KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

VERSLO INFORMATIKOS KATEDRA

Nerijus Ambrazas

Pasiekiamų būsenų grafo sudarymo sudėtingumo tyrimas

Magistro darbas

Darbo vadovas

dr. D. Makackas

Kaunas, 2008

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

VERSLO INFORMATIKOS KATEDRA

Nerijus Ambrazas

Pasiekiamų būsenų grafo sudarymo sudėtingumo tyrimas

Magistro darbas

Vadovas
doc. dr. D. Makackas
2008-05-23

Recenzentas
doc. dr. E. Valakevičius
2008-05-26

Atliko
IFM- 2/1 gr. stud.
Nerijus Ambrazas
2008-05-20

Kaunas, 2008

TURINYS

Paveikslėliai

ĮVADAS

Šiuo metu realiojo laiko sudėtingų sistemų projektavimui keliami dideli saugumo, patikimumo ir našumo reikalavimai, todėl vis plačiau taikomi formalieji metodai. Projektuojant tokias sistemas formalūs metodai naudojami siekiant sukurti modelius, kurie leistų validuoti, verifikuoti bei modeliuoti analizuojamas sistemas. Sudėtingų realiojo laiko sistemų (valdikliai, tinklo protokolai) tikrinimui naudojami įvairūs verifikavimo būdai. Tradiciniai verifikavimo būdai neleidžia atlikti pilno realiojo laiko sistemų tikrinimo. Pagrindinis jų trūkumas tas, kad tradiciniai verifikavimo būdai neįvertina sistemos funkcionavimo laike. Sistemos veiksena analizuojama tiriant tos sistemos funkcionavimo trajektorijas [1,2]: $S_0, e_1, S_1, e_2, S_2 \ldots$, čia S_i – sistemos būsenos žymė, o e_i – įvykis, keičiantis sistemos būseną. Norint pabrėžti ir laiko momentą, kada įvyksta įvykis, keičiantis sistemos būseną, naudojamos sekos $S_0, e_1(t_1), S_1, e_2(t_2), S_2 \ldots$, čia t_i – laiko momentas, kada įvyko atitinkamas įvykis e_i. Šios sekos yra naudojamos nagrinėjant sistemas, kurių operacijų trukmės yra determinuotos, ir šių sistemų funkcionavimas aprašomas viena trajektorija.

Šiame darbe nagrinėjamos sistemos, kurių operacijų trukmė priklauso tam tikram intervalui ir jų veiksena aprašoma tokio pavidalo trajektorijomis: $S_0, e_1(I_1), S_1, e_2(I_2), S_2 \ldots$, čia I_i – laiko intervalas. Be to, šios trajektorijos pasižymi tokia savybe: nesvarbu, kuriuo laiko momentu iš intervalo I_i įvyks įvykis e_i, tolimesnis sistemos funkcionavimas nepriklausys nuo pasirinkto laiko momento. Šių sistemų aprašyti viena trajektorija nebeįmanoma, nes yra be galo daug operacijos pabaigos laiko momentų (kontinuumo galios aibė). Tačiau tokias sistemas galima aprašyti suskaičiuojama sekų aibe [3]. Taigi atsiranda galimybė sukurti programines priemones tokių sistemų veiksenų analizei.

Darbe nagrinėjamas realiojo laiko sistemų, specifikuotų agregatiniu metodu, verifikavimo uždavinys. Sprendžiant šį uždavinį, naudojama pasiekiamų būsenų grafo sudarymo metodika, leidžianti įvertinti laiko intervalus, kuriais įvyksta sistemoje apibrėžti įvykiai.

Darbo tikslas yra patikrinti pasiekiamų būsenų grafo sudarymo metodikos, skirtos kompiuterinei realizacijai, matematines prielaidas ir atlikti šio metodo panaudojimo efektyvumo tyrimą. Be to, įvertinti Simplekso optimizavimo metodo naudojamo įvykių įvykimo sritims nustatyti tinkamumą.

Tam, kad šis tikslas būtų pasiektas, reikėjo atlikti tokias užduotis:

1. Išanalizuoti agregatinio specifikavimo metodo savybes.
2. Išnagrinėti pasiekiamų būsenų grafo sudarymo metodiką, leidžiančią įvertinti laiko intervalus, kuriais įvyksta sistemoje apibrėžti įvykiai.
3. Sukurti kompiuterinio realizavimo prototipus pasiekiamų būsenų grafo sudarymui.

4. Atlikti eksperimentus su realiojo laiko sistemų pavyzdžiais, siekiant įvertinti metodo panaudojimo efektyvumą.
5. Suformuluoti ir įrodyti teiginius apie pasiekiamų būsenų grafo sudarymo sudėtingumą.

Mokslinis naujumas. Šiame darbe tirtas realiojo laiko sistemų pasiekiamų būsenų grafo sudarymo metodas, kai sistemos aprašomos agregatiniu metodu. Tyrimas parodė, kad naudoti Simplekso optimizavimo metodą įvykių įvykimo sritims nustatyti nėra gerai, nes didinant modeliavimo laiką Simplekso optimizavimo metodo procedūros laikas neišlieka konstanta, o auga eksponentiškai didėjant laiko momentų apribojimų aibei (šiuo atveju negalima taikyti Simplekso metodo apribojimų aibės mažinimo algoritmo). Be to, išnagrinėti visas sistemos būsenas per priimtiną darbo laiką pavyksta tik kai kuriems modelio parametrų reikšmių rinkiniams.

Testinių pavyzdžių atveju parodyta, kad uždavinio sprendimo laikas priklauso nuo agregatinėje specifikacijoje esančių operacijų trukmių.

Šio metodo realizavimui siūloma naudoti lygiagrečius skaičiavimus, nes šis metodas leidžia tai daryti.

Darbo struktūra. Pirmajame darbo skyriuje pateikta sudėtingų sistemų formalizavimo metodų, imitacinių modelių apžvalga. Antrajame skyriuje aptariamas pasiekiamų būsenų grafo sudarymo metodika. Aptariamas dviejų laiko momentų palyginimas bei aprašomas optimizavimo uždavinio sprendimo Simplekso metodu panaudojimas lyginant laiko momentus. Aptartas būsenos skaičiavimo atskiras atvejis bei metodika ekvivalenčių būsenų paieškai. Trečiajame skyriuje suformuluojami ir įrodomi 4 teiginiai apie pasiekiamų būsenų grafo sudėtingumą. Ketvirtajame skyriuje aptariama pasiekiamų būsenų grafo programinė realizacija, pateikti realizacijos pavyzdžiai. Penktajame skyriuje analizuojami gauti rezultatai sprendžiant tris uždavinius, tiriant resursų panaudojimą bei kiekybinius įvertinimus.

1 SUDĖTINGOS SISTEMOS IR JŲ MODELIAVIMAS

Sudėtingos sistemos yra sistemų teorijos mokslo (systemics) objektas, kuri tiria bendrąsias sudėtingų gyvosios gamtos, socialinių ir kt. sistemų savybes. Dar iki šiol neegzistuoja vieningos nuomonės dėl sudėtingos sistemos apibrėžimo. Publikacijose pateikiama visa eilė apibrėžimų.

Sudėtinga sistema – tai sistema, kuri turi būti analizuojama pagal daugelį veikiančių komponentų, turinčių santykinai daug ryšių tarp jų, taip kad kiekvieno komponento veikimas priklauso nuo kitų komponentų veikimo [4].

Sudėtinga sistema – tai sistema charakterizuojama dideliu kiekiu tarpusavyje veikiančių komponentų (agentų (*agent*), procesų ir kt.), kurių bendras veikimas yra netiesinis (nėra išvedamas sumuojant individualius komponentus). Dažnai tai sistema, turinti hierarchinę struktūrą.

Sistema - tai esybių (pvz., žmonių, mašinų,…) rinkinys, kurios tarpusavyje sąveikauja siekdamos tam tikrą loginę prasmę turinčio tikslo [5].

Apibrėžiant sudėtingas sistemas, išskiriama keletas bendrų esminių savybių [6]:

- Vientisumas. Visi sistemos elementai yra sujungti į bendrą visumą su integruotomis charakteristikomis.
- Dalumas. Sistemą kaip visumą galima sąlyginai suskirstyti arba padalinti į atskiras posistemes.
- Unikalumas. Kiekviena sistema, net labai panaši į kitą, yra unikali ir nepakartojama.
- Įvairumas. Kiekvienas elementas ir jo elgsena sistemoje kažkuo skiriasi nuo kitų.
- Neapibrėžtumas. Vienu ir tuo pačiu metu neįmanoma fiksuoti visų sistemos charakteristikų. Žinodami sistemos elementų veikseną negalime daryti prielaidų apie visos sistemos veikseną. Sistemos elgesys nėra nulemtas vieno (pagrindinio) valdiklio.
- Sudėtingumas. Sistema susideda iš didelio kiekio tarpusavyje sąveikaujančių elementų. Sistemos elementų skaičių galima nustatyti tik specialiais metodais, sistemos sandara nėra lengvai ir paprastai nustatoma.

Šiuo metu sudėtingų sistemų projektavimui keliami dideli saugumo, patikimumo ir našumo reikalavimai, todėl vis plačiau taikomi formalieji metodai, leidžiantys sudaryti kuriamų sistemų formaliąsias specifikacijas. Įvairūs metodai taikomi sistemų formaliajam specifikavimui: automatiniai modeliai, sąveikaujančių procesų algebra, Petri tinklai, DEVS (angl. discreet event specification), atkarpomis tiesiniai agregatai ir kt.

Sudėtingų sistemų formaliosios specifikacijos nagrinėjamos dviem požiūriais: elgsenos ir funkcionavimo. Elgsenos analizės metu tikrinamos visos galimos sistemos trajektorijos, o tai leidžia patikrinti, ar specifikacija sudaryta teisingai. Teisingumas tikrinamas įvairiais validavimo ir verifikavimo metodais. Funkcionavimo analizė atliekama sukuriant sistemos imitacinį modelį [7].

1.1 Realiojo laiko sistemos ir jų projektavimas

Panagrinėkime sistemas, kuriose duomenis reikia apdoroti tam tikru laiko momentu ar laiko intervalais. Pvz., lėktuvas naudoja akselerometro impulsų seką, kad būtų nustatyta jo pozicija. Toks įvykis, kaip temperatūros viršijimas atominėje elektrinėje, reikalauja greito atsako į įvykį. Tarkime, yra rezervuotas bilietas skrydžiui. Lėktuvas išskrenda po 5 min. Jūs prieinate prie langelio ir oro linijų tarnautojas išduoda bilietą per minutę. Ar tai realiojo laiko sistema? Taip, visos paminėtos sistemos yra realiojo laiko sistemos, nes informacija turi būti apdorota per tam tikrą laiko intervalą arba sistema neveiks. Realiojo laiko sistema apibrėžiama, kaip sistema turi tenkinti aiškius atsako laiko apribojimus, o jei netenkins, tai gali iššaukti rimtas pasekmes, įskaitant sistemos gedimą [8].

Kas yra „neveikianti“ sistema? Lėktuvo ar atominės elektrinės atveju, tai yra pražūtingos pasekmės. Jei gedimas įvyksta tokioje sistemoje, kaip bankomatas, čia sugedusi sistema neturi rizikos faktoriaus. Sistemos trikis apibrėžiamas, kaip sistemos negalėjimas veikti pagal sistemos specifikaciją, kai sistema negali tenkinti vieno ar daugiau apribojimų nurodytų sistemos specifikacijoje.

Griežti laikiniai apribojimai ir savalaikiai atsakymai yra pagrindiniai realiojo laiko sistemas apibrėžiantys faktoriai. Taigi, realiojo laiko sistema turi griežtas, fiksuotas laikines sąlygas. Veiksmai turi būti atlikti atsižvelgiant į apibrėžtas laikines sąlygas arba sistema neveiks. Realiojo laiko sistema teisingai funkcionuos, tik tuo atveju, jei ji grąžins teisingą rezultatą arba atliks teisingus veiksmus per apibrėžtas laikines sąlygas. Galima ir kitu būdu apibrėžti šią sistemą. Realiojo laiko sistema yra tokia, kurios veikimo loginis korektiškumas pagrįstas tiek jos išėjimų korektiškumu, tiek ir laikinių apribojimų tenkinimu [8]. Galima teigti, kad veiksmų atlikimo laikas yra vienas svarbiausių reikalavimų iš visų realiojo laiko sistemai keliamų reikalavimų.

Panagrinėsime dvi realiojo laiko sistemų kategorijas: realiojo laiko sistemas su griežtais reikalavimais ir realiojo laiko sistemas su tikimybiniais reikalavimais.

Realiojo laiko sistemą su griežtais reikalavimais

Realiojo laiko sistemą su griežtais reikalavimais galima apibrėžti kaip sistemą, kuri privalo atlikti tam tikrus veiksmus per nustatytą laiką T, t. y., jei $\tau = \tau(e)$ – sistemos reakcijos laikas į įvykį e, tai turi būti tenkinama sąlyga: $\tau < T$ [9]. Šiose sistemose turi būti užtikrintas teisingas veiksmų atlikimas per tiksliai nurodytą laiko tarpą. Veiksmų atlikimo tvarka taip pat yra svarbi. Net ir tuo atveju, jei veiksmai atliekami teisingai, bet jie užtrunka ilgiau nei per iš anksto nustatytą laiko tarpą, veiksmų atlikimo procesas nutraukiamas. Tam, kad užtikrintumėme šių reikalavimų išpildymą, reikia apriboti visus galimus užlaikymus sistemoje.

Realiojo laiko sistemos su tikimybiniais reikalavimais

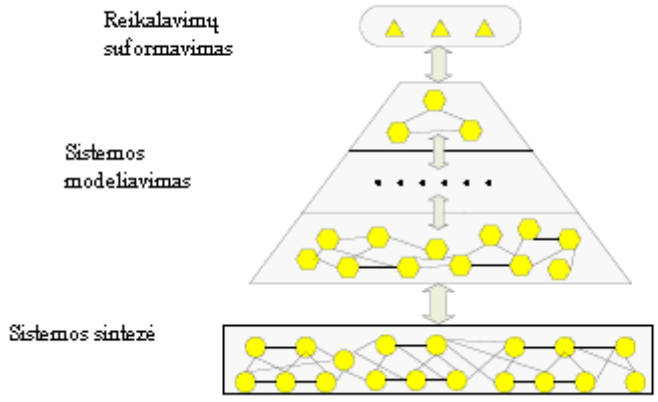
Realiojo laiko sistemos su tikimybiniais reikalavimais yra mažiau apribojamos laikinėmis sąlygomis negu realiojo laiko sistemos su griežtais apribojimais. Šiose sistemose tam tikroms užduotims yra suteikiamas prioritetas prieš kitas užduotis, bet veiksmų užlaikymai jose taip pat turi būti apriboti. Realiojo laiko sistemų su tikimybiniais reikalavimais atveju, jeigu procesas buvo atliktas teisingai, tačiau jo atlikimas užtruko ilgiau negu iš anksto nustatytą laiko tarpą, rezultatas vis tiek gali būti priimtinas.

Realiojo laiko sistemų su tikimybiniais reikalavimais įgyvendinimas reikalauja tikslaus užduočių realizavimo plano. Pirma, sistema turi turėti prioritetų sistemą, pagal kurią realiojo laiko procesams suteiktų didžiausią prioritetą. Ir šis prioritetas neturi mažėti laikui einant. Antra, veiksmų atlikimo trukmės turi būti kuo trumpesnės. Kuo mažesnis užlaikymas, tuo greičiau realiojo laiko procesai gali būti įvykdyti [9].

Realiojo laiko sistemos su tikimybiniais reikalavimais turi „švelnesnius“ reikalavimus, t. y. jose reikalaujama, kad sistema reaguotų per iš anksto numatytą laiką tik su tam tikra tikimybe *p,* t. y. išpildoma sąlyga $P(\tau<T) = p$, čia $\tau = \tau(e)$ – sistemos reakcijos laikas į įvykį *e, T*– reakcijos laikas į įvykį e.

Realiojo laiko sistemų projektavimas

Dėl realiojo laiko sistemų sudėtingumo ir griežtų laiko reikalavimų tokių sistemų projektavimas yra daugiažingsnis procesas, kai sistema yra specifikuojama ir analizuojama skirtingais abstrakcijos lygiais nuo reikalavimų suformavimo iki realizacijos (1 pav).



1 pav. Projektavimo procesas

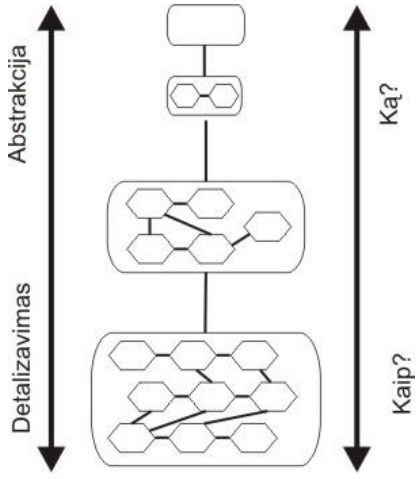
Projektavimo procesas paprastai apima reikalavimų suformavimą, sistemos modeliavimą ir sistemos sintezę [10]. Formuojant reikalavimus, sistema specifikuojama abstrakčiame lygyje, apibrėžiami sistemos poreikiai ir apribojimai. Modeliuojant sistemą, naudojama daugiau skirtingų abstrakcijos lygių, daromi projekto spendimai per daugelį etapų ir galiausiai pasiūlomas tinkamas

galutinis projekto spendimas, kuris tarnauja kaip planas realizacijos įgyvendinimui. Sistemos sintezės metu, modelis paverčiamas realizacija, kuri kaip tikimasi turės reikiamas savybes.

Sistemos reikalavimai apibrėžia sistemos poreikius ir apribojimus, kurie turi atsispindėti sistemos realizacijoje kaip jos savybės, kurios adekvačios realios sistemos savybėms [11]. Paprastai reikalavimai rašomi natūralia kalba, tačiau dėl natūralios kalbos dviprasmiškumo kompleksines sąvokas sudėtinga apibrėžti. Dėl to kyla klaidų ir papildomų iteracijų projektavimo procese. Formali semantika naudojama kaip būdas išspręsti dviprasmybių problemą. Formalios kalbos panaudojimas specifikuojant reikalavimus, padidina specifikacijos reikalavimų suprantamumą, palengvina reikalavimų logiškumo ir išbaigtumo tikrinimą, pagerina reikalavimų trasavimą kiekviename projektavimo etapo žingsnyje.

Sistemos modeliavimas yra kūrybinis procesas. Modeliuojant sistemą, turi būti aiškiai apibrėžti reikalavimai, pilnai ištirta projektuojama erdvė ir galiausiai sugalvotas projekto modelis. Projekto modelis yra pagrindas sistemos įgyvendinimui, kurio pasisekimas priklauso nuo pačio modelio. Dėl realiojo laiko sistemų potencialaus sudėtingumo, tokių sistemų modeliavimas vykdomas per daug žingsnių, kai palaipsniui apibrėžiami norimi pageidaujami trokštami sistemos aspektai ir analizuojamas sistemos veikimas kiekviename etape, atliekamos modelio transformacijos, išlaikančios esmines sistemos savybes per visus projektavimo etapus.

Abstrakcija ir detalizavimas yra dvi transformacijos, vykdomos sistemos modeliavimo metu (2 pav.).



2 pav. Modeliavimo veiklos

Abstrakcija – tai veikla, kai bandoma panaikinti (paslėpti) neesminę informaciją, kas sustiprina modelio suprantamumą. Pagrindinis abstrakcijos tikslas yra sustiprinti modelio suprantamumą, leidžiantį daryti reikiamus sprendimus. Detalizavimas – veikla, kai modelis papildomas detalėmis ir tuo pačiu sumažinamas atotrūkis tarp modelio ir realizacijos. Kitais žodžiais tariant, abstrakcijos lygyje paaiškinama, ką sistema (jos komponentai) turi daryti, tuo tarpu detalizavimo lygyje paaiškinama, kaip sistemos (komponentų) funkcionalumas gali būti

pasiektas. Abiejų veiklų tikslas sumažinti atotrūkį tarp sistemos trokštamų savybių (kokia sistema turėtų būti) ir realizacijos (kaip sistema funkcionuos).

1.2 Sistemų formalizavimo metodai

Formalūs metodai naudojami sukurti neformalaus pasaulio formalius apibrėžimus ir aprašymus. Formalūs teiginiai konstruojami naudojant logiką, aibių teoriją ir diskrečiąją matematiką, kad glaustai ir nedviprasmiškai apibrėžtų realiojo pasaulio objektus (*define part of the real world*).

Programų inžinerijoje ir kompiuterių moksle formalūs metodai apibrėžiami kaip matematika pagrįsta metodika, skirta programinės ar techninės įrangos sistemų specifikavimui, plėtojimui ir verifikavimui. Formalizavimas ypač reikšmingas sudėtingų sistemų projektavime (kur svarbus saugumas ir patikimumas), kad būtų užtikrintas teisingas sistemos tobulinimo procesas ir išvengta klaidų atsiradimo. Formalių metodų panaudojimo motyvacija grindžiama tuo, kad atitinkamos matematinės analizės vykdymas sistemos projektavimui gali įnešti patikimumo [12]. Šie metodai efektyviausi ankstyvosiose reikalavimų ir specifikacijų dokumentų ruošimo stadijose, tačiau gali būti naudojami ir jau įgyvendintos sistemos formalizavimui.

Formalūs metodai gali būti naudojami aprašant sistemą bet kuriame lygyje [13]:

- Sistema specifikuojama formaliai, bet tolesnėse stadijose tobulinama neformaliai. Dažniausiai tai mažiausiai resursų reikalaujantis ir duodantis pakankamai gerus rezultatus atvejis.
- Formalus tobulinimas ir verifikavimas naudojamas kuriant sistemą. Sukurta formali specifikacija naudojama kaip gairės, kol konkreti sistema bus sukurta (realizuota programiškai ar aparatūriškai). Formali specifikacija gali būti panaudota kaip pagrindas įrodyti specifikacijos savybes (t.y. darant išvadas apie kuriamos sistemos korektiškumą). Tai turi privalumų, nes nekorektiškų sistemų projektai gali būti peržiūrėti dar prieš atliekant pagrindinį investavimą į projekto įgyvendinimą. Šis naudojimo atvejis tinkamiausias toms sistemoms, kuriose ypatingas dėmesys turi būti skiriamas saugumui ar patikimumui.
- Teoremų įrodymo priemonės yra panaudojamos tam, kad būtų garantuotos visiškai formalios automatiškai patikrinamos tiesos. Tai labai brangus panaudojimo atvejis, tačiau praktikoje dažnai atsiperkantis, ypač jei klaidų kainos sistemoje labai didelės (pvz., kritinių mikroprocesoriaus dalių projektavimas).

Paprastai modeliuojamų sistemų sudėtingumas visapusiško formalizavimo procesą padaro labai sudėtingu ir didelių sąnaudų sprendimui reikalaujančiu uždaviniu [14]. Kaip alternatyva yra pasiūlyta įvairių nesudėtingų formalizavimo metodų, kuriuose akcentuojamas dalinis formalizmas ir dėmesys nukreipiamas į jų pritaikomumą. Tokių supaprastintų metodų pavyzdžiais yra Alloy

objektinių modelių notacija, B-Metodas, Procesų algebra, Agentų modeliai, Esterela, Petri tinklai, VDM įrankių rinkinys, Z notacija ir kt..

Alloy specifikacijos kalba paremta predikatų logika. Ji skirta nesudėtingos struktūros modeliams aprašyti. Alloy labiausiai pritaikyta programinės įrangos mikro modeliams aprašyti, tam kad būtų galima patikrinti jų teisingumą [15].

B-Metodas paremtas abstraktaus mechanizmo notacija, kuri dažnai naudojama kuriant programinę įrangą. Kaip ir Alloy specifikacija, B-Metodas yra susijęs su Z notacija. B-Metodas naudojamas daugelyje kritinio saugumo reikalaujančiose sistemose. Metodui taikyti sukurta galingų įrankių, padedančių specifikuoti, projektuoti, analizuoti ar generuoti programinį kodą.

Procesų algebra yra lygiagrečių sistemų formalaus modeliavimo metodų šeima. Kanalų naudojimas komunikacijoms aprašyti yra viena iš savybių, išskiriančių procesų algebrą iš kitų lygiagretumą aprašančių modelių, tokių kaip Petri tinklai ar agentų modeliai.

Agentų modeliai yra matematinis lygiagrečių skaičiavimų modelis [16]. Šiame modelyje agentu vadinamas lygiagrečių skaičiavimų primityvas. Kaip atsaką į gautą žinutę, agentas gali atlikti vietinius sprendimus, sukurti daugiau agentų, išsiųsti daugiau žinučių ir nuspręsti, kaip atsakyti į kitą gautą žinutę. Agentų modelis gali būti naudojamas kaip karkasas lygiagretumo aiškinimui teoriniame lygmenyje, bei kaip teorinis pagrindimas praktinių lygiagrečių sistemų realizacijoms.

Esterel yra vienalaikė programavimo kalba, skirta sudėtingoms sistemoms kurti. Ji tinka valdymo modeliams projektuoti. Esterel kompiliatoriai gali transliuoti šia kalba parašytas programas ir generuoti C programinį kodą arba techninės įrangos aprašus (VHDL arba Verilog). Sukurta keletas kompiliatorių (SCADE, Esterel Studio). Esterel kalbos privalumai – tikslus programinis valdymas laike, patikimas lygiagretumo specifikavimas valdymo sistemoms, visiškai deterministiniai modeliai, baigtinių būsenų kalba gali būti naudojama tiek programinei, tiek techninei įrangai aprašyti. Esterel kalbos trūkumai – aprašymas baigtinėmis būsenomis riboja lankstumą, sudėtingas duomenų apdorojimas, tinkamumas tik gana paprastiems sprendimus atliekantiems kontroleriams, sudėtingas kompiliavimo procesas.

Lustre – Esterel gimininga formalizavimo kalba, sukurta tų pačių kūrėjų, skirta sinchroniniais duomenų srautais valdomoms reaguojančioms sistemoms aprašyti. Ji naudojama aprašant kritines valdymo sistemas– lėktuvuose, sraigtasparniuose ir atominėse elektrinėse.

Petri tinklai puikiai tinka lygiagrečiai paskirstytų sistemų modeliavimui. Tai modeliavimo kalba, kuri orientuoto grafo su paaiškinimais pavidalu vaizduoja paskirstytos sistemos struktūrą. Petri tinklas turi viršūnes, perėjimo mazgus ir orientuotus lankus, jungiančius viršūnes su perėjimo mazgais.

VDM (Vienna Development Method) – sistemų kūrimo metodas, paremtas formalia specifikacija, parašyta VDM-SL specifikavimo kalba [17]. Metodo naudojimui yra sukurti ir palaikomi įrankiai, taip pat sukurtas praplėtimas objektiškai orientuotam programavimui – VDM++. Metodas naudingas modeliais pagrįstoms sistemoms aprašinėti, tačiau netinkamas, jei sistema laikinė.

Z notacija yra formali specifikavimo kalba, naudojama kompiuterinių sistemų aprašymui ir modeliavimui. Pagrindinis Z tikslas – išsami ir aiški kompiuterinės programos specifikacija ir numatytos programos veiksenos formalus įrodymas. Z išrasta 1970 m. Oksfordo universitete, ji remiasi aibių teorija ir predikatų logika.

1.3 Sudėtingų sistemų formalizavimas agregatiniu metodu

Realiojo laiko sistemų specifikavimui galima naudoti atkarpomis tiesinį agregatinį metodą [18]. Metodo esmė yra ta, kad sistemą galima suskaidyti į posistemes, kurių kiekvieną galima aprašyti penketu: būsenų aibe *Z*, įėjimų signalų aibe *X*, išėjimų signalų aibe *Y*, perėjimo operatoriumi *H* ir išėjimo operatoriumi *G*. Atkarpomis tiesinių agregatų funkcionavimas yra aprašomas laiko momentų *T* aibe. Būsena, įėjimo signalas ir išėjimo signalas yra laiko funkcijos [19].

Agregato būsena aprašoma vektoriumi, kurio koordinatės gali įgyti reikšmes iš jų apibrėžimo srities. Viena būsena skiriasi nuo kitos bent vienos koordinatės reikšme. Yra skiriamos dviejų tipų būsenos vektoriaus koordinatės: diskrečių koordinačių vektorius $N(t) = (n_1(t), n_2(t), ..., n_r(t))$, ir tolydinių koordinačių vektorius $W(t) = (w_1(t), w_2(t), ..., w_g(t))$. Tolydinės koordinatės yra susijusios vidiniais įvykiais, žyminčiais operacijos pabaigą. Agregato evoliucija aprašoma perėjimo ir išėjimo operatoriais *H* ir *G*, įvedant įvykio ir operacijos sąvokas. Įvykis ir su juo susijęs agregato būsenos pasikeitimas – tai momentinis veiksmas, kai tuo tarpu operacija turi trukmę. Įvykių, galinčių įvykti agregate, aibė E skaidoma į nesusikertančius poaibius E' ir E''. Poaibį E' sudaro įvykiai, kurie įvyksta atėjus signalams iš įėjimo signalų aibės *X*. Šie įvykiai vadinami išoriniais. Įvykiai iš poaibio E'' vadinami vidiniais. Jie fiksuoja operacijų, vykstančių agregate, pabaigą. Operacijos trukmė – tai atsitiktinis dydis su nurodyta pasiskirstymo funkcija. Jei t_m – j-osios operacijos pradžios momentas, tai einamuoju momentu jį atitinka tolydinė koordinatė nustatoma taip: $w_j(t_m) = t_m + \xi_j^{\alpha}$, čia ξ_j^{α} – atsitiktinio dydžio ξ_j su žinoma pasiskirstymo funkcija $F_j(t) = P(\xi_j < t)$ reikšmė. Taigi $w_j(t_m)$ reikšmė lygi j-osios operacijos trukmei. Jei laiko momentu t_m ši operacija nevykdoma, tai $w_j(t_m) = \varnothing$ [19].

Šį metodą išsamiai aprašė Rusijos mokslininkai N.Buslenko ir I.Kovalenko [18]. Prof. H.Pranevičius, pasiūlė metodo modifikaciją, t.y. papildė agregatinį aprašymą valdančiomis sekomis, kas sudarė prielaidas patogiam šių sistemų modelių realizavimui kompiuteriu.

Ši specifikacija naudojama dviem tikslams: imitaciniams modeliams sudaryti, sistemai validuoti bei verifikuoti. Validavimas ir verifikavimas remiasi pasiekiamų būsenų grafo sudarymu [20,21].

1.3.1 Agregatinių specifikacijų verifikavimas

Agregatinėms specifikacijoms verifikuoti yra sukurti pasiekiamų būsenų ir invariantų metodai [7]. Pasiekiamų būsenų metodo esmę sudaro tai, kad turint sistemos agregatinę specifikaciją generuojama visų galimų sistemos trajektorijų aibė. Tada šios trajektorijos nagrinėjamos sistemos tiriamų savybių atžvilgiu. Taikant invariantų metodą, reikia sudaryti sistemos invariantą ir patikrinti, ar jis yra teisingas visose galimose sistemos būsenose.

Agregatinių specifikacijų verifikavimas pasiekiamų būsenų metodu

Sistemai, specifikuotai agregatinių metodu, pasiekiamų būsenų grafas sudaromas taip [7]:

1. Atliekama visų agregatų kompozicija. Ją atlikus gaunamas agregatas, kuris neturi įėjimų ir išėjimų signalų. Po kompozicijos $E'=\varnothing$ ir $E''\neq\varnothing$. Agregatinėje specifikacijoje tolydusis laikas panaikinamas šitaip: visos tolydžiosios dedamosios $w\{e, t_m)$ keičiamos į $w\{e) = 1$, jei operacija aktyvi, ir $w(e) = 0$, jei operacija neaktyvi. Po tokių pokyčių agregato tolydžiosios dedamosios rodo, ar operacijos, kurios iššaukia atitinkamus įvykius, yra aktyvios ar ne. Agregato būsena įgyja pavidalą $z = \{\nu; w(e_1), w(e_2), ..., w\{e_k)\}$.
2. Nusakoma pradinių būsenų aibė.
3. Apibrėžiama galutinių būsenų aibė.
4. Pasiekiamų būsenų grafas susideda iš viršūnių aibės V ir lankų aibės L. Viršūnėje apibrėžiama sistemos būsena, o lankas yra trejetas: būsena z, iš kurios lankas išeina, įvykis, kuris sukelia perėjimą e, būsena z', t. y. (z, e, z'), čia $z' = H(z, e)$.
5. Sudaroma nenagrinėtų būsenų aibė V_N, kur pradžioje randasi visos pradinės būsenos. Paskui imama būsena $z \in V_N$ ir sudaroma tiek lankų, kiek yra aktyviųjų operacijų, t. y. gaunama aibė $L=\{(z, e, H(z, e)) \mid e \in E'', w(e) = 1\}$. Nenagrinėtų viršūnių aibė V_N

papildoma būsenomis, į kurias pereinama iš *z* būsenos, bet kurios dar nebuvo nagrinėtos.

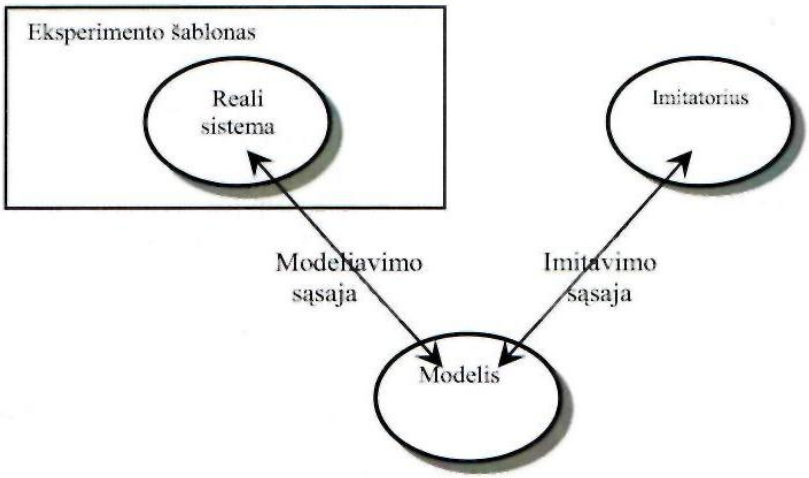
Grafas baigiamas sudaryti, kai nenagrinėtų būsenų aibė tampa tuščia, t. y. nelieka nenagrinėtų viršūnių. Pasiekiamų būsenų grafas leidžia analizuoti šias sistemos savybes:

- *Pasiekiamumą,* t. y. kokiu būdu iš pradinės būsenos galima pasiekti galinę būseną.
- *Būsenos koordinačių apribojimus,* t. y. galima nustatyti, ar būsenos koordinatės neišeina už iš anksto nustatytų ribų.
- *Tolydžiųjų koordinačių reikalingumą, t.* y. reikia nustatyti, ar specifikacijoje nėra aprašytų tolydžiųjų dedamųjų, kurios niekada negeneruoja vidinių įvykių.
- *Aklaviečių paiešką,* t. y. galima nustatyti, ar yra būsenų, iš kurių niekur neišeinama.
- *Ciklų paiešką,* t. y. galima nustatyti, ar nepatenkama į uždarą ciklą, kuriame kartojasi tam tikra įvykių seka.

1.3.2 DEVS modelis

Naudojant DEVS formalizavimą, pirmiausia reikia apibrėžti atominius (pagrindinius) modelius, iš kurių vėliau kuriami jungtiniai modeliai, taip pat nusakyti tų modelių sąsajas hierarchinėje struktūroje [7]. Atominis DEVS neturi jokių komponentų, tik matematiškai aprašytą elgsenos specifikaciją. Diskrečiųjų įvykių sistema aprašoma kaip deterministinių perėjimų tarp sistemos būsenų seka. Taip pat apibrėžiama, kaip sistema reaguoja į išorinį įėjimą (įvykius) ir kaip generuoja išėjimą (įvykius). Jungtinis DEVS sudaromas iš vieno ar kelių atominių DEVS. Be to, jungtinis DEVS gali būti pakeistas atominiu DEVS su ekvivalenčia elgsena. Jungtinis DEVS gali būti naudojamas daug sudėtingesniems DEVS komponentams sudaryti. Tokiuose modeliuose reikia apibrėžti ryšius tarp komponentų.

Paveikslėlyje parodyta imitacinių modelių sudarymo schema (3 pav.) [7]:



3 pav. Imitacinių modelių sudarymo schema

Pagrindiniai schemos komponentai yra šie:

- Modelis - instrukcijų, pagal kurias gaunami realią sistemą atitinkantys duomenys, rinkinys. Modelio elgsena yra rinkinys visų įmanomų duomenų, kurie gali būti gaunami tiksliai vykdant modelio instrukcijas.
- Imitatorius - modelio instrukcijų vykdymas modelio elgsenai generuoti.
- Eksperimento šablonas apibrėžia tam tikras sąlygas, kuriomis sistema stebima ir atliekami eksperimentai.
- Modeliavimo sąsajos, susiejančios realią sistemą ir modelį. Nuo jų priklauso, kaip gerai modelis atitinka modeliuojamą sistemą. Modeliavimo sąsajos teisingumas gali būti patikrintas formaliaisiais verifikavimo metodais.
- Imitavimo sąsajos, susiejančios modelį ir imitatorių. Imitatorius vykdo modelio instrukcijas.

2 PASIEKIAMŲ BŪSENŲ GRAFO SUDARYMO METODAS

Pagrindinė pasiekiamų būsenų grafo funkcija agregatinėms specifikacijoms [22] yra atlikti sistemų funkcionalumo analizę bei validavimą. Realiojo laiko sistemų, specifikuotų agregatiniu metodu, verifikavimo uždavinys yra nagrinėtas literatūroje [19], kur yra išanalizuota pasiekiamų būsenų grafo sudarymo metodika, leidžianti įvertinti laiko intervalus, kuriais įvyksta sistemoje apibrėžti įvykiai. Nagrinėjamu atveju pasiekiamų būsenų grafo viršūnė aprašo vieną būseną ir susideda iš trijų dedamųjų:

- Tolydinės dedamosios $z_v(t_m)$. Ji apibrėžta agregatinėje specifikacijoje, kur kiekvienai aktyviai komponentei nurodomas laiko intervalas, kada gali baigtis atitinkama operacija.
- Diskretinės būsenos dedamosios $\nu(t_m)$. Ji taipogi apibrėžta agregatinėje specifikacijoje.
- Laiko momentų apribojimų aibės R. Šios aibės elementai yra $t_i - t_j < \alpha$ pavidalo nelygybės. Čia α - konstanta, t_i, t_j - laiko momentai, kuriais įvyko įvykiai.

Lankas jungiantis dvi viršūnes nusako:

- kada perėjimas gali įvykti (kada gali baigtis operacija),
- koks įvykis iššaukia šį perėjimą;

Pasiekiamų būsenų grafo sudarymo algoritmai buvo aptarti T. Milinio ir U.Leonavičiūtės magistriniuose darbuose. Pasinaudojant pateiktais algoritmais ir juos patikslinant, šiame darbe pasiekiamų būsenų grafo sudarymo algoritmas realizuojamas kompiuteriu.

2.1 PBG sudarymo algoritmas

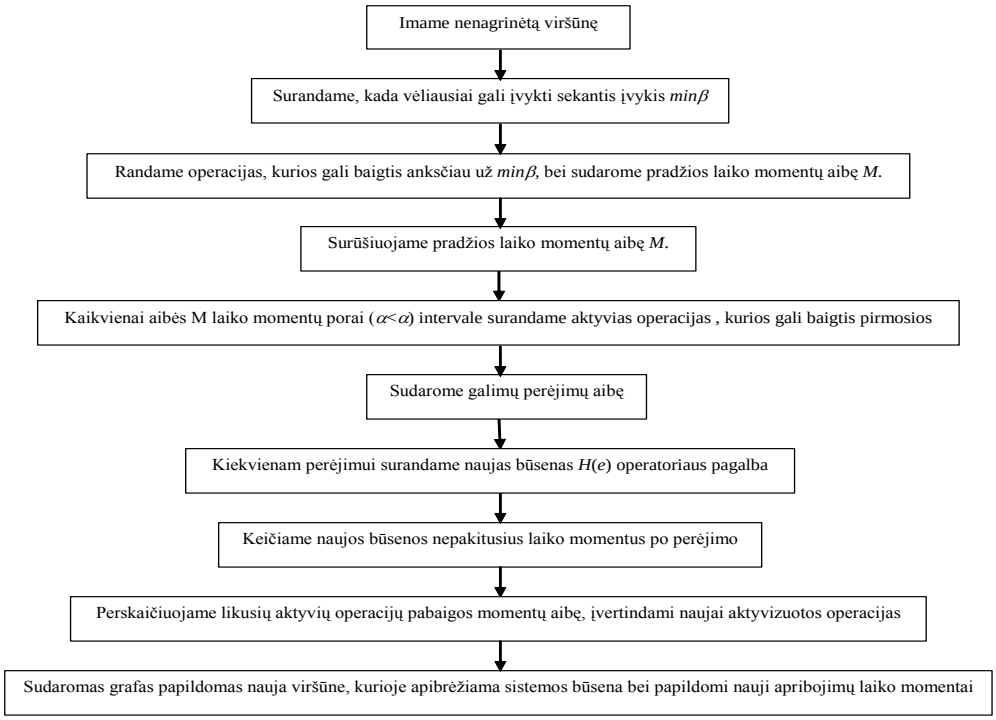
Sudarant pasiekiamų būsenų grafą (PBG) yra svarbu išanalizuoti vieną viršūnę. Visos kitos viršūnės yra analizuojamos analogiškai. PBG vienos viršūnės analizė atliekama taip:

1. Randame, kada vėliausiai gali įvykti sekantis įvykis $\beta = \min_{1 \le i \le n} \beta_i$. β randamas palyginus visų nagrinėjamos būsenos operacijų O_i galinius laiko momentus β_i. Jei reikia, lygindami laiko momentus, remiamės R aibės nelygybėmis. Plačiau kaip palyginami du laiko momentai aptarsime 2.2 skyriuje.
2. Randame aibę aktyvių operacijų O, kurios gali baigtis anksčiau už β. Tam yra lyginami visi operacijų O pradžios laiko momentai α_i su β ir laiko momentai mažesni už β įtraukiami į aibę. Jei reikia, palygindami du laiko momentus, remiamės R aibės

nelygybėmis. Taigi, jei $\alpha_i < \beta$, tai operacija O_i gali baigtis pirmoji. α_i įtraukiamas į pradžios laiko momentų aibę $M = \{\alpha_i | \alpha_i < \beta, i = 1,2,\ldots,n\}$.

3. Surūšiuojame aibės M elementus didėjimo tvarka. Tam reikalinga atlikti dviejų laiko momentų palyginimą bei esant reikalui remtis R aibės nelygybėmis. Taip gauname surūšiuotą pradžios momentų aibę $\alpha_1^* < \alpha_2^* < \ldots < \alpha_{M+1}^* = \beta$.
4. Kiekvienam M aibės intervalui $I_i = \alpha_i^* < \alpha_{i+1}^*, i = 1,2,...n-1$ randame aktyvias operacijas, kurios gali baigtis pirmosios bei surandame galimų perėjimų aibę (e_i, I_i).
5. Kiekvienai porai (e_i, I_i) sudaroma nauja viršūnė, kuri sujungiama lanku su nagrinėjama viršūne. Naujos viršūnės būsena apskaičiuojama, remiantis perėjimo operatoriumi $H(e_i)$ specifikuotu agregatinėje specifikacijoje. Naujos viršūnės apribojimų aibė formuojama prirašant prie buvusios nelygybės naujas: $\alpha_i^* < t_m$, $t_m < \alpha_{i+1}^*$, čia. m įvykio eilės numeris.
6. Visų operacijų, kurios buvo aktyvios ir liko nepakeistos po įvykio, pradžios intervalai turi būti pataisyti. Tai atliekama vietoje esamo pradžios laiko momento įrašant laiko momentą t_m.
7. Patikrinama ar aktyvios operacijos sužadinamas įvykis neprasidės anksčiau nei prieš tai buvusių aktyvių operacijų pabaigos laiko momentas β. Tam yra lyginamas naujai sugeneruoto įvykio minimalus laiko momentas $t_m + \alpha$ su nagrinėjamu intervalu $t_m \in (\alpha_i^*, \alpha_{i+1}^*)$. Detaliau tai aptarsime 2.1.3 skyriuje.

Pasiekiamų būsenų grafo sudarymo algoritmas pateiktas 4 paveikslėlyje.



4 pav. Pasiekiamų būsenų grafo sudarymo algoritmas

2.2 Laiko momentai ir jų palyginimas

Vienas svarbiausių ir dažniausiai atliekamų veiksmų, bandant įvertinti realiojo laiko sistemos veikseną – dviejų laiko momentų palyginimas.

Laiko momentas aprašomas dviem dedamosiomis - atskaitos taško laiko koordinate t_i bei jo poslinkiu laiko ašyje α. Apibrėžkime laiko momentų aibę T:

$$T = \left\{ t_i + \alpha \mid \alpha \in \mathsf{R}_i, i \in I \right\}, \text{ čia } i \in \mathsf{N},\ \mathsf{R}_i \in \mathsf{R}_0^+.$$

Be to, priimame sąlygą, kad jei $i < j$, tada ir $t_i < t_j$.

Laiko momentus, kurių ta pati atskaitos taško koordinatė t_i, t.y. indeksai i yra vienodi, galime vienareikšmiškai įvertinti, kuris yra mažesnis. Mažesnis bus tas laiko momentas, kurio poslinkis yra mažesnis. Pvz., jei $T_1 = \left\{ \tau \mid \tau = t_m + \alpha \right\}$ ir $T_2 = \left\{ \tau \mid \tau = t_m + \beta \right\}$, o $\alpha < \beta$, tada $T_1 < T_2$.

Tačiau sprendžiant realiojo laiko sistemas PBG metodu dažnai tenka lyginti laiko momentus, kurių atskaitos taško koordinatė nėra vienoda, t.y. tenka lyginti laiko momentus $t_i + \alpha$ ir $t_j + \beta$.

Jeigu norime palyginti du laiko momentus T_1 ar T_2 kuris yra mažesnis, ir žinome, kad $T_1 = t_i + \alpha$, $T_2 = t_j + \beta$, $i > j$, $\alpha > \beta$, tai tokius laiko momentus galima palyginti vienareikšmiškai. Kaip minėjome, jei $i > j$, tada ir atskaitos taškas $t_i > t_j$. Kadangi dar ir poslinkis $\alpha > \beta$, tai abi T_1 laiko momento dedamosios yra didesnės už T_2. Taigi galima vienareikšmiškai pasakyti, kad T_2 laiko momentas yra mažesnis.

Dažnai pasitaiko ir tokia situacija, kad $T_1 = t_i + \alpha$, $T_2 = t_j + \beta$, $i > j$, o $\alpha < \beta$. Šioje situacijoje vienareikšmiškai pasakyti, kuris laiko momentas yra mažesnis, neįmanoma.

Norint įvertinti šiuos laiko momentus, kuris iš jų yra mažesnis, reikalingos papildomos sąlygos apie laiko koordinates. Papildomas sąlygas sudaro apribojimų aibės R nelygybės. Apribojimų nelygybių aibė: $t_0 + \alpha < t_i < t_0 + \beta$, $t_{i_1} + \delta_{i_1,i} < t_i < t_{i_2} + \varepsilon_{i_2,i}$, čia $i_1, i_2 < i < \max(m,n)$.

Nustatyti, kuris laiko momentas yra mažesnis, galima įvertinant skirtumą $T_1 - T_2 = t_i - t_j - (\alpha_j - \alpha_i)$. Jei šis skirtumas yra mažesnis už 0, tada $T_1 < T_2$. Tuo atveju, kai skirtumas didesnis už 0, tada $T_1 > T_2$. Reikia įvertinti skirtumą $t_i - t_j$, nes kitą sumos dedamąją $\alpha_j - \alpha_i$ galime nustatyti iš karto, kadangi $\alpha_i, \alpha_j \in \mathsf{R}_0^+$.

Laiko momentų skirtumui $t_i - t_j$ įvertinti, sudaromas tiesinio programavimo uždavinys [23], kuris sprendžiamas Simplekso metodu. Uždavinio nelygybių sistemą sudaro būseną aprašančių

apribojimų aibė, o tikslo funkciją – nagrinėjamų laiko momentų skirtumas. Išsprendus sudarytą tiesinio programavimo uždavinį, gaunamas šių dviejų laiko momentų skirtumo minimumo ir maksimumo įvertis.

Taigi sprendžiant tiesinio programavimo uždavinius galima gauti skirtumo $t_i - t_j$ įvertį iš viršaus ir apačios:

- ieškant įverčio iš apačios, spręsime tiesinio programavimo uždavinį $c_1 = \min\left(t_i - t_j\right)$, o apribojimų aibę sudarys tokia nelygybių sistema:

$$\begin{cases} t_1 > \alpha, \\ t_1 < \beta, \\ t_i - t_{i_1} > \delta_{i_1,i}, \\ t_i - t_{i_2} < \varepsilon_{i_2,i}, & \text{čia } i = 1,\dots l,\ i_1, i_2 < i, \\ t_{j_1} - t_{j_2} < \eta_{j_2} - \gamma_{j_2}, & \text{čia } j_1, j_2 < l, \\ 0 < t_1 < t_2 < \dots < t_l; \end{cases}$$

- ieškant įverčio iš viršaus, spręsime tiesinio programavimo uždavinį $c_2 = \max\left(t_i - t_j\right)$, o apribojimų aibę sudarys tokia nelygybių sistema:

$$\begin{cases} t_1 > \alpha, \\ t_1 < \beta, \\ t_i - t_{i_1} > \delta_{i_1,i}, \\ t_i - t_{i_2} < \varepsilon_{i_2,i}, & \text{čia } i = 1,\dots l,\ i_1, i_2 < i, \\ t_{j_1} - t_{j_2} < \eta_{j_2} - \gamma_{j_2}, & \text{čia } j_1, j_2 < l, \\ 0 < t_1 < t_2 < \dots < t_l. \end{cases}$$

Išsprendus šiuos uždavinius gauname, kad $c_1 < t_i - t_j < c_2$. Dabar galima atsakyti į klausimą kas mažiau: T_1 ar T_2. Galimi trys atvejai:

1. jei $\alpha_j - \alpha_i < c_1$, tada $T_1 - T_2 = t_i - t_j - \left(\alpha_j - \alpha_i\right) > c_1 - \left(\alpha_j - \alpha_i\right) > 0$, iš čia seka, kad $T_1 = t_i + \alpha_i > t_j + \alpha_j = T_2$;
2. jei $\alpha_j - \alpha_i > c_2$, tada $T_1 - T_2 = t_i - t_j - \left(\alpha_j - \alpha_i\right) < c_2 - \left(\alpha_n - \alpha_m\right) < 0$, iš čia seka, kad $T_1 = t_i + \alpha_i < t_j + \alpha_j = T_2$;

3. kai $c_1 < \alpha_j - \alpha_i < c_2$, vienareikšmiškai pasakyti kuris laiko momentas (T_1 ar T_2) yra mažesnis, negalima. Tai parodysime taip: pirmu atveju laikysime, kad $t_i - t_j = \frac{c_1 + \alpha_j - \alpha_i}{2}$, o antru atveju – $t_i - t_j = \frac{c_2 + \alpha_j - \alpha_i}{2}$

Pirmuoju atveju $T_1 - T_2 = t_i - t_j - (\alpha_j - \alpha_i) = \frac{c_1 + \alpha_j - \alpha_i}{2} - (\alpha_j - \alpha_i) = \frac{c_1 - (\alpha_j - \alpha_i)}{2} < 0$, o antruoju atveju $T_1 - T_2 = t_i - t_j - (\alpha_j - \alpha_i) = \frac{c_2 + \alpha_j - \alpha_i}{2} - (\alpha_j - \alpha_i) = \frac{c_2 - (\alpha_j - \alpha_i)}{2} > 0$.

Žemiau pateikti dviejų laiko momentų palyginimo pseudo kodai.

Aiškių laiko momentų palyginimas

dim tm,αm ← (t_m,α_m)

```
dim tn,αn ← (tn,αn)
if (tm >= tn) & (αm >= αn)
       return (tn,αn)
if if (tm <= tn) & (αm <= αn)
       return (tm,αm)
```

Palyginimas, jei nepavyksta palyginti laiko momentų pirmuoju metodu

```
dim tm,αm ← (tm,αm)
dim tn,αn ← (tn,αn)
tf ← tm - tn              // tikslo funkcija
a,b ← a1<t1<b1, a2<t2<b2, ...  // apribojimų aibė
c1 ← simplex(a,b,tf)    // tikslo f-jos minimumas
c2 ← simplex(a,b,(-1)*tf)    // tikslo f-jos maksimumas
if (αn - αm < c1)
        return (tn,αn)
if (αn - αm > c2)
       return (tm,αm)
if (c1 < αn - αm < c2)
       return (tm,αm, c1<tn-tm<αn-αm; tn,αn, αn-αm <tn-tm<c2)
```

2.3 Simplekso metodas ir apribojimų aibės optimizavimas

Generuojant PBG ir didėjant trajektorijų medžiui, kiekviena nauja būsena nauju laiko momentu aprašoma vis didesne laiko momentų apribojimų aibe R. Skaičiuojant naujas būsenas ir vertinant dviejų laiko momentų apribojimus, tiesinio programavimo uždavinio aibę sudaro vis daugiau nelygybių. Sprendžiant šį uždavinį Simplekso metodu reikia atlikti vis daugiau iteracijų norint gauti įvertį. Tam, kad optimizuoti nelygybių sistemą, sumažinant lygčių skaičių ir

nepaveikiant uždavinio sprendimo rezultato, T.Milinio [23] magistriniame darbe buvo pasiūlytas ir įrodytas algoritmas, kurio pagalba būtų atrenkamos nelygybės įtakojančios sprendinį.

Šio metodo idėja yra tokia:

1. Sudaroma pradinė aibė *K*, kurioje saugomi analizuojami laiko momentų atskaitos taškų indeksai i, t.y. $K = \{i_1, i_2, \ldots, i_n\}$, čia $i \in \mathsf{N}$. Šioje aibėje saugomi tik nesikartojantys indeksai;
2. Surandamas maksimalus indeksas aibėje *K*, $n = \max K$;
3. Maksimalus indeksas pašalinamas iš aibės ir įtraukiamas į naują aibę *L*, kurioje saugomi reikalingi įtraukti į apribojimų aibę momentų atskaitos taškų indeksai.
4. Laiko momentų apribojimų aibėje *R* išrenkami laiko momentai, kuriais išreiškiamas laiko momentas t_n (čia *n* yra maksimalus indeksas) ir šių laiko momentų indeksais papildoma aibė *K*.
5. Jeigu *K* aibėje yra daugiau nei vienas elementas, kartojamas ciklas nuo 2 punkto.
6. Jeigu *K* aibėje liko vienas elementas, į apribojimų aibę įtraukiamas likusio elemento laiko momento t_n (čia *n* yra likęs aibės *K* elementas) laikinis apribojimas $\alpha < t_n < \beta$.
7. Į apribojimų aibę įtraukiami visos apribojimų aibės *R* nelygybės, kurios išreiškiamos laiko momentais t_n, kur n priklauso *L* aibei, $n \in \mathsf{L}$.

Pseudo kodas, minimalios apribojimų srities radimui

```
dim K[] ← set (t_i, t_j,...)   // lyginamų laiko momentų masyvas
dim L[] // tusčias masyvas laiko indeksų kurie turės būti įtraukti
dim R[] ← set (t_{i_1} + δ_{i_1,i} < t_i < t_{i_2} + ε_{i_2,i}) // apribojimų aibė
dim A[] //apribojimų aibė
while length(K) > 1
    Kmax ← max(K)
    Remove Kmax from K
    Add Kmax to L
    for i=1 to length(R)
        if R[i] ∈ t_Kmax && K not contains i
           add i to K
for i=1 to length(L)
    for j=1 to length(R)
        if R[j] ∈ t_i
           add R[j] to A
add t_K[0] to A
return A
```

2.4 Būsenos skaičiavimo atskiras atvejis

Sistemos būsenos nagrinėjimo metu gali susidaryti tokia situacija, kad aktyvios operacijos sužadinamas įvykis prasidės ir baigsis anksčiau nei prieš tai buvusių aktyvių operacijų pabaigos laiko momentas. Tokiu būdu, toliau analizuojant sistemą toks įvykis būtų tariamai išanalizuotas, o sistemos veiksena apibrėžiama netiksliai. Atliekant sistemos analizę būtina atlikti patikrinimą, ar generuojamas naujas įvykis nėra būtent toks. Esant tokiam atvejui, reikalingas patikslinimų įvedimas.

Siekiant patikrinti šį atvejį, sugeneravus naują būseną reikia patikrinti, ar visos įvykių sugeneruotos operacijos neįvyks anksčiau nei buvusių aktyvių operacijų pabaigos laiko momentas. Tam yra surandamas minimalus naujai sugeneruotų operacijų pradžios laiko momentas $t_n + \alpha$. Reikia palyginti, ar šis laiko momentas prasidės vėliau nei buvusios viršūnės analizuojamas intervalas $t_n \in (\alpha_k, \beta_k)$, t.y. ar $\alpha_k > \beta_k - \alpha$. Jeigu ši sąlyga yra tenkinama, sužadinti įvykiai prasidės vėliau nei prieš tai buvusių aktyvių operacijų pabaigos laiko momentas ir nieko keisti nereikia. Jeigu ši sąlyga yra netenkinama, vadinasi sužadinamas įvykis įvyks anksčiau. Dėl to esama apribojimų aibę bei viršūnę reikia skaidyti į dvi apribojimų aibes su tokiais apribojimais:

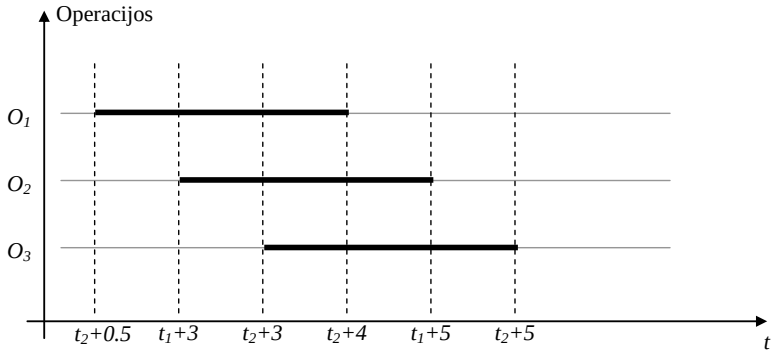
$t_n \in (\alpha_k, \beta_k - \alpha)$, $R \cup \{\alpha_k < t_n < \beta_k - \alpha\} \cup \{\beta_k > t_n + \alpha\}$

$t_n \in (\beta_k - \alpha, \beta_k)$, $R \cup \{\beta_k - \alpha < t_n < \beta_k\} \cup \{\beta_k < t_n + \alpha\}$.

Į apribojimų aibę antrojo apribojimo, t.y. $\{\beta_k > t_n + \alpha\}$ pirmuoju atveju ir $\{\beta_k < t_n + \alpha\}$ - antruoju, įtraukti nebūtina, nes jį pilnai padengia pirmasis apribojimas.

Pavyzdys:

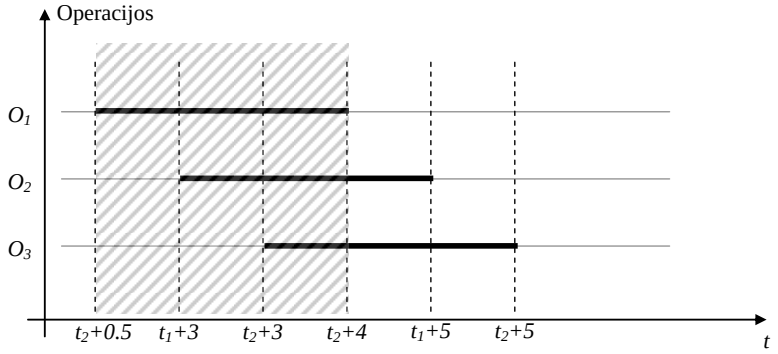
Pamėginkime detaliau išanalizuoti tokią situaciją, tokio įvykio atsiradimo aplinkybes ir veiksmus, kurių reiktų imtis. Tarkime, kad turime būseną $T = \left(0; (t_2 + 0{,}5, t_2 + 4); (t_1 + 3, t_1 + 5); (t_2 + 3, t_2 + 5); R\right)$. $R = \{t_1 + 0.5 < t_2 < t_1 + 2{,}5\} \cup \{t_0 + 0.5 < t_1 < t_0 + 3\}$.



5 pav. *T* būsenos grafinis atvaizdas

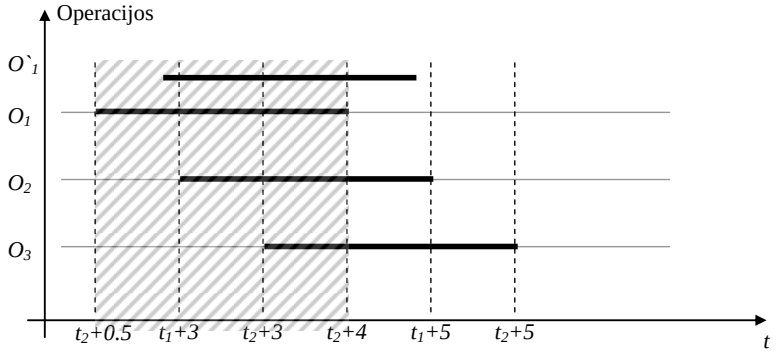
Būsenos analizė pradedama įvykių pabaigos laiko momentų minimumo paieška. Tarkime, kad laiko momentų eiliškumas toks, koks pateiktas 5 paveikslėlyje. Šiuo atveju, $\beta = \min(t_2 + 4, t_1 + 5, t_2 + 5) = t_2 + 4$, kai $t_1 + 0.5 < t_2 < t_1 + 1$. Surandame ir surūšiuojame pradžios momentus mažesnius už β $M = \{\alpha_i | \alpha_i < \beta, i = 1,2,\ldots,n\} = t_2 + 0{,}5 < t_1 + 3 < t_2 + 3 < \beta$

Gaunamas nagrinėjamas intervalas – $t_3 \in (t_2 + 0.5; t_2 + 4)$ (6 pav.).



6 pav. Nagrinėjamas laiko intervalas

Aktyvios operacijos – O_1, O_2, O_3. Nagrinėkime tik vieną operaciją iš galimų – O_1 ir tarkime, kad ši operacija susijusi su įvykiu e. Taip pat, įvykis e generuoja naują būseną laiko intervale $t_3 \in (t_2 + 0{,}5; t_1 + 3)$: $T' = (0; (t_3 + 0{,}5, t_3 + 4); (t_1 + 3, t_1 + 5); (t_2 + 3, t_2 + 5); R)$. Anksčiausiai sugeneruota nauja operacija gali pasidėti: $t_3 + 0{,}5$ laiko momentu. Taigi gaunasi, kad , jei $t_2 + 0{,}5 < t_1 + 3 - 0{,}5$, nauja operacija O_1 būtinai prasidės anksčiau nei nagrinėjamas laiko intervalas pasibaigs (7 pav.). To pasėkoje, kitame analizuojamame intervale, įvykio e sugeneruota būsena O_1 nebegalės būti aktyvi. Tai veda prie klaidingos sistemos veiksena analizės.



7 pav. Įvykio *e* generuojama operacija O^I

Taigi sugeneravus naują operaciją, visada būtina patikrinti, ar šios operacijos įvykimo intervalo pabaigos laiko momentas nėra mažesnis už analizuojamo laiko intervalo pabaigą β.

Susidarius tokiai situacijai, nagrinėjamą intervalą reikia sumažinti iki naujosios operacijos pradžios laiko momento.

2.5 Pasiekiamų būsenų grafo sudarymo algoritmas baigtiniame laiko intervale

Daugeliu atveju mus domina sistemos elgsena baigtiniame laiko intervale, t.y. norime žinoti, kaip elgsis sistema per *T* laiko vienetų nuo pradinio laiko. Generuodami pasiekiamų būsenų grafą, sistemos analizę laiko intervale $(t_0, t_0 + T)$ atliekame sekančiai:

- Naudodamiesi pasiekiamų būsenų grafo generavimo algoritmu pateiktu skyriuje 2.1 yra nagrinėjama nenagrinėta viršūnė ir skaičiuojami perėjimai į naujas viršūnes.
- Kiekvienoje naujai sugeneruotoje viršūnėje įvertinamas išnagrinėtas laiko intervalas nuliniu laiko momentu $(t_0 + \alpha, t_0 + \beta)$. Laiko intervalas yra įvertinamas pagal naujai sugeneruotos viršūnės apribojimų aibę, įstatant aukščiau gautų atitinkamų viršūnių apribojimus. $(t_i + \chi, t_j + \delta)$.
- Jeigu gauname, kad įvertintos viršūnės laiko intervalo pradžia yra didesnė už analizuojamo intervalo pabaigą $t_0 + T < t_0 + \alpha$, tai baigiame viršūnės skaičiavimus ir teigiame, kad viršūnė išanalizuota laiko intervale $(t_0, t_0 + T)$.

2.6 Pasiekiamų būsenų grafo sudarymo algoritmas ieškant ekvivalenčių viršūnių

Pasiekiamų būsenų grafo modeliavimo metu didėjant modeliavimo trukmei neaprėžtai auga viršūnių skaičius. Buvo pastebėta, kad toje pačioje šakoje pradeda kartotis tos pačios diskretinės komponentės bei tolydinės komponentės yra panašios tam tikro laiko periodo atžvilgiu. Nuspręsta pabandyti patikrinti, ar sugeneruota viršūnė nėra ekvivalenti kuriai nors prieš tai buvusiųjų. Jei viršūnė ekvivalenti, tai toliau analizuoti nuo šios viršūnės nereikia, nes kartojasi jau išanalizuotos situacijos.

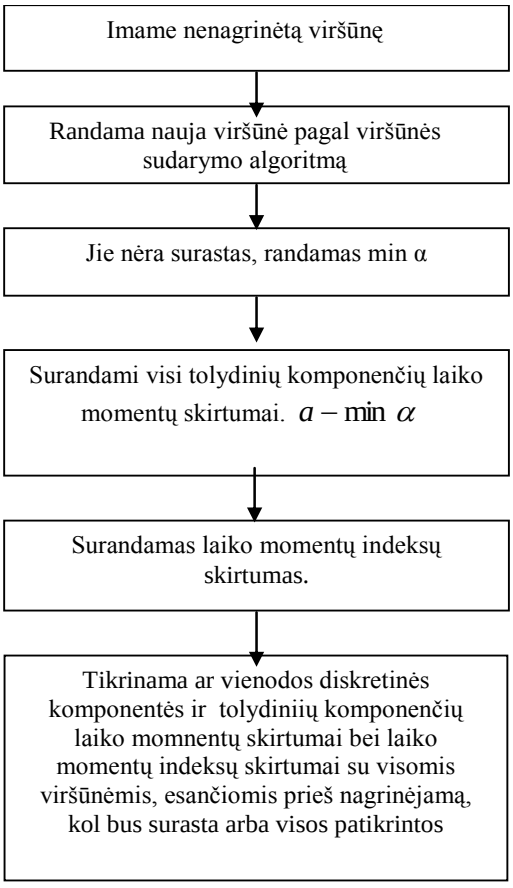
Naudodamiesi vienos viršūnės analizės metodu pateiktu 2.1 skyriuje, išanalizuojame vieną viršūnę rasdami perėjimus ir naujas viršūnes, į kurias galimi perėjimai iš šios būsenos. Tada patikriname ar sugeneruotos naujos būsenos nėra ekvivalenčios kuriai nors iš prieš tai buvusiai būsenai. Norint įsitikinti ar būsenos yra ekvivalenčios, turime palyginti dviejų būsenų tolydines ir diskretines komponentes. Palyginti diskretines komponentes nėra sunku, nes jų skaičius yra baigtinis ir jos gali įgyti tik skaitines reikšmes. Norint palyginti tolydines komponentes, reikia patikrinti, ar kiekvienai tolydinei komponentei galioja sąlyga:

$$\left\{a - \min_l \alpha_l^i \middle| \alpha_k^i < a < \beta_k^i \right\} = \left\{a - \min_l \alpha_l^j \middle| \alpha_k^j < a < \beta_k^j \right\}.]$$

Reikia surasti minimumą iš visų esamų operacijų pradžios laiko momentų, kad būtų gautas tas pats atskaitos laiko momentas. Reikia iš kiekvienos tolydinės komponentės laiko intervalo galo atimti atskaitos laiko momentą ir jį palyginti su tokia pačia tolydine komponente kitos viršūnės būsenoje. Be to, reikia įvertinti momentų indeksų skirtumą tarp generuojamo viršūnės laiko momento ir viršūnės atskaitos laiko momento. Jeigu būsenos generuojamas laiko momentas yra t_n, o minimumo laiko momentas yra $t_m + \alpha$, vertiname skirtumą $n - m$. Šis skirtumas turi būti vienodas abiejose lyginamose viršūnėse.

Vertinant skirtumą $a - \min_l \alpha_l^i$, dažnai neįmanoma vienareikšmiškai jo palyginti, nes gaunamas laiko momentas $t_i - t_j + \alpha$. Norint palyginti $t_i - t_j$ laiko momentų skirtumą, reikia jį įvertinti iš viršaus bei iš apačios. Įverčius skaičiuoti sudaromas tiesinio programavimo uždavinys, kurio tikslo funkcija yra $t_i - t_j$. Uždavinys sprendžiamas Simplekso metodu. Išsprendus uždavinį gautus įverčius iš viršaus ir apačios galime lyginti su kitos būsenos tolydinės komponentės įverčiais.

Ekvivalenčių būsenų radimo algoritmas pateiktas 8 paveikslėlyje.



8 pav. Ekvivalenčių būsenų radimo algoritmas

3 ALGORITMO SUDĖTINGUMO ANALIZĖ

Norint tiksliau nusakyti sistemos būsenos kitimus, reikia ne tik žinoti įvykių seką, bet ir kada tie įvykiai įvyksta. Tai ypač aktualu, kai nagrinėjamos realiojo laiko sistemos. Šiuo atveju sistemos elgesį nusako tokio pavidalo trajektorijos: $\rho = s_0, e_0(t_0), s_1, e_1(t_1), s_2, e_2(t_2), s_3, \ldots$ ir jos vadinamos pėdsakais.

Du pėdsakus $\rho^1 = s_0^1, e_0^1(t_0^1), s_1^1, e_1^1(t_1^1), \ldots$ ir $\rho^2 = s_0^2, e_0^2(t_0^2), s_1^2, e_1^2(t_1^2), \ldots$ laikome ekvivalenčiais, jei jie atitinka tą pačią trajektoriją ir $t_j^1 < t_{j+1}^2$ ir $t_j^2 < t_{j+1}^1$, $\forall j > 0$.

Pėdsakų generuotą ekvivalentumo klasę vadiname elgsena ir žymime:

$$\pi = s_0, e_0[I_0], s_1, e_1[I_1], s_2, e_2[I_2], s_3, e_3[I_3], \ldots,$$

čia I_j – įvykio e_j įvykimo laiko momentų aibės [3].

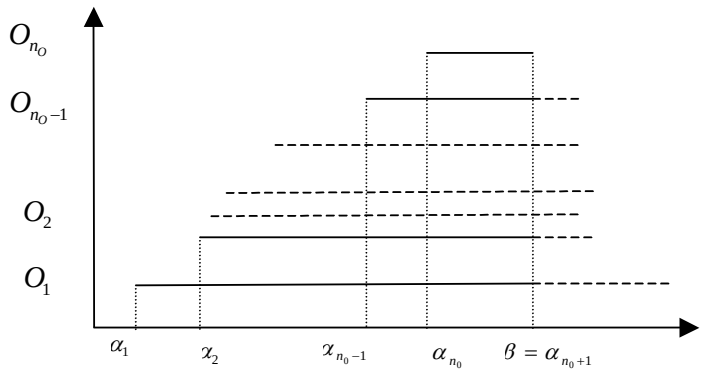
Pasiekiamų būsenų grafą sudaro elgsenų šeima, kurių pirmasis sekos narys vienodas. Jei dviejų elgsenų pradiniai keli elementai sutampa ir įvykiai yra iš tų pačių intervalų, tuomet grafe tos būsenos sutapatinamos tose pačiose viršūnėse. Kyla klausimas, kiek reikia išnagrinėti viršūnių, kad galėtume atsakyti į klausimą, ar visos situacijos yra išnagrinėtos iki tam tikro laiko momento $t_0 + T, T \in R$ nuo modeliavimo pradžios t_0. Tikslui pasiekti reikia atsakyti į klausimą, koks maksimalus galimų perėjimų skaičius iš viršūnės.

Šiam klausimui atsakyti suformuluosime teiginį.

Δ 1 teiginys. Tarkime, kad n_0 - maksimalus operacijų skaičius būsenoje S, tada maksimalus naujų perėjimų skaičius: $n_S(n_0) = \mathrm{O}(n_0^2)$.

Įrodymas:

∇ Tarkime, kad nagrinėjamoje viršūnėje yra n_0 aktyvių operacijų $O_1, O_2, ..., O_{n_o}$, kurios gali generuoti perėjimą. Atitinkamų operacijų tolydinės komponentės $\alpha_i < w_i < \beta$ ir $\alpha_1 < \alpha_2 < ... < \beta$. Šio atvejo situacija pavaizduota 9 paveikslėlyje.



9 pav. Situacija, kai yra n_0 aktyvių operacijų

Remiantis grafo sudarymo algoritmu pateiktu 2.4 skyriuje intervalas $I = \bigcup_{i=1}^{n_O} I_i$, čia $I_i = (\alpha_i, \alpha_{i+1})$ su visais $i = \overline{1, n_O}$.

Sudaromos poros, koks įvykis kuriame laiko intervale įvyks:

(e_1, I_1)

$(e_1, I_2), (e_2, I_2)$

$(e_1, I_3), (e_2, I_3), (e_3, I_3)$

…

$(e_1, I_{n_{O-1}}), (e_2, I_{n_{O-1}}), (e_3, I_{n_{O-1}}), ..., (e_{n_{O-1}}, I_{n_{O-1}})$

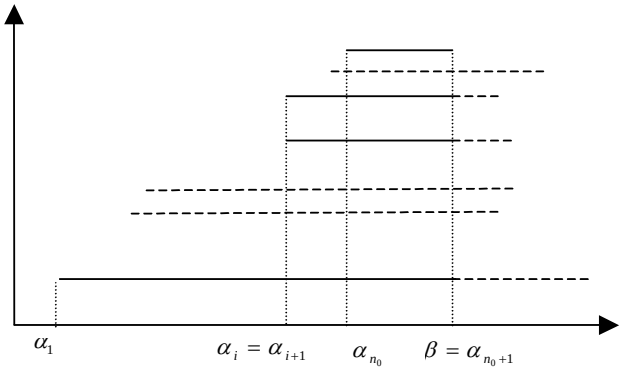
$(e_1, I_{n_O}), (e_2, I_{n_O}), (e_3, I_{n_O}), ..., (e_{n_{O-1}}, I_{n_O}), (e_{n_0}, I_{n_O})$

Nesunku pastebėti, kad tai yra aritmetinės progresijos (d = 1) pirmų n_0 narių suma:

$$\frac{n_O(n_O+1)}{2}$$

Kadangi, kiekvienai porai (e_i, I_i) remiantis algoritmu gali būti du perėjimai ir papildomai du perėjimai vertinant laiką, tai maksimalus skaičius perėjimų nagrinėjamu atveju yra: $n_S(n_O) = n_O(n_O+1)$. $n_S(n_0) = 2n_0(n_0+1)$

Liko įrodyti, kad kitais atvejais perėjimų skaičius yra mažesnis, t.y. egzistuoja α_i, α_{i+1} , kai $\alpha_i = \alpha_{i+1}$ (10 pav.).



10 pav. Situacija, kai $\alpha_i = \alpha_{i+1}$

Šiuo atveju *I* dalinasi į n_{0-1} intervalų, o reikia parodyti, kad:

$$\frac{n_O(n_O+1)}{2} > \frac{(n_O-1)n_0}{2} + (n_0 - i)$$

$$n_O(n_O+1) > (n_O-1)n_0 + 2n_0 - 2i$$

$$n_O(n_O+1) > (n_O-1+2)n_0 - 2i$$

$$n_O(n_O+1) > (n_O+1)n_0 - 2i \; .$$

Ši nelygybė teisinga su visomis *i* reikšmėmis, kai $i = \overline{1,n}$.

Δ 2 teiginys. Būsenų grafo viršūnių skaičiaus augimo greičio priklausomybė nuo įvykių skaičiaus *k*:

$$n_k(n_a) = \mathrm{O}(n_s^{\,k-1}(n_a)),$$

čia n_a - maksimalus operacijų skaičius būsenoje.

∇ Tarkime , kad n_a - maksimalus aktyvių operacijų skaičius būsenose. Remiantis pirmu teiginiu gavome, kad perėjimų skaičius yra:

$$n_a(n_a+1)$$

Kadangi kiekviename žingsnyje viršūnių skaičius išauga $n_a(n_a+1)$ kartų, tai $i-1$ perėjime buvo q_{i-1} viršūnių. Radus visus galimus perėjimus, viršūnių skaičius *i*-ajame žingsnyje $q_i = q_{i-1}n_a(n_a+1)$, t.y. geometrinė progresija. Jos *n* narių suma bus lygi

$$n_k(n_a) = \frac{n_a^{\,k}(n_a+1)^k - 1}{n_a(n_a+1)-1} = \frac{n_s^{\,k}(n_a)-1}{n_s(n_a)-1} \; .$$

Δ 3 teiginys. Maksimalus įvykių skaičius elgsenoje per laiko tarpą *T* yra lygus:

$$n(T) = \sum_{i=1}^{n_0} \left[\frac{T}{\alpha_i} \right],$$

čia α_i - įvykio e_i valdančios sekos minimumas.

∇ Blogiausia situacija, kai visos operacijos aktyvios pradiniu laiko momentu t_0 ir perėjimų operatoriai $H(e_i)$ keičia tik w_i operaciją tokiu būdu:

$$w_i(t_m) = w_i(t_m - 0) + \xi_m, \;\; \alpha_i < \xi_m$$

Šiuo atveju per laiko tarpą T įvykis e_i įvyks $\left[\frac{T}{\alpha_i}\right]$ kartų.

Δ 4 teiginys. Norint garantuoti sistemos analizę laiko intervale $(t_0, t_0 + T)$, reikia išnagrinėti ne daugiau kaip:

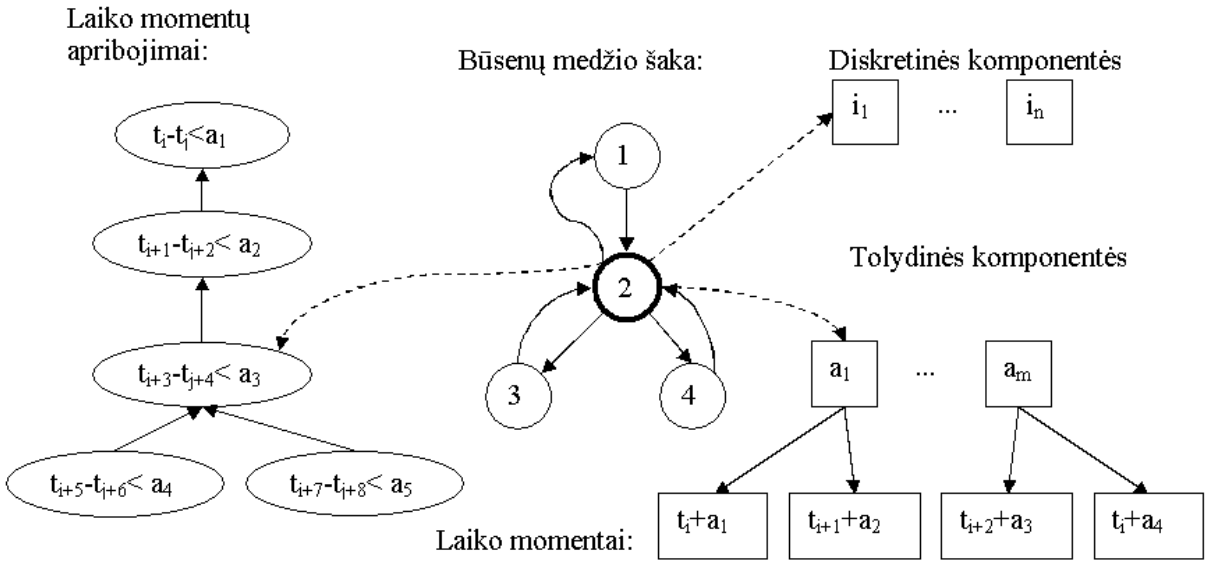
$$S(T, n_a) = \mathrm{O}(n_s^{n(T)-1}(n_a)).$$

∇ Įrodymas seka iš 2 ir 3 teiginių.

4 PBG GENERAVIMO ALGORITMO KOMPIUTERINĖ REALIZACIJA

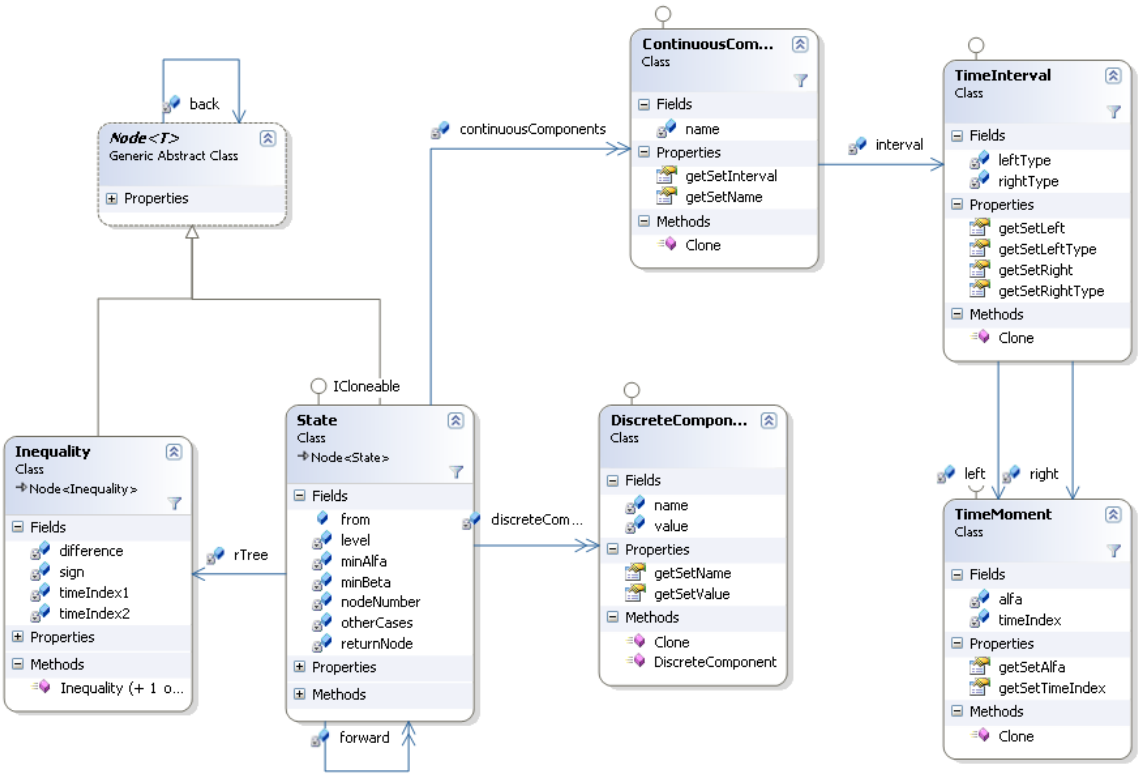
4.1 Duomenų struktūros klasių diagrama

Realizuojant pasiekiamų būsenų grafo algoritmą, vienas svarbiausiu uždavinių buvo sukurti struktūrą duomenų saugojimui. Darbe pasiūlyta dviejų medžių struktūra (11 pav.). Čia duomenys saugomi nelygybių ir būsenų medžiuose. Būsenų medžio viršūnė susideda iš tokių komponenčių: diskretinės komponentės, tolydinės komponentės ir laiko momentų apribojimų aibės R. Kiekviena šio medžio viršūnė saugo perėjimus į sekančias viršūnes bei sugrįžimą į prieš tai buvusią viršūnę. Diskretinės ir tolydinės komponentės yra kintamo dydžio sąrašai, kurie leidžia realizuoti uždavinius specifikuotus įvairiausiomis specifikacijomis. Laiko momentų apribojimams saugoti naudojamas atskiras medis, kadangi tas pats apribojimas gali būti naudojamos skirtingose viršūnėse. Apribojimų medis saugo tik sugrįžimus atgal, nes uždaviniui spręsti pakanka prieš tai buvusių apribojimų.



11 pav. PBG duomenų struktūros schema

Klasių diagrama duomenų struktūrai realizuoti pateikta 12 paveikslėlyje. Realizuoti du medžiai - būsenų medis (*State*) ir nelygybių medis (*Inequality*). Nelygybių medis susideda iš dviejų laiko momentų indeksų ir įverčio tarp jų. Šis medis turi tik atgalinį ryšį. Būsenų medis susideda iš sąrašo diskretinių komponenčių (*Diskrete component*), tolydinių komponenčių (*Continuous component*) bei nuorodos į nelygybių medį. Būsenų medis turi atgalinį ryšį bei sąrašą ryšių į sekančias būsenas. Tolydinė komponentė savyje turi laiko intervalą (*Time interval*). Laiko intervalas susideda iš dviejų laiko momentų (*Time moment*).



12 pav. Klasių diagrama duomenų struktūrai realizuoti

4.2 Prototipų realizacija

Prototipai buvo realizuoti C# programavimo kalba, remiantis .NET framework 3.5 technologijomis. Šie prototipai realizuoti kaip vienos gijos programa. Programos realizacijos langas ir pagrindiniai jo komponentai pateikti 13a paveikslėlyje.

Prototipuose realizuoti trys uždaviniai:

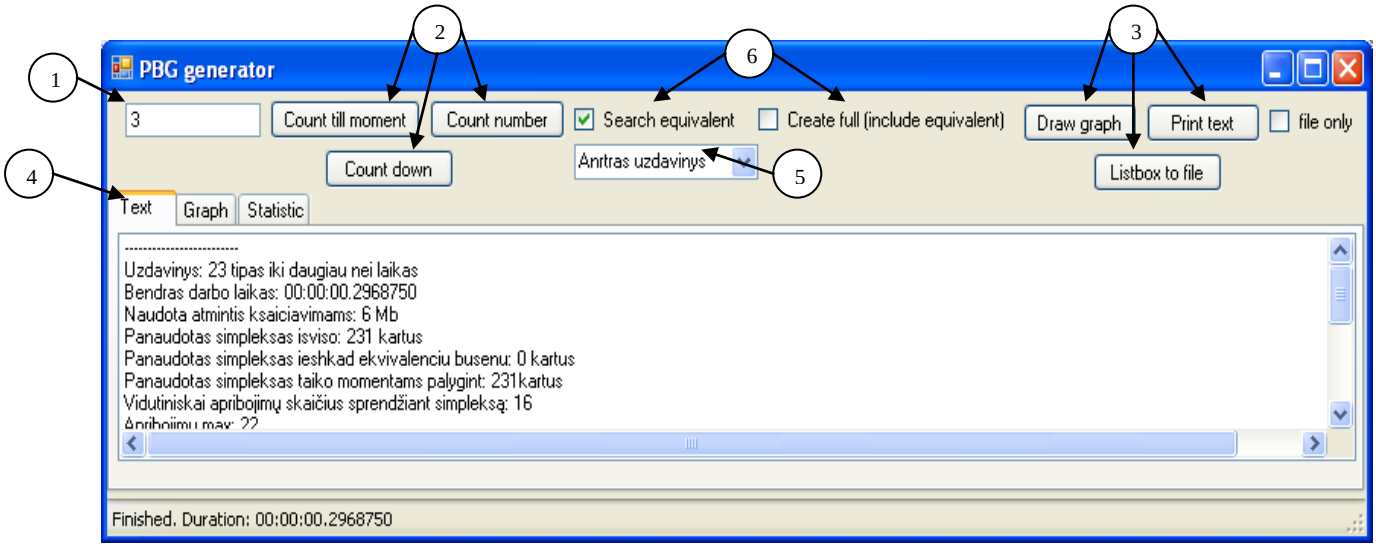
1. dvikanalė aptarnavimo sistema su su pirmo tipo valdančiąja seka,
2. dvikanalė aptarnavimo sistema su su antro tipo valdančiąja seka,
3. gamybinė sistema, susidedanti iš penkių aptarnaujančių paraiškas įrenginių (A, B, C, D, E).

Spendžiamas uždavinys nurodomas 5 laukelyje.

Kiekvienam uždaviniui nustatomi papildomi parametrai (laukelis 6): *Search equivalent* – ieškomos ekvivalenčios būsenos; *Create full* – sukuriamas pilnas grafas pasinaudojant ekvivalenčiomis būsenomis.

Atlikti skaičiavimus galima trimis būdais (laukelis 2): *Count till moment* – skaičiuojamas laiko intervalas nuo 0 iki įvesto T (įvedamas laukelyje 1) laiko momento; *Count number* - analizuojamos viršūnės iš eilės viena po kitos, naujai sugeneruotą statant į eilės galą (analizavimo ilgis nurodomas laukelyje 1); *Count down* – analizuojama viena šaka žemyn iki kol randama ekvivalenti būsena.

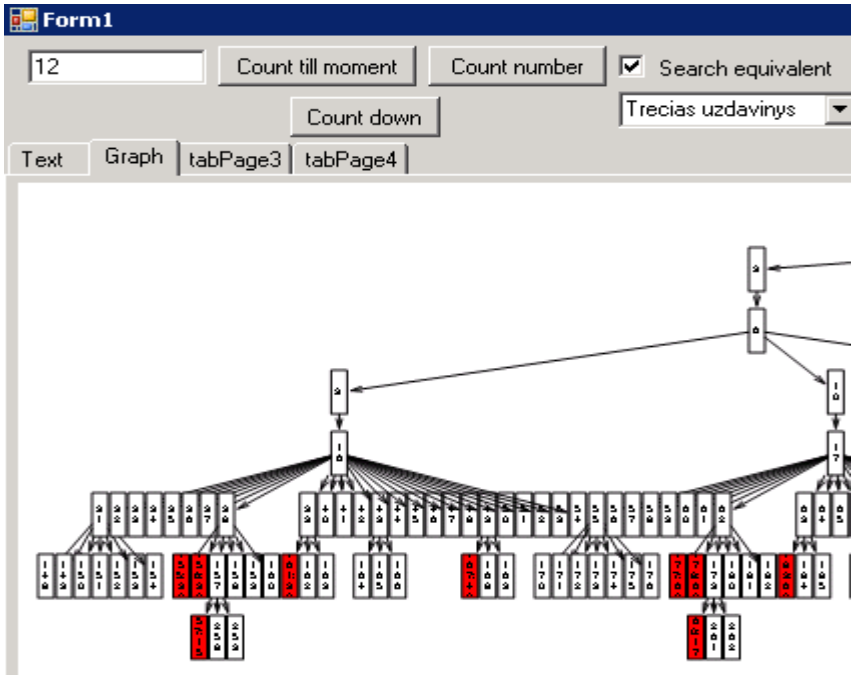
Skaičiavimo rezultatams pristatyti, gali būti pasirinkta (laukelis 3) statistika apie skaičiavimus, pateikiant pasiekiamų būsenų grafą kaip tekstą ekrane ir/arbe faile arba pateikiant grafinį grafo atvaizdą (13b pav).



13a pav. Pagrindinis programos langas

13a paveikslėlyje pateikti programos nustatymai, kai nagrinėjama dvikanalė aptarnavimo sistema su antro tipo valdančiąja seka. Sprendžiant uždavinį buvo ieškoma ekvivalenčių būsenų bei sprendžiamas laiko intervalas iki 3 laiko momentų ir pateikiami statistiniai programos rezultatai, tokie kaip sprendimo laikas, Simplekso panaudojimas ir t.t.

13b paveikslėlyje pateikti programos skaičiavimo rezultatai, kai buvo analizuojama gamybinė sistema, ieškomos ekvivalenčios būsenos bei sprendžiamas laiko intervalas iki 12 laiko momentų.



13b pav. Realizacijos rezultatų pavyzdys

Baltai pavaizduotos suskaičiuotos viršūnės analizuojamame laiko intervale, raudonai - ekvivalenčios viršūnės prieš tai esančioms. Visos viršūnės pavaizduotos su jų numeriais, o ekvivalenčios viršūnės – su viršūnės numeriu ir ekvivalenčios viršūnės numeriu.

Daugiau realizacijos pavyzdžių pateikta 2 priede.

4.3 Simplekso metodo apribojimų aibė

Realizuojant pasiekiamų būsenų grafo sudarymo prototipus, buvo pastebėtos situacijos, kad naudojant [23] darbe pateiktą algoritmą susidaro tokio tipo situacijos:

Jei turime būseną *121: (0;(t_4+0,t_3+4);(t_4+3,t_4+5);(t_2+3,t_2+5);R121)*

ir jos bendra apribojimų aibė:

$t_0+0.5 < t_1 < t_0+3.5$ *(1)*

$t_1+0.5 < t_2 < t_1+2.5$ *(2)*

$t_1+1 < t_2 < t_1+2.5$ *(3)*

$t_1+1 < t_2 < t_1+2$ *(4)*

$t_2+0.5 < t_3 < t_1+2.5$ *(5)*

$t_1+3 < t_4 < t_2+3$ *(6),*

ir skaičiuojant būsenos minimumus, lyginant laiko momentus, prisideda papildomas apribojimas:

$t_2+1 < t_3 < t_2+1.5$ (7).

Šiuo atveju, lyginant laiko momentus t_2 ir t_4 ir taikant Simplekso metodo apribojimų optimizavimo algoritmą (2.2 skyrius), gauname, kad pradinė aibė *K* susideda iš indeksų: *4,2*. Imame didžiausią laiko indeksą *4* ir įtraukiame (6) lygtį į apribojimų aibę, o laiko indeksus *1* ir *2* į aibę *K*. Kadangi aibėje *K* jau egzistuoja laiko momento indeksas *2*, tai po šios operacijos aibėje *K* bus laiko momentų indeksai *2 ir 1*.

Dar kartą imame didžiausią laiko indeksą, šiuo atveju - *2*. Į apribojimų aibę įtraukiame lygtis (4), (3) ir (2), o į aibę *K* laiko indeksą 1. *K* aibėje po šios operacijos liko vienas laiko indeksas, todėl turėtume įtraukti jo įvertį, sprendžiant tiesinio programavimo uždavinį. Šiuo atveju tai būtų: $0.5<t_1<3.5$.

Bandome rasti įvertį *t4-t2* iš viršaus ir iš apačios ir gauname:

$Min(t_4-t_2) = 1$

$Max(t_4-t_2)=3$

ir $t_4=3.5$, $t_2=2.5$, $t_1=0.5$

t3 lieka neįvertintas.

Į (*5)* ir (*7)* lygtis įrašę gautus rezultatus, gauname, kad: (*5)*$3<t_3<3$ *ir (7)* $3,5<t_3<4$.

Matome, kad neįvertinus t_3 laiko momento, gaunamas intervalo trūkis, kuris vėliau gali neleisti įvertinti reikiamų laiko momentų susijusių su t_3 laiko momentu.

Sprendžiant uždavinį ir į apribojimų aibę įtraukus visas lygtis, gauname:

$Min(t_4-t_2) = 1.5$

$Max(t_4-t_2)=3$

ir $t_4=3.5$, $t_3=3$, $t_2=2$, $t_1=0.5$.

Šiuo atveju visos nelygybės tenkina sąlygas. Taigi galime daryti išvadą, kad iš apribojimų aibės negalima atmesti nei vienos lygties, kurią galima išreikšti per bet kurį laiko momentą. Kadangi generuojant PBG laiko momentai visada išreiškiami vienas per kitą, tai į apribojimų aibę reikia įtraukti visas nelygybes. Akivaizdu, apribojimų aibėje yra nelygybių, kurios neturi įtakos uždavinio sprendimui, o tik padidina metodo sprendimo laiką (reikia perrinkti daugiau daugiamačio briaunainio viršūnių). Vadinasi, reiktų ieškoti šio algoritmo modifikacijų, kurios leistų sumažinti Simplekso metodo apribojimų nelygybių skaičių ir tiktų visais atvejais.

5 REALIZUOTOS SISTEMOS IR JŲ SUDĖTINGUMO TYRIMAS

Šiame darbe pristatomi trijų uždavinių tyrimai. Buvo tiriama dvikanalė aptarnavimo sistema su dvejomis skirtingomis valdančiomis sekomis bei gamybinė sistema, susidedanti iš penkių aptarnaujančių paraiškas įrenginių.

Sprendžiant kiekvieną iš jų buvo vertinami:

- Resursai reikalingi skaičiavimams atlikti. Vertinami bendri resursai uždaviniui bei atskiroms procedūroms – Simplekso metodo sprendimo, ekvivalentumo ieškojimo. Skaičiavimams atlikti buvo naudojamas kompiuteris su vienu procesoriumi bei 8 GB operatyvinės atminties. Laikinius resursus būtų galima gerokai pagerinti realizavus programą su daug gijų bei naudojant daugiaprocesorines sistemas. Tai atlikti būtų nesudėtinga, nes kiekviena grafo viršūnė gali būti skaičiuojam nepriklausomai nuo kitų.
- Kiekybiniai įvertinimai. Vertinamas išspręstų viršūnių kiekis, panaudotų algoritmų kiekis, apribojimų kiekis.

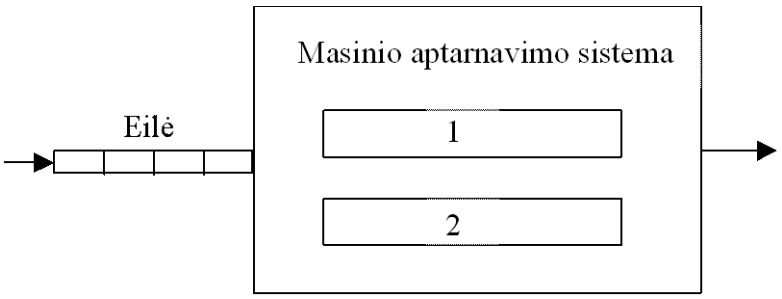
5.1 Dvikanalė aptarnavimo sistema (pirmas atvejis)

Dvikanalėje sistemoje (14 pav.) yra vienas įėjimas su eile ir du aptarnavimo įrenginiai. Atėję pranešimai statomi į eilę ir kai tik atsilaisvina nors vienas aptarnavimo įrenginys, pranešimas perduodamas jam. Jei laisvi abu įrenginiai, pranešimas perduodamas pirmajam įrenginiui. Šios sistemos agregatinė specifikacija pateikta 1 priede.

Dvikanalės sistemos realizacijoje pirmiausiai nustatomos pradinės sąlygos:

įvykių valdančios sekos: $4<\alpha_i<6$, $3<\beta_i<5$, $2<\varphi_i<4$, $\forall i=1,2,3$;
apribojimų aibė $R_0=\varnothing$;

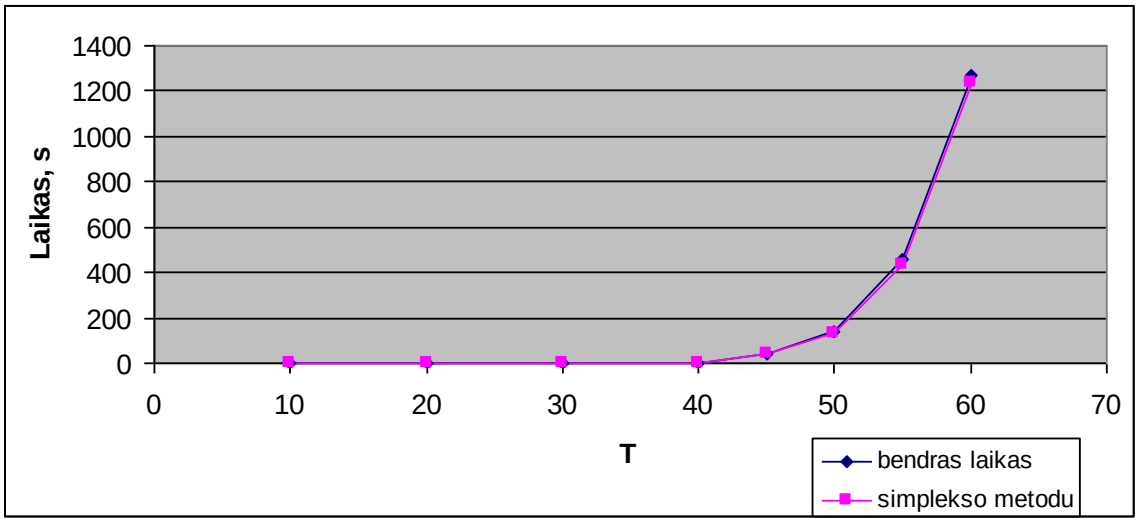
Grafo generavimas pradedamas nuo pradinės būsenos, kuri nusakyta specifikacijoje. Šiuo atveju pradiniu laiko momentu t_0 viršūnė yra tokio pavidalo: $1:\left(0;\left(t_0+4,t_0+6\right),\varnothing,\varnothing;R_0\right)$, čia $R_0=\varnothing$.



14 pav. Dvikanalė aptarnavimo sistema

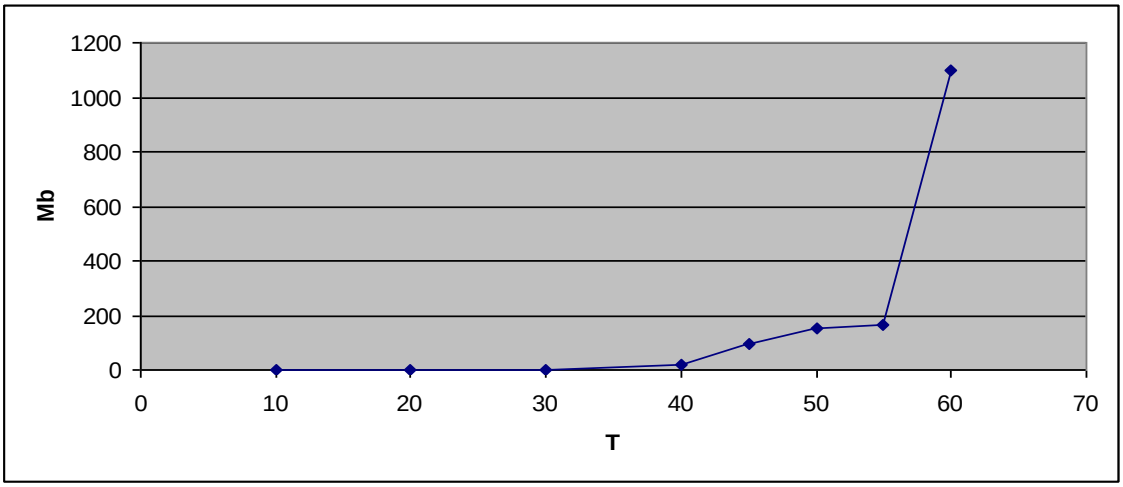
Sistemos įvertinimas, ją analizuojant įvairiuose laiko intervaluose $(t_0, t_0 + T)$

1. *Resursų įvertinimai*



15 pav. Skaičiavimo trukmė laiko intervaluose

Programos skaičiavimo laikas auga eksponentiniu greičiu, nagrinėjant vis didesnį laiko intervalą (15 pav.). Nagrinėjant laiko intervalus iki 55 laiko vienetų jis nesiekia minutės, o pasiekus 60 laiko vienetų - skaičiavimo trukmė viršija daugiau nei 20 min. Pagrindinės laiko sąnaudos išnaudojamos sprendžiant tiesinio programavimo uždavinius Simplekso metodu. Toks staigus laiko didėjimas atsiranda dėl didelio analizuojamo viršūnių skaičiaus.

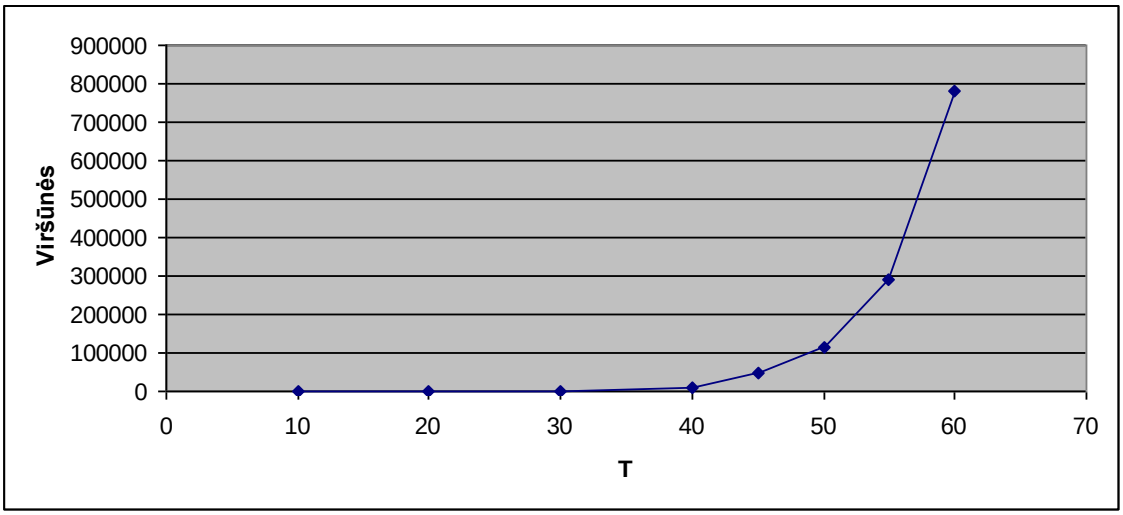


16 pav. Atminties naudojimas laiko intervaluose

Grafike pateiktas (16 pav.) kompiuterio atminties sunaudojimas sprendžiant uždavinį skirtingose laiko intervaluose nėra labai tikslus. *.Net* technologijose objektų naikinimui yra automatinis mechanizmas, vadinamas *Grabage collector*, kuris vykdomas pačios operacinės sistemos automatiškai tam tikrais laiko momentais. Tai akivaizdžiai matosi grafiko taške ties 55 laiko vienetu. Grafike matomas žymus atminties didėjimas, kuris susijęs su vis didėjančiu viršūnių kiekiu, kurį reikia saugoti atmintyje. Atminties kiekio naudojimą būtų galima mažinti, nesaugant

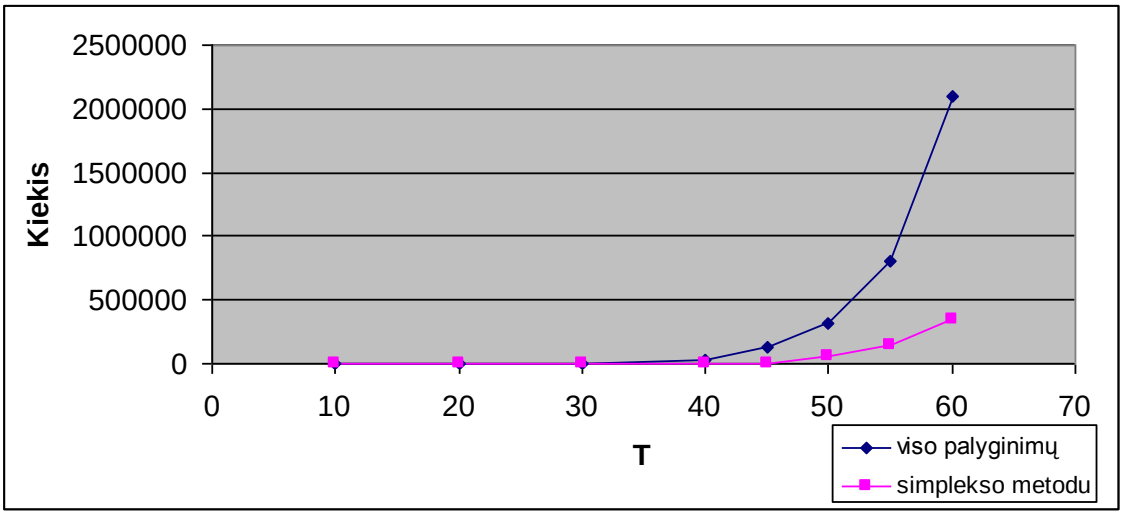
atmintyje nereikalingų tolesniuose skaičiavimuose rezultatų, o juos rašyti į failus ar duomenų bazes.

2. *Kiekybiniai įvertinimai*



17 pav. Grafo viršūnių skaičius

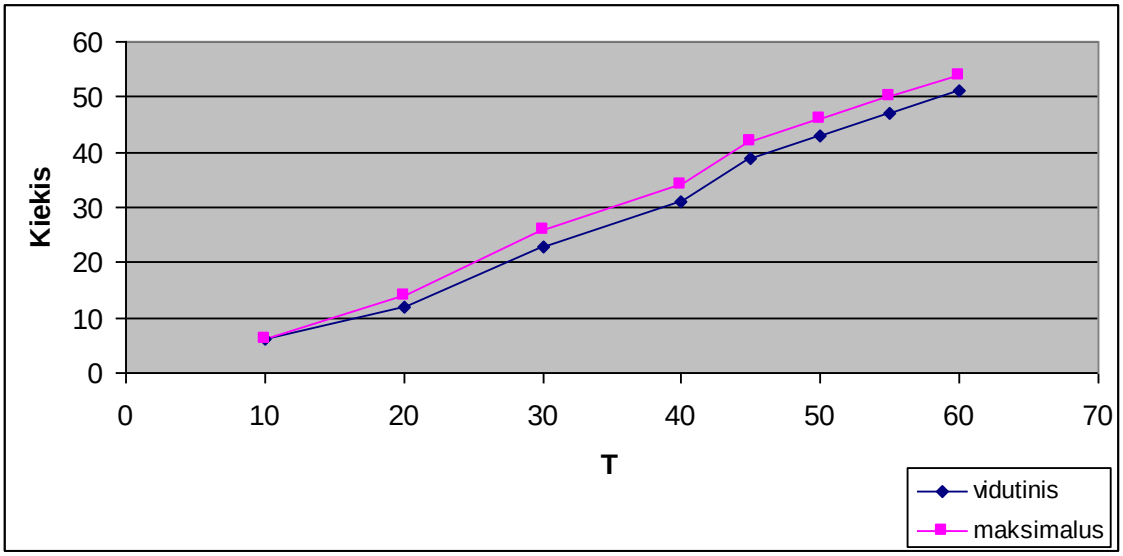
Analizuojant grafo viršūnių skaičiaus priklausomybę nuo vertinamo laiko intervalo $(t_0, t_0 + T)$ (17 pav.) buvo nustatyta, kad viršūnių skaičius didėja eksponentiniu greičiu. Sugeneruojamas grafas su daugiau nei 100000 viršūnių, kad įvertinti laiko intervalą iki 50 laiko vienetų. Įvertinant intervalą iki 60 laiko vienetų, viršūnių skaičius siekia apie 800000. Akivaizdu, kad ir grafo skaičiavimo greitis auga tokiu pačiu greičiu. Toks viršūnių didėjimas neprieštarauja 4 teiginiui.



18 pav. Laiko momentų palyginimų skaičius

Generuojant grafą dažnai tenka atlikti dviejų laiko momentų palyginimą (žiūr. 2.2 skyrių). Grafike (18 pav.) parodyta kiek buvo iš viso lyginama laiko momentų bei kiek iš jų buvo lyginama naudojantis Simplekso metodu. Reikia pastebėti, kad Simplekso metodas lyginant du laiko

momentus buvo panaudotas ketvirtadaliui palyginimų, tuo tarpu kitus laiko momentus buvo galima įvertinti vienareikšmiškai. Be to, kaip žinoma, pagrindinės laiko sąnaudos vykdant grafo generavimą laiko intervale yra, būtent, sprendžiant tiesinio programavimo uždavinius, Simplekso metodu (15 pav.).



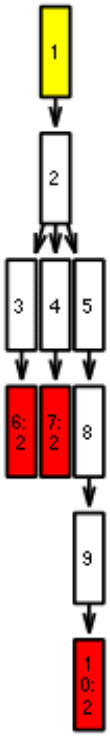
19 pav. Lygčių skaičius sprendžiant Simplekso metodu

Tiek maksimalus, tiek vidutinis lygčių skaičius generuojant grafą vis didesniame laiko intervale (19 pav.) auga tiesiškai. Tai yra dėl to, kad generuojami viršūnių perėjimai yra baigtinio ir to paties dydžio.

Sistemos įvertinimas, kai sistema analizuojama laiko intervaluose $(t_0, t_0 + T)$ *bei ieškomos ekvivalenčios būsenos.*

Sprendžiant šią sistemą PBG metodu ir ieškant ekvivalenčių būsenų algoritmu pateiktu 2.6 skyriuje, sistema yra išsprendžiama išanalizavus 7 viršūnes. Tam yra atliekami 33 laiko momentų palyginimai, iš kurių 6 kartus vertinama Simplekso metodo pagalba. Be to 2 kartus naudojamas Simplekso metodas lyginant būsenų ekvivalentiškumą. Visiems skaičiavimams atlikti reikalinga mažiau nei sekundė. Gautas PBG su ekvivalenčiomis būsenomis yra pateiktas 20 paveikslėlyje, kur galime matyti, kad:

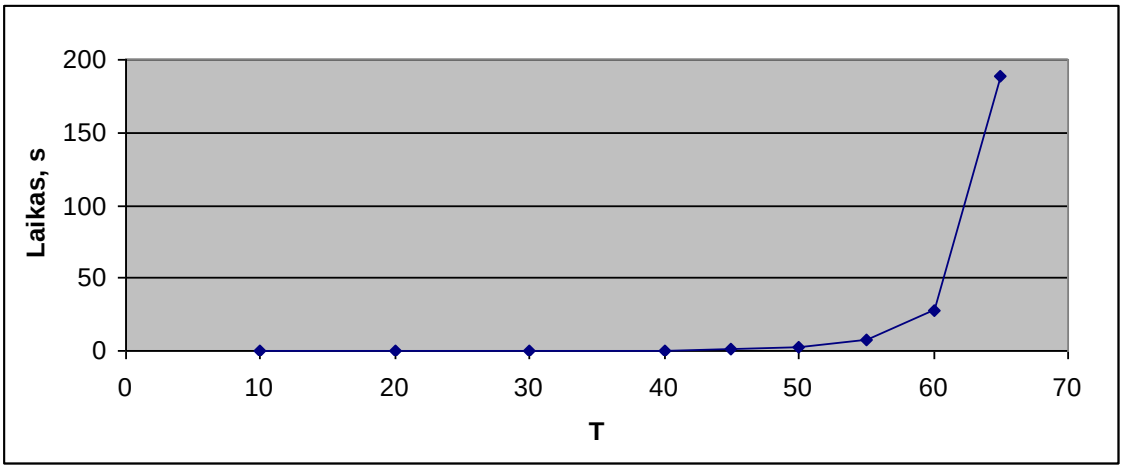
- 1 viršūnė – yra pradinė būsena,
- 6 viršūnė - ekvivalenti 2,
- 7 viršūnė – ekvivalenti 2,
- ir 10 viršūnė taipogi ekvivalenti 2 būsenai.



20 pav. Ekvivalenčių būsenų PBG

Sugeneravus tokį PBG su ekvivalenčiomis būsenomis, galime teikti, kad sistema yra pilnai išspręsta. Turint tokį PBG, galima susidaryti grafą pageidaujamame laiko intervale. Sudarant grafą pageidaujamame laiko intervale nereikalinga atlikti jokių sudėtingų skaičiavimų, o tiesiog reikia perskaičiuoti laiko momentų indeksus sekančiai būsenai pagal ekvivalenčią būseną.

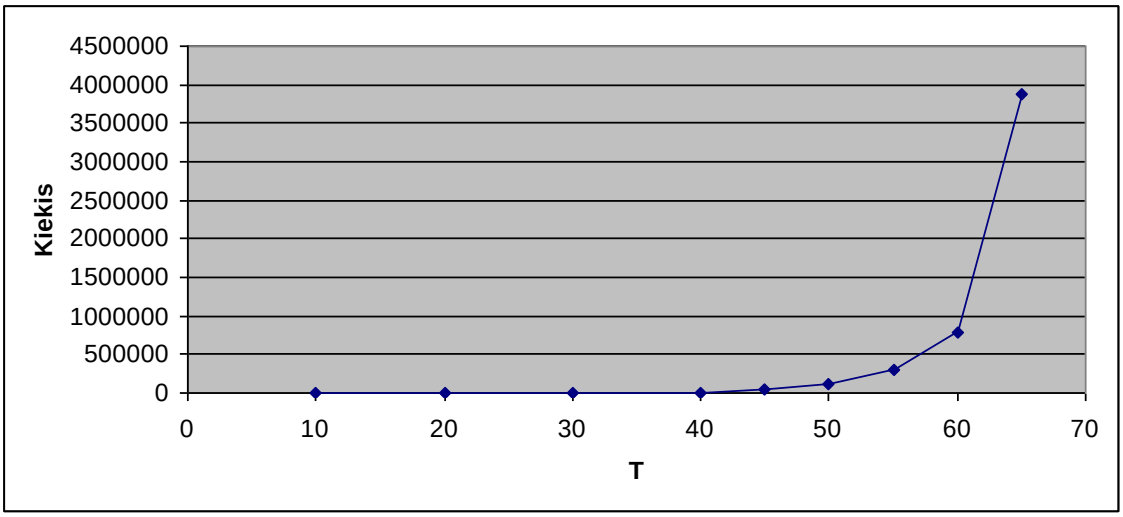
Sistemos įvertinimas, kai sistema analizuojama laiko intervaluose $(t_0, t_0 + T)$ *ir grafas generuojamas iš ekvivalenčių būsenų PBG*



21 pav. Skaičiavimo trukmė laiko intervaluose

Šiuo atveju (21 pav.) skaičiavimo trukmė žymiai sumažėjo palyginus su skaičiavimo trukme, kai sistema buvo analizuojama laiko intervaluose ir nebeieškoma ekvivalenčių būsenų (15 pav.).

Kai pirmu atveju sugeneruoti PBG laiko intervale iki 60 laiko vienetų reikėjo maždaug 20 min, tai generuojant grafą tame pačiame laiko intervale iš ekvivalentaus PBG, tai užtruko tik apie puse minutės. Simplekso metodas sprendžiant šį uždavinį buvo panaudotas tik 8 kartus. Visos kitos viršūnės buvo suskaičiuotos pakeičiant tik laiko momentų indeksus. Akivaizdu, kad nėra tikslinga naudoti sprendimo būdą, kai nėra ieškomos ekvivalenčios būsenos.



22 pav. Grafo viršūnių skaičius

Viršūnių skaičius (22 pav.) auga ypač dideliu greičiu skirtinguose laiko intervaluose. Galima pastebėti, kad viršūnių skaičius padidėja nuo 3-5 kartų vertinant skirtingus laiko intervalus su laiko vienetų skirtumu 5 vienas tarp kito.

5.2 Dvikanalė aptarnavimo sistema (antras atvejis)

Analizuojama ta pati dvikanalė aptarnavimo sistema pateikta 1 priede, tik pakeičiamos pradinės sąlygos. Dvikanalės sistemos realizacijoje nustatomos tokios pradinės sąlygos:

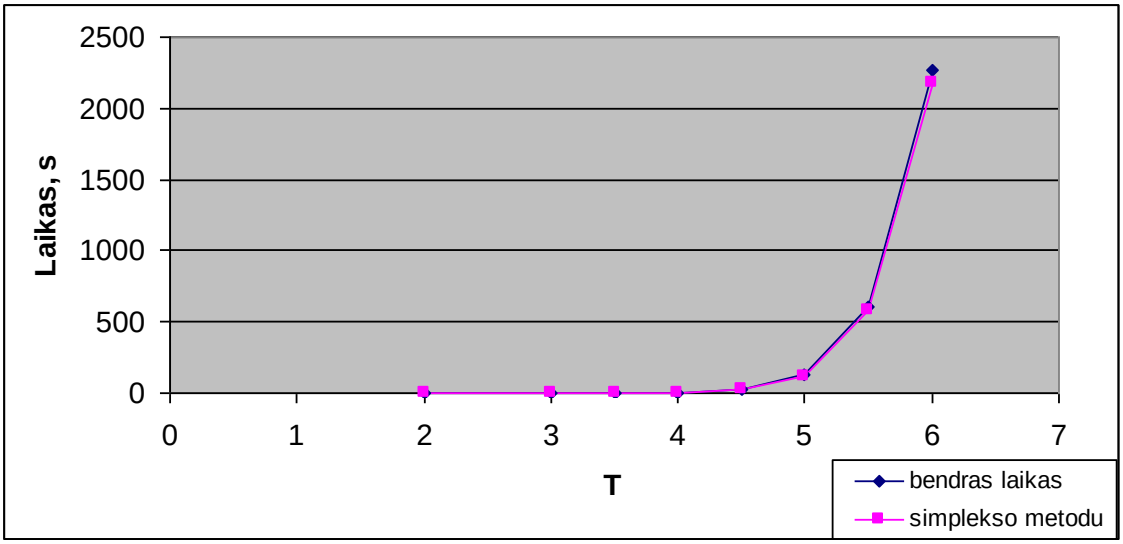
įvykių valdančios sekos: $0{,}5 < \alpha_i < 4,\ 3 < \beta_i < 5,\ 3 < \varphi_i < 5,\ \forall i = 1,2,3;$

apribojimų aibė $R_0 = \varnothing$;

Grafo generavimas pradedamas nuo pradinės būsenos, kuri nusakyta specifikacijoje. Šiuo atveju pradiniu laiko momentu t_0 viršūnė yra tokio pavidalo: $1:(0;(t_0 + 0{,}5, t_0 + 4), \varnothing, \varnothing; R_0)$, čia $R_0 = \varnothing$.

Sistemos įvertinimas, analizuojant įvairiuose laiko intervaluose $(t_0, t_0 + T)$

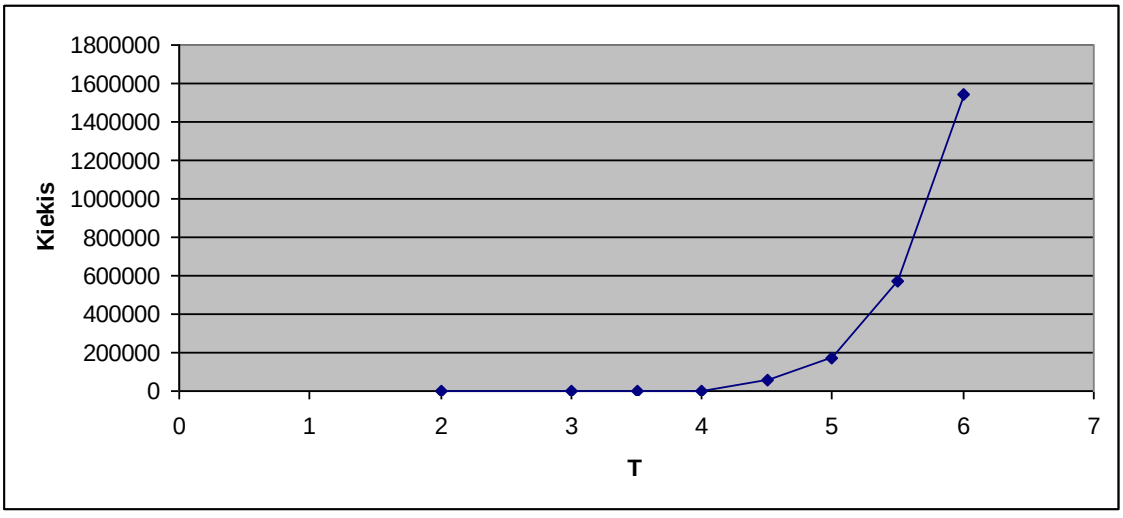
1. *Resursų įvertinimai*



23 pav. Skaičiavimo trukmė laiko intervaluose

Skaičiavimo trukmė kaip ir anksčiau gautuose rezultatuose daugiausiai priklauso nuo Simplekso metodo sprendimo laiko (23 pav.). Sudarant PBG laiko intervale iki 6 laiko momentų, skaičiavimai užtruko 37 min. Tokio rezultato ganėtinai mažame laiko intervale buvo galima tikėtis dėl šio uždavinio operacijų trukmės minimumo, kuris yra 0,5 (3 teiginys). Galima daryti išvadą, kad uždavinio sprendimo laikas labai priklauso nuo pasirinktų valdančiųjų sekų operacijų trukmių.

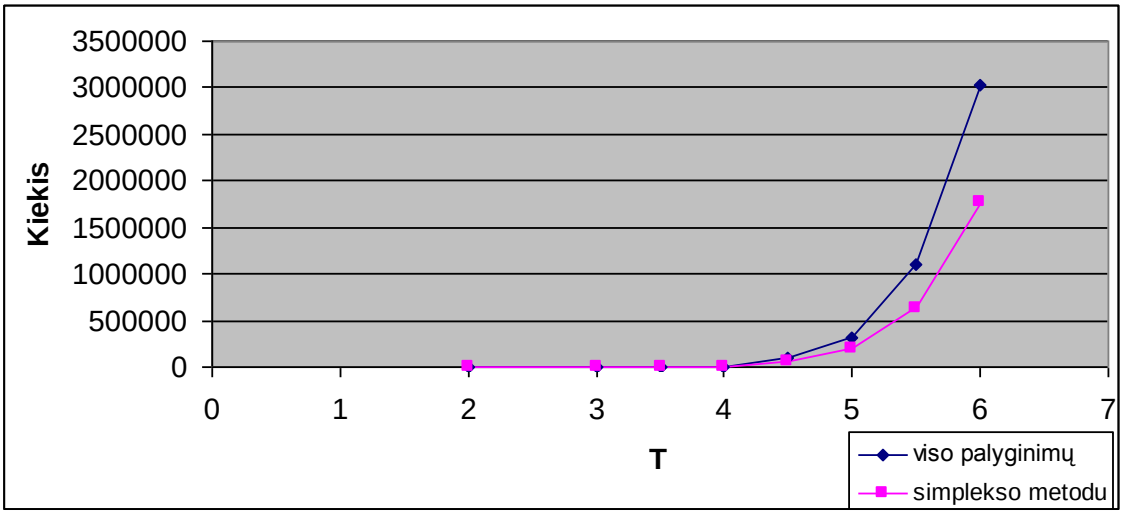
3. *Kiekybiniai įvertinimai*



24 pav. Grafo viršūnių skaičius

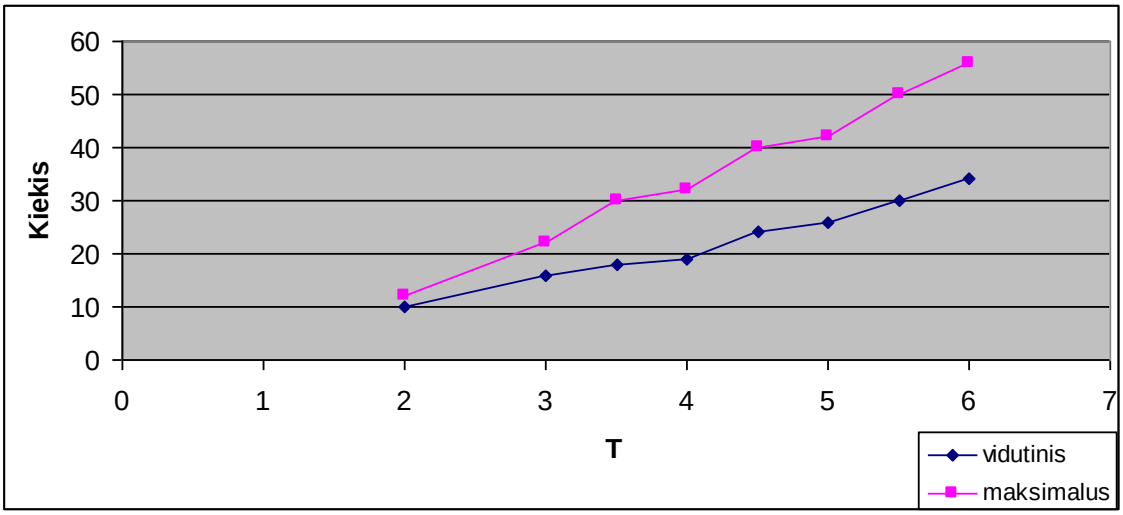
Generuojant grafą iki 5,5 laiko momentų buvo sukurta 600000 grafo viršūnių (24 pav.). Lyginant su pirmuoju uždaviniu pastebėsime, kad laiko sąnaudos sprendžiant šį uždavinį tam

pačiam viršūnių kiekiui surasti buvo daugiau nei du kartus didesnės. Paanalizuokime, kodėl taip atsitiko.



25 pav. Laiko momentų palyginimų skaičius

Šiuo atveju kiekviename laiko intervale reikia palyginti žymiai daugiau laiko momentų ir daugiau nei pusę laiko momentų lyginama naudojant Simplekso metodą (25 pav.). O kadangi pagrindinės laiko sąnaudos sprendžiant uždavinį ir yra Simplekso metodui spręsti, tai įtakoja ilgesnį uždavinio spendimo laiką.

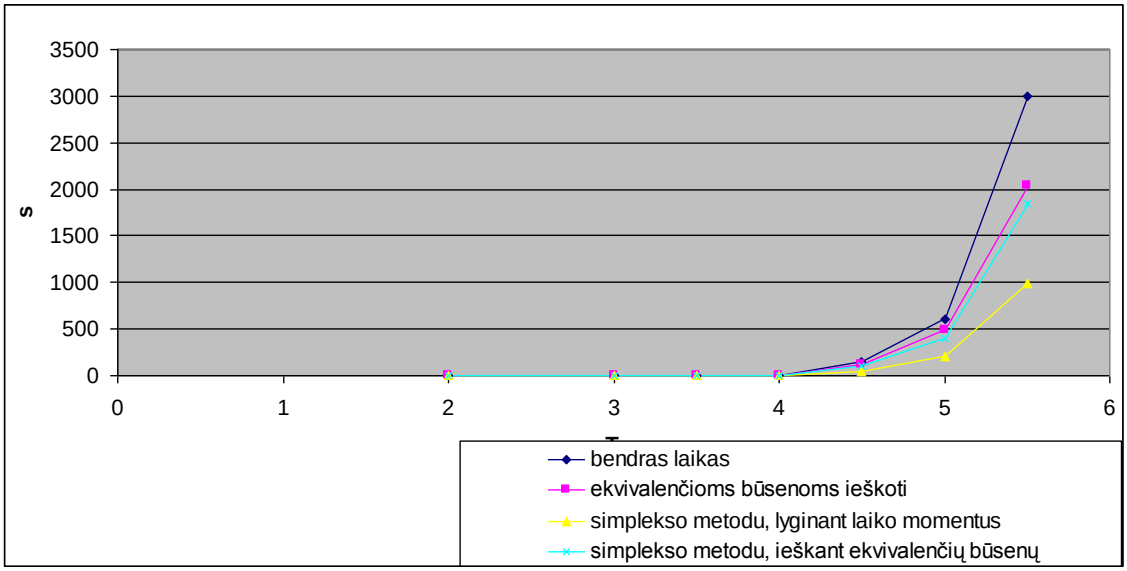


26 pav. Lygčių skaičius sprendžiant Simplekso metodu

Lygčių skaičius sprendžiant Simplekso metodu auga vienodai didinant laiko intervalus (26 pav.). Tačiau matome, kad maksimalus lygčių skaičius auga žymiai greičiau nei vidutinis. Šį reiškinį galima būtų paaiškinti tuo, kad atsiranda vis daugiau viršūnių, kurios skyla lyginant du laiko momentus.

Sistemos įvertinimas, kai sistema analizuojama laiko intervaluose $(t_0, t_0 + T)$ *bei ieškomos ekvivalenčios būsenos*

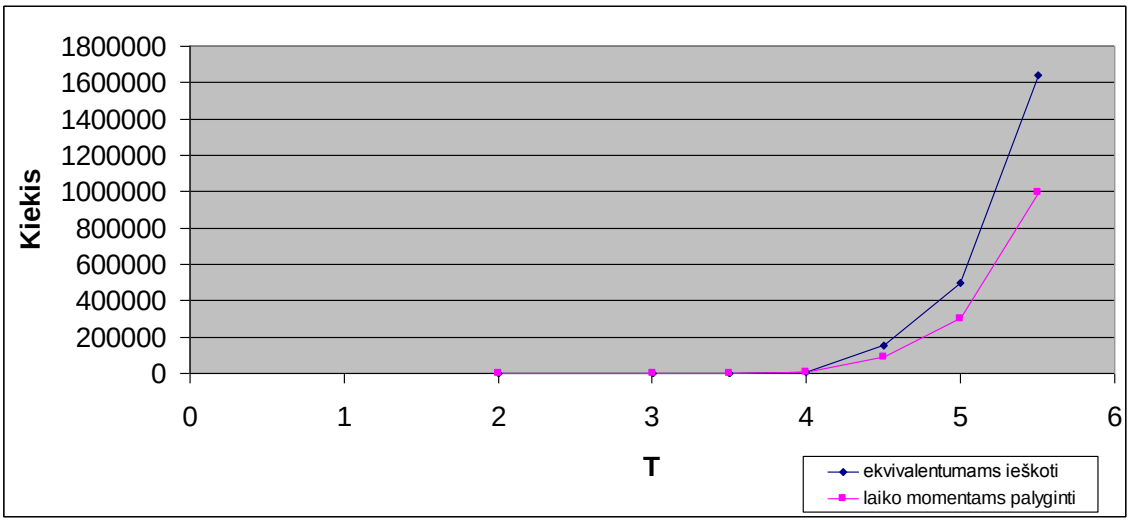
1. *Resursų įvertinimai*



27 pav. Skaičiavimo trukmė laiko intervaluose

Kaip matome iš grafiko (27 pav.), ieškant ekvivalenčių būsenų, laikas sugeneruoti PBG žymiai padidėjo. Jei imsime intervalą iki 5,5 laiko momentų, sugeneruoti grafą šiame laiko intervale nebeieškant ekvivalenčių būsenų pakako ~10 minučių, o dabar tai užtruko net 48 min. Lyginimas Simplekso metodu dviejų laiko momentų užtruko panašiai. Tačiau atsirado laikas, reikalingas patikrinti būsenų ekvivalentumui. Kaip minėjome 2.6 skyriuje, ekvivalentumui įvertinti taip pat naudojamas Simplekso metodo įvertis. Ieškant ekvivalentumų ir sprendžiant Simplekso metodu, buvo užtrukta dvigubai daugiau laiko nei palyginant laiko intervalus.

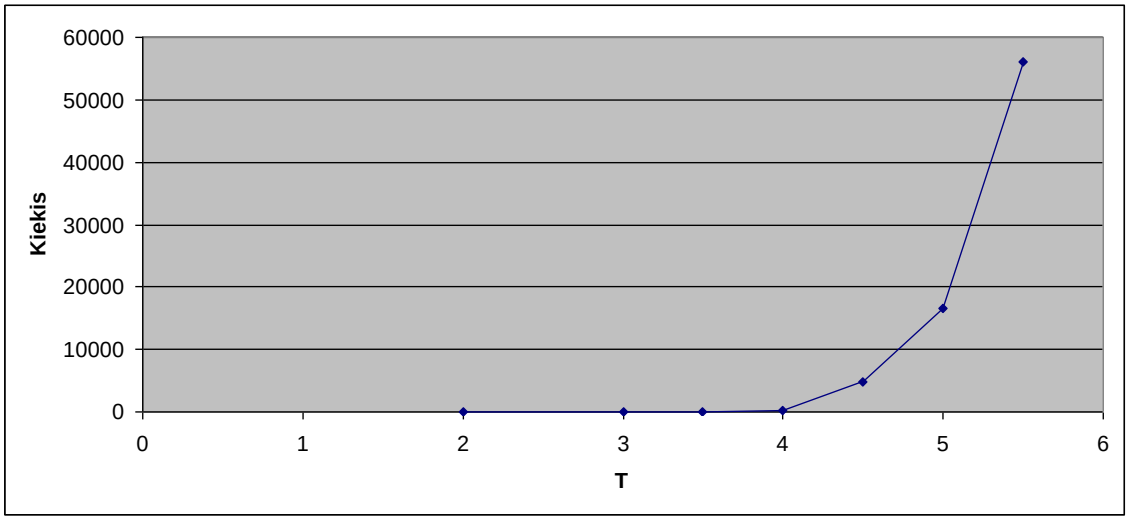
2. *Kiekybiniai įvertinimai*



28 pav. Simplekso metodo panaudojimas

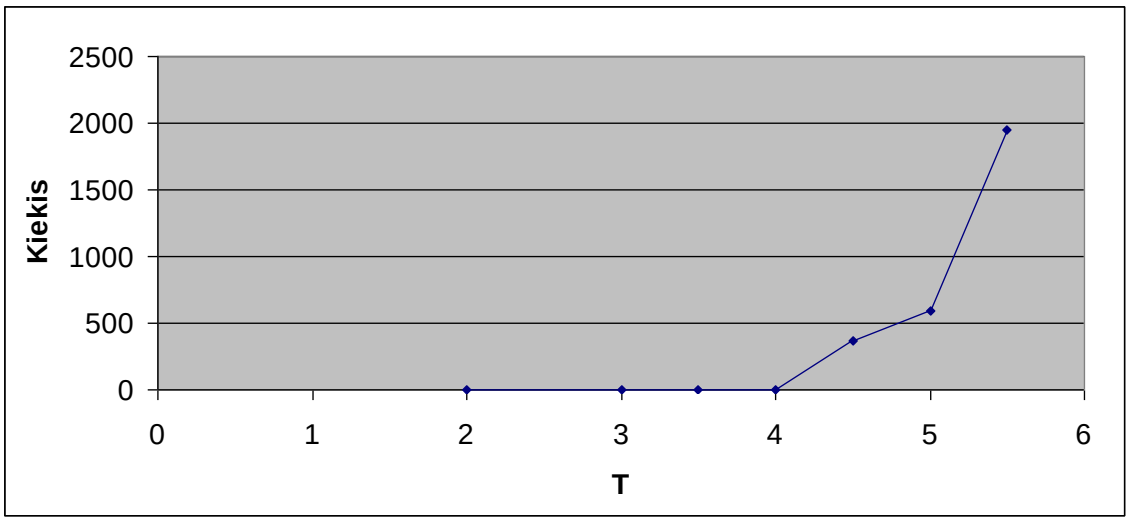
Simplekso metodas ekvivalentiškoms būsenoms ieškoti buvo panaudotas daug daugiau kartų nei lyginant du laiko momentus (28 pav.). Tai gerokai pablogina uždaviniui reikalingų resursų

panaudojimą, o ypač laiko sąnaudas. Atrodytų lyg nevertėtų ieškoti ekvivalenčių būsenų, tačiau prisiminus pirmą uždavinį, matome, kaip pagreitėja skaičiavimo trukmė suradus ekvivalentines būsenas.



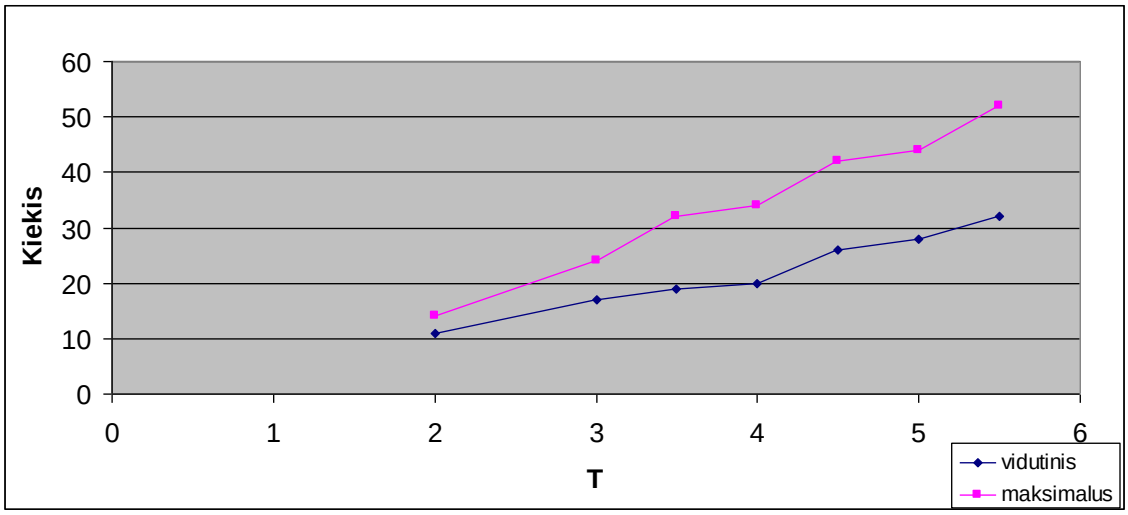
29 pav. Grafo viršūnių skaičius

Didinant vertinamus laiko momentų intervalus, matomas viršūnių skaičiaus mažėjimas lyginant sprendimo algoritmą su ekvivalenčiomis bei be ekvivalenčių būsenų (29 pav. ir 24 pav.). Tai yra dėl to, kad randamos ekvivalenčios būsenos, ir toliau šių viršūnių analizuoti nebereikia.



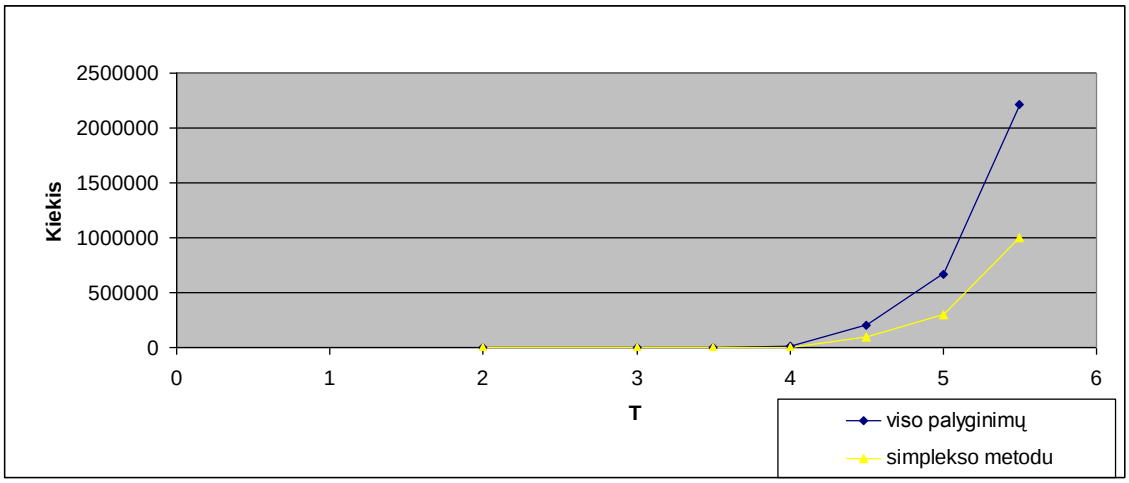
30 pav. Ekvivalenčių grafo viršūnių skaičius

Šiuo atveju didėja ir surastų ekvivalenčių būsenų kiekis (30 pav.). Tai leidžia daryti prielaidą, kad ekvivalenčių būsenų ieškojimo darbo laiko sąnaudos sprendžiant šį uždavinį, didinant laiko intervalus, duotų geresnių rezultatų.



31 pav. Apribojimų skaičius

Apribojimų skaičius padidėja nežymiai lyginant su sprendimu, kai nėra ieškoma ekvivalenčių būsenų (31 pav. ir 26 pav.). Galima daryti išvadą, kad apribojimų skaičius sprendžiant Simplekso metodu išliko vidutiniškai panašus. O tai leidžia daryti prielaidą, kad vieno Simplekso metodo sprendimo laikas išlieka toks pats.

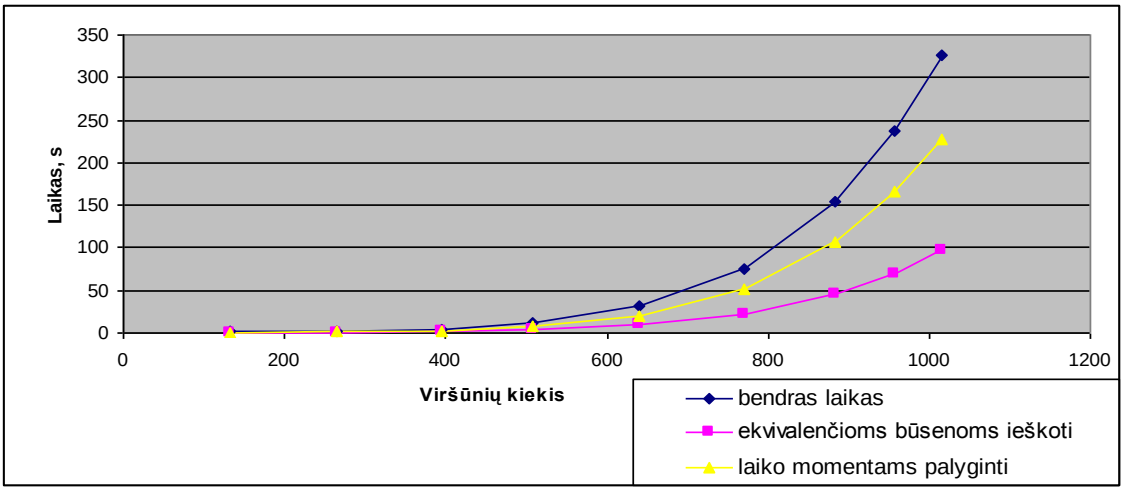


32 pav. Laiko momentų palyginimo skaičius

Laiko momentų palyginimų skaičius bei palyginimai naudojant Simplekso metodą, kai vertinami tie patys laiko intervalai ir ieškant ekvivalenčių būsenų (32 pav.), gerokai išaugo lyginant su algoritmu, kai nebuvo ieškomos ekvivalenčios būsenos (25 pav.). Imant intervalą iki 5,5 laiko momentų teko lyginti virš dviejų milijonų laiko momentų, kai tuo tarpu neieškant ekvivalenčių būsenų tai buvo atliekama per milijono kartų. Nagrinėjamu atveju, ieškant ekvivalenčių būsenų, reikia turėti naujai sugeneruotos būsenos $\min_l \alpha_l^i$, kad būtų galima įvertinti visų operacijų pradžios ir galo laiko momentų skirtumus $a - \min_l \alpha_l^i$. Skirtumas $a - \min_l \alpha_l^i$ reikalingas lyginant tolydinių komponenčių ekvivalentiškumą.

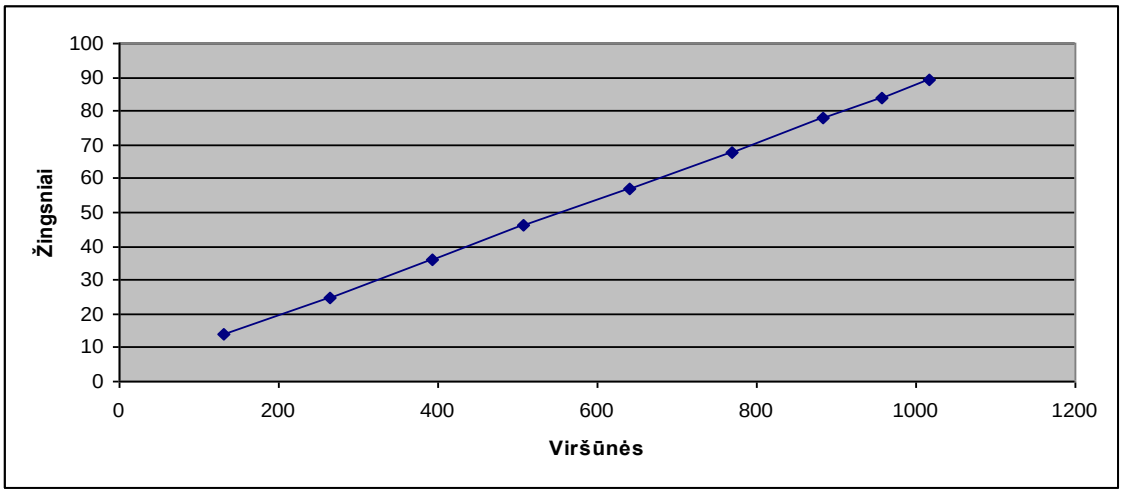
Sistemos įvertinimas, visada generuojant grafą viena šaka žemyn

Šiuo atveju nagrinėjamas toks uždavinio sprendimas, kai grafas generuojamas visada žemyn viena šaka, kol randama ekvivalenti būsena arba išanalizuojamas nurodytas viršūnių skaičius. Jei randama ekvivalenti būsena, grįžtama analizuoti kitos šakos viena viršūne aukštyn. Šis uždavinys buvo sprendžimas, siekiant išanalizuoti Simplekso metodo sprendimo laiką priklausomai nuo apribojimų aibės. Tyrimams buvo pasirinkta dvikanalė aptarnavimo sistema (antras atvejis), nes nagrinėjamu atveju ekvivalenčių būsenų pradiniuose intervaluose nėra.



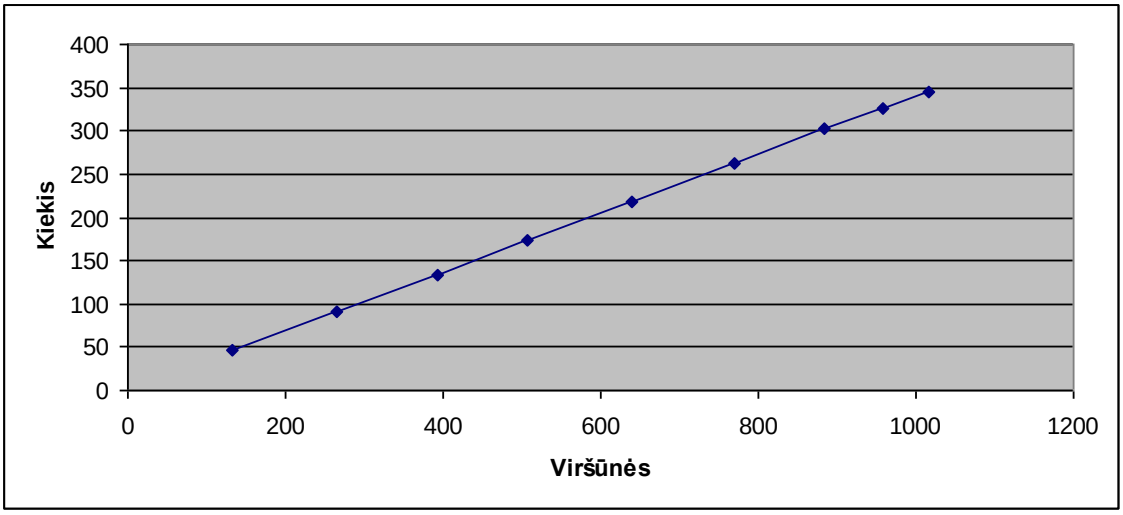
33 pav. Skaičiavimo trukmė priklausomai nuo išspręstų viršūnių skaičiaus

Generuojant grafą viena šaka žemyn, sprendimo laikas auga greičiau, nei viršūnių kiekis. Generuojamoje šakoje taipogi daugiausiai laiko užtrunka palyginti du laiko momentus, naudojant Simplekso metodą (33 pav.).



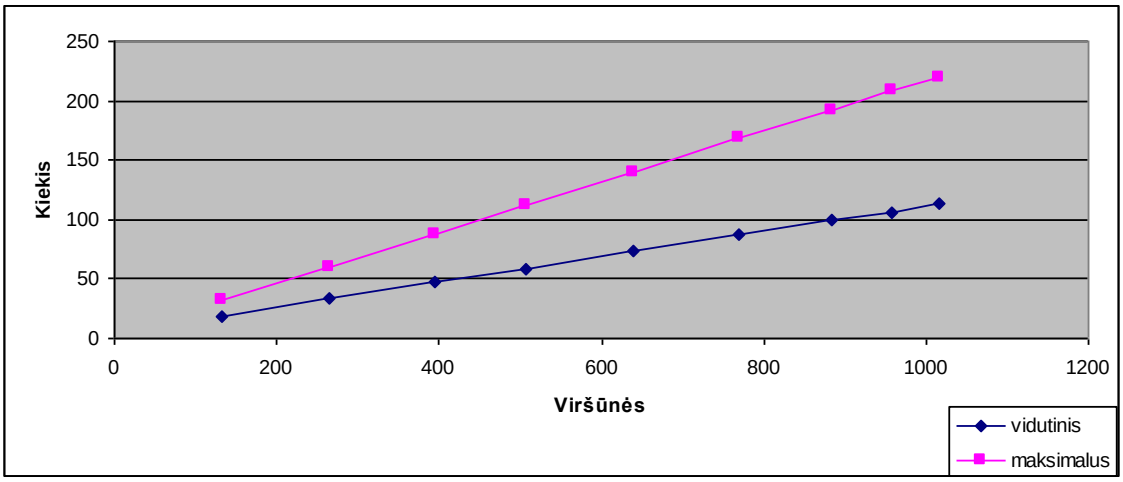
34 pav. Žingsnių skaičius

Žingsnių skaičius tiesiškai priklauso nuo išspęstų viršūnių skaičiaus (34 pav.). Nagrinėjamu atveju yra apie 9 žingsniai šimtui viršūnių.



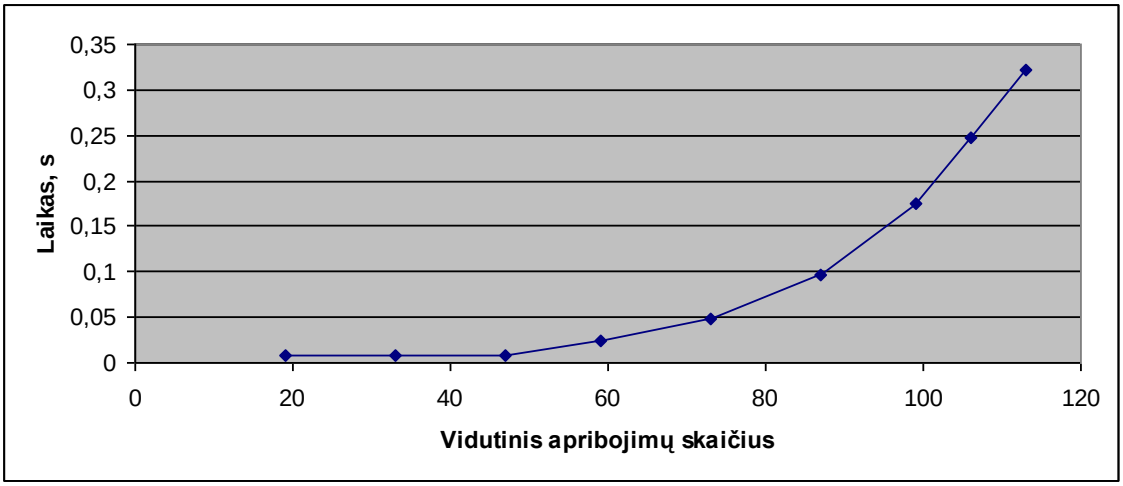
35 pav. Uždavinio sprendimas, kai naudojamas Simplekso metodas

Simplekso metodo panaudojimo kiekis ir išspręstų viršūnių kiekis taipogi kinta tiesiškai (35 pav.). Tai reiškia, kad didesnio sprendimo laiko Simplekso metodu neįtakoja palyginimų skaičius.



36 pav. Apribojimų skaičius sprendžiant Simplekso metodu

Maksimalus apribojimų skaičius sprendžiant Simplekso metodu auga daug greičiau nei vidutinis (36 pav.). Tai reiškia, kad sekančiam žingsnyje reikia spręsti Simplekso uždavinį su vis didesniu apribojimų skaičiumi.



37 pav. Vieno simplekso metodo sprendimo laikas

Simplekso optimizavimo metodo procedūros laikas auga eksponentiškai, didėjant laiko momentų apribojimų aibei (nėra konstanta), todėl naudoti Simplekso optimizavimo metodą įvykių įvykimo sritims nustatyti nėra gerai.

5.3 Lanksti gamybinė sistema

Lanksti gamybinė sistema susideda iš penkių aptarnaujančių paraiškas įrenginių *(A,B,C,D,E)*. *A* įrenginio paraiškos aptarnavimo trukmė $t_A \in (2;4)$; atitinkamai *B* - $t_B \in (5;7)$; *C* - $t_C \in (14;16)$; *D* - $t_D \in (9;11)$; *E* - $t_E \in (4;6)$. Nauja paraiška į sistemą patenka laiko intervalais $e \in (3;5)$. Visų aptarnavimo įrenginių darbo bei naujų paraiškų patekimo į sistemą tikimybių pasiskirstymas laiko intervaluose tolygus.

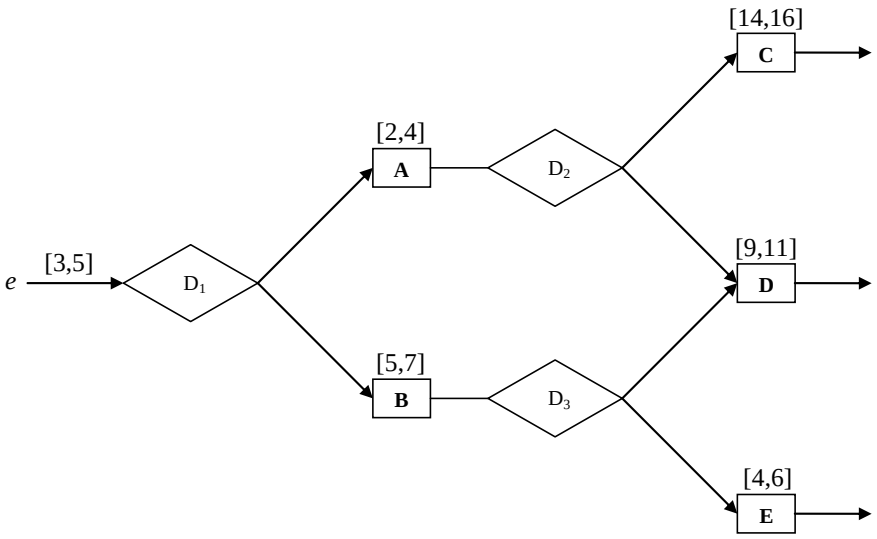
Patekusią į sistemą paraišką, gali aptarnauti du įrenginiai: *A* arba *B*. Paraiškų skirstymas įrenginiams atliekamas pagal du faktorius: įrenginių aptarnavimo spartą bei užimtumą. Jei sparčiau aptarnaujantis įrenginys *A* užimtas, paraiška aptarnaujam*a B* įrenginyje. Jei užimtas ir *B* įrenginys, paraiška neaptarnaujama ir atmetama.

Kai paraiška baigiama aptarnauti *A* įrenginyje, ji perduodama į *C* arba *D* įrenginius. Jei abu įrenginiai laisvi, perduodama į *D*, priešingu atveju - į *C*. Kai paraiška baigiama aptarnauti *B* įrenginyje, ji perduodama į *D* arba *E* įrenginius. Jei abu įrenginiai laisvi, perduodama į *D*, priešingu atveju - į *E*.

Jei *A* arba *B* įrenginys baigęs paraiškos aptarnavimą neturi kam jos atiduoti, įrenginys blokuojamas. Paraiška lieka jame iki tol, kol atsiranda laisvas kitas įrenginys, galintis pratęsti aptarnavimą. Tol, kol įrenginys blokuotas, naujos paraiškos aptarnavimas negali būti pradėtas. Eilių prieš įrenginius nėra.

C, D ir *E* įrenginiams baigus darbą, paraiška laikoma visiškai aptarnauta ir iš sistemos pašalinama.

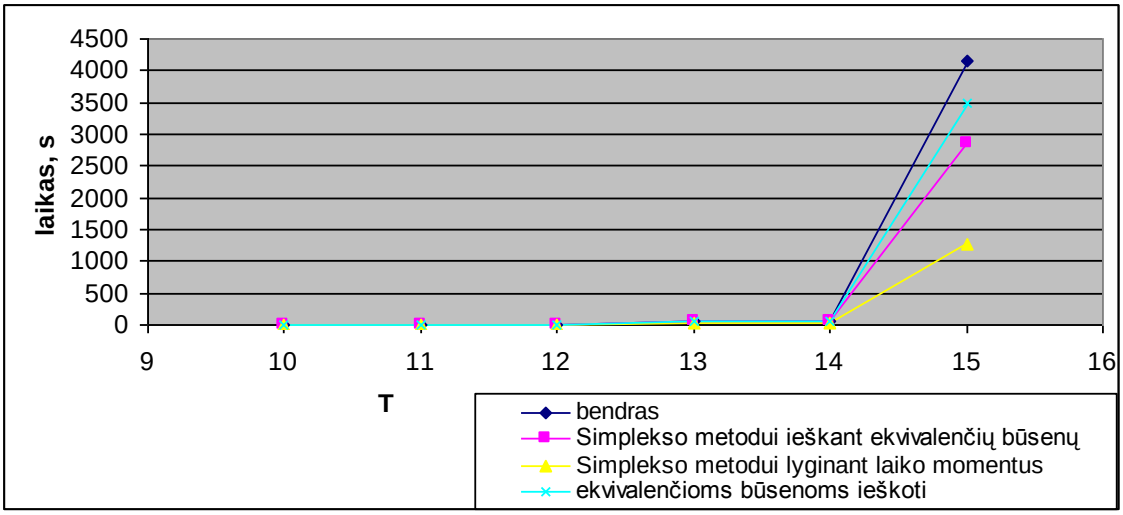
Lanksčios gamybinės sistemos schema pavaizduota 38 paveikslėlyje.



38 pav. Lanksčios gamybinės sistemos schema

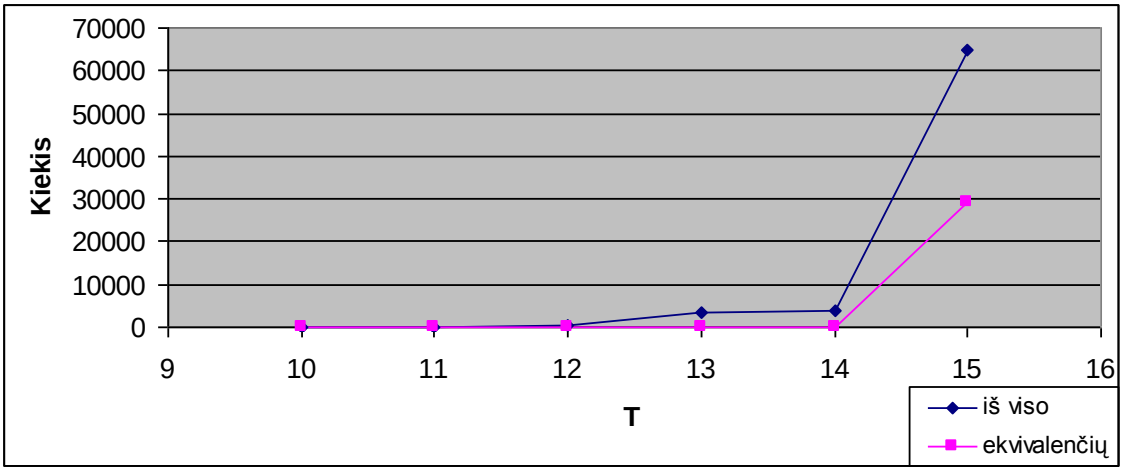
Sistemos įvertinimas, kai sistema analizuojama laiko intervaluose $(t_0, t_0 + T)$ *bei ieškomos ekvivalenčios būsenos*

1. *Resursų įvertinimai*



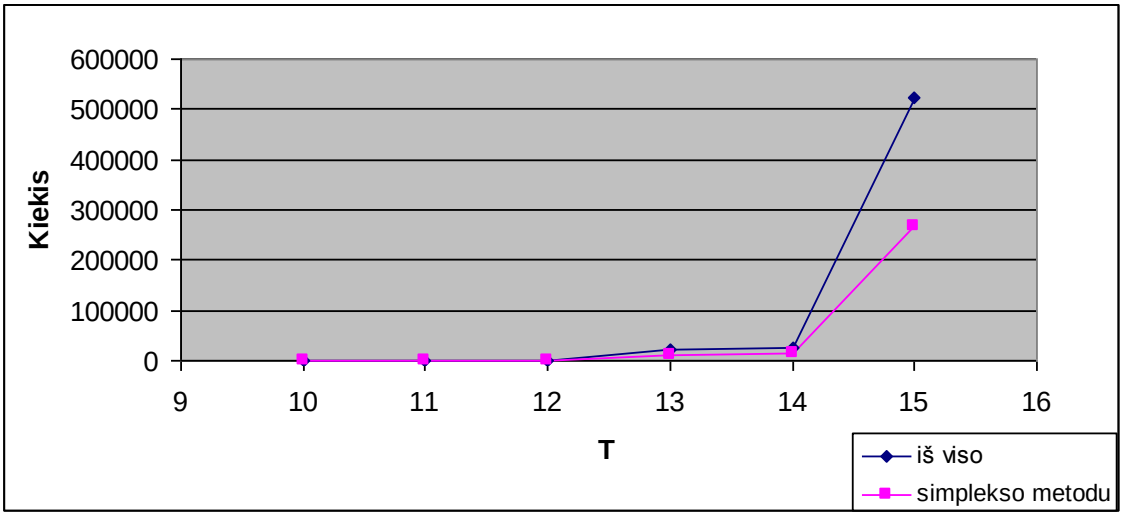
39 pav. Skaičiavimo trukmė laiko intervaluose

Skaičiavimo trukmė laiko intervale iki 15 laiko vienetų labai staigiai padidėja (39 pav.). Imant intervalą iki 14 laiko vienetų, skaičiavimo trukmė siekia apie vieną minutę, o skaičiuoti laiko intervalą iki 15 laiko vienetų jau užtrunka apie 1 valandą ir 8 min. Tokį rezultatą galima paaiškinti, nes pasiekus 14 laiko vienetų intervalą (pagal specifikaciją) aktyviomis gali tapti visos būsenos operacijos ir gali būti sugeneruota iki 60 perėjimų, kas neprieštarauja 1 teiginiui. Tai, vėlgi, galime daryti išvadą, kad uždavinio sprendimo laikas priklauso nuo pačio uždavinio specifikacijos. Plačiau, dėl ko didėja sprendimo laikas, panagrinėsime kituose pavyzdžiuose.



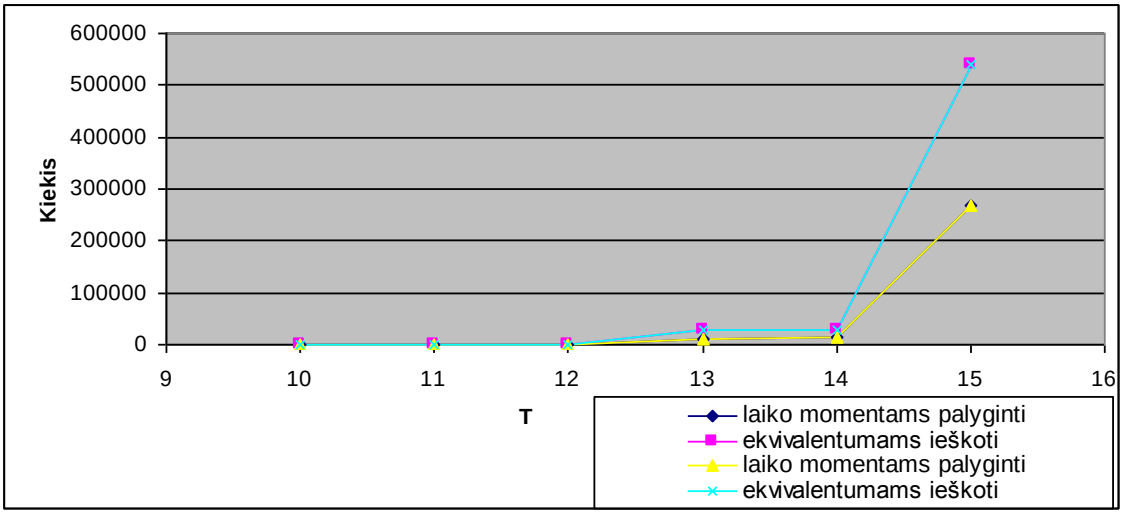
40 pav. Grafo viršūnių skaičius

Sugeneruotų viršūnių skaičius, nagrinėjant intervalą iki 15 laiko momentų, išauga labai daug kartų (40 pav.). Kaip jau buvo minėta, tai atsitinka dėl to, kad pagal specifikaciją nuo 14 vienetų gali aktyviomis tapti visos operacijos, kas įtakoja kur kas didesnį perėjimų skaičių. Akivaizdžiai matoma tendencija, kad beveik pusė iš surastų būsenų yra ekvivalenčios. Akivaizdu, kad sprendžiant uždavinį be ekvivalenčių būsenų išnagrinėti reiktų kur kas daugiau viršūnių.



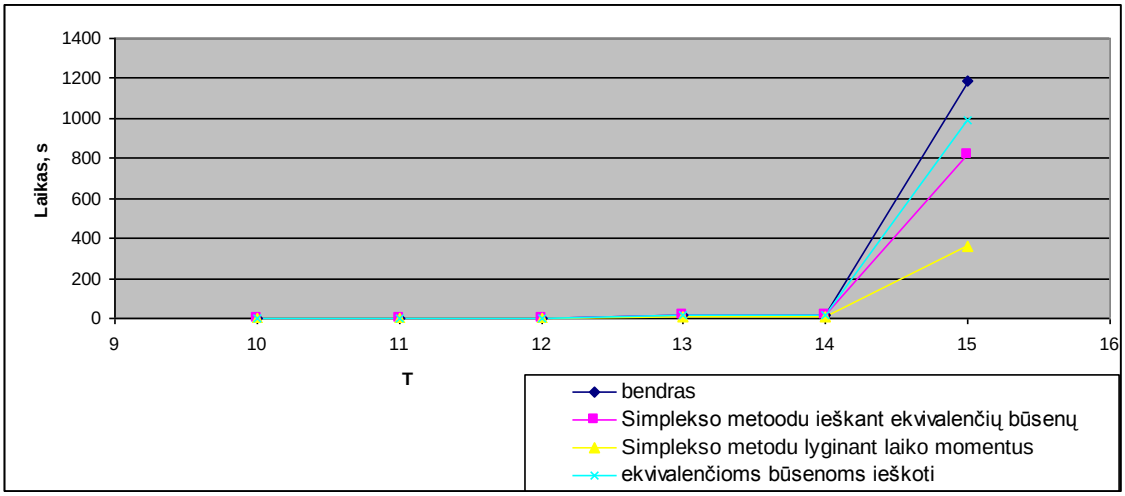
41 pav. Palyginimų skaičius

Pusei laiko momentų palyginimų (41 pav.) buvo panaudotas Simplekso metodas, kad palyginti laiko momentus intervaluose. Palyginimų santykis labai panašus, kaip ir antrajame uždavinyje. Išanalizavus intervalą iki 15 laiko momentų reikėjo atlikti virš 500000 laiko momentų palyginimų ir apie 250000 palyginimų buvo atlikta naudojant Simplekso metodą.



42 pav. Simplekso metodo panaudojimas

Analogiškai kaip ir nagrinėtuose uždaviniuose, Simplekso metodas buvo beveik dvigubai daugiau naudojamas ieškant ekvivalenčių būsenų nei dviejų laiko momentų palyginimams (42 pav.).



43 pav. Uždavinio sprendimo laikas naudojant 4 gijas

Kad įsitikintume uždavinio išlygiagretinimo galimybėmis, buvo realizuotas prototipas su 4 gijomis tam pačiam uždaviniui spręsti. Uždaviniui spręsti buvo naudojamas 4 branduolių kompiuteris. Buvo sprendžiamas gamybinės sistemos uždavinys ieškant ekvivalenčių būsenų. Kaip buvo tikėtasi, pastebėtas žymus laiko sumažėjimas (43 pav.) - net iki 3,5 karto. Jei naudojant vieną giją uždavinys buvo sprendžiamas beveik 1 valandą ir 10 min. (iki 15 laiko vienetų), tai, naudojant 4 gijas, tam pačiam uždaviniui išspręsti užteko 20 min.

6 DARBO REZULTATAI IR IŠVADOS

- Sukurti pasiekiamų būsenų grafo kompiuterinio realizavimo prototipai.
- Suformuluoti ir įrodyti teiginiai apie pasiekiamų būsenų grafo sudarymo algoritmo sudėtingumą:
    - Maksimalaus vienos būsenos galimų perėjimų skaičiaus.
    - Grafo viršūnių skaičiaus augimo priklausomybės nuo įvykių skaičiaus.
    - Maksimalaus įvykių skaičiaus elgsenoje per laiko tarpą T..
    - Maksimalaus viršūnių skaičiaus laiko intervale.
- Atlikti eksperimentai su realiojo laiko sistemomis, kurie įvertino pasiekiamų būsenų grafo sudarymo metodo panaudojimo efektyvumą ir parodė, kad:
    - uždavinio sprendimo laikas priklauso nuo agregatinėje specifikacijoje esančių operacijų trukmių;
    - naudoti Simplekso optimizavimo metodą įvykių įvykimo sritims nustatyti nėra gerai, nes didinant modeliavimo laiką Simplekso optimizavimo metodo procedūros laikas neišlieka konstanta, o auga eksponentiškai didėjant laiko momentų apribojimų aibei (šiuo atveju negalima taikyti Simplekso metodo apribojimų aibės mažinimo algoritmo).
- Algoritmas mažinantis apribojimų skaičių palyginant du laiko momentus [23] ne visais atvejais veikia korektiškai. Kai buvo minėta, sprendžiant dviejų laiko momentų palyginimo uždavinį „rankiniu“ būdu, dažniausiai dalis nelygybių nenaudojamos. Taigi reiktų ieškoti šio algoritmo modifikacijų, kurios leistų sumažinti Simplekso metodo apribojimų nelygybių skaičių ir tiktų visais atvejais.
- Nors programuoti aukšto lygio programavimo kalbomis, tokiomis kaip C#, Java, VB ar C++ valdomo kodo yra patogu, tačiau šios kalbos tik dalinai tinkamos realizuoti sistemas, kurios reikalauja didelių skaičiavimo resursų, nes neįmanoma tiesiogiai valdyti resursų, tokių kaip atmintis.
- Sukurti pasiekiamų būsenų grafo prototipai gali būti taikomi ir kitų sistemų specifikuotų agregatiniu metodu analizei, paliekant duomenų struktūrą aprašančias klases ir skaičiavimo logiką, keičiant tik būsenos struktūrą ir perėjimo operatorius.

LITERATŪRA

[1] Alur, R.; Henzinger T. A. A Really Temporal Logic. Symposium on Foundations of Computer Science. 1989, p. 164–169.

[2] Schneider S. Timed CSP: Theory and Practice. 1992, p. 640 – 675.

[3] Makackas, D. Agregatinio metodo taikymas realiojo laiko sistemų funkcionavimo charakteristikoms įvertinti: daktaro disertacijos darbas. KTU, informatikos fakultetas. [Kaunas], 2003.

[4] Iwasaki, Y., & Simon, H.A. Causality and model abstraction. Artificial Intelligence, Publisher: Elsevier Science Publishers Ltd., 1994.

[5] Awerill, M.; Kelton, D. Simulation Modeling and Analysis. Third Edition. New York, ISBN 0070-59-292-6. 2000, p. 17.

[6] Boccara, N. Modeling Complex Systems. New York, ISBN 0-387-40462-7. 2004, p. 38.

[7] Pranevičius, H. Sudėtingų sistemų formalizavimas ir analizė. Kaunas, 2008, p. 230

[8] Phillip A. Laplante. Real-Time Systems Design and Analysis.: Wiley-IEEE Press, 3 rd. ed. Hardcover 2004. 528 p. ISBN: 978-0-471-22855-4.

[9] Silberschatz A., Galvin P.B. Operating System Concepts.: Addison Weslwy Longman, Inc., 1998.

[10] Huang, J., Voeten, J., Florescu, O., et all. Predictability in real-time system development. [žiūrėta 2007-03-27] Prieiga per internetą: <http://www.esi.nl/projects/boderc/publications/of_book05.pdf>

[11] Kotonya, G. and Sommerville, I. Requirement Engineering: Processes and Techniques.: John Wiley & Sons, Inc., New York. 1998. p.

[12] Lamport, L., Lynch, N. Distributed computing: modes and methods. Handbook of theoretical computer science. 1990.

[13] Wikipedia, the free encyclopedia [žiūrėta 2006-02-17], prieiga per internetą: <http://en.wikipedia.org/wiki/Formal_methods>

[14] Dunstan, M., Kelsey, T., Linton, S. Lightweight formal methods for computer algebra systems. Germany. ISBN 1-58113-003-3. 1998.

[15] The Alloy Analyzer [žiūrėta 2007-01-27], prieiga per internetą: <http://alloy.mit.edu>

[16] Hewitt, C., Bishop, P., Steigei, R. A. Universal Modular Actor Formalism for Artificial Intelligence. 3rd. ed. International Joint Conference on Artificial Intelligence. Standford, California, 1973, p. 235-245.

[17] Agerholm, S., Larsen, P. A. Lightweight Approach to Formal Methods. Proceedings of the International Workshop on Current Trends in Applied Formal Method: Applied Formal Methods. ISBN 3-540-66462-9. London, 1998.

[18] Buslenko N., Kovalenko I. Lectures on comlex systems. Moscow, 1973.

[19] Pranevičius H., Makackas D. Laikinis pasiekiamų būsenų grafas alternuojančio bito protokolo agregatinei specifikacijai// Lietuvos mokslas ir pramonė, Informacinės technologijos – 98: konferencijos pranešimų medžiaga. Kaunas, 1998, p.291-293.

[20] Pranevičius H., Pilkauskas V., Chmieliauskas A. Aggregate approach for specification and analysis of computer network protocols. Kaunas, Technologija, 1994.

[21] Wang K., Pranevičius H. Applications of AI to Production Engineering. Kaunas: Technologija, 1997.

[22] Motus, L. Timing analysis of real-time software. Leidykla “Redwood Books”, 1994.

[23] Milinis, T. Realiojo laiko sistemų veiksenos įvertinimas: magistro darbas, KTU, informatikos fakultetas. [Kaunas], 2006.

SANTRAUKA ANGLŲ KALBA

Complexity analysis of reachable state graph creation
Summary

The work deals with a verification task of real time system specified by aggregate method. While solving the task, a technique for creation a reachable state graph is used. The technique permits to evaluate intervals of time when the defined system events occur.

Reachable state graph creation algorithms are analysed in the work. A data structure used in prototype software is presented in the work too. Assertions about a complexity of algorithm for reachable state graph creation are formulated and proved. These assertions concern maximum number of transitions from single state, dependency of number of graph verteles growth on a number of events, maximum number of events in a behaviour during time interval, and maximum number of vertexes during time interval.

Analysis of automated creation of graphs for three test systems is presented. It is shown that Simplex optimisation procedure for comparison of time intervals can be used only in certain cases.

PRIEDAI

1 priedas. Agregatinės specifikacijos

Dvikanalės aptarnavimo sistemos agregatinė specifikacija

1. Specifikacija sudaryta iš vieno agregato, todėl įėjimo ir išėjimo signalų bei išorinių įvykių aibės yra tuščios, t. y. $X=\varnothing$, $E'=\varnothing$, $Y=\varnothing$.
2. Vidinių įvykių aibė $E''=\{e_1,e_2,e_3\}$, čia e_1 – nauja paraiška, e_2 – baigė aptarnavimą pirmas kanalas, e_3 – baigė aptarnavimą antras kanalas.
3. Valdančios sekos: $e_1 \mapsto \alpha_0,\alpha_1,\alpha_2\ldots$, $e_2 \mapsto \beta_0,\beta_1,\beta_2\ldots$, $e_3 \mapsto \gamma_0,\gamma_1,\gamma_2\ldots$.
4. Diskretinė agregato būsena: $\nu(t)=(n(t))$, čia $n(t)$ – laukiančių aptarnavimo paraiškų skaičius.
5. Tolydinę būseną sudaro trys komponentės: $z_\nu(t)=(w(e_1,t),w(e_2,t),w(e_3,t))$, čia $w(e_1,t)$ – laiko momentas, kada atsiras nauja paraiška sistemoje, kuris atitinka operaciją O_1; $w(e_2,t)$ – laiko momentas, kada bus baigtas aptarnavimas pirmajame kanale; $w(e_3,t)$ – laiko momentas, kada bus baigtas aptarnavimas antrajame kanale.
6. Pradinė būsena: $n(t_0)=0$, $w(e_1,t_0)=t_0+\alpha_0$, $w(e_2,t_0)=t_0$, $w(e_3,t_0)=t_0$.
7. Parametrai: s – maksimalus eilės ilgis.

$H(e_1)$:

$$n(t_m)=\begin{cases} n(t_m-0)+1, & t_m<w(e_2,t_m-0)\wedge t_m<w(e_3,t_m-0)\wedge n(t_m-0)<s;\\ 0, & \text{kitu atveju;}\end{cases}$$

$$w(e_1,t_m)=t_m+\alpha_m;$$

$$w(e_2,t_m)=\begin{cases} t_m+\beta_m, & t_m>w(e_2,t_m-0);\\ w(e_2,t_m-0), & \text{kitu atveju;}\end{cases}$$

$$w(e_3,t_m)=\begin{cases} t_m+\gamma_m, & t_m<w(e_2,t_m-0)\wedge t_m>w(e_3,t_m-0);\\ w(e_3,t_m-0), & \text{kitu atveju;}\end{cases}$$

$H(e_2)$:

$$n(t_m)=\begin{cases} n(t_m-0)-1, & n(t_m-0)>0;\\ 0, & \text{kitu atveju;}\end{cases}$$

$$w(e_2,t_m)=\begin{cases} t_m+\beta_m, & n(t_m-0)>0;\\ w(e_2,t_m-0), & \text{kitu atveju;}\end{cases}$$

$H(e_3)$:

$$n(t_m)=\begin{cases} n(t_m-0)-1, & n(t_m-0)>0;\\ 0, & \text{kitu atveju;}\end{cases}$$

$$w(e_3,t_m)=\begin{cases} t_m+\gamma_m, & n(t_m-0)>0;\\ w(e_3,t_m-0), & \text{kitu atveju;}\end{cases}$$

Lanksčios gamybinės sistemos agregatinė specifikacija

1. Įėjimo signalų aibė: $X = \varnothing$.
2. Išėjimo signalų aibė: $Y = \varnothing$.
3. Išorinių įvykių aibė: $E' = \varnothing$.
4. Vidinių įvykių aibė: $E'' = \{e, e_A, e_B, e_C, e_D, e_E\}$,
   kur e – naujos paraiškos sistemoje atsiradimas;
   e_i – įrenginys i baigė paraiškos aptarnavimą.
5. Valdančios sekos:
   $e \mapsto S_0, \ldots, S_n, \ldots$, $e_A \mapsto S_0^A, \ldots, S_n^A, \ldots$, $e_B \mapsto S_0^B, \ldots, S_n^B, \ldots$, $e_C \mapsto S_0^C, \ldots, S_n^C, \ldots$,
   $e_D \mapsto S_0^D, \ldots, S_n^D, \ldots$, $e_E \mapsto S_0^E, \ldots, S_n^E, \ldots$.
   kur $S_i \in [3,5]$ – laiko tarpas, tarp naujų paraiškų atsiradimo sistemoje;
   $S_i^A \in [2,4]$ – A įrenginio paraiškos aptarnavimo laikas;
   $S_i^B \in [5,7]$ – B įrenginio paraiškos aptarnavimo laikas;
   $S_i^C \in [14,16]$ – C įrenginio paraiškos aptarnavimo laikas;
   $S_i^D \in [9,11]$ – D įrenginio paraiškos aptarnavimo laikas;
   $S_i^E \in [4,6]$ – E įrenginio paraiškos aptarnavimo laikas.
6. Diskreti būsenos dedamoji: $\nu(t) = \{n_A(t), n_B(t), n_C(t), n_D(t), n_E(t), b_A(t), b_B(t)\}$
   kur $n_i(t)$ – i įrenginio eilėje laukiančių paraiškų skaičius;
   $b_A(t) = \begin{cases} 1, \text{ kai A įrenginys blokuotas,} \\ 0, \text{ kai A įrenginys neblokuotas;} \end{cases}$
7. Tolydinė būsenos dedamoji: $z_\nu(t) = \{\omega(e,t), \omega(e_A,t), \omega(e_B,t), \omega(e_C,t), \omega(e_D,t), \omega(e_E,t)\}$
   kur $\omega(e,t)$ – laiko momentas, kada atvyks paraiška;
   $\omega(e_i,t)$ – laiko momentas, kada i įrenginys baigs paraiškos aptarnavimą.
8. Parametrai:
   l_i – i įrenginio eilės maksimalus ilgis;
9. Pradinė būsena: $z(t_0) = \{0,0,0,0,0,0,0, t_0 + S_0, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing\}$

Perėjimo ir išėjimo operatoriai:
Nauja paraiška pateko į sistemą
$H(e)$:

$$\omega(e, t_m) = t_m + S_m,$$

$$n_A(t_m) = \begin{cases} n_A(t_m) + 1, & (\omega(\boldsymbol{e_A}, t_m) > t_m \vee b_A(t_m - 0) = 1) \wedge (\omega(e_B, t_m) > t_m \vee b_B(t_m - 0) = 1) \wedge \\ & \wedge\, n_A(t_m) < l_A, \\ n_A(t_m - 0), & \text{kitu atveju,} \end{cases}$$

$$n_B(t_m) = \begin{cases} n_B(t_m) + 1, & (\omega(e_A, t_m) > t_m \vee b_A(t_m - 0) = 1) \wedge (\omega(e_B, t_m) > t_m \vee b_B(t_m - 0) = 1) \wedge \\ & \wedge\, n_A(t_m) = l_A \wedge n_B(t_m) < l_B, \\ n_B(t_m - 0), & \text{kitu atveju,} \end{cases}$$

$$n_C(t_m) = n_C(t_m - 0),\ n_D(t_m) = n_D(t_m - 0),\ n_E(t_m) = n_E(t_m - 0),$$

$$\omega(e_A, t_m) = \begin{cases} t_m + S_m^A, & \omega(e_A, t_m - 0) < t_m \wedge b_A(t_m - 0) = 0, \\ \omega(e_A, t_m - 0), & \text{kitu atveju,} \end{cases}$$

$$\omega(e_B,t_m)=\begin{cases} t_m+S_m^B, & (\omega(e_B,t_m-0)<t_m \wedge b_B(t_m-0)=0)\wedge \\ & \wedge(\omega(e_A,t_m-0)>t_m \vee b_A(t_m-0)=1), \\ \omega(e_B,t_m-0), & \text{kitu atveju.} \end{cases}$$

Įrenginys A baigė paraiškos aptarnavimą:

$H(e_A)$:

$$\omega(e_A,t_m)=\begin{cases} t_m+S_m^A, & n_C(t_m-0)<l_C \vee n_D(t_m-0)<l_D \wedge n_A(t_m-0)>0, \\ \omega(e_A,t_m-0), & \text{kitu atveju,} \end{cases}$$

$$b_A(t_m)=\begin{cases} 1, & n_C(t_m-0)=l_c \wedge n_D(t_m-0)=l_D \wedge \omega(e_C,t_m)>t_m \wedge \omega(e_D,t_m)>t_m \\ 0, & \text{kitu atveju,} \end{cases}$$

$$n_A(t_m)=\begin{cases} n_A(t_m-0)-1, & n_C(t_m-0)<l_C \vee n_D(t_m-0)<l_D \wedge n_A(t_m-0)>0, \\ n_A(t_m-0), & \text{kitu atveju,} \end{cases}$$

$$n_C(t_m)=\begin{cases} n_C(t_m)+1, & \omega(e_D,t_m)>t_m \wedge \omega(e_C,t_m)>t_m \wedge n_D(t_m)=l_D \wedge n_C(t_m)<l_C, \\ n_C(t_m-0), & \text{kitu atveju,} \end{cases}$$

$$n_D(t_m)=\begin{cases} n_D(t_m)+1, & \omega(e_D,t_m)>t_m \wedge \omega(e_C,t_m)>t_m \wedge n_D(t_m)<l_D, \\ n_D(t_m-0), & \text{kitu atveju,} \end{cases}$$

$$\omega(e_C,t_m)=\begin{cases} t_m+S_m^C, & \omega(e_C,t_m-0)<t_m \wedge \omega(e_D,t_m-0)>t_m, \\ \omega(e_C,t_m-0), & \text{kitu atveju,} \end{cases}$$

$$\omega(e_D,t_m)=\begin{cases} t_m+S_m^D, & \omega(e_D,t_m-0)<t_m, \\ \omega(e_D,t_m-0), & \text{kitu atveju.} \end{cases}$$

Įrenginys B baigė paraiškos aptarnavimą:

$H(e_B)$:

$$\omega(e_B,t_m)=\begin{cases} t_m+S_m^B, & n_D(t_m-0)<l_D \vee n_E(t_m-0)<l_E, \\ \omega(e_B,t_m-0), & \text{kitu atveju,} \end{cases}$$

$$b_B(t_m)=\begin{cases} 1, & n_D(t_m-0)=l_D \wedge n_E(t_m-0)=l_E \wedge \omega(e_D'',t_m)>t_m \wedge \omega(e_E'',t_m)>t_m \\ 0, & \text{kitu atveju,} \end{cases}$$

$$n_D(t_m)=\begin{cases} n_D(t_m)+1, & \omega(e_E,t_m)>t_m \wedge \omega(e_D,t_m)>t_m \wedge n_E(t_m)=l_E \wedge n_D(t_m)<l_D, \\ n_D(t_m-0), & \text{kitu atveju,} \end{cases}$$

$$n_E(t_m)=\begin{cases} n_E(t_m)+1, & \omega(e_E,t_m)>t_m \wedge \omega(e_D,t_m)>t_m \wedge n_E(t_m)<l_E, \\ n_E(t_m-0), & \text{kitu atveju,} \end{cases}$$

$$\omega(e_D,t_m)=\begin{cases} t_m+S_m^D, & \omega(e_D,t_m-0)<t_m \wedge \omega(e_E,t_m-0)>t_m, \\ \omega(e_D,t_m-0), & \text{kitu atveju,} \end{cases}$$

$$\omega(e_E,t_m)=\begin{cases} t_m+S_m^E, & \omega(e_E,t_m-0)<t_m, \\ \omega(e_E,t_m-0), & \text{kitu atveju.} \end{cases}$$

Įrenginys C baigė paraiškos aptarnavimą:

$H(e_C)$:

$$n_C(t_m)=\begin{cases} n_C(t_m)-1, & \omega(e_C,t_m-0)<t_m, \\ n_C(t_m-0), & \text{kitu atveju,} \end{cases}$$

$$\omega(e_C,t_m)=\begin{cases} t_m+S_m^C, & n(t_m-0)>0, \\ \omega(e_C,t_m-0), & \text{kitu atveju,} \end{cases}$$

$b_A(t_m)=0$,

$$n_A(t_m)=\begin{cases} n_A(t_m)-1, & b_A(t_m-0)=1\wedge n_A(t_m)>0, \\ n_A(t_m-0), & \text{kitu atveju,} \end{cases}$$

$$\omega(e_A,t_m)=\begin{cases} t_m+S_m^A, & n_A(t_m)>0\wedge b_A(t_m-0)=1, \\ \omega(e_A,t_m-0), & \text{kitu atveju.} \end{cases}$$

Įrenginys E baigė paraiškos aptarnavimą:

$H(e_E)$:

$$n_E(t_m)=\begin{cases} n_E(t_m)-1, & \omega(e_E,t_m-0)<t_m, \\ n_E(t_m-0), & \text{kitu atveju,} \end{cases}$$

$$\omega(e_E,t_m)=\begin{cases} t_m+S_m^E, & n(t_m-0)>0, \\ \omega(e_E,t_m-0), & \text{kitu atveju,} \end{cases}$$

$b_B(t_m)=0$,

$$n_B(t_m)=\begin{cases} n_B(t_m)-1, & b_B(t_m-0)=1\wedge n_B(t_m)>0, \\ n_B(t_m-0), & \text{kitu atveju,} \end{cases}$$

$$\omega(e_B,t_m)=\begin{cases} t_m+S_m^B, & n_B(t_m)>0\wedge b_B(t_m-0)=1, \\ \omega(e_B,t_m-0), & \text{kitu atveju.} \end{cases}$$

Įrenginys D baigė paraiškos aptarnavimą:

$H(e_D)$:

$$n_D(t_m)=\begin{cases} n_D(t_m)-1, & \omega(e_D,t_m-0)<t_m, \\ n_D(t_m-0), & \text{kitu atveju,} \end{cases}$$

$$\omega(e_D,t_m)=\begin{cases} t_m+S_m^D, & n(t_m-0)>0, \\ \omega(e_D,t_m-0), & \text{kitu atveju,} \end{cases}$$

$b_B(t_m)=0$,

$$n_A(t_m)=\begin{cases} n_A(t_m)-1, & b_A(t_m-0)=1\wedge n_A(t_m)>0, \\ n_A(t_m-0), & \text{kitu atveju,} \end{cases}$$

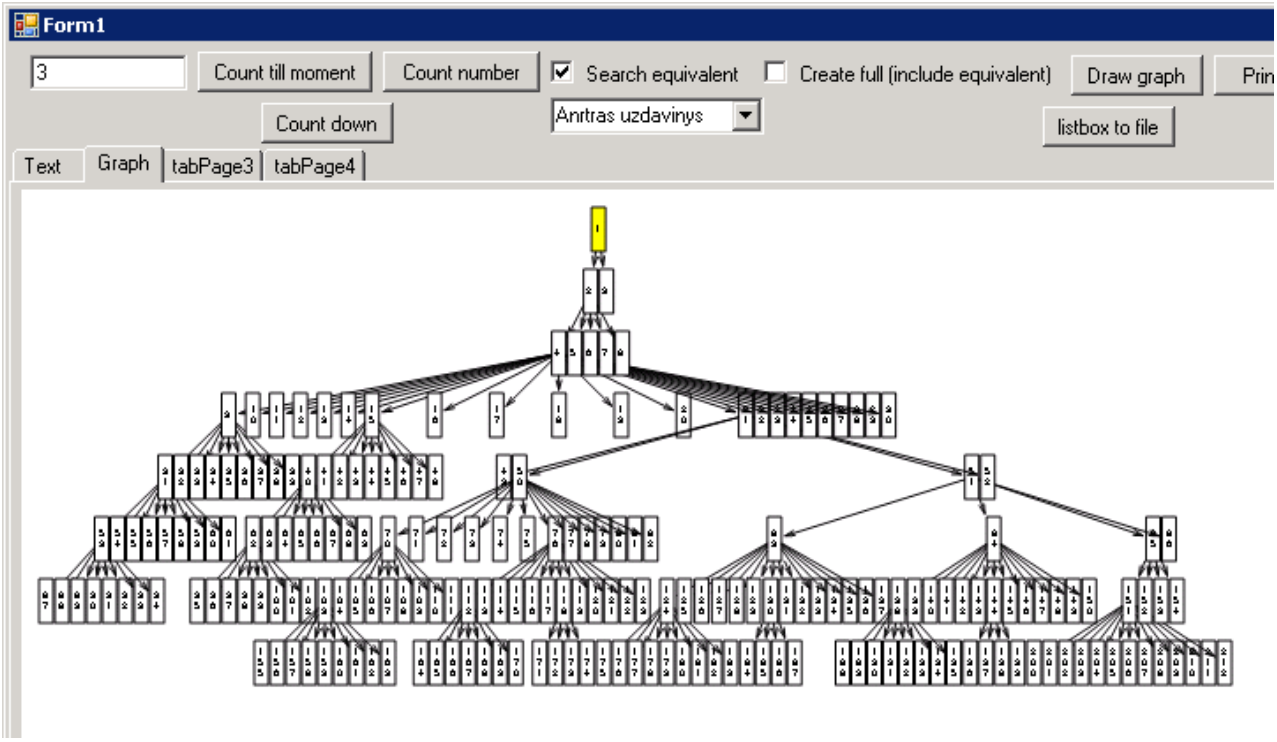
$$\omega(e_A,t_m)=\begin{cases} t_m+S_m^A, & n_A(t_m)>0\wedge b_A(t_m-0)=1, \\ \omega(e_A,t_m-0), & \text{kitu atveju,} \end{cases}$$

$$n_B(t_m)=\begin{cases} n_B(t_m)-1, & b_B(t_m-0)=1\wedge n_B(t_m)>0, \\ n_B(t_m-0), & \text{kitu atveju,} \end{cases}$$

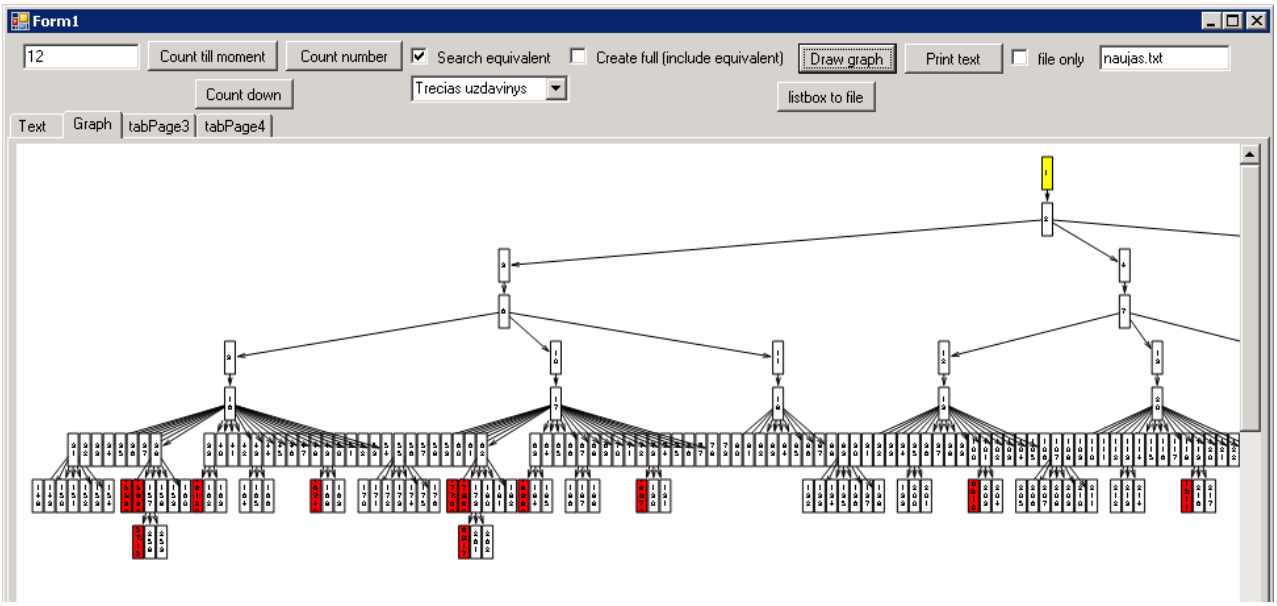
$$\omega(e_B,t_m)=\begin{cases} t_m+S_m^B, & n_B(t_m)>0\wedge b_B(t_m-0)=1, \\ \omega(e_B,t_m-0), & \text{kitu atveju.} \end{cases}$$

2 priedas. Realizacijos pavyzdžiai

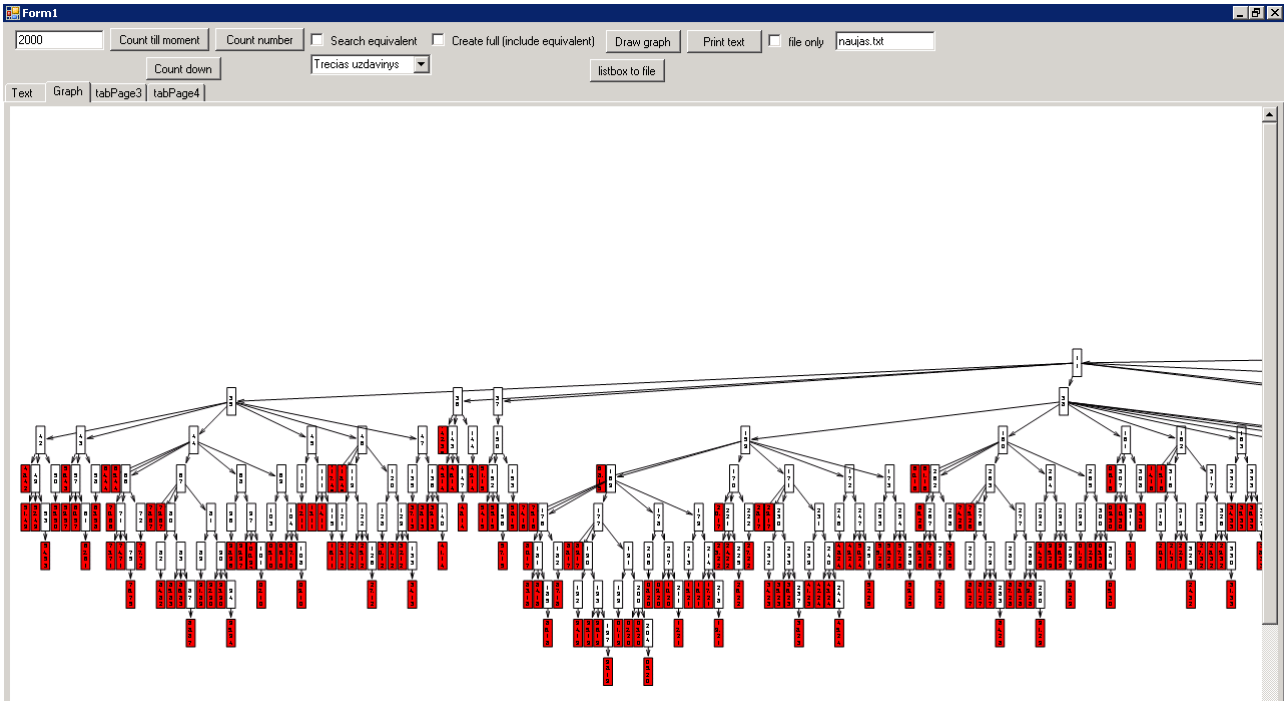
1 pavyzdys. Analizuojama dvikanalė aptarnavimo sistema (antras atvejis) ir nagrinėjamas laiko intervalas iki 3 laiko momentų. Buvo ieškoma ekvivalenčių būsenų, kurių, šiuo atveju, surasti nepavyko. Kaip analizės rezultatas pateiktas grafas.



2 pavyzdys. Analizuojama gamybinė sistema ir nagrinėjamas laiko intervalas iki 12 laiko momentų. Sprendžiant uždavinį ieškota ekvivalenčių būsenų, kurios paveikslėlyje pažymėtos raudonai. Kaip analizės rezultatas pateiktas grafas.



3 pavyzdys. Analizuojama gamybinė sistema ir sprendžiamas uždavinys, kai ieškoma ekvivalenčių būsenų viena šaka žemyn. Galima pastebėti, kad visais atvejais yra randamos ekvivalenčios būsenos.



4 pavyzdys. Analizuojama dvikanalė aptarnavimo sistema (pirmas atvejis) ir nagrinėjamas laiko intervalas iki 23 laiko momentų. Ekvivalenčių būsenų nėra ieškoma. Kaip analizės rezultatas pateiktas grafas. Grafe pastebimas šakų pasikartojimas.