_{\min}$	U_{vid}	$U_{\max}$	σ_U	$t_{\min}$	t_{vid}	$t_{\max}$	σ_t
RT1	88,9508	91,750618	94,0717	0,937	0,125	0,17909	0,265	0,02869
RT2	90,4642	92,862212	94,8722	0,92724	0,109	0,48439	1,373	0,26741
RT3	89,8558	91,463572	93,1014	0,83183	0,109	0,16567	0,25	0,03035
RT4	90,0446	92,009234	94,4574	0,87245	0,047	0,11248	0,328	0,04695
RT5	90,4063	92,117148	94,0717	0,83131	0,093	0,16396	0,25	0,04167
RT6	91,3269	93,033315	94,8722	0,75284	0,078	0,37113	0,999	0,19107
RT7	89,8558	91,701097	93,3969	0,80385	0,078	0,14383	0,281	0,03951
RT8	90,3714	92,216258	94,4574	0,84048	0,031	0,09173	0,265	0,04105
RT9	88,7732	91,456656	94,5685	0,95103	0,001	0,04558	0,109	0,0234
RT10	89,0006	92,362805	94,3478	0,94674	0,016	0,08019	0,187	0,03485
RT11	87,6438	90,815558	93,3755	1,03104	0,001	0,04043	0,094	0,02041
RT12	87,5873	91,212786	93,854	1,13004	0,001	0,02793	0,063	0,01262
RT13	88,9508	91,756176	94,0717	0,93467	0,125	0,178	0,265	0,02802
RT14	90,083	92,832768	94,8722	0,90857	0,125	0,48002	1,435	0,26634
RT15	89,5073	91,497726	93,7345	0,87613	0,109	0,16256	0,281	0,03305
RT16	89,7978	91,970984	94,4574	0,88254	0,047	0,11294	0,327	0,04758
RT17	89,3415	91,66311	94,0717	0,9567	0,078	0,21404	0,531	0,10779
RT18	90,7981	92,760414	94,8722	0,93306	0,063	0,52245	2,839	0,47314
RT19	89,1839	91,384094	93,3534	0,85658	0,078	0,17956	0,546	0,10332
RT20	89,7988	91,911956	94,4574	0,92661	0,031	0,08252	0,406	0,05342
RT21	88,4567	91,785514	93,3588	0,88364	0,094	0,16629	0,25	0,02959
RT22	90,9098	92,883043	95,0709	0,82997	0,109	0,38235	1,201	0,18004
RT23	89,0798	91,277152	95,0267	0,91789	0,078	0,1454	0,218	0,02977
RT24	88,7622	91,823945	95,0377	0,98226	0,031	0,10062	0,234	0,0337
MAX	91,3269	93,033315	95,0709	1,13004	0,125	0,52245	2,839	0,47314
M IN	87,5873	90,815558	93,1014	0,75284	0,001	0,02793	0,063	0,01262



1 pav. Praloštas laikas (s) mažiausią t_{vid} duodančios RT atžvilgiu



2 pav. Per vieną sekundę užpildomų procentų kiekis



3 pav. Kartai, kiek RT_i laikas, kurio reikia 1% užpildymui, yra didesnis už RT bazinės

2 PRIEDAS. PAIEŠKOS VARIANTŲ 2D IR 3D PAKAVIME TYRIMAS

2 lentelė

Vidutinis konteinerio užpildymas esant skirtingoms M_2 ir M_3 kombinacijoms

		M3									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
M2	**1**	91,0037	91,1096	91,16	91,198	91,3595	91,5941	91,7285	91,8443	91,9197	91,9124
	2	92,7241	92,9198	92,9611	92,9762	93,0587	93,1958	93,243	93,3014	93,3256	93,3959
	3	93,7676	93,9055	93,9912	94,0226	94,0694	94,1972	94,3233	94,3884	94,4185	94,3761
	4	94,2513	94,3375	94,3877	94,3809	94,5404	94,6667	94,756	94,7977	94,817	94,8313
	5	94,4276	94,5131	94,5548	94,6158	94,7338	94,8818	94,942	94,9959	95,0093	95,0294
	6	94,4496	94,5739	94,6062	94,7096	94,7826	94,8997	95,0263	95,0928	95,1004	95,0996
	7	94,4495	94,5663	94,6013	94,7241	94,792	94,9068	95,0429	95,1288	95,1261	95,098
	8	94,4495	94,5663	94,6013	94,7241	94,792	94,8986	95,0429	95,1274	95,1252	95,0942
	9	94,4495	94,5663	94,6013	94,7241	94,792	94,8986	95,0429	95,1274	95,1252	95,0942
	10	94,4495	94,5663	94,6013	94,7241	94,792	94,8986	95,0429	95,1274	95,1252	95,1011

3 lentelė

Vidutinis uždavinio sprendimo laikas esant skirtingoms M_2 ir M_3 kombinacijoms

		M3									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
M2	**1**	0,00125	0,00312	0,00624	0,00952	0,01466	0,0209	0,02761	0,03463	0,03776	0,04009
	2	0,0028	0,00811	0,01513	0,0231	0,03651	0,04883	0,06521	0,08112	0,08768	0,09609
	3	0,00748	0,02309	0,04352	0,06864	0,09844	0,13853	0,18361	0,22012	0,23992	0,28407
	4	0,02153	0,06599	0,12948	0,20436	0,28861	0,39078	0,49983	0,58594	0,6326	0,83304
	5	0,04992	0,14789	0,27113	0,43961	0,61136	0,79404	0,98873	1,20979	1,26158	1,48154
	6	0,06973	0,19141	0,3471	0,52182	0,74303	0,94724	1,19169	1,37858	1,48497	1,81071
	7	0,0741	0,20265	0,36255	0,54148	0,76519	0,97439	1,2115	1,41306	1,56609	1,85095
	8	0,07348	0,20529	0,36488	0,54725	0,7697	0,99061	1,2101	1,41166	1,56876	1,85641
	9	0,07473	0,20436	0,36473	0,54445	0,76331	0,98686	1,21447	1,41056	1,55642	1,84565
	10	0,07332	0,20249	0,36847	0,546	0,76425	0,98639	1,209	1,40853	1,55799	1,83848

4 lentelė

Vidutinio konteinerio užpildymo pokytis (horizontalus kitimas)

		M3									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
M2	**1**	-	0,1059	0,0504	0,038	0,1615	0,2346	0,1344	0,1158	0,0754	-0,0073
	2	-	0,1957	0,0413	0,0151	0,0825	0,1371	0,0472	0,0584	0,0242	0,0703
	3	-	0,1379	0,0857	0,0314	0,0468	0,1278	0,1261	0,0651	0,0301	-0,0424
	4	-	0,0862	0,0502	-0,0068	0,1595	0,1263	0,0893	0,0417	0,0193	0,0143
	5	-	0,0855	0,0417	0,061	0,118	0,148	0,0602	0,0539	0,0134	0,0201
	6	-	0,1243	0,0323	0,1034	0,073	0,1171	0,1266	0,0665	0,0076	-0,0008
	7	-	0,1168	0,035	0,1228	0,0679	0,1148	0,1361	0,0859	-0,0027	-0,0281
	8	-	0,1168	0,035	0,1228	0,0679	0,1066	0,1443	0,0845	-0,0022	-0,031
	9	-	0,1168	0,035	0,1228	0,0679	0,1066	0,1443	0,0845	-0,0022	-0,031
	10	-	0,1168	0,035	0,1228	0,0679	0,1066	0,1443	0,0845	-0,0022	-0,0241

5 lentelė

Vidutinio uždavinio sprendimo laiko pokytis (horizontalus kitimas)

		M3									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
M2	**1**	-	0,00187	0,00312	0,00328	0,00514	0,00624	0,00671	0,00702	0,00313	0,00233
	2	-	0,00531	0,00702	0,00797	0,01341	0,01232	0,01638	0,01591	0,00656	0,00841
	3	-	0,01561	0,02043	0,02512	0,0298	0,04009	0,04508	0,03651	0,0198	0,04415
	4	-	0,04446	0,06349	0,07488	0,08425	0,10217	0,10905	0,08611	0,04666	0,20044
	5	-	0,09797	0,12324	0,16848	0,17175	0,18268	0,19469	0,22106	0,05179	0,21996
	6	-	0,12168	0,15569	0,17472	0,22121	0,20421	0,24445	0,18689	0,10639	0,32574
	7	-	0,12855	0,1599	0,17893	0,22371	0,2092	0,23711	0,20156	0,15303	0,28486
	8	-	0,13181	0,15959	0,18237	0,22245	0,22091	0,21949	0,20156	0,1571	0,28765
	9	-	0,12963	0,16037	0,17972	0,21886	0,22355	0,22761	0,19609	0,14586	0,28923
	10	-	0,12917	0,16598	0,17753	0,21825	0,22214	0,22261	0,19953	0,14946	0,28049

6 lentelė

Vidutinio konteinerio užpildymo pokytis (vertikalus kitimas)

		M3									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
M2	**1**	-	-	-	-	-	-	-	-	-	-
	2	1,7204	1,8102	1,8011	1,7782	1,6992	1,6017	1,5145	1,4571	1,4059	1,4835
	3	1,0435	0,9857	1,0301	1,0464	1,0107	1,0014	1,0803	1,087	1,0929	0,9802
	4	0,4837	0,432	0,3965	0,3583	0,471	0,4695	0,4327	0,4093	0,3985	0,4552
	5	0,1763	0,1756	0,1671	0,2349	0,1934	0,2151	0,186	0,1982	0,1923	0,1981
	6	0,022	0,0608	0,0514	0,0938	0,0488	0,0179	0,0843	0,0969	0,0911	0,0702
	7	-0,0001	-0,0076	-0,0049	0,0145	0,0094	0,0071	0,0166	0,036	0,0257	-0,0016
	8	0	0	0	0	0	-0,0082	0	-0,0014	-0,0009	-0,0038
	9	0	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	0,0069

7 lentelė

Vidutinio uždavinio sprendimo laiko pokytis (vertikalus kitimas)

		M3									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
M2	**1**	-	-	-	-	-	-	-	-	-	-
	2	0,00155	0,00499	0,00889	0,01358	0,02185	0,02793	0,0376	0,04649	0,04992	0,056
	3	0,00468	0,01498	0,02839	0,04554	0,06193	0,0897	0,1184	0,139	0,15224	0,18798
	4	0,01405	0,0429	0,08596	0,13572	0,19017	0,25225	0,31622	0,36582	0,39268	0,54897
	5	0,02839	0,0819	0,14165	0,23525	0,32275	0,40326	0,4889	0,62385	0,62898	0,6485
	6	0,01981	0,04352	0,07597	0,08221	0,13167	0,1532	0,20296	0,16879	0,22339	0,32917
	7	0,00437	0,01124	0,01545	0,01966	0,02216	0,02715	0,01981	0,03448	0,08112	0,04024
	8	-0,00062	0,00264	0,00233	0,00577	0,00451	0,01622	-0,0014	-0,0014	0,00267	0,00546
	9	0,00125	-0,00093	-0,00015	-0,0028	-0,00639	-0,00375	0,00437	-0,0011	-0,01234	-0,01076
	10	-0,00141	-0,00187	0,00374	0,00155	0,00094	-0,00047	-0,00547	-0,00203	0,00157	-0,00717

3 PRIEDAS. PAKUOJAMŲ DĖŽIŲ KIEKIO ĮTAKOS TYRIMAS

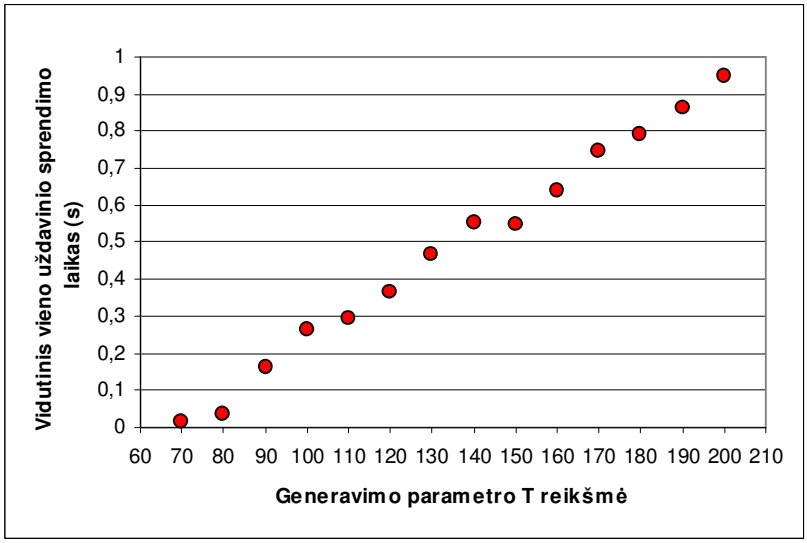
8 lentelė

Pakavimo rodiklių reikšmės priklausomai nuo T reikšmės

T reikšmės didinimo etapas	Generavimo parametras T(%)	n^*	r^{**}	$U_{\min}$	U_{vid}	$U_{\max}$	σ_U	$t_{\min}$	t_{vid}	$t_{\max}$	σ_t
0	**70**	43,04	0,01	70,0086	72,1876	78,5676	2,01541	0	0,01622	0,28	0,032779
1	**80**	48,83	0,05	80,02	81,7587	88,3423	1,81662	0	0,03743	1,217	0,140318
2	**90**	54,71	4,56	85,703	88,3362	91,0617	1,08441	0,015	0,16084	1,248	0,184772
3	**100**	60,81	10,41	87,7939	90,4944	92,6288	1,01691	0,016	0,26457	3,9	0,528649
4	**110**	67,21	17,02	89,5308	91,8238	94,0874	0,889353	0,031	0,29203	1,716	0,307492
5	**120**	73,13	22,8	90,8935	92,8019	94,7826	0,847214	0,047	0,36317	2,917	0,404106
6	**130**	79,5	28,87	91,6157	93,6067	95,1452	0,756358	0,046	0,46551	3,339	0,461618
7	**140**	85,12	34,62	92,08	94,1496	96,2562	0,752322	0,047	0,55271	3,432	0,535441
8	**150**	91,03	40,59	92,8645	94,7241	96,3658	0,656432	0,078	0,54615	2,84	0,459904
9	**160**	96,64	47,01	93,4397	95,1455	96,8606	0,676187	0,078	0,63898	3,198	0,515126
10	**170**	102,82	52,2	93,9452	95,5581	96,8085	0,624162	0,14	0,7474	5,569	0,694715
11	**180**	108,9	58,49	94,2104	95,9388	97,2488	0,554904	0,109	0,78998	3,651	0,614656
12	**190**	114,83	63,88	94,7891	96,1451	97,6606	0,592197	0,094	0,86097	4,087	0,670003
13	**200**	120,83	69,94	95,0603	96,4065	97,6401	0,515673	0,11	0,94708	4,336	0,691124

* Vidutinis sugeneruotų dėžių skaičius

** Vidutinis nesupakuotų dėžių skaičius



4 pav. Vidutinis vieno uždavinio sprendimo laikas esant skirtingoms T reikšmėms

4 PRIEDAS. KROVINIO TIPO ĮTAKOS TYRIMAS

9 lentelė

Vidutinis sugeneruotų dėžių kiekis esant skirtingoms T reikšmėms

Generavimo parametras T(%)	Stipriai įvairiarūšis krovinys (k = 0)	Silpnai įvairiarūšis krovinys (k = 10)	Vienarūšis krovinys (k = 1)
70	27,06	29,45	49,11
80	30,86	33,7	56,05
90	34,64	37,84	62,95
100	38,3	41,92	69,92
110	41,87	45,82	76,85
120	45,68	50,01	83,75
130	49,58	53,99	90,71
140	53,04	58,03	97,67
150	56,62	61,89	104,65
160	60,32	66,01	111,6
170	64,13	69,86	118,47
180	67,99	73,67	125,44
190	71,71	77,69	132,37
200	75,29	81,5	139,34

5 PRIEDAS. PROGRAMOS INSTRUKCIJA VARTOTOJUI

Failo *Instrukcija.txt* turinys:

```
_____________________________________________________________________________
Jei sprendžiamo konteinerio užpildymo uždavinio (toliau CLP) pradiniai duomenys
yra įvedami iš tekstinio duomenų failo, tuomet skaičiavimams atlikti pasirenkama
skiltis <Duomenys iš failo>.

Duomenų failo struktūra:
-----Duomenys.txt-----
W
H
D
w1 h1 d1
w2 h2 d2
w3 h3 d3
... ...
----------------------
Paaiškinimas:
1. Pirmoje failo eilutėje yra nurodomas konteinerio plotis W.
2. Antroje failo eilutėje yra nurodomas konteinerio aukštis H.
3. Trečioje failo eilutėje yra nurodomas konteinerio ilgis D.
4. Kitose failo eilutėse yra surašomi dėžių matmenys (plotis wj, aukštis hj ir
   ilgis dj), atskirti bent vienu tarpu.

Duomenų failas ir rezultatų failas parenkami spragtelėjus meniu punktą <Failai>
ir pasirinkus atitinkamai <Duomenų failas> arba <Rezultatų failas> bei nurodant
failų buvimo vietą. Nepasirinkus šių failų, neleidžiama pradėti skaičiavimų.
Uždavinio sprendimui taikomų dažnumo funkcijų ir prioriteto taisyklių parinkimas
atliekamas skiltyje <Euristikos parametrai>.
Skaičiavimams pradėti paspaudžiamas mygtukas <Vykdyti>.
Skaičiavimo rezultatai bus išsaugomi vartotojo anksčiau nurodytame faile lentelių
pavidalu bei parodomi ekrane.
Programos darbas sėkmingai užbaigiamas pasirinkus meniu punktą <Baigti>.
_____________________________________________________________________________
Jei sprendžiamo CLP pradiniai duomenys turi būti generuojami programos vykdymo
metu, tuomet skaičiavimams atlikti pasirenkama skiltis <Generuoti duomenys>.

Rezultatų failas parenkamas spragtelėjus meniu punktą <Failai> ir pasirinkus
<Rezultatų failas> bei nurodant failo buvimo vietą. Nepasirinkus šio failo,
neleidžiama pradėti skaičiavimų.
Programoje realizuota galimybė sugeneruotus duomenis (konteinerio ir dėžių
išmatavimus) išsaugoti naujame faile, atitinkančiame anksčiau aptarto failo
Duomenys.txt struktūrą. Tam reikia programos lange uždėti varnelę prie
<Išsaugoti sugeneruotus duomenis faile>. Taip sudarytą failą vėliau galima
sėkmingai naudoti programos <Duomenys iš failo> skiltyje.

Generuojant duomenis turi būti įvesti šie parametrai:
1. konteinerio plotis W, aukštis H ir ilgis D;
2. a - minimali dėžės dimensija, b - maksimali dėžės dimensija; dežių išmatavimai
   generuojami pagal tolygųjį diskretųjį intervale [a; b] skirstinį.
3. generavimo parametras T; dėžės generuojamos tol, kol jų bendras tūris viršija
   T% konteinerio tūrio.
4. krovinio tipo parametras k; kai k=0 - stipriai įvairiarūšis; kai k=1 -
   vienarūšis; kai k=2,3,... - silpnai įvairiarūšis (k<<n, n - sugeneruotų dėžių
   kiekis).

Uždavinio sprendimui taikomų dažnumo funkcijų ir prioriteto taisyklių parinkimas
atliekamas skiltyje <Euristikos parametrai>.
Skaičiavimams pradėti paspaudžiamas mygtukas <Vykdyti>.
Skaičiavimo rezultatai bus išsaugomi vartotojo anksčiau nurodytame faile lentelių
pavidalu bei parodomi ekrane.
Programos darbas sėkmingai užbaigiamas pasirinkus meniu punktą <Baigti>.
_____________________________________________________________________________

Euristikos tyrimui atlikti pasirenkama skiltis <Testavimas>.

Ekperimentiniai duomenys yra generuojami pagal tą pačią metodiką, kuri buvo
aptarta prie skilties <Generuoti duomenys>, todėl turi būti įvestos tų pačių
parametrų (1)-(4) reikšmės.

Šiuo atveju papildomai prašoma įvesti dar du parametrus, t.y.:
5. realizacijų skaičius, apibrėžiantis sprendžiamų pakavimo uždavinių kiekį;
6. laiko limitas (sekundėmis), kuri viršijus skaičiavimai yra nutraukiami.

Uždavinio sprendimui taikomų dažnumo funkcijų ir prioriteto taisyklių parinkimas
atliekamas skiltyje <Euristikos parametrai>.
Skaičiavimams pradėti paspaudžiamas mygtukas <Vykdyti>.
Skaičiavimo rezultatai bus išsaugomi faile TestRez.txt, kuris, jei dar nebuvo
sukurtas, sukuriamas pagrindiniame programos kataloge.
Programos darbas sėkmingai užbaigiamas pasirinkus meniu punktą <Baigti>.
```

6 PRIEDAS. PROGRAMOS KODAS

Šiame priede yra pateikiami esminiai euristikos procedūrų programinio kodo fragmentai. Tuo tarpu originalų ir išsamų programinio kodo variantą galima rasti literatūros šaltinyje [14].

```
//---------------------------------------------------------------------------
//                              Procedūra FINDVECT
//---------------------------------------------------------------------------
state*  TForm1::findvect(stype ws, state *f, state *l)
{
  state *m;
  if (f > l) error("tuščia aibė");
  if (f->wsum >  ws) return NULL;
  if (l->wsum <= ws) return l;
  while (l - f > SYNC) {
    m = f + (l - f) / 2;
    if (m->wsum > ws) l = m-1; else f = m;
  }
  while (l->wsum > ws) l--;
  return l;
}
//---------------------------------------------------------------------------
//                              Procedūra PUSH
//---------------------------------------------------------------------------
void TForm1::push(allinfo *a, int side, item *f, item *l)
{
  interval *pos;
  if (l+1 == f) return;
  switch (side) {
    case LEFT : pos = a->intv1; (a->intv1)++; break;
    case RIGHT: pos = a->intv2; (a->intv2)--; break;
  }
  if (a->intv1 == a->intv2)  error("");
  pos->f = f; pos->l = l;
}
//---------------------------------------------------------------------------
//                              Procedūra POP
//---------------------------------------------------------------------------
void TForm1::pop(allinfo *a, int side, item **f, item **l)
{
  interval *pos;
  switch (side) {
    case LEFT : if (a->intv1 == a->intv1b)  error("");
                        (a->intv1)--; pos = a->intv1; break;
    case RIGHT: if (a->intv2 == a->intv2b)  error("");
                        (a->intv2)++; pos = a->intv2; break;
  }
  *f = pos->f; *l = pos->l;
}
//---------------------------------------------------------------------------
//                              Procedūra IMPROVESOLUTION
//---------------------------------------------------------------------------
void TForm1::improvesolution(allinfo *a, state *v)
{
  if (v->wsum > a->c)  error("");
  if (v->psum < a->z)  error("");
  a->z      = v->psum;
  a->zwsum  = v->wsum;
  a->ovect  = v->vect;
  memcpy(a->ovitem, a->vitem, sizeof(item *) * MAXV);
}
//---------------------------------------------------------------------------
//                              Procedūra DEFINESOLUTION
//---------------------------------------------------------------------------
void TForm1::definesolution(allinfo *a)
{
  register item *i, *m;
  item *f, *l;
  stype psum, wsum;
  btype j, k;
  if (a->firsttime) {
    if (a->z == a->lb) { a->welldef = TRUE; return; }
    a->zstar = a->z;
    a->wstar = a->zwsum;
    for (i = a->h+1, m = a->b; i != m; i++) i->k = 1;
    for (i = a->b, m = a->litem+1; i != m; i++) i->k = 0;
    a->firsttime = FALSE;
  }
  psum = a->z;
  wsum = a->zwsum;
  f    = a->fsort - 1;
```

```
  l    = a->lsort + 1;
  for (j = 0; j < MAXV; j++) {
    k = a->ovect & ((btype) 1 << j);
    i = a->ovitem[j]; if (i == NULL) continue;
    if (i->k == 1) {
      if (i > f) f = i;
      if (k) { psum += i->p; wsum += i->w; i->k = 0; }
    } else {
      if (i < l) l = i;
      if (k) { psum -= i->p; wsum -= i->w; i->k = 1; }
    }
  }
  a->welldef = (psum == a->psumb) && (wsum == a->wsumb);
  if (!a->welldef) {
    a->fsort = f + 1;
    a->lsort = l - 1;
    a->intv1 = a->intv1b;
    a->intv2 = a->intv2b;
    a->c  = wsum;
    a->z  = psum - 1;
    a->ub = psum;
  }
}
//---------------------------------------------------------------------------
//                              Procedūra PARTSORT
//---------------------------------------------------------------------------
void TForm1::partsort(allinfo *a, item *f, item *l, stype ws, int what)
{
  register itype mp, mw;
  register item *i, *j, *m;
  register stype wi;
  int d;
  d = l - f + 1;
  if (d > 1) {
    m = f + d / 2;
    if (DET(f->p, f->w, m->p, m->w) < 0) SWAP(f, m);
    if (d > 2) {
      if (DET(m->p, m->w, l->p, l->w) < 0) {
        SWAP(m, l);
        if (DET(f->p, f->w, m->p, m->w) < 0) SWAP(f, m);
      }
    }
  }
  if (d > 3) {
    mp = m->p; mw = m->w; i = f; j = l; wi = ws;
    for (;;) {
      do { wi += i->w; i++; } while (DET(i->p, i->w, mp, mw) > 0);
      do {             j--; } while (DET(j->p, j->w, mp, mw) < 0);
      if (i > j) break;
      SWAP(i, j);
    }
    if (wi <= a->cstar) {
      if (what ==  SORTALL) partsort(a, f, i-1, ws, what);
      if (what == PARTIATE) push(a, LEFT, f, i-1);
      partsort(a, i, l, wi, what);
    } else {
      if (what ==  SORTALL) partsort(a, i, l, wi, what);
      if (what == PARTIATE) push(a, RIGHT, i,  l);
      partsort(a, f, i-1, ws, what);
    }
  }
  if ((d <= 3) || (what == SORTALL)) {
    a->fpart  = f;
    a->lpart  = l;
    a->wfpart = ws;
  }
}
//---------------------------------------------------------------------------
//                              Procedūra HASCHANCE
//---------------------------------------------------------------------------
booleann TForm1::haschance(allinfo *a, item *i, int side)
{
  register state *j, *m;
  register ptype p, w, r;
  register stype pp, ww;
  if (a->d.size == 0) return FALSE;
  if (side == RIGHT) {
    if (a->d.fset->wsum <= a->c - i->w) return TRUE;
    p = a->ps; w = a->ws;
    pp = i->p - a->z - 1; ww = i->w - a->c;
    r = -DET(pp, ww, p, w);
    for (j = a->d.fset, m = a->d.lset + 1; j != m; j++) {
      if (DET(j->psum, j->wsum, p, w) >= r) return TRUE;
    }
  } else {
    if (a->d.lset->wsum > a->c + i->w) return TRUE;
```

```
    p = a->pt; w = a->wt;
    pp = -i->p - a->z - 1; ww = -i->w - a->c;
    r = -DET(pp, ww, p, w);
    for (j = a->d.lset, m = a->d.fset - 1; j != m; j--) {
      if (DET(j->psum, j->wsum, p, w) >= r) return TRUE;
    }
  }
  return FALSE;
}
//---------------------------------------------------------------------------
//                              Procedūra MULTIPLY
//---------------------------------------------------------------------------
void TForm1::multiply(allinfo *a, item *h, int side)
{
  register state *i, *j, *k, *m;
  register itype p, w;
  register btype mask0, mask1;
  state *r1, *rm;
  if (a->d.size == 0) return;
  if (side == RIGHT) { p = h->p; w = h->w; } else { p = -h->p; w = -h->w; }
  if (2*a->d.size + 2 > MAXSTATES)  error("");
  a->vno++;
  if (a->vno == MAXV) a->vno = 0;
  mask1 = ((btype) 1 << a->vno);
  mask0 = ~mask1;
  a->vitem[a->vno] = h;
  r1 = a->d.fset; rm = a->d.lset; k = a->d.set1; m = rm + 1;
  k->psum = -1;
  k->wsum = r1->wsum + abs(p) + 1;
  m->wsum = rm->wsum + abs(w) + 1;
  for (i = r1, j = r1; (i != m) || (j != m); ) {
    if (i->wsum <= j->wsum + w) {
      if (i->psum > k->psum) {
        if (i->wsum > k->wsum) k++;
        k->psum = i->psum; k->wsum = i->wsum;
        k->vect = i->vect & mask0;
      }
      i++;
    } else {
      if (j->psum + p > k->psum) {
        if (j->wsum + w > k->wsum) k++;
        k->psum = j->psum + p; k->wsum = j->wsum + w;
        k->vect = j->vect | mask1;
      }
      j++;
    }
  }
  a->d.fset = a->d.set1;
  a->d.lset = k;
  a->d.size  = a->d.lset - a->d.fset + 1;
}
//---------------------------------------------------------------------------
//                              Procedūra SIMPREDUCE
//---------------------------------------------------------------------------
void TForm1::simpreduce(int side, item **f, item **l, allinfo *a)
{
  register item *i, *j, *k;
  register ptype pb, wb;
  register ptype q, r;
  register int redu;
  if (a->d.size == 0) { *f = *l+1; return; }
  if (*l < *f) return;
  pb = a->b->p; wb = a->b->w;
  q = DET(a->z+1-a->psumb, a->c-a->wsumb, pb, wb);
  r = -DET(a->z+1-a->psumb, a->c-a->wsumb, pb, wb);
  i = *f; j = *l;
  redu = 0;
  if (side == LEFT) {
    k = a->fsort - 1;
    while (i <= j) {
      if (DET(j->p, j->w, pb, wb) > r) {
        SWAP(i, j); i++; redu++;
      } else {
        SWAP(j, k); j--; k--;
      }
    }
    *l = a->fsort - 1; *f = k + 1;
  } else {
    k = a->lsort + 1;
    while (i <= j) {
      if (DET(i->p, i->w, pb, wb) < q) {
        SWAP(i, j); j--; redu++;
      } else {
        SWAP(i, k); i++; k++;
      }
    }
```

```
    *f = a->lsort + 1; *l = k - 1;
  }
}
//---------------------------------------------------------------------------
//                              Procedūra REDUCESET
//---------------------------------------------------------------------------
void TForm1::reduceset(allinfo *a)
{
  register state *i, *m, *k;
  register ptype ps, ws, pt, wt, r;
  stype z, c;
  state *r1, *rm, *v;
  item *f, *l;
  if (a->d.size == 0) return;
  r1 = a->d.fset; rm = a->d.lset;
  v  = findvect(a->c, r1, rm);
  if (v == NULL) v = r1 - 1;
  else { if (v->psum > a->z) improvesolution(a, v); }
  c = a->c; z = a->z + 1; k = a->d.setm;
  if (a->s < a->fsort) {
    if (a->intv1 == a->intv1b) {
      ps = PMAX; ws = WMAX;
    } else {
      pop(a, LEFT, &f, &l);
      if (f < a->ftouch) a->ftouch = f;
      ps = f->p; ws = f->w;
      simpreduce(LEFT, &f, &l, a);
      if (f <= l) {
        partsort(a, f, l, 0, SORTALL); a->fsort = f;
        ps = a->s->p; ws = a->s->w;
      }
    }
  } else {
    ps = a->s->p; ws = a->s->w;
  }
  if (a->t > a->lsort) {
    if (a->intv2 == a->intv2b) {
      pt = PMIN; wt = WMIN;
    } else {
      pop(a, RIGHT, &f, &l);
      if (l > a->ltouch) a->ltouch = l;
      pt = l->p; wt = l->w;
      simpreduce(RIGHT, &f, &l, a);
      if (f <= l) {
        partsort(a, f, l, 0, SORTALL); a->lsort = l;
        pt = a->t->p; wt = a->t->w;
      }
    }
  } else {
    pt = a->t->p; wt = a->t->w;
  }
  r = DET(z, c, ps, ws);
  for (i = rm, m = v; i != m; i--) {
    if (DET(i->psum, i->wsum, ps, ws) >= r) {
      k--; *k = *i;
    }
  }
  r = DET(z, c, pt, wt);
  for (i = v, m = r1 - 1; i != m; i--) {
    if (DET(i->psum, i->wsum, pt, wt) >= r) {
      k--; *k = *i;
    }
  }
  a->ps = ps; a->ws = ws;
  a->pt = pt; a->wt = wt;
  a->d.fset = k;
  a->d.lset = a->d.setm - 1;
  a->d.size = a->d.lset - a->d.fset + 1;
}
//---------------------------------------------------------------------------
//                              Procedūra INITFIRST
//---------------------------------------------------------------------------
void TForm1::initfirst(allinfo *a, stype ps, stype ws)
{
  state *k;
  a->d.size  = 1;
  a->d.set1  = palloc1(MAXSTATES, sizeof(state));
  a->d.setm  = a->d.set1 + MAXSTATES - 1;
  a->d.fset  = a->d.set1;
  a->d.lset  = a->d.set1;
  k = a->d.fset;
  k->psum   = ps;
  k->wsum   = ws;
  k->vect   = 0;
}
//---------------------------------------------------------------------------
```

```
//                              Procedūra INITVECT
//---------------------------------------------------------------------------
void TForm1::initvect(allinfo *a)
{
  register btype i;
  for (i = 0; i < MAXV; i++) a->vitem[i] = NULL;
  a->vno = MAXV-1;
}
//---------------------------------------------------------------------------
//                              Procedūra FINDBREAK
//---------------------------------------------------------------------------
void TForm1::findbreak(allinfo *a)
{
  register item *i, *m;
  register stype psum, wsum, c;
  psum = 0; wsum = 0; c = a->cstar;
  for (i = a->h+1, m = a->litem+1; ; i++) {
    if (i == m) break;
    if (wsum + i->w > c) break;
    psum += i->p; wsum += i->w;
  }
  a->fsort   = a->fpart;
  a->lsort   = a->lpart;
  a->ftouch  = a->fpart;
  a->ltouch  = a->lpart;
  a->b       = i;
  a->psumb   = psum;
  a->wsumb   = wsum;
  a->zwsum   = 0;
  a->dantzig = (i == m ? psum : psum + ((c - wsum) * (stype) i->p) / i->w);
  for (i = a->b, m = a->litem+1; i != m; i++) {
    if (wsum + i->w <= c) { psum += i->p; wsum += i->w; }
  }
  if (psum > a->lb) {
    for (i = a->h+1, m = a->b; i != m; i++) i->k = 1;
    for (i = a->b, m = a->litem+1, wsum = a->wsumb; i != m; i++) {
      i->k = 0; if (wsum + i->w <= c) { wsum += i->w; i->k = 1; }
    }
    a->lb    = psum;
    a->z     = psum;
    a->zstar = psum;
    a->wstar = wsum;
  } else {
    a->z     = a->lb;
    a->zstar = a->lb;
    a->wstar = a->cstar;
  }
  a->c = a->cstar;
}
//---------------------------------------------------------------------------
//                              Procedūra MINKNAP
//---------------------------------------------------------------------------
void TForm1::minknap(allinfo *a, item *fitem, stype c)
{
  interval *inttab;
  a->h = fitem-1;
  a->cstar = c;
  a->lb    = 0;
  inttab   = palloc2(sizeof(interval), SORTSTACK);
  a->intv1 = a->intv1b = &inttab[0];
  a->intv2 = a->intv2b = &inttab[SORTSTACK - 1];
  a->fsort = a->litem; a->lsort = a->h+1;
  partsort(a, a->h+1, a->litem, 0, PARTIATE);
  findbreak(a);
  a->ub = a->dantzig;
  a->firsttime = TRUE;
  for (;;) {
    a->s = a->b-1;
    a->t = a->b;
    initfirst(a, a->psumb, a->wsumb);
    initvect(a);
    reduceset(a);
    while ((a->d.size > 0) && (a->z != a->ub)) {
      if (a->t <= a->lsort) {
            if (haschance(a, a->t, RIGHT)) multiply(a, a->t, RIGHT);
            (a->t)++;
      }
      reduceset(a);
      if (a->s >= a->fsort) {
            if (haschance(a, a->s, LEFT)) multiply(a, a->s, LEFT);
            (a->s)--;
      }
      reduceset(a);
    }
    pfree(a->d.set1);
    definesolution(a);
```

```
    if (a->welldef) break;
  }
  pfree(inttab);
}
//-----------------------------------------------------------------------------
//                              Procedūra FINDDIMENSIONS
//-----------------------------------------------------------------------------
void TForm1::finddimensions(allinfo *a)
{
  register item *i, *m;
  register itype mindim, maxdim;
  mindim = a->mx + a->my + a->mz; maxdim = 0;
  for (i = a->g, m = a->litem+1; i != m; i++) {
    if (i->dx < mindim) mindim = i->dx;
    if (i->dy < mindim) mindim = i->dy;
    if (i->dz < mindim) mindim = i->dz;
    if (i->dx > maxdim) maxdim = i->dx;
    if (i->dy > maxdim) maxdim = i->dy;
    if (i->dz > maxdim) maxdim = i->dz;
  }
  a->mindim = mindim;
  a->maxdim = maxdim;
}
//-----------------------------------------------------------------------------
//                              Procedūra FINDMINMAX
//-----------------------------------------------------------------------------
void TForm1::findminmax(allinfo *a)
{
  register item *i, *m;
  register itype mindim, maxdim;
  mindim = maxdim = a->g->dx;
  for (i = a->g, m = a->litem+1; i != m; i++) {
    if (i->dx < mindim) mindim = i->dx;
    if (i->dy < mindim) mindim = i->dy;
    if (i->dx > maxdim) maxdim = i->dx;
    if (i->dy > maxdim) maxdim = i->dy;
  }
  a->mindim = mindim;
  a->maxdim = maxdim;
}
//-----------------------------------------------------------------------------
//                              Procedūra FINDFITS
//-----------------------------------------------------------------------------
stype TForm1::findfits(allinfo *a, itype dz)
{
  register item *i, *m;
  register stype vol;
  vol = a->mcut * dz;
  for (i = a->g, m = a->litem+1; i != m; i++) {
    if ((i->dx <= dz) || (i->dy <= dz) || (i->dz <= dz)) vol -= VOL(i);
  }
  return MAX(0, vol);
}
//-----------------------------------------------------------------------------
//                              Procedūra GIVEDIM
//-----------------------------------------------------------------------------
void TForm1::givedim(item *i, int h, itype *x, itype *y, itype *z)
{
  switch (h) {
    case 0: *x = i->dy; *y = i->dz; *z = i->dx; break;
    case 1: *x = i->dx; *y = i->dz; *z = i->dy; break;
    case 2: *x = i->dx; *y = i->dy; *z = i->dz; break;
  }
}
//-----------------------------------------------------------------------------
//                              Procedūra BESTMATCH
//-----------------------------------------------------------------------------
void TForm1::bestmatch(item *s, item *i, itype z, itype mindim,
               item **pair, int *h1, int *k1, double *mfill)
{
  register item *j;
  itype ix, iy, iz, jx, jy, jz, sx, sy, sz;
  double fill, jfill;
  int h, k, a[3], b[3];
  *pair = NULL; *h1 = *k1 = 0;
  if (i->dx + mindim > z) return;
  *mfill = VOL(i)/(double)(i->w*(long)z);
  a[0] = i->dx; a[1] = i->dy; a[2] = i->dz;
  for (j = s; j != i; j++) {
    if (i->dx + j->dx > z) continue;
    jfill = VOL(j)/(double)(j->w*(long)z);
    b[0] = j->dx; b[1] = j->dy; b[2] = j->dz;
    for (h = 0, k = 2; h != 3; h++) {
      while (a[h] + b[k] > z) { k--; if (k < 0) break; }
      if (k < 0) break;
      iz = a[h]; jz = b[k]; sz = iz + jz;  if (sz > z)  error("");
```

```
      switch (h) {
        case 0: ix = a[1]; iy = a[2]; break;
        case 1: ix = a[0]; iy = a[2]; break;
        case 2: ix = a[0]; iy = a[1]; break;
      }
      switch (k) {
        case 0: jx = b[1]; jy = b[2]; break;
        case 1: jx = b[0]; jy = b[2]; break;
        case 2: jx = b[0]; jy = b[1]; break;
      }
      sx = MAX(ix, jx); sy = MAX(iy, jy);
      fill = (VOL(i)+VOL(j))/(double)(sx*(long)sy*z);
      if ((fill > *mfill + epsilon) && (fill > jfill + epsilon)) {
        *h1 = h; *k1 = k; *mfill = fill; *pair = j;
      }
    }
  }
}
//---------------------------------------------------------------------------
//                              Procedūra TURNBOXES
//---------------------------------------------------------------------------
item* TForm1::turnboxes(allinfo *a, itype z)
{
  register item *i, *m, *s;
  itype ix, iy, iz, jx, jy, jz, sx, sy, sz;
  double mfill;
  int h1, k1;
  item *j;
  ix = iy = iz = jx = jy = jz = 0;
  for (i = s = a->g, m = a->litem+1; i != m; i++) {
    if (i->dx > i->dy) SWAPI(i->dx, i->dy);
    if (i->dy > i->dz) SWAPI(i->dy, i->dz);
    if (i->dx > i->dy) SWAPI(i->dx, i->dy);
    if (i->dx > i->dy || i->dy > i->dz) error("");
    i->x = i->y = i->z = i->p = 0; i->k = FALSE; i->pair = NULL;
    if (i->dx > z) { SWAP(s,i); s++; continue; }
    if (i->dz <= z) { i->w = i->dx * i->dy; continue; }
    if (i->dy <= z) { i->w = i->dx * i->dz; continue; }
    if (i->dx <= z) { i->w = i->dy * i->dz; continue; }
    error("");
  }
  if (s > a->litem) { printitems(a); error(""); }
  for (i = a->litem; i != s-1; i--) {
    bestmatch(s, i, z, a->mindim, &j, &h1, &k1, &mfill);
    if (j != NULL) {
      givedim(i, h1, &ix, &iy, &iz); givedim(j, k1, &jx, &jy, &jz);
      if (ix * (long) iy < jx * (long) jy) { SWAP(i,j); SWAPI(h1,k1); }
      j->p = 0; i->p = (VOL(i)+VOL(j))/100;
      givedim(i, h1, &ix, &iy, &iz); givedim(j, k1, &jx, &jy, &jz);
      sx = MAX(ix, jx); sy = MAX(iy, jy);
      sz = iz + jz; if (sz > z)  error("bad pair");
      j->dx = jx; j->dy = jy; j->dz = jz;
      j->x  = ix; j->y  = iy; j->z  = iz;
      i->dx = sx; i->dy = sy; i->dz = sz;
      i->pair = s; SWAP(s,j); s++;
    } else {
      i->p = VOL(i)/100;
      if (i->dz <= z) continue;
      if (i->dy <= z) { SWAPI(i->dy, i->dz); continue; }
      if (i->dx <= z) { SWAPI(i->dx, i->dz); SWAPI(i->dx, i->dy); }
    }
  }
  return s;
}
//---------------------------------------------------------------------------
//                              Procedūra RESTORELAYER
//---------------------------------------------------------------------------
item* TForm1::restorelayer(allinfo *a, item *fitem, itype dz)
{
  register item *i, *j, *m, *s;
  stype vol;
  itype sx;
  for (i = fitem, m = a->litem+1; i != m; i++) {
    i->p = 0;
    if ((i->k) && (i->pair)) { i->pair->w = i->w; i->pair->p = i->pair->k = 1; }
  }
  for (i = fitem, m = a->litem+1; i != m; i++) {
    if (i->pair != NULL) {
      j = i->pair; sx = MAX(j->x, j->dx);
      if (sx != i->dx) { SWAPI(j->x, j->y); SWAPI(j->dx, j->dy); }
      i->dx = j->x; i->dy = j->y; i->dz = j->z;
      j->x  = i->x; j->y  = i->y; j->z  = i->z+i->dz;
      if (i->dx*(long)i->dy < j->dx*(long)j->dy)  error("");
      i->pair = NULL;
    }
  }
```

```
  s = fitem; vol = a->lvol;
  for (i = fitem, m = a->litem+1; i != m; i++) {
    if (i->k) { vol -= VOL(i); i->k = dz; SWAP(s,i); s++; } else { i->w = 0; }
  }
  if (vol > a->rectloss)  error(");
  a->rectloss = vol;
  return s;
}
//---------------------------------------------------------------------------
//                              Procedūra FINDLARGEST
//---------------------------------------------------------------------------
void TForm1::findlargest(allinfo *a, itype *t, itype maxd, int no)
{
  register item *i, *m;
  int *occ, j, k, lj, mx;
  itype *h;
  occ = palloc3(sizeof(int), a->maxdim+1);
  for (j = a->maxdim, lj = a->mindim-1; j != lj; j--) occ[j] = 0;
  //-------------------------------------------------------------------
  if (kodasS_d==1){
     for (i = a->g, m = a->litem+1; i != m; i++) {
          occ[i->dx]++; occ[i->dy]++; occ[i->dz]++;
     }
  }
  if (kodasS_d==2){
     for (i = a->g, m = a->litem+1; i != m; i++) {
          if ((i->dx >= i->dy) && (i->dx >= i->dz))
              occ[i->dx]++;
          else
              if (i->dy >= i->dz)
                  occ[i->dy]++;
              else occ[i->dz]++;
     }
  }
  //-------------------------------------------------------------------
  j = MIN(maxd, a->maxdim);
  if (kodasS_p==1){
      for (h = t, k = 0, lj = a->mindim-1, mx = 0; j != lj; j--) {
          if (occ[j] > 0) { *h = j; h++; k++; mx = occ[j]; if (k == no) break; }
      }
  }
  if (kodasS_p==2){
      for (h = t, k = 0, lj = a->mindim-1, mx = 0; j != lj; j--) {
          if (occ[j] > k) { *h = j; h++; k++; mx = occ[j]; if (k == no) break; }
      }
  }
  if (kodasS_p==3){
    int *occ_copy;
    int *occ_index;
    int z;
    occ_copy = palloc3(sizeof(int), a->maxdim+1);
    occ_index = palloc3(sizeof(int), a->maxdim+1);
    for (z = a->maxdim, lj = a->mindim-1; z != lj; z--)
    {
         occ_copy[z] = occ[z];
         occ_index[z]=z;
    }
    QuickSort(occ_copy,occ_index,a->mindim,a->maxdim);
    h = t;   mx = 0;
    for (k=0; k<no; k++)
    {
      if (a->mindim+k != a->maxdim)
      {
         *h = occ_index[a->mindim+k]; h++; mx = occ_copy[a->mindim+k];
      }
      else break;
    }
     pfree(occ_copy);
     pfree(occ_index);
  }
  if (k < no) *h = 0;
  pfree(occ);
}
//---------------------------------------------------------------------------
//                              Procedūra FINDCOMMON
//---------------------------------------------------------------------------
void TForm1::findcommon(allinfo *a, itype *t, itype maxd, int no)
{
  register item *i, *m;
  int *occ, j, k, lj, mx;
  itype *h;
  occ = palloc3(sizeof(int), a->maxdim+1);
  for (j = a->maxdim, lj = a->mindim-1; j != lj; j--) occ[j] = 0;
  //-------------------------------------------------------------------
  if (kodasJ_d==1){
     for (i = a->g, m = a->litem+1; i != m; i++) {
```

```
          occ[i->dx]++; occ[i->dy]++;
      }
   }
   if (kodasJ_d==2){
      for (i = a->g, m = a->litem+1; i != m; i++) {
          if (i->dx > i->dy)
              occ[i->dx]++;
          else
              occ[i->dy]++;
      }
   }
   //---------------------------------------------------------------------------
   j = MIN(maxd, a->maxdim);
   if (kodasJ_p==1){
      for (h = t, k = 0, lj = a->mindim-1, mx = 1; j != lj; j--) {
          if (occ[j] > 0) { *h = j; h++; k++; mx = occ[j]; if (k == no) break; }
      }
   }
   if (kodasJ_p==2){
      for (h = t, k = 0, lj = a->mindim-1, mx = 1; j != lj; j--) {
          if (occ[j] > k)
          {
              *h = j; h++; k++; mx = occ[j];
              if (k == no) break;
          }
      }
   }
   if(k < no) *h = 0;
   pfree(occ);
}
//---------------------------------------------------------------------------
//                              Procedūra PREPARE
//---------------------------------------------------------------------------
item* TForm1::prepare(allinfo *a, item *fitem, itype dx)
{
   register item *i, *m, *s;
   itype no;
   for (i = s = fitem, m = a->litem+1, no = 0; i != m; i++) {
     i->k = FALSE;
     if (i->dy < i->dx) SWAPI(i->dx, i->dy);
     if (i->dy <= dx) SWAPI(i->dx, i->dy);
     if (i->dx > dx) { SWAP(s,i); s++; continue; }
     i->w = i->dy;
     no++;
   }
   return s;
}
//---------------------------------------------------------------------------
//                              Procedūra GIVEX
//---------------------------------------------------------------------------
stype TForm1::givex(allinfo *a, item *fitem, itype x, itype y, itype z,
            itype *dx, booleann xdir)
{
   register item *i, *m, *s;
   stype vsum;
   itype d;
   s = a->g; vsum = 0; d = 0;
   for (i = fitem, m = a->litem+1; i != m; i++) {
     if (i->k) {
       i->x = x; i->y = y; i->z = z; i->w = z;
       if (i->dx > d) d = i->dx;
       if (xdir) SWAPI(i->dx, i->dy);
       if (xdir) x += i->dx; else y += i->dy;
       vsum += i->p * (long) 100;
       SWAP(s,i); s++;
     }
   }
   if (d < *dx) { *dx = d; }
   a->g = s;
   return vsum;
}
//---------------------------------------------------------------------------
//                              Procedūra ONESTRIP
//---------------------------------------------------------------------------
stype TForm1::onestrip(allinfo *a, itype x, itype y, itype z,
                itype *dx, itype dy, booleann xdim)
{
   item *fitem;
   fitem = prepare(a, a->g, *dx);
   if (fitem > a->litem) return 0;
   a->knapsacks++;
   minknap(a, fitem, dy);
   return givex(a, fitem, x, y, z, dx, xdim);
}
//---------------------------------------------------------------------------
//                              Procedūra SAVESOL
```

```
//---------------------------------------------------------------------------
void TForm1::savesol(allinfo *a, stype loss)
{
  item *i, *m;
  stype vol;
  for (i = a->g, m = a->litem+1; i != m; i++) { i->x = i->y = i->z = i->k = 0; }
  vol = a->lvol; m = a->litem+1;
  for (i = a->rstart; i != m; i++) if (i->k) vol -= i->p * (long)100;
  if (vol != loss) { printitems(a); error(""); }
  if (vol < 0) { printitems(a); error("");  }
  memcpy(a->frect, a->fitem, a->n * sizeof(item));
  a->rectloss = loss;
}
//---------------------------------------------------------------------------
//                              Procedūra STRIPRECURS
//---------------------------------------------------------------------------
void TForm1::striprecurs(allinfo *a, stype oldloss, booleann xdir,
                itype x, itype y, itype z, itype dz)
{
  item *saveg, *gafter;
  itype c, b, mx, my, md, dx, h, x1, y1, d[MCUT2];
  stype loss, vol;
  mx = a->mx - x;
  my = a->my - y;
  md = MIN(mx, my);
  saveg = prepare(a, a->g, md);
  c = (xdir ? mx : my);
  b = (xdir ? my : mx);
  if (saveg > a->litem) {
    loss = oldloss + mx * (long)my * dz;
    if (loss < a->rectloss) savesol(a, loss);
  } else {
    findcommon(a, d, b, MCUT2);
    for (h = 0, dx = d[0]+1; h != MCUT2; h++) {
      if (d[h] == 0) break;
      if (d[h] >= dx) continue;
      dx = d[h]; a->g = saveg;
      vol = onestrip(a, x, y, z, &dx, c, xdir);
      loss = oldloss + dx * (long)c * dz - vol;
      if (loss >= a->rectloss) continue;
      if (loss < 0) { printitems(a); error(""); }
      if (dx == 0) a->g = a->litem+1;
      x1 = (xdir ? x : x+dx); y1 = (xdir ? y+dx : y); gafter = a->g;
      striprecurs(a, loss, (md < 3*a->maxdim/2 ? xdir : !xdir), x1, y1, z, dz);
      if (md < 3*a->maxdim/2) continue;
      a->g = gafter; striprecurs(a, loss, xdir, x1, y1, z, dz);
    }
  }
}
//---------------------------------------------------------------------------
//                              Procedūra RECTANGLE
//---------------------------------------------------------------------------
stype TForm1::rectangle(allinfo *a, itype z, itype dz)
{
  a->rstart   = a->g;
  a->g        = turnboxes(a, dz);
  a->lvol     = a->mx * (long) a->my * dz;
  a->rectloss = a->lvol;
  findminmax(a); savesol(a, a->lvol);
  striprecurs(a, 0, TRUE, 0, 0, z, dz);
  memcpy(a->fitem, a->frect, a->n * sizeof(item));
  a->g = restorelayer(a, a->rstart, dz);
  return a->rectloss;
}
//---------------------------------------------------------------------------
//                              Procedūra BRANCH
//---------------------------------------------------------------------------
booleann TForm1::branch(allinfo *a, item *f, itype z, stype oldloss, int levels)
{
  itype d[MCUT3], dz;
  stype loss;
  double fill;
  booleann ok;
  int j;
  a->iterates++; a->g = f;
  finddimensions(a);
  if ((a->mz - z < a->maxdim) || (a->g > a->litem)) {
    dz = a->mz - z;
    if ((dz < a->mindim) || (a->g > a->litem)) loss = oldloss + a->mcut * dz;
    else {
      loss = oldloss + findfits(a, dz);
      if (loss < a->minloss) loss = oldloss + rectangle(a, z, dz);
    }
    if (loss < a->minloss) {
      a->minloss = loss; a->miss = a->litem - a->g + 1; a->lev = levels;
      fill = (a->mvol - loss) / (double) a->mvol;
```

```
      memcpy(a->fsol, a->fitem, a->n * sizeof(item));
    }
    return (a->miss == 0);
  }
  ok = FALSE;
  findlargest(a, d, a->mz - z, MCUT3);
  for (j = 0; j != MCUT3; j++) {
    a->g = f; dz = d[j]; if (dz <= a->maxdim/2) continue;
    loss = oldloss + rectangle(a, z, dz);
    if (loss < a->minloss) ok = branch(a, a->g, z+dz, loss, levels+1);
    if (ok) break;
  }
  return ok;
}
//---------------------------------------------------------------------------
//                              Procedūra INITITEMS
//---------------------------------------------------------------------------
void TForm1::inititems(allinfo *a, int *w, int *h, int *d)
{
  item *i, *m;
  int j;
  for (i = a->fitem, m = a->litem+1, j = 0; i != m; i++, j++) {
    i->no = j+1; i->dx = w[j]; i->dy = h[j]; i->dz = d[j];
  }
  for (i = a->fitem, m = a->litem+1; i != m; i++) {
    i->x = i->y = i->z = i->p = i->w = i->k = 0; i->pair = NULL;
  }
}
//---------------------------------------------------------------------------
//                              Procedūra RETURNTITEMS
//---------------------------------------------------------------------------
void TForm1::returnitems(allinfo *a, int *w, int *h, int *d,
                 int *x, int *y, int *z, int *k, int *vol)
{
  stype svol;
  item *i, *m;
  int j;
  for (i = a->fsol, m = a->lsol+1, svol = 0; i != m; i++) {
    if (i->k) svol += VOL(i);
    j = i->no-1;
    w[j] = i->dx; h[j] = i->dy; d[j] = i->dz;
    x[j] = i->x; y[j] = i->y; z[j] = i->z; k[j] = i->k;
  }
  *vol = svol;
}
//---------------------------------------------------------------------------
//                              Procedūra CONTLOAD
//---------------------------------------------------------------------------
void TForm1::contload(int n, int W, int H, int D,
              int *w, int *h, int *d,
              int *x, int *y, int *z, int *k,
              int *vol)
{
  allinfo a;
  item *t;
  t           = palloc4(n, sizeof(item));
  a.n         = n;
  a.fitem     = t;
  a.litem     = t + n - 1;
  a.mx        = W;
  a.my        = H;
  a.mz        = D;
  a.fsol      = palloc4(n, sizeof(item));
  a.lsol      = a.fsol + n - 1;
  a.frect     = palloc4(n, sizeof(item));
  a.mcut      = W * (long) H;
  a.mvol      = a.mcut * D;
  a.minloss   = a.mvol;
  a.iterates  = 0;
  a.knapsacks = 0;
  inititems(&a, w, h, d);
  a.g = a.fitem;
  branch(&a, a.fitem, 0, 0, 1);
  returnitems(&a, w, h, d, x, y, z, k, vol);
  pfree(t);
  pfree(a.fsol);
  pfree(a.frect);
}
//---------------------------------------------------------------------------
//                              Procedūra CHECKSOL
//---------------------------------------------------------------------------
void TForm1::checksol(item *f, item *l, int W, int H, int D, int vol)
{
  ofstream fr(RezultatuFailas, ios::app);
  item *i, *j, *m;
  int svol;
```

```
  for (i = f, m = l+1, svol = 0; i != m; i++) {
    if (!i->k) continue;
    svol += VOL(i);
    if ((i->x + i->dx > W) || (i->y + i->dy > H) || (i->z + i->dz > D)) {
      fr << "bloga dėžės nr=" << i->no <<" padėtis [dx,dy,dz]=["<< i->dx << "," << i->dy << "," << i-
>dz << "], [x,y,z]=[" << i->x << "," << i->y << "," << i->z << "]." << endl;
      exit(-1);
    }
    for (j = f; j != m; j++) {
      if (i == j) continue;
      if (i->no == j->no) { fr << "padubliuota dėžė nr=" << i->no << endl; exit(-1); }
      if (!j->k) continue;
      if ((i->x + i->dx > j->x) && (j->x + j->dx > i->x) &&
          (i->y + i->dy > j->y) && (j->y + j->dy > i->y) &&
          (i->z + i->dz > j->z) && (j->z + j->dz > i->z)) {
        fr << "dėžė nr=" << i->no << "ir nr=" << j->no << "persidengia: [" << i->dx << "," << i->dy <<
"," << i->dz << "], [" << j->dx << "," << j->dy << "," << j->dz << "]" << endl;
        exit(-1);
      }
    }
  }
  if (vol != svol) {
    fr << "neteisinga tikslo funkcijos reikšmė: svol="<< svol << ", vol=" << vol << endl; exit(-1);
  }
  fr.close();
}
//---------------------------------------------------------------------------
//                              Procedūra RANDOMTYPE
//---------------------------------------------------------------------------
void TForm1::randomtype(item *i, int mindim, int maxdim)
{
  i->dx = randm(maxdim-mindim+1) + mindim;
  i->dy = randm(maxdim-mindim+1) + mindim;
  i->dz = randm(maxdim-mindim+1) + mindim;
  i->x = i->y = i->z = i->k = 0;
}
//---------------------------------------------------------------------------
//                              Procedūra MAKETEST
//---------------------------------------------------------------------------
void TForm1::maketest(item *f, item **l, int *mx, int *my, int *mz,
              int mindim, int maxdim, int fillpct, int maxtyp)
{
  item t[MAXITEMS];
  register item *i, *j, *m;
  register stype vol;
  int no;
  if(PageControl1->ActivePage==TabSheet2){
  *mx = StrToInt(Edit5->Text);
  *my = StrToInt(Edit6->Text);
  *mz = StrToInt(Edit7->Text);
  }
  if(PageControl1->ActivePage==TabSheet6){
  *mx = StrToInt(Edit8->Text);
  *my = StrToInt(Edit9->Text);
  *mz = StrToInt(Edit10->Text);
  }
  for (i = t, m = t+maxtyp; i != m; i++) {
     randomtype(i, mindim, maxdim);
  }
  vol = (fillpct * *mx * (double) *my * *mz) / 100;
  for (i = f, no = 1; ; i++, no++) {
    //if (i == *l) { ShowMessage("per mažas masyvas"); exit(-1); }
    if (maxtyp == 0)
    {
       randomtype(i, mindim, maxdim);
    }
    else
    {
       j  = t + randm(maxtyp); *i = *j;
    }
    i->no = no; vol -= VOL(i);
    if (vol < 0) { *l = i; break; }
  }
}
//---------------------------------------------------------------------------
//                              Procedūra PREPAREITEMS
//---------------------------------------------------------------------------
void TForm1::prepareitems(item *f, item *l, int *w, int *h, int *d)
{
  item *i;
  int k;
  for (i = f, k = 0; i != l+1; i++, k++) {
    w[k] = i->dx; h[k] = i->dy; d[k] = i->dz;
  }
}
//---------------------------------------------------------------------------
```

```
//                              Procedūra UPDATEITEMS
//---------------------------------------------------------------------------
void TForm1::updateitems(item *f, item *l, int *w, int *h, int *d,
                 int *x, int *y, int *z, int *k, int *miss)
{
  item *i;
  int j;
  *miss = 0;
  for (i = f, j = 0; i != l+1; i++, j++) {
    i->dx = w[j]; i->dy = h[j]; i->dz = d[j];
    i->x  = x[j]; i->y  = y[j]; i->z  = z[j]; i->k = k[j];
    if (!k[j]) (*miss)++;
  }
}
//---------------------------------------------------------------------------
```