KAUNO TECHNOLOGIJOS UNIVERSITETAS

MATEMATIKOS IR GAMTOS MOKSLŲ FAKULTETAS

MATEMATINIO MODELIAVIMO KATEDRA

Mindaugas Bražėnas

APROKSIMAVIMAS FAZINIAIS SKIRSTINIAIS BEI JŲ TAIKYMAS APTARNAVIMO SISTEMOMS MODELIUOTI

Magistro darbas

Vadovas
prof.dr.E. Valakevičius

KAUNAS, 2014

KAUNO TECHNOLOGIJOS UNIVERSITETAS

MATEMATIKOS IR GAMTOS MOKSLŲ FAKULTETAS

MATEMATINIO MODELIAVIMO KATEDRA

TVIRTINU
Katedros vedėjas
prof.dr. E.Valakevičius
2012 06 02

APROKSIMAVIMAS FAZINIAIS SKIRSTINIAIS BEI JŲ TAIKYMAS APTARNAVIMO SISTEMOMS MODELIUOTI

Taikomosios matematikos magistro baigiamasis darbas

Vadovas
prof.dr.E. Valakevičius
2014 0601

Recenzentas
dr. V. Pilkauskas
2014 06 01

Atliko
FMMM-2 gr. stud.
M. Bražėnas
2014 05 30

KAUNAS, 2014

KVALIFIKACINĖ KOMISIJA

Pirmininkas: Juozas Augutis, profesorius (VDU)

Sekretorius: Eimutis Valakevičius, profesorius (KTU)

Nariai: Arūnas Barauskas, dr., direktoriaus pavaduotojas (UAB „Danet Baltic“)

Vytautas Janilionis, docentas (KTU)

Zenonas Navickas, profesorius (KTU)

Kristina Šutienė, docentė (KTU)

Jonas Valantinas, profesorius (KTU)

Approximation by phase-type distributions and their usage in modeling queueing systems / supervisor prof. dr. E.Valakevičius; Department of Mathematical Modelling, Faculty of Mathematics and Natural Sciencies, Kaunas University of Technology. – Kaunas, 2014. –82p.

SUMMARY

The method of modelling nonmarkovian systems using phase –type distributions is proposed. If the duration of time between two events in a system has the exponential distribution then the performance of the system can be described bythe continuous time Markov chain. However, in practice this condition is not satisfied. The direct modelling of such systems by continuous time Markov chains is impossible. In such cases an arbitrary continuous distribution of a positive random variable can be approximated with a sequence and mixture of exponential distributions which result in phase-type distribution.Consequently, the originalnonmarkovian system can by approximated by markovian one.The process of constructing models of some queuing systems withproposed method ispresented.

The following tasks and problems were solved in this work. First of all, an algorithm for generating continuous time Markov chains of a specific system was developed and tested. The problem of approximation of a general type distributionfunction by the phase-type distribution was investigated. Several optimization algorithms for estimating parameters of phase-type distribution were compared. Also, a research on how approximation accuracy depends on the form of density function was carried out. In order to reduce computation time,innovative technologies were applied. All calculations were performed by programs written in Java and C++ programming languages.

The literature of this topic was reviewed. It was noticed that many researchers investigatethe approximation of ageneral distribution function by a phase-type distributionswith a specific structure. In this worka general structure of phase-type distribution was used and a new approach to estimate the optimalparameters of phase-type distribution with the optimization algorithm of local unimodal sampling was proposed.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLĖLIŲ SĄRAŠAS

ĮVADAS

Pristatome ne Markovo tipo sistemų modeliavimas panaudojant fazinius skirstinius.Sistemą, kurioje laiko tarpai tarp betkurių dviejų įvykių turi eksponentinį skirstinį, galima modeliuoti tolydaus laiko Markovo grandine. Jeigu šis reikalavimas nėra išpildomas –sistemos tiesioginis modeliavimas tolydaus laiko Markovo grandine yra negalimas. Tokiais atvejais teigiamų atsitiktinių dydžių skirstiniai yra aproksimuojami ekspenentinių skirstinių mišiniais ir sąsūkomis – faziniais skirstiniais. Tokiu būdu, originali ne Markovo tipo sistema gali būti aproksimuojama kita Markovo tipo sistema.Darbe parodoma kaip galima modeliuoti paprastas aptarnavimo sistemas.

Darbe buvo sprendžiami įvairūs uždaviniai. Visų pirma, buvo sukurtas algoritmas, kurio pagalba sukonstruojama tolydaus laiko Markovo grandinė pasirinktai sistemai. Nagrinėta skirstinių aproksimavimo faziniais skirstiniais problema. Buvo palyginti keli paieškos algoritmai fazinio skirstinio optimaliems parametrams surasti.Stebėta kaip aproksimavimo faziniais skirstiniais tikslumas priklauso nuo aproksimuojamo skirsntio tankio funkcijos formos. Atliekant tyrimus neapsieita be inovatyvių technologijų taikymo, kurių dėka pavyko ženkliai sutrumpinti skaičiavimo trukmę.Tyrimams atlikti buvo parašytos programos (Java ir C++ kalbomis).

Apžvelgta su šia tema susijusi literatūrą. Pastebėta, kad daugeliu atvejų tiriamas skirstnių aproksimavimas tam tikros struktūros faziniais skirstiniais. Šiame darbe skirstiniai buvo aproksimuojami naudojant bendrąją fazinio skirstinio struktūrą, kurios parametrai surandami lokalia paieška pagrįsto metodo pagalba.

Darbą sudaro trys dalys.

- Bendrojoje dalyje trumpai pateikiama informacija apie tiriamus paieškos metodus, tolydaus laiko markovo grandines, fazinius skirstinius, aptarnavimo sistemas.
- Tiriamojoje dalyje atliekamas fazinių skirstinių parametrų paieškos metodų palyginimas, sukonstruojamos ir analizuojamos įvairių aptarnavimo sistemų markovo proceso grandinės.

Darbo pabaigoje pateikiamos gautų tyrimo rezultatų išvados.

1 BENDROJI DALIS

1.1 Paieškos metodai

Šiame skyriuje aptarsime tris paieškos metodus: atsitiktinės paieškos metodą (RND), dalelių spiečiaus metodą (PSO) bei lokalios paieškos metodą (LUS).

Paieškos metodo tikslas yra surasti tokias kintamųjų vektoriaus $\boldsymbol{x} = [x_1, x_2, \dots, x_n]$, $x_i \in \boldsymbol{R}$, $i = \overline{1,n}$ reikšmes su kuriomis tikslo funkcija $g(\boldsymbol{x})$ įgytų kiek galima mažesnę reikšmę. Kintamųjų reikšmėms taikomi apribojimai:

$$l_i \le x_i \le u_i, i = \overline{1,n}$$

čia: l_i, u_i – yra atitinkamai *i*-tojo kintamojo apatinė ir viršutinė ribos.

Pažymėjimai, kurie bus naudojami toliau aprašomuose paieškos algoritmuose:

- $\boldsymbol{x}^*$ - geriausias surastas sprendinys
- g^*- geriausio surasto sprendinio tikslo funkcijos reikšmė ($g^* = g(\boldsymbol{x}^*)$)
- N – dalelių kiekis
- Ic – iteracijų kiekis

1.1.1 Atsitiktinės paieškos metodas (RND)

Tai vienas iš pačių paprasčiausių paieškos metodų.

Algoritmas:

1. Nustatyti pradinę tikslo funkcijos reikšmę: $g^* \coloneqq \infty$.
2. Nustatyti pradinės iteracijos numerį:$k \coloneqq 1$.
3. Atsitiktinai sugeneruoti N vektorių:

$$\boldsymbol{x}_j, \{\boldsymbol{x}_j\}_i \sim U(l_i, u_i), i = \overline{1,n}, j = \overline{1,N}$$

4. Apskaičiuoti sugeneruotų vektorių tikslo funkcijų reikšmes:

$$g_j \coloneqq g(\boldsymbol{x}_j), j = \overline{1,N}$$

5. Atnaujinti geriausią sprendinį $\boldsymbol{x}^*$bei jo tikslo funkcijos reikšmę:

```
for j from 1 to N
if g_j < g* then
        x* := x_j, g* := g_j
endif
```

6. Patikrinti ar atliktos visos iteracijos: jeigu $k < Ic$, $k \coloneqq k + 1$ ir grįžti į 3 žingsnį.
7. Algoritmo pabaiga. Geriausias surastas sprendinys $\boldsymbol{x}^*$ bei jo tikslo funkcijos reikšmė g^*.

Metodo privalumas: gali būtų taikomas tiek tolydžioms tiek diskrečioms tikslo funkcijoms minimizuoti. Trūkumas: geriausias surastas sprendinys $\boldsymbol{x}^*$ gali labai lėtai konverguoti link globalaus sprendinio.

1.1.2 Dalelių spiečiaus metodas (PSO)

Pateikiame literatūroje [6] aprašyto dalelių spiečiaus metodo algoritmą. Parametrų erdvę tyrinėja N dalelių, kurios viena su kita tam tikru būdų pasidalina informacija apie geriausio surasto sprendinio vietą.

Metodo parametrai:

- N – dalelių kiekis
- $C_1 > 0$ – koeficientasnusakantis dalelės trauką link jau jos aplankytos geriausios pozicijos
- $C_2 > 0$ – koeficientasnusakantis dalelės trauką link surastos globalios geriausios pozicijos
- $W \in (0; 1)$ – koeficientas nusakantis dalelės inertiškumą

Atskiras daleles nurodysime žymėjimu: $(*)_j$. Indeksas $j = \overline{1, N}$ – dalelės numeris, vietoj žvaigždutės bus įrašomas simbolis, nurodantis tam tikrą dalelės charakteristiką:

- $(\boldsymbol{x})_1, (\boldsymbol{x})_2, \dots, (\boldsymbol{x})_N$ – dalelių pozicijos
- $(\boldsymbol{x})_1^B, (\boldsymbol{x})_2^B, \dots, (\boldsymbol{x})_N^B$ – kiekvienos dalelės geriausia aplankyta pozicija
- $(g)_1, (g)_2, \dots, (g)_N$ – dalelių tikslo funkcijos reikšmės
- $(g)_1^B, (g)_2^B, \dots, (g)_N^B$ – kiekvienos dalelės geriausios aplankytos pozicijos tikslo funkcijos reikšmė
- $(\boldsymbol{v})_1, (\boldsymbol{v})_2, \dots, (\boldsymbol{v})_N$ – dalelių greičių vektoriai

Paieškos metodo algoritmas:

1. Nustatyti pradinę tikslo funkcijos reikšmę: $g^* \coloneqq \infty$.
2. Nustatyti pradinės iteracijos numerį: $k \coloneqq 1$.
3. Sugeneruoti *N* dalelių pradines pozicijas:
$$\{(\boldsymbol{x})_j\}_i \sim U(l_i, u_i); \quad i = \overline{1, n}, j = \overline{1, N}$$
4. Nustatyti kiekvienos dalelės pradinę geriausią aplankytą poziciją:
$$(\boldsymbol{x})_j^B \coloneqq (\boldsymbol{x})_j, j = \overline{1, N}$$
5. Apskaičiuoti kiekvienos dalelės tikslo funkcijos reikšmę:
$$(g)_j \coloneqq g\big((\boldsymbol{x})_j\big), j = \overline{1, N}$$
6. Nustatyti kiekvienos dalelės geriausios aplankytos pozicijos pradinę tikslo funkcijos reikšmę:

$$(g)_j^B \coloneqq (g)_j \ , j = \overline{1,N}$$

7. Nustatyti pradinius dalelių greičio vektorius:

$$(\boldsymbol{v})_j \coloneqq \mathbf{0}, j = \overline{1,N}$$

8. Surasti geriausią poziciją:

forjfrom1 to N
if $(g)_j < g^*$ *then*
$\boldsymbol{x}^* \coloneqq (\boldsymbol{x})_j, g^* \coloneqq (g)_j$
endif

9. Nustatyti iteracijos numerį: $k \coloneqq 2$.
10. Perskaičiuoti dalelių greičio vektorius:

$$(\boldsymbol{v})_j \coloneqq W(\boldsymbol{v})_j + C_1 r_1\big((\boldsymbol{x})_j^B - (\boldsymbol{x})_j\big) + C_2 r_2\big(\boldsymbol{x}^* - (\boldsymbol{x})_j\big), j = \overline{1,N}$$

čia: $r_1, r_2 \sim U(0,1)$ – atsitiktiniai dydžiai sugeneruojami kiekvienai dalelei atskirai.

11. Perskaičiuoti dalelių pozicijas:

$$(\boldsymbol{x})_j \coloneqq (\boldsymbol{x})_j + (\boldsymbol{v})_j, j = \overline{1,N}$$

12. Pakoreguoti naujas dalelių pozicijas (bei atitinkamus jų greičio vektorius), jeigu jos išeina iš apibrėžimo srities $[\boldsymbol{l}, \boldsymbol{u}]$:

for j from 1 to N
for i from 1 to n
if $\{(\boldsymbol{x})_j\}_i < l_i$ *then*
$\{(\boldsymbol{x})_j\}_i \coloneqq l_i, \{(\boldsymbol{v})_j\}_i \coloneqq -\{(\boldsymbol{v})_j\}_i$
elseif $\{(\boldsymbol{x})_j\}_i > u_i$ *then*
$\{(\boldsymbol{x})_j\}_i \coloneqq u_i, \{(\boldsymbol{v})_j\}_i \coloneqq -\{(\boldsymbol{v})_j\}_i$
endif

13. Apskaičiuoti kiekvienos dalelės tikslo funkcijos reikšmę:

$$(g)_j \coloneqq g\big((\boldsymbol{x})_j\big), j = \overline{1,N}$$

14. Atnaujinti kiekvienos dalelės geriausią aplankytą poziciją bei ją atitinkančią tikslo funkcijos reikšmę:

for j from 1 to N
if $(g)_j < (g)_j^B$ *then*
$(\boldsymbol{x})_j^B \coloneqq (\boldsymbol{x})_j, (g)_j^B \coloneqq (g)_j$
endif

15. Atnaujinti geriausią poziciją bei jos tikslo funkcijos reikšmę:

forjfrom1 to N
if$(g)_j < g^*$*then*
$\boldsymbol{x}^* \coloneqq (\boldsymbol{x})_j, g^* \coloneqq (g)_j$
endif

16. Jeigu $k < Ic$, tuomet $k \coloneqq k + 1$ ir grįžti į 10 žingsnį.
17. Algoritmo pabaiga. Geriausias surastas sprendinys $\boldsymbol{x}^*$ bei jo tikslo funkcijos reikšmė g^*.

Literatūroje[7] rašoma, kad keičiant parametro W reikšmę galima reguliuoti dalelių inertiškumą. Juo W reikšmė yra didesnė – tuo dalelės yra inertiškesnės – ko pasekoje metodas labiau ištyrinėja parametrų erdvę globaliu mastu. Jeigu priešingai, W yra mažesnis – dalelės tampa ne tokios inertiškos ir greičiau konverguoja link geriausio surasto sprendinio vietos. Šio algoritmo trūkumas yra tai, kad procesas gali konverguoti link lokalaus ekstremumo, gerai prieš tai neištyręs visos parametrų erdvės. Todėl siūloma parametro W reikšmę vykdant iteracijas tolygiai keisti nuo 0.9 iki 0.4. Tokiu būdų iš pradžių, kol W reikšmė yra didesnė, labiau ištyrinėjama visa parametrų erdvę, o W reikšmei vis mažėjant – vis geriau patikslinamaoptimalaus sprendinio vieta.

Literatūroje galima surasti įvarių šio paieškos metodo modifikacijų, kelios iš jų:

- Geriausias sprendinys (į kurį dalelės judėjimo greitį netiesiogiai nusako parametras C_2) gali būti parenkamas ne iš visų dalelių, bet iš tam tikro jų poaibio, kuris kiekvienai dalelei gali būti skirtingas.
- Dalelių aibę gali sudaryti keli nepriklausomi dalelių poaibiai, iš kurių vienas yra pagrindinis ir gali pasinaudoti informacija apie geriausią surastą sprendinį kitose dalelių aibėse.
- Įvairios strategijos metodo parametrams keisti:
    - kai parametro reikšmės išlieka pastovios.
    - kai parametro reikšmės tolygiai kinta paieškos metu.
    - kai parametro reikšmės kiekvienos iteracijos metu atsitiktinai sugeneruojamos pagal tam tikrą skirstinį.

1.1.3 Lokalios paieškos metodas (LUS)

Aptarsime lokalia paieškapagrįstą paieškos metodą LUS (*local unimodal sampling*). Metodo parametrai:

- N – dalelių kiekis
- β – parametras, kuris nusako kaip greitai mažėja lokalios paieškos zona

Kaip ir aprašant PSO metodo veikimą, dalelę žymėsime kaip $(*)_j, j = \overline{1,N}$. Dalelių charakteristikos:

- $(\boldsymbol{x})_1, (\boldsymbol{x})_2, \dots, (\boldsymbol{x})_N$ – dalelių pozicijos
- $(\boldsymbol{d})_1, (\boldsymbol{d})_2, \dots, (\boldsymbol{d})_N$ – kiekvienos dalelės paieškos zonos spindulys
- $(\boldsymbol{y})_1, (\boldsymbol{y})_2, \dots, (\boldsymbol{y})_N$ – dalelių poslinkių vektoriai
- $(\boldsymbol{z})_1, (\boldsymbol{z})_2, \dots, (\boldsymbol{z})_N$ – dalelių pozicijos po paslinkimo
- $(T)_1, (T)_2, \dots, (T)_N$ – kiekviena dalelė „atsimena“ kiek kartų nepavyko pagerinti jos pozicijos
- $(g)_1, (g)_2, \dots, (g)_N$ – dalelių tikslo funkcijos reikšmės

Paieškos metodo algoritmas:

1. Nustatyti pradinę tikslo funkcijos reikšmę: $g^* \coloneqq \infty$.
2. Nustatyti:
$$(\boldsymbol{d})_j \coloneqq \boldsymbol{u} - \boldsymbol{l}, (T)_j \coloneqq 0;\ j = \overline{1,N}$$
3. Nustatyti pradinės iteracijos numerį: $k \coloneqq 1$.
4. Atsitiktinai sugeneruoti pradines dalelių pozicijas:
$$\{(\boldsymbol{x})_j\}_i \sim U(l_i, u_i);\quad i = \overline{1,n}, j = \overline{1,N}$$
5. Apskaičiuoti dalelių tikslo funkcijos reikšmes:
$$(g)_j \coloneqq g\big((\boldsymbol{x})_j\big), j = \overline{1,N}$$
6. Nustatyti pradinės iteracijos numerį: $k \coloneqq 2$.
7. Sugeneruoti dalelių poslinkio vektorius:
$$\{(\boldsymbol{y})_j\}_i \sim U\left(-\{(\boldsymbol{d})_j\}_i, \{(\boldsymbol{d})_j\}_i\right);\quad i = \overline{1,n}, \qquad j = \overline{1,N}$$
8. Atnaujinti dalelių pozicijas:

for j from 1 to N

$$(\boldsymbol{z})_j \coloneqq (\boldsymbol{x})_j + (\boldsymbol{y})_j$$

for i from 1 to n

if $\{(\boldsymbol{z})_j\}_i < l_i$ *then*

$$\{(\boldsymbol{z})_j\}_i := l_i$$

elseif $\{(\boldsymbol{z})_j\}_i > u_i$ *then*

$$\{(\boldsymbol{z})_j\}_i := u_i$$

endif

if $g((\boldsymbol{z})_j) < (g)_j$ *then*

$$(\boldsymbol{x})_j := (\boldsymbol{z})_j, (g)_j := g((\boldsymbol{z})_j)$$

else

$$(T)_j := (T)_j + 1, (\boldsymbol{d})_j := \left(\frac{1}{2}\right)^{\frac{\beta}{(T)_j}} \cdot (\boldsymbol{d})_j$$

endif

endfor

9. Jeigu $k < Ic$, $k := k + 1$ ir grįžti į 7 žingsnį.
10. Surasti geriausią poziciją:

for j from 1 to N

if $(g)_j < g^*$ *then*

$$\boldsymbol{x}^* := (\boldsymbol{x})_j, g^* := (g)_j$$

endif

11. Algoritmo pabaiga. Geriausias surastas sprendinys $\boldsymbol{x}^*$ bei jo tikslo funkcijos reikšmė g^*.

Čia aprašytas metodas yra bendresnis atvejis metodo pateikto literatūroje [8], kur aprašomas atvejis kai $N = 1$.

Įvedame du papildomus parametrus iteracijų kiekiui nusakyti:

- Ic_{main} – pagrindinių iteracijų kiekis
- Ic_{sub} – tarpinių iteracijų kiekis

Metodo iteracijų kiekis paskaičiuojamas taip: $Ic := Ic_{main} \cdot Ic_{sub}$. Tokiu būdu keisdami dydį Ic_{sub} galime reguliuoti kiek atliekama tikslo funkcijos įvertinimų kiekvienoje pagrindinėje iteracijoje. O tai vėliau leis nesunkiai palyginti šį metodą su kitais, pareikalaujant, kad kiekvienos iteracijos (LUS metodo atveju – pagrindinės iteracijos) metu būtų atliekamas vienodas tikslo funkcijų įvertinimų kiekis.

1.2 Tolydaus laiko Markovo grandinė

Trumpai apžvelgsime literatūroje [3],[4] pateikta informacija apie tolydaus laiko Makovo grandines, kurias žymėsime santrumpa CTMC (*continuous time Markov chain*).

Apibrėžiame sistemos būsenų aibę $C = \{c_i | i = 1,2, \dots, n\}$.

Apibrėžimas.*Tolydaus laiko procesas* $\{X_t, t \geq 0\}$, *kai dydis*X_t *laikui bėgant atsitiktinai įgyja reikšmes iš būsenų aibės* C*ir* $P(X_0 = c_i) = \pi_i(0)$, *vadinamas tolydaus laiko Markovo grandie jeigu tenkinamos dvi sąlygos:*

(a) su visais galimais laikų rinkiniais$t_0 \leq t_1 \leq \dots, \leq t_k, k \in \boldsymbol{N}$ *galioja savybė:*

$$P\left(X_{t_k} = x_k | X_{t_0} = x_0, X_{t_1} = x_1, \dots, X_{t_{k-1}} = x_{k-1}\right) = P\left(X_{t_k} = x_k | X_{t_{k-1}} = x_{k-1}\right)$$

(b) visiems laikams $t \geq 0$ *galioja savybė:*

$$\forall i, j \in \{1,2, \dots, n\} : \ P\left(X_{t+s} = c_j | X_t = c_i\right) = P\left(X_t = c_j | X_0 = c_i\right)$$

Pasinaudojant apibrėžimo savybe *(b)* galima apibrėžti perėjimo tikimybių matricą $P(t), t \geq 0$:

$$\{P(t)\}_{ij} = P\left(X_t = c_j | X_0 = c_i\right) (1.1)$$

Perėjimų tikimybių matrica tenkiną savybę:

$$P(t + s) = P(t)P(s) (1.2)$$

Įrodymas:

►

$$\{P(t+s)\}_{ij} = P\left(X_{t+s} = c_j | X_0 = c_i\right) =$$

$$= \sum_{k=1}^{n} P\left(X_{t+s} = c_j | X_t = c_k, X_0 = c_i\right) P(X_t = c_k | \, X_0 = c_i) =$$

$$= \sum_{k=1}^{n} P\left(X_{t+s} = c_j | X_t = c_k\right) P(X_t = c_k | \, X_0 = c_i) = \sum_{k=1}^{n} \{P(s)\}_{kj} \{P(t)\}_{ik} = \{P(s)P(t)\}_{ij}$$

▲

Lygtį (1.2)išdiferencijavę pagal kintamąjį *t*ir įstatę $t = 0$ bei pažymėję $Q = P`(0)$ gauname lygtį (1.3) :

$$P`(s) = QP(s) (1.3)$$

Arba (1.2)išdiferencijavę pagal kintamąjį s ir įstatę $s = 0$ bei pažymėję $Q = P`(0)$ gauname lygtį (1.4) :

$$P`(t) = P(t)Q \qquad (1.4)$$

Pastaba $P(0) = I$. Šios lygtys(1.3,1.4) turi tą patį sprendinį:

$$P(t) = e^{tQ} = \sum_{k=0}^{\infty} \frac{(tQ)^k}{k!} (1.5)$$

Bei perėjimų tarp būsenų intensyvumų matricą $[n \times n]$:

$$Q = \begin{bmatrix} q_{11} & \cdots & q_{1n} \\ \vdots & \ddots & \vdots \\ q_{n1} & \dots & q_{nn} \end{bmatrix} (1.6)$$

Reikšmė $q_{ij} i, j = \overline{1,n}$, kai $i \neq j$ nurodo kiek vidutiniškai įvyksta perėjimų iš i-tosios į j-tąją būseną per tam tikrą laiko vienetą. Tikimybė, kad per vienetinį laiko vienetą įvyks $k \geq 0$ peršokimų turi Pusono skirstinį (1.7):

$$P(K = k\ ) = e^{-q_{ij}} \sum_{r=0}^{k} \frac{{q_{ij}}^r}{r!} (1.7)$$

Tuo tarpu laiko tarpai tarp peršokimų yra pasiskirstę pagal eksponentinį skirstinį (1.8):

$$P(X < x) = 1 - e^{-q_{ij}x} (1.8)$$

Laikas praleistas i-tojoje būsenoje, taip pat turi eksponentinį skirstinį su intensyvumu$-q_{ii}$.

Perėjimų intensyvumų matricos (1.6) elementai tenkina sąlygas:

- $q_{ij} \geq 0,\ i \neq j$
- $\sum_{j=1}^{m+1} q_{ij} = 0,\ i = \overline{1,n}$ (1.9)
- $q_{ii} = -\sum_{j=1, j\neq i}^{m+1} q_{ij},\ i = \overline{1,n}$
- $0 \leq -q_{ii} \leq +\infty,\ i = \overline{1,n}$

Procesas prasideda *i*-tojoje būsenoje su tikimybe $\pi_i(0), i = \overline{1,n}$. Tolydaus laiko Markovo grandinei nusakyti pakanka nurodyti būsenų aibę, perėjimų intensyvumų matricą bei pradinių tikimybių vektorių :

$$CTMC(C, \boldsymbol{\pi}(0), Q)$$

čia : $\boldsymbol{\pi}(0) = [\pi_1(0), \pi_2(0), \dots, \pi_n(0)]$

Laikui bėgant sistemos būsena atsitiktinai kinta. Tikimybė, kad laiko momentu *t* sistema bus tam tikroje būsenoje nusakoma tikimybių vektoriumi $\boldsymbol{\pi}(t), t > 0$, kuris paskaičiuojamas pagal formulę:

$$\boldsymbol{\pi}(t) = \boldsymbol{\pi}(0) \cdot P(t) = \boldsymbol{\pi}(0) \cdot e^{tQ}, t \geq 0 \qquad (1.9)$$

Apibrėžimas.*Sugeneriančąja Markovo grandinės būsena vadinama būsena į kurią patekus iš jos jau niekada neišeinama. Markovo grandinė turinti bent vieną sugeneriančią būseną vadinama sugenerenčiąja Markovo grandine.*

Apibrėžimas. *Markovo gradinė vadinama ergodine jeigu, jos būsenos sudaro vieną klasę ir ji nėra segerenčioji Markovo gradinė.*

Jeigu nagrinėjama Markovo grandinė yra ergodine, laikui bėgant jos elgsena nusistovi ir nepriklauso nuo to kokioje būsenoje procesas prasidėjo. Tokiu atveju galima paskaičiuoti finalines būsenų tikimybes, pagal apibrėžimą (1.10):

$$\boldsymbol{\pi}^* = \lim_{n\to\infty} \boldsymbol{\pi}(t) \text{ (1.10)}$$

Arba skaitiniu būdų, išsprendus lygčių sistemą (1.11):

$$\boldsymbol{\pi}^* \cdot Q = 0, \quad \sum_{i=1}^{n} \pi_i^* = 1 \qquad (1.11)$$

1.3 Fazinio tipo skirstiniai

1.3.1 Tankio ir pasiskirstymo funkcijos bei jų savybės

Pasinaudodami literatūros šaltiniu [1] pateikiame svarbiausią informaciją apie fazinius skirstinius, kuriuos nurodysime sutrumpinimu PTD (*phase-type distribution*).

Apibrėžimas.*Fazinis skirstinys nusako laiką per kurį pasiekiama vienintelėsugeriančioji tolydaus laiko Markovo grandinės būsena. Fazinis skirstinys nurodomas žymėjimuPTD*$(\boldsymbol{\pi}, T)$*, kur* $\boldsymbol{\pi}$*– pradinių tikimybių vektorius,*T *– perėjimo intensyvumų matrica.*

Apibrėžime minimą tolydaus laiko Markovo grandinę sudaro m tarpinių būsenų$c_1, c_2, \dots, c_m$ ir viena sugeriančioji būsenac_{m+1}, formaliai galima užrašyti taip$CTMC(\{c_1, c_2, \dots, c_m, c_{m+1}\}, [\pi_1, \pi_2, \dots, \pi_m, 0], Q)$, kurperėjimų intensyvumų matrica$Q$yra sudaryta iš komponenčių (1.12):

$$Q = \begin{bmatrix} T & \boldsymbol{t} \\ \mathbf{0} & 0 \end{bmatrix} \text{ (1.12)}$$

čia$T$ – perėjimų tarp tarpinių būsenų intensyvumų matrica, $\boldsymbol{t}$ – perėjimo intensyvumų iš tarpinių būsenų į sugeriančią būseną vektorius-stulpelis, $\mathbf{0}$ – nulinis vektorius-eilutė, turintis *m* nulinių elementų.

$$T = \begin{bmatrix} t_{11} & \cdots & t_{1m} \\ \vdots & \ddots & \vdots \\ t_{m1} & \cdots & t_{mm} \end{bmatrix}, \boldsymbol{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_m \end{bmatrix} \text{ (1.13)}$$

Pasinaudojant parėjimo matricos Q savybėmis (1.9) vektorių $\boldsymbol{t}$galima išreikšti per matricą T (1.14):

$$\boldsymbol{t} = (I - T)\mathbf{1} \text{ (1.14)}$$

Markovo procesas prasideda *i*-tojoje tarpinėje būsenoje su tikimybe π_i. Laiko tarpas nuo proceso pradžios iki momento kai pasiekiama sugeriančioji būsena turi fazinį skirstinį (1.15):

$$X \sim PTD(\boldsymbol{\pi}, T), \qquad \boldsymbol{\pi} = [\pi_1, \pi_2, \dots, \pi_m] \qquad (1.15)$$

Fazinio skirstinio charakteristikos:

- skirstinio funkcija: $F(x;\ \boldsymbol{\pi}, T) = 1 - \boldsymbol{\pi} e^{xT} \mathbf{1}$
- tankio funkcija: $f(x;\ \boldsymbol{\pi}, T) = \boldsymbol{\pi} e^{xT} \boldsymbol{t}$
- vidurkis: $E(X) = -\boldsymbol{\pi} T^{-1} \mathbf{1}$ (1.16)
- dispersija: $D(X) = 2\boldsymbol{\pi} T^{-2} \mathbf{1} - (\boldsymbol{\pi} T^{-1} \mathbf{1})^2$
- k-tasis momentas: $E(X^k) = (-1)^{\boldsymbol{k}} k!\, \boldsymbol{\pi} T^{-k} \mathbf{1}$

Fazinį skirstinį galima nusakyti ir kitu būdu naudojantis pradinių tikimybių vektoriumi $\boldsymbol{\pi}$, perėjimo tarp būsenų tikimybių matrica P ir vektoriumi $\boldsymbol{s} = [s_1, s_2, \dots, s_m, 0], (s_i = -t_{ii}, i = \overline{1, m})$. Vektorius $\boldsymbol{s}$nurodo, kiek laiko (t.y. i-tojoje būsenoje praleidžiamas laikas yra pasiskirstęs eksponentiškai, $P(X < x) = 1 - e^{-s_i x}$) procesas praleidžia tam tikroje būsenoje, prieš peršokdamas į kitą. Turėdami matricą Q nesunkiai galime surasti perėjimų tikimybių matricąP (1.17):

$$\{P\}_{ij} = -\frac{q_{ij}}{q_{ii}}, i \neq j;\ i, j = \overline{1, m+1} \qquad (1.17)$$

Šią fazinio skirstinio reprezentaciją naudosime fazinių skirstinių struktūrų schemose, tuo tarpu visus skaičiavimus atliksime naudodamiesi fazinio skirstinio intensivumų matricą T.

Pavyzdys. Tarkime turime fazinį skirstinį su parametrais:

$$\boldsymbol{\pi} = [0.5;\ 0.5], \qquad \boldsymbol{T} = \begin{bmatrix} -6 & 3 \\ 3 & -8 \end{bmatrix}$$

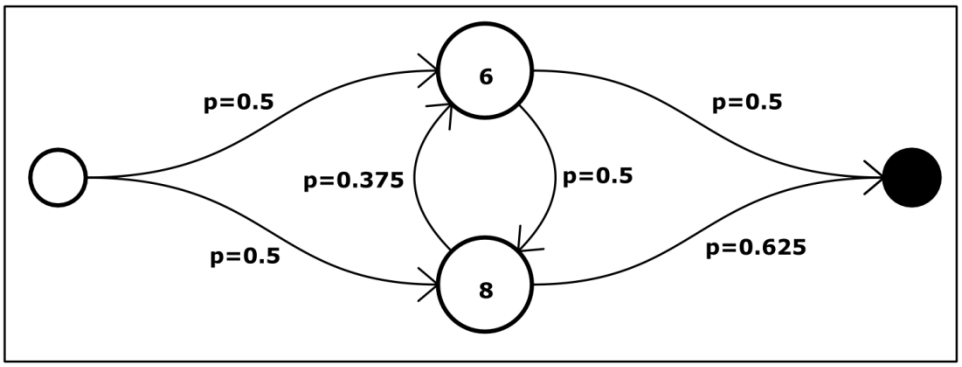
Suformuojame vektorių $\boldsymbol{s}$**:**

$$\boldsymbol{s} = [6;\ 8]$$

Suskaičiuojame perėjimo tikimybių matricą *P*:

$$P = \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0.375 & 0 & 0.625 \\ 0 & 0 & 0 \end{bmatrix}$$

Tuomet šį fazinį skirstinį schematiškai galima pavaizduoti taip:



1.1 pav. Fazinio skirstinio schemos pavyzdys.

Schemoje (1.1 pav.) kairėje pusėje pavaizduotas apskritimas vaizduoja vietą iš kurios prasideda procesas, dešinėje pusėje esantis juodas skritulys – tai sugeriančioji būsena. Per vidurį esantys apskritimai reprezentuoja fazinio skirstinio tarpines būsenas (dar vadinamas fazėmis), kurios yra numeruojamos $1, 2, \ldots$ nuo viršaus į apačią. Šiuose apskritimuose surašytos vektoriaus $\boldsymbol{s}$reikšmės. Galimus būsenų pasikeitimus rodo rodyklės. Iš būsenos galima patekti į kitas būsenas su nurodytomis tikimybėmis. Laikas per kurį procesas pasiekia sugeriančią būseną yra atsitiktinis dydis $X \sim PTD(\boldsymbol{\pi}, T)$.

1.3.2 Skirstinių struktūros

Fazinį skirstinį charakterizuoja jo struktūra. Panagrinėsime visų galimų unikalių fazinio skirstinio struktūrų generavimo uždavinį.

Apibrėžiame loginio kintamojo matricą $\dot{T}$ (1.18):

$$\dot{T} = \begin{bmatrix} \dot{t}_{11} & \cdots & \dot{t}_{1m} \\ \vdots & \ddots & \vdots \\ \dot{t}_{m1} & \cdots & \dot{t}_{mm} \end{bmatrix}, \qquad \dot{t}_{ij} \in \{0, 1\} \qquad (1.18)$$

Matricos $\dot{T}$ elementai apibūdina matricą T bei vektorių $\boldsymbol{t}$tokių būdu. Kai $i \neq j$:

$$\dot{t}_{ij} = 0 \leftrightarrow t_{ij} = 0, \dot{t}_{ij} = 1 \leftrightarrow t_{ij} > 0 \qquad (1.19)$$

Papildomai apibrėžiamas vektorius $\dot{\boldsymbol{\pi}}$(1.20):

$$\dot{\boldsymbol{\pi}} = [\dot{\pi}_1, \dot{\pi}_2, \ldots, \dot{\pi}_m], \ \ \dot{\pi}_i \in \{0, 1\} \qquad (1.20)$$

Kuris apibūdina pradinių tikimybių vektorių $\boldsymbol{\pi}$taip (1.21):

$$\dot{\pi}_i = 0 \leftrightarrow \pi_i = 0, \dot{\pi}_i = 1 \leftrightarrow \pi_i > 0 \qquad (1.21)$$

Pasinaudodami apibrėžtais dydžiais $\dot{T}, \dot{\boldsymbol{\pi}}$galima nusakyti bet kokią fazinio skirstinio strutūrą.Apibrėžiame fazinio skirstinio struktūros ryšių kiekį $L(\dot{\boldsymbol{\pi}}, \dot{T})$ pagal formulę (1.22):

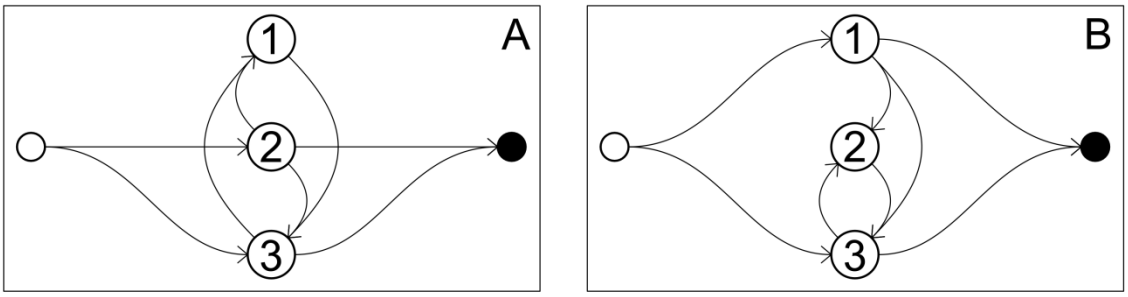
$$L(h) = \sum_{i=1}^{m} \sum_{j=1}^{m} \dot{t}_{ij} + \sum_{i=1}^{m} \dot{\pi}_i, \qquad h = [\dot{\boldsymbol{\pi}}, \dot{T}] \quad (1.22)$$

Sugeneruojame visus galimus ryšių variantus$h \in H_0$, $card(H_0) = 2^{m(m+1)}$. Iš aibės H_0 elementų suformuojamas poaibis H_1 į kurį patenka tie elementai h, kurie tenkina dvi sąlygas:

- h apibrėžia struktūrą, kurioje galima patekti į bet kurią tarpinę būseną
- h apibrėžia struktūrą, kurioje iš betkurios tarpinės būsenos galima pasiekti sugeriančią būseną

Galiausiai iš aibės H_1 atrenkami tie h elementai, kurie reprezentuoja unikalias struktūras ir surašomi į aibę H_2, kuriojebetkurios dvi struktūros nėra ekvivalenčios.

Apibrėžimas.*Dvi struktūros h_A ir h_B vadinamos ekvivalenčiomis (žymėsime $h_A \sim h_B$) jeigu galima surasti tokį struktūros h_A būsenų pernumeravimą, kad gautosios $h_{\hat{A}}$ duomenys sutaptų su struktūros h_B duomenimis, t.y.:* $\dot{\boldsymbol{\pi}}_{\hat{A}} \equiv \dot{\boldsymbol{\pi}}_{\boldsymbol{B}},\ \dot{T}_{\hat{A}} \equiv \dot{T}_B$.



1.2 pav. Ekvivalenčių struktūrų pavyzdys.

Pateiktame ekvivalenčių struktūrų pavyzdyje (1.2 pav.) matome, kad sukeitę vietomis *A* struktūros pirmąją ir antrąją būsenas gausime struktūrą *B*.

Unikalias struktūras iš aibės H_2 surikiuojame ryšių kiekio $L(h)$ didėjimo tvarka. Kiekvienai struktūrai priskiriame kodą: „**struct-x-y-z**“. Kur „x“ nurodo būsenų (fazių) kiekį, „y“ – struktūroje esančių ryšių kiekį ir „**z**“ – struktūros numerį tarp struktūrų turinčių tą patį ryšių kiekį. Sugeneruotos struktūros pateiktos 1 priede. Betkokiam faziniam skirstiniui $PTD(\boldsymbol{\pi}, T)$ galima surasti jo struktūrą $[\dot{\boldsymbol{\pi}}, \dot{T}]$ ir pritaikius ekvivalenčių struktūrų apibrėžimą nustatyti struktūros kodą.

Akivaizdu, kad turėdami betkokį fazinį skistinį $PTD(\boldsymbol{\pi}_A, T_A)$ ir pernumeravę jo tarpines būsenas gausime kitą ekvivalentų fazinį skirstinį $PTD(\boldsymbol{\pi}_B, T_B)$. Vadinasi tas pats fazinis skirstinys gali būti nusakomas įvairiais parametrų $\boldsymbol{\pi}, T$ rinkiniais. Todėl galima įvesti surikiuoto fazinio skirstinio sąvoką.

1.2 Apibrėžimas.*Fazinis $PTD(\boldsymbol{\pi}, T)$ skirstinys vadinamas surikuotu, jeigu $t_{11} \le t_{22} \le \ldots \le t_{mm}$ ir yra žymimas $PTD(\overline{\boldsymbol{\pi}}, \overline{T})$. O parametrų transfromacija $\Phi: \boldsymbol{\pi}, T \to \overline{\boldsymbol{\pi}}, \overline{T}$vadinama fazinio skirstinio parametrų rikiavimo transformacija.*

Transformaciją Φ nusako vektorius $\boldsymbol{\varphi} = [\varphi_1, \varphi_2, \dots, \varphi_\mathrm{m}]$, čia φ_i yra tarpinės būsenos numeris. O pati transformacija atliekama pagal formules (1.23):

$$\bar{\pi}_i \coloneqq \pi_{\varphi_\mathrm{i}}, \qquad \bar{T}_{i,j} \coloneqq T_{\varphi_\mathrm{i},\varphi_j}; i,j = \overline{1,m} \qquad (1.23)$$

1.3.3 Parametrų suradimas

Fazinio skirstinio parametrų ieškoti galima įvairiais metodais, keletas iš jų:

- EM (*expectation maximization*) metodas [9], kuris maksimizuoja tikėtinumą, kad imtis buvo sugeneruota tam tikro fazinio skirstinio.
- Momentų sulyginimo metodas [10], kurio pagalba analitiniu būdų išskaičiuojami fazinio skirstinio parametrai, su kuriais susilygina aproskimuojamo ir fazinio skirstinių tam tikras kiekis pirmųjų momentų.
- Fazinio skirstinio parametrų ieškojimas panaudojant vektoriaus optimizavimo metodus.

Šiame darbe apsiribota fazinių skirstinių parametrų ieškojimu taikant 1.1 skyrelyje aptartaisoptimizavimo metodais.

Tarkime, kad reikia surasti fazinio skistinį$PTD(\boldsymbol{\pi}, T)$, kuris kiek galima geriau aproksimuotų tam tikrą skirstinį. Pažymime: $f_{PTD}(x;\ \boldsymbol{\pi}, T)$ – fazinio skirstinio tankio funkcija, $f(x;\ \boldsymbol{\theta})$ – aproksimuojamo skirstinio tankio funkcija, kur $\boldsymbol{\theta}$ – parametrų vektorius.

Ieškosime tokių fazinio skirstinio parametrų $\boldsymbol{\pi}, T$ reikšmių, su kuriomis minimizuojamas plotas tarp tankio funkcijų $f_{PTD}(x;\ \boldsymbol{\pi}, T)$ ir $f(x;\ \boldsymbol{\theta})$. Formaliai ši statistika užrašoma taip:

$$S = \int_0^\infty |f_{PTD}(x;\ \boldsymbol{\pi}, T) - f(x;\ \boldsymbol{\theta})|\, dx \qquad (1.24)$$

Pastarojo integralo (1.24) integravimas yra pernelyg komplikuotas, dėl sudėtingos fazinio skirstio tankio funkcijos išraiškos. Todėl naudojamas šios statistikos įvertis $\hat{S}$:

$$\hat{S}(x_{end},\ \Delta x;\ \boldsymbol{\pi}, T) = \sum_{k=1}^{n} |f_{PTD}(x_k);\ \boldsymbol{\pi}, T) - f(x_k;\ \boldsymbol{\theta})|\Delta x; x_k = (k - 0.5)\Delta x,$$

$$n = \left\lfloor \frac{x_{end}}{\Delta x} \right\rfloor \qquad (1.25)$$

čia:x_{end} – nurodo reikšmę iki kurios atliekamas diskretizavimas, Δx – diskretizavimo žingsnelio dydis.

Diskretizavimo parametrai šiame darbe parenkami rankiniu būdu atsižvelgiant į tai kaip stipriai pakinta diskretizuoto skirstinio vidurkis bei standartinis nuokrypis. Diskretizuoto skirstinio vidurkis (1.25):

$$E[\hat{X}] = \hat{\mu} = \frac{1}{n}\sum_{k=1}^{n} x_k f(x_k;\, \boldsymbol{\theta})\Delta x \qquad (1.25)$$

Diskretizuoto skirstio standartinis nuokrypis (1.26):

$$Std[\hat{X}] = \hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n} (x_k - \hat{\mu})^2 f(x_k;\, \boldsymbol{\theta})\Delta x} \qquad (1.26)$$

Nustatome vaizdavimą tarpvektoriaus $\boldsymbol{x}$ir fazinio skirstio $PTD(\boldsymbol{\pi}, T)$ (turinčio mbūsenų /fazių) parametrų $\boldsymbol{\pi}, T$ (1.27):

$$\begin{aligned}\boldsymbol{x} = (x_1, x_2, \dots, x_{m\cdot(m+1)}) \rightarrow \boldsymbol{\pi} = \frac{[x_1, x_2, \dots, x_m]}{x_1 + x_2 + \dots + x_m}, t_1 = x_{m+1}, t_{12} = x_{m+2}, \dots, t_{1m} = x_{2\cdot m}, t_{21} \\ = x_{2\cdot m+1}, t_2 = x_{2\cdot m+2}, t_{23} = x_{2\cdot m+3}, \dots, t_{2m} = x_{3\cdot m}, \dots, t_{m1} = x_{m^2+1}, t_{m2} \\ = x_{m^2+2}, \dots, t_m = x_{m^2+m} \qquad (1.27)\end{aligned}$$

Kadangi fazinio skirstinio parametrų ieškome baigtinėje erdvėje, reikia iš anksto pasirinkti koks gali būtų maksimulus perėjimo tarp būsenų intensyvumas λ_{max}. Nusakome parametrų vektoriaus $\boldsymbol{x}$apibrėžimo sritį (1.28):

$$\boldsymbol{l} = [0,0, \dots, 0], \boldsymbol{u} = \left[\underbrace{1,1, \dots, 1}_{m}, \underbrace{\lambda_{max}, \lambda_{max}, \dots, \lambda_{max}}_{m^2}\right] \qquad (1.28)$$

Pasinaudodami vaizdavimu (1.27) ,vektoriaus $\boldsymbol{x}$apibrėžimo sritimi (1.28)bei apibrėžta tikslo funkcija (1.25) fazinio skirstinio parametrų reikšmių galime ieškoti skyrelyje 1.1 aptartais paieškos metodais.

1.4 Markovo tipo sistemos būsenų generavimo algoritmas

Apibrėžimas. *Sistema$\langle S \rangle$ vadinama Markovo tipo sistema jeigu jos elgseną galima sumodeliuoti Markovo gradine.*

Kitaip tariant, sistema yra Markovo tipo, jeigu sistemos praleidžiamas laikas bet kurioje būsenoje turi eksponentinį skirstinį.

Sistemą $\langle S \rangle$ čia suprantame, kaip taisyklių visumą, kuri nusako kaip sistema keičia savo būseną. Sistemos būsena nurodoma vektoriumi $\boldsymbol{v} = (k_1, k_2, \dots, k_N)$, kur kintamasis k_i yra vadinamas k-tąja būsenos koordinate ir gali įgyti reikšmes $k_i \in \{0, 1, \dots, n_i\}$. Koordinačių kiekis$N$ bei jų maksimalios reikšmes $n_i$yra pasirenkamos konkrečios sistemos modeliavimo metu.

Apibrėžiame sistemos $\langle S \rangle$visų galimų būsenų aibę:

$$V^+ = \{\boldsymbol{v} = (k_1, k_2, \dots, k_N) \mid k_i \in \{0, 1, \dots, n_i\}, i = \overline{1, N}\} \qquad (1.29)$$

Nesunku pastebėti, kad ši aibė (1.29) iš viso turi $r_{max} = card(V^+) = \prod_{i=1}^{N}(n_i + 1)$ elementų, kuriuos sunumeruojame:

$$I_v = \omega(\boldsymbol{v}) = \sum_{i=1}^{N-1}\left(k_i \prod_{j=i+1}^{j=N}(n_i+1)\right) + k_N \in \{0,1,\dots,r_{max}-1\} \qquad (1.30)$$

Atvirkštinis vaizdavimas $\omega^{-1}(I_v)$:

$$\boldsymbol{v} = \omega^{-1}(I_v) = \left(a_1, a_2\%(n_2+1), \dots, a_{N-1}\%(n_{N-1}+1), I_v\%(n_N+1)\right) \qquad (1.31)$$

$$a_{N-1} = \frac{I_v - I_v\%(n_N+1)}{n_N+1}, a_{N-2} = \frac{a_N - a_N\%(n_{N-2}+1)}{n_{N-2}+1}, \dots, a_1 = \frac{a_2 - a_2\%(n_2+1)}{n_2+1}$$

čia % yra sveikų skaičių dalybos liekana.

Reikia surasti sistemos $\langle S\rangle$ būsenų aibę V ir pradinių tikimybių vektorių $\boldsymbol{\pi}(0)$ bei perėjimų tarp būsenų intensyvumų matricą Q. Loginio kintamojo vektoriumi $\boldsymbol{u} = \left(u_0, u_1, \dots, u_{r_{max}-1}\right), u_{I_v} \in \{false, true\}, I_v = \overline{0, r_{max}-1}$žymėsime jau išnagrinėtas būsenas. Apibrėžiame algoritme naudojamas aibes: V^{init} – sistemos startinių būsenų aibė, V – sistemos pasiekiamų būsenų aibė, V^{np} – būsenų, kurios bus nagrinėjamas sekančios iteracijos metu, aibė, ir V^{tmp} – pagalbinė būsenų aibė. Šiose aibėse yra laikomi būsenų numeriai I_v. Perėjimų tarp visų būsenų intensyvumų matrica $Q^+ = \{q_{ij}^+\};\ i,j = \overline{0, r_{max}-1}$.

Algoritmas:

1. Visas būsenas pažymėti kaip neišnagrinėtas:
$$u_{I_v} \coloneqq false\ , I_v = \overline{0, r_{max}-1}$$
2. Pasinaudojant sistemos $\langle S\rangle$ apibrėžimu nustatyti pradinių tikimybių vektorių $\boldsymbol{\pi}(0)$.
3. Pasinaudojant sistemos $\langle S\rangle$ apibrėžimu nustatyti startinių būsenų aibę V^{init}.
4. Priskirti pradines reikšmes:
$$V \coloneqq V^{init}, V^{tmp} \coloneqq V^{init}, \{q_{ij}^+\} \coloneqq 0;\ i,j = \overline{0, r_{max}-1}$$
5. Priskirti aibei V^{np} būsenas, kurias reikia išnagrinėti:
$$V^{np} \coloneqq V^{tmp}, \qquad V^{tmp} \coloneqq \emptyset$$
6. Tegu $\{V^{np}\}_p$ bus p-tasis aibės V^{np} elementas:
$$p \coloneqq 1$$
7. Surasti nagrinėjamos būsenos koordinačių vektorių:
$$\boldsymbol{v} \coloneqq \omega^{-1}\left(\{V^{np}\}_p\right),\ I_v \coloneqq \omega(\boldsymbol{v})$$
8. Tegu V^* bus būsenų, kurios gali būti pasiekiamos iš būsenos $\boldsymbol{v}$, aibė:$V^* \coloneqq \emptyset$, ir λ^* - perėjimų intensyvumų iš būsenos $\boldsymbol{v}$ į atitinkamas būsenas aibėje V^*, aibė: $\lambda^* = \emptyset$.
9. Pasinaudojant sistemos $\langle S\rangle$ apibrėžimu surasti aibes: $V^* = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k\}, \lambda^* = \{\lambda_1, \lambda_2, \dots, \lambda_k\}$.
10. Atnaujinti perėjimų intensyvumų matricą:
$$q^+_{I_v I_{v_i}} \coloneqq \lambda_i;\ I_{v_i} = \omega(\boldsymbol{v}_i), i = \overline{1,k}$$

11. Dar neišnagrinėtas būsenas iš aibės V^* įtraukti į aibes V ir V^{tmp}:

for i from 1 to k
$I_{v_i} \coloneqq \omega(\boldsymbol{v}_i)$
if $u_{I_{v_i}} = false$ *then*
$V^{tmp} \coloneqq V^{tmp} \cup \{I_{v_i}\}$,
$V \coloneqq V \cup \{I_{v_i}\}$
endif

12. Pažymėti dabartinę būseną kaip išnagrinėtą:

$$u_{I_v} \coloneqq true$$

13. Jeigu $p < card(V^{np})$, tada $p \coloneqq p + 1$ ir grįžti į 7 žingsnelį.
14. Jeigu $V^{tmp} \neq \emptyset$ grįžti į 5 žingsnelį.
15. Sukonstruoti pasiekiamų sistemos būsenų aibę V ir pradinių tikimybių vektorių $\boldsymbol{\pi}(0)$:

for i from 0 to $card(V) - 1$
$I \coloneqq \omega(\{V\}_i)$
$\pi_i \coloneqq \pi_I^+$
for j from $0\ card(V) - 1$
$J \coloneqq \omega(\{V\}_j)$,
$q_{ij} \coloneqq q_{IJ}^+$
endfor
endfor

16. Algoritmo pabaiga.

Atlikę pateiktą algoritmą gauname tolydaus laiko Markovo grandinę$CTMC(V, \boldsymbol{\pi}(0), Q)$. Sekančiuose skyreliuose panagrinėsime, kaip atliekami 2,3 ir 9 algoritmo žingsneliai konkrečioms sistemoms.

1.5 Aptarnavimo sistemų modeliai

Aptarnavimo sistema aptarnauja paraiškas. Gyvenime dažnai sutinkamų aptarnavimo sistemų pavyzdžiai: pirkėjai laukia eilėse prie parduotuvės kasų; mašinos degalinėje laukia, kada galės įsipilti degalų; skaičiavimo užduotys laukia eilėje, kol būs įvykdytos vieno iš procesorių ir pan.

Aptarnavimo sistemą (pagal Kendall‘o klasifikaciją) nusako parametrų rinkinys [2]:

$$A/B/m/K/n/D \qquad (1.11)$$

čia: $A \in \{M, PTD, D, G\}$ – skirstinys, pagal kurį yra pasiskirstę laiko tarpai tarp paraiškų atėjimų, $B \in \{M, PTD, D, G\}$ – paraiškų aptarnavimo trukmių skirstinys, $m \in \boldsymbol{N}$ – aptarnavimo punktų kiekis, $K \in \boldsymbol{N}$ – bendra sistemos talpa (t.y. maksimalus paraiškų kiekis sistemoje), n – paraiškų šaltinių kiekis, D – sistemos struktūra.

Parametras Agali įgyti vieną iš reikšmių:

- M – kuomet laiko trukmės tarp paraiškų atėjimų yra pasiskirstę pagal eksponentinį skirstinį
- PTD – kuomet laiko trukmės tarp paraiškų atėjimų yra pasiskirstę pagal fazinį skirstinį

- D – kuomet laiko trukmės tarp paraiškų atėjimų yra pasiskirstę pagal tolygųjį skirstinį
- G – kuomet laiko trukmės tarp paraiškų atėjimų yra pasiskirstę pagal betkokį skirstinį

Parametras B gali įgyti vieną iš reikšmių:

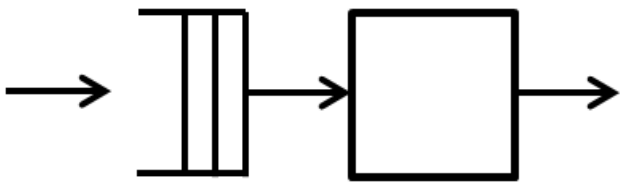
- M – kuomet paraiškų aptarnavimo laikai yra pasiskirstę pagal eksponentinį skirstinį
- PTD – kuomet paraiškų aptarnavimo laikai yra pasiskirstę pagal fazinį skirstinį
- D – kuomet paraiškų aptarnavimo laikai yra pasiskirstę pagal tolygųjį skirstinį
- G – kuomet paraiškų aptarnavimo laikai yra pasiskirstę pagal betkokį skirstinį

Aptarnavimo sistemos modeliavimas tikslas – veikimo charakteristikų radimas. Charakteristikos:

- Paraiškų kiekis sistemoje
- Paraiškų kiekis laukimo eilėje
- Laikas per kurį paraiška yra aptarnaujama
- Laikas, kurį paraiška praleidžia eilėje
- Aptarnavimo punkto užimtumas
- Ir kt.

Nagrinėjant konkretų atvejį siekiama surasti charakteristikų analitinius arba empirinius skirstinius, arba jeigu pakanka bent jau vidurkį, dispersiją.

Šiame darbe apsiribosime paprastos aptarnavimo sistemos (kurią sudaro viena paraiškų laukimo eilė ir vienas aptarnavimo punktas) nagrinėjimu - $A/B/1/K$, kai $K = \infty$ - parametrą K praleisime.



1.3 pav. Aptarnavimo sistemos su viena laukimo eile ir vienu aptarnavimo punktu schema

Sistemos charakteristikų galima ieškoti įvairiais būdais:

- Atliekant sistemos veikimo imitaciją
  - Trūkumai: imitacinis modeliavimas gali būti labai reiklus skaičiavimo resursams, o dėl nekokybiškų atsitiktinių skaičių generavimo – gauti rezultatai gali būti nepatikimi
  - Privalumai: galima praktiškai tirti betkokią sistemą
- Charakteristikų reikšmes skaičiuojant analitiniu būdų
  - Trūkumai: ne visoms aptarnavimo sistemos yra išvesto charakteristikų analitinės formulės

- Privalumai: greitai apskaičiuojamos tikslios charakteristikų reikšmės

- Atliekant sistemos modeliavimą arba aproksimavimą Markovo tipo sistema
  - Trūkumai: reiklumas skaičiavimo resursams sparčiai auga bent kiek sudėtingesnėms sistemoms; tiksliai modeliuoti gali tik Markovo tipo sistemas, pvz.: $M/M/n/K$, $PTD/PTD/n/K$
  - Privalumai: ne Markovo tipo sistemas $G/G/n/K$ galima aproksimuoti Markovo tipo sistemomis $PTD/PTD/n/K$ kurių charakteristikos turėtų būti panašios į originalios sistemos charakteristikas; sistemos būsenų grafas gali padėti geriau pažinti sistemoje vykstančius procesus

Šiame darbe pagrindinis dėmesys skiriamas aptarnavimo sistemų modeliavimui Markovo tipo sistemomis.

Apsiribosime šių charakteristikų skaičiavimu:

- $E[L]$ – vidutinis laukimo eilėje laukiančių paraiškų kiekis

Įvedame pažymėjimus:

- atsitiktinis dydis ΔT_a – laiko trukmė tarp paraiškų atėjimų
- atsitiktinis dydis ΔT_b – laikas per kurį aptarnaujama paraiška
- L_{max} – maksimali laukimo eilės talpa ($L_{max} = K - 1$)
- $(k_1, k_2, \dots, k_N)$ – vektorius, kuris nurodo konkrečią sistemos būseną

1.5.1 M/M/1/Kaptarnavimo sistemos modeliavimas

Konstruosime tolydaus laiko Markovo grandinę aptarnavimo sistemai $M/M/1/K$, schema pateikta 1.3 pav. Laiko tarpai tarp paraiškų atėjimų turi eksponentinį skirstinį: $\Delta T_a \sim E(\lambda = \lambda_a)$, paraiškų aptarnavimo laikai taip pat pasiskirstę eksponentiškai: $\Delta T_b \sim E(\lambda = \lambda_b)$. Kadangi sistema daugiausiai gali talpinti K paraiškų (įskaitant ir aptarnaujamą) tai maksimalus laukimo eilės ilgis yra $L_{max} = K - 1$.

Atėjusi paraiška, jeigu aptarnavimo punktas yra tuščias, priimama aptarnavimui. Jeigu paraiška ateina tuo metu kai aptarnavimo punktas yra užimtas ir laukimo eilėje dar yra vietų – paraiška patalpinama laukimo eilės gale, priešingu atveju ji prarandama. Aptarnauta paraiška yra išleidžiama iš aptarnavimo punkto ir jeigu laukimo eilėje yra paraiškų, priima paraišką iš eilės pradžios aptarnavimui. Nei aptarnautų nei prarastų paraiškų kiekiai nėra fiksuojami, paraiškų šaltinis yra neišsenkantis.

Šios sistemos visas galimas būsenas galima identifikuoti indeksų vektoriumi $v = (k_1, k_2)$, kur $k_1 = 0, 1, \dots, L_{max}$– nurodo eilėje laukiančių paraiškų kiekį, $k_2 = 0, 1$ – nurodo ar aptarnavimo punktas aptarnauja paraišką (kai $k_2 = 0$ – aptarnavimo punktas neaptarnauja paraiškos, kai $k_2 = 1$ – aptarnavimo punktas aptarnauja paraišką).

Iš pradžių aptarnavimo punktas yra tuščias ($k_2 = 0$), o eilėje nėra laukiančių paraiškų ($k_1 = 0$), vadinasi pradinė Markovo grandinės būsena yra $(0,0)$. Kadangi tai vienintelė pradinė būsena, tai jos tikimybė lygi vienetui.

Šioje sistemoje gali įvykti du įvykiai: e_1) atėjo paraiška, e_2) paraiška aptarnauta. Vadovaujantis šiais įvykiais reikia surasti visas būsenas (bei intensyvumus, su kuriais jos yra pasiekiamos) iš tam tikros būsenos $\boldsymbol{v} = (k_1, k_2)$.

Įvykio e_1 metu sugeneruojamos būsenos surandamos pagal algoritmą:

(1) if $k_2 = 0$ *then*
(2) $V^* := V^* \cup \{(0,1)\}$,
(3) $\lambda^* := \lambda^* \cup \{\lambda_a\}$
(4) elseif $k_1 < L_{max}$ *then*
(5) $V^* := V^* \cup \{(k_1 + 1,1)\}$,
(6) $\lambda^* := \lambda^* \cup \{\lambda_a\}$
(7) endif

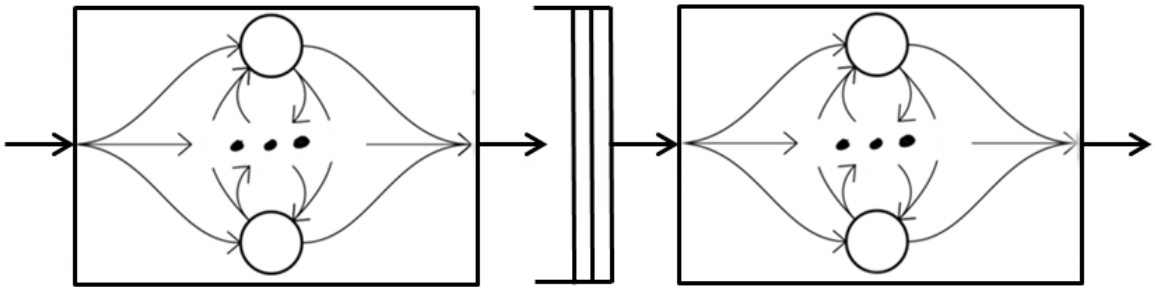
Įvykio e_2 metu sugeneruojamos būsenos surandamos pagal algoritmą:

(1) if $k_1 = 0$ *and* $k_2 = 1$ *then*
(2) $V^* := V^* \cup \{(0,0)\}$,
(3) $\lambda^* := \lambda^* \cup \{\lambda_b\}$
(4) elseif $k_1 > 0$ *then*
(5) $V^* := V^* \cup \{(k_1 - 1,1)\}$,
(6) $\lambda^* := \lambda^* \cup \{\lambda_b\}$
(7) endif

Informacija apie sukonstruotas būsenos (bei intensyvumai su kuriais jos yra pasiekiamos) yra toliau naudojama tolydžios Markovo grandinis konstravimo algoritme (žr. skyr. 1.4).

1.5.2PTD/PTD/1/Kaptarnavimo sistemos modeliavimas

Aptarnavimo sistemoje $PTD/PTD/1/K$ laiko tarpai tarp paraiškų atėjimų turi fazinį skirstinį $\Delta T_a \sim PTD\left(\boldsymbol{\pi}^{(a)}, T^{(a)}\right)$, kuris turi $m^{(a)}$ tarpines būsenas (fazes). Paraiškų aptarnavimo laikai taip pat yra pasiskirstę pagal fazinį skirstnį: $\Delta T_b \sim PTD\left(\boldsymbol{\pi}^{(b)}, T^{(b)}\right)$ su $m^{(b)}$ tarpinėmis būsenomis (fazėmis). Maksimali laukimo eilės talpa $L_{max} = K - 1$.Kai $m^{(a)} = 1$ ir $m^{(b)} = 1$ši sistema tampa sistema $M/M/1/K$. Aptarnavimo sistemos schema:



1.4 pav. Aptarnavimo PTD/PTD/1/K sistemos schema

Paraiškos prieš patekdamos į aptarnavimo punktą arba laukimo eilę (jeigu aptarnavimo punktas yra užimtas) turi praeiti pro fazinio skirstinio (a) būsenų grafą. Paraiškos į fazinio skirstinio (a) būsenų grafą patenka su begaliniu intensyvumu, t.y. tą pačią akimirką, kai paraiška palieka fazinio skirstinio (a) būsenų grafą į jį patenka nauja paraiška. Iš fazinio skirstinio (a) būsenų grafo išėjusi paraiška patenka į aptarnavimo punktą (kuriame yra patalpintas fazinio skirstinio (b) būsenų grafas), jeigu pastarasis yra neužimtas. Jeigu aptarnavimo punktas yra užimtas , o laukimo eilėje dar yra laisvų vietų paraiška patalpinama į eilės galą, priešingu atveju paraiška yra prarandama. Tuo tarpu kai paraiška praeina pro fazinio skirstinio (b) būsenų grafą laikoma, kad ji yra aptarnauta ir palieka sistemą. Jeigu laukimo eilėje yra laukiančių paraiškų, pirmoji paraišką laukianti eilėje priimama į aptarnavimo punktą.

Šios sistemos betkurią būseną galima nusakyti vektoriumi $\boldsymbol{v} = (k_1, k_2, k_3)$. Kur $k_1 \in \{0, m^{(a)}\}$ – nurodo kurioje fazinio skirstinio (a) būsenoje yra paraiška, jeigu $k_1 = 0$ – fazinio skirstinio (a) būsenų grafe paraiškos nėra; $k_2 \in \{0, L_{max}\}$ – nurodo laukimo eilėje esančių paraiškų kiekį; $k_3 \in \{0, m^{(b)}\}$ – nurodo kurioje fazinio skirstinio (b) būsenoje yra paraiška, jeigu $k_3 = 0$ – fazinio skirstinio (b) būsenų grafe paraiškos nėra.

Sistemoje yra galimi keturių tipų įvykiai:

e$_1$) Paraiška esanti fazinio skirstinio (a) būsenų grafe pereina iš vienos būsenos į kitą

e$_2$) Paraiška išeina iš fazinio skirstinio (a) būsenų grafo

e$_3$) Paraiška esanti fazinio skirstino (b) būsenų grafe pereina iš vienos būsenos į kitą

e$_4$) Paraiška išeina iš fazinio skirstinio (b) būsenų grafo

Sistemai startuojant viena paraiška jau yra fazinio skirstinio (a) būsenų grafe. Todėl sistemos pradinių būsenų kiekis yra $card(V^{init}) = \sum_{i=1}^{m^{(a)}} \mathbf{1}_{\{\pi_i^{(a)}>0\}}$. Į aibę V^{init}patenka visos būsenos $\forall i = 1, .., m^{(a)}(i, 0, 0)$, kurioms $\pi_i^{(a)} > 0$o šių būsenų pradinės tikimybės - tai atitinkamos $\pi_i^{(a)}$ reikšmės.

Reikia surasti visas pasiekiamas būsenas iš tam tikros būsenos, nusakytos koordinačių vektoriumi $\boldsymbol{v} = (k_1, k_2, k_3, k_4)$.

Įvykio e_1 metu sugeneruojamos būsenos surandamos pagal algoritmą:

(1) for i *from* 1 *to* $m^{(a)}$
(2) if $k_1 > 0, t_{k_1 i}^{(a)} > 0$ *then*
(3) $V^* := V^* \cup \{(i, k_2, k_3)\}$,
(4) $\lambda^* := \lambda^* \cup \left\{t_{k_1 i}^{(a)}\right\}$
(5) endif

Įvykio e_2 metu sugeneruojamos būsenos surandamos pagal algoritmą:

(1) if $k_3 = 0$ *then*
(3) for j *from* 1 *to* $m^{(b)}$
(4) for s *from* 1 *to* $m^{(a)}$
(5) if $k_1 > 0, t_{k_1}^{(a)} > 0, \pi_j^{(b)} > 0, \pi_s^{(a)} > 0$ *then*
(6) $V^* := V^* \cup \{(s, 0, j)\}$,
(7) $\lambda^* := \lambda^* \cup \left\{t_{k_1}^{(a)} \pi_j^{(b)} \pi_s^{(a)}\right\}$
(8) endif
(9) else
(10) for s *from* 1 *to* $m^{(a)}$
(11) if $k_1 > 0, t_{k_1}^{(a)} > 0, \pi_s^{(a)} > 0$ *then*
(12) $V^* := V^* \cup \{(s, k_2 + 1, k_3, k_4)\}$,
(13) $\lambda^* := \lambda^* \cup \left\{t_{k_1}^{(a)} \pi_s^{(a)}\right\}$
(14) endif
(15) endif

Įvykio e_3 metu sugeneruojamos būsenos surandamos pagal algoritmą:

(1) for i *from* 1 *to* $m^{(b)}$
(2) if $k_3 > 0, t_{k_3 i}^{(b)} > 0$ *then*
(3) $V^* := V^* \cup \{(k_1, k_2, i)\}$,
(4) $\lambda^* := \lambda^* \cup \left\{t_{k_3 i}^{(b)}\right\}$
(5) endif

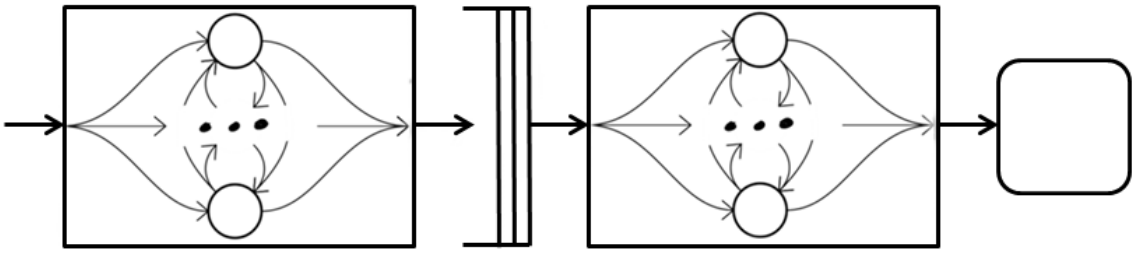
Įvykio e_4 metu sugeneruojamos būsenos surandamos pagal algoritmą:

(1) if $k_2 = 0$ *then*
(2) if $k_3 > 0, t_{k_3}^{(b)} > 0$ *then*
(3) $V^* := V^* \cup \{(k_1, 0, 0)\}$,
(4) $\lambda^* := \lambda^* \cup \left\{t_{k_3}^{(b)}\right\}$
(5) endif
(6) else
(7) for s *from* 1 *to* $m^{(b)}$
(8) if $k_3 > 0, t_{k_3}^{(b)} > 0, \pi_s^{(b)} > 0$ *then*
(9) $V^* := V^* \cup \{(k_1, k_2 - 1, s)\}$,
(10) $\lambda^* := \lambda^* \cup \left\{t_{k_3}^{(b)} \pi_s^{(b)}\right\}$
(11) endif
(12) endif

Sugeneruotų būsenų V^* aibė (bei juos atitinkantys intensyvumai) yra toliau naudojami konstruojant sistemos Markovo grandinę pagal anksčiau pateiktą algoritmą (žr. skyr. 1.4).

1.5.3 PTD/PTD/1/Kaptarnavimo sistemos modeliavimas (kai fiksuojamas aptarnautų paraiškų kiekis)

Nagrinėsime truputį pakeistą $PTD/PTD/1/K$ tipo sistemą (žr. 1.5.2), kurioje yra sekamas aptarnautų paraiškų kiekis. Beto, yra žinomas į sistemą ateisiančių paraiškų kiekis Ec. Sistemos schema:



pav. 1.4 PTD/PTD/1/Kaptarnavimo sistemos schema

Vėlgi kaip ir $PTD/PTD/1/K$ sistemoje laiko tarpai tarp paraiškų atėjimų turi fazinį skirstinį $\Delta T_a \sim PTD\left(\boldsymbol{\pi}^{(a)}, T^{(a)}\right)$, kuris turi $m^{(a)}$ tarpines būsenas/fazes. Paraiškų aptarnavimo laikai yra pasiskirstę pagal fazinį skirstnį: $\Delta T_b \sim PTD\left(\boldsymbol{\pi}^{(b)}, T^{(b)}\right)$ su $m^{(b)}$ tarpinėmis būsenomis/fazėmis. Maksimalus paraiškų kiekis sistemoje K nusako ateisiančių paraiškų kiekį, t.y. $K = Ec$. Maksimali laukimo eilės talpa $L_{max} = K - 1$. Šioje sistemoje yra negalimas paraiškos praradimas dėl perpildytos laukimo eilės.

Paraiška išėjusi iš fazinio skirstinio (a) būsenų grafo gali patekti į laukimo eilė (jeigu fazinio skirstinio (b) būsenų grafe yra kita paraiška) arba gali būtų priimama aptarnavimui (t.y. patenka į fazinio skirstinio (b) būsenų grafą). Jeigu ši paraiška nėra paskutinė (t.y. sistemoje esančių paraiškų kiekis neviršija Ec) tai fazinio skirstinio (a) grafe automatiškai atsiranda nauja paraiška. Aptarnauta paraišką keliauja į paraiškų talpyklą, ir jeigu dar yra paraiškų laukiančių eilėje, pirmoji priimama aptarnavimui. Šios sistemos esminė detalė yra ta, kad aptarnautos paraiškos nepalieka sistemos, jos yra saugomos talpykloje, jų kiekis fiksuojamas.

Visas galimas būsenas galima nusakyti koordinačių vektoriumi $\boldsymbol{v} = (k_1, k_2, k_3, k_4)$. Kur $k_1 \in \{0, m^{(a)}\}$ – nurodo kurioje fazinio skirstinio (a) būsenoje yra paraiška, jeigu $k_1 = 0$ – fazinio skirstinio (a) būsenų grafe paraiškos nėra; $k_2 \in \{0, L_{max}\}$ – nurodo laukimo eilėje esančių paraiškų kiekį; $k_3 \in \{0, m^{(b)}\}$ – nurodo kurioje fazinio skirstinio (b) būsenoje yra paraiška, jeigu $k_3 = 0$ – fazinio skirstinio (b) būsenų grafe paraiškos nėra; k_4 – nurodo aptarnautų paraiškų kiekį. Papildomai apibrėžiamas išvestinis dydis $c(\boldsymbol{v})$, kuris nurodo kiek sistemoje yra paraiškų:

$$c(\boldsymbol{v}) = \mathbf{1}_{\{k_1>0\}} + k_2 + \mathbf{1}_{\{k_3>0\}} + k_4$$

Sistemoje yra galimi keturi įvykiai:

e_1) Paraiška esanti fazinio skirstinio (a) būsenų grafe pereina iš vienos būsenos į kitą

e_2) Paraiška išeina iš fazinio skirstinio (a) būsenų grafo

e_3) Paraiška esanti fazinio skirstino (b) būsenų grafe pereina iš vienos būsenos į kitą

e_4) Paraiška išeina iš fazinio skirstinio (b) būsenų grafo

Sistemai startuojant viena paraiška jau yra fazinio skirstinio (a) būsenų grafe. Todėl sistemos pradinių būsenų kiekis yra $card\left(V^{init}\right) = \sum_{i=1}^{m^{(a)}} \mathbf{1}_{\left\{\pi_i^{(a)}>0\right\}}$. Į aibę V^{init} patenka visos būsenos $\forall i = 1,..,m^{(a)}(i,0,0,0)$, kurioms $\pi_i^{(a)} > 0$ o šių būsenų pradinės tikimybės - tai atitinkamos $\pi_i^{(a)}$ reikšmės.

Reikia surasti visas pasiekiamas būsenas iš tam tikros būsenos, nusakytos koordinačių vektoriumi$\boldsymbol{v} = (k_1, k_2, k_3, k_4)$.Kiekvienas įvykis savitai pakeičia sistemos būseną.

Įvykioe$_1$metu sugeneruojamos būsenos surandamos pagal algoritmą:

(1) for i *from* 1 *to* $m^{(a)}$
(2) if $k_1 > 0, t_{k_1 i}^{(a)} > 0$ *then*
(3) $V^* := V^* \cup \{(i, k_2, k_3, k_4)\}$,
(4) $\lambda^* := \lambda^* \cup \left\{t_{k_1 i}^{(a)}\right\}$
(5) endif

Įvykio e$_2$metu sugeneruojamos būsenos surandamos pagal algoritmą:

(1) if $k_3 = 0$ *then*
(2) *if* $c(\boldsymbol{v}) < E$ *then*
(3) *for* j *from* 1 *to* $m^{(b)}$
(4) *for* s *from* 1 *to* $m^{(a)}$
(5) *if* $k_1 > 0, t_{k_1}^{(a)} > 0, \pi_j^{(b)} > 0, \pi_s^{(a)} > 0$ *then*
(6) $V^* := V^* \cup \{(s, 0, j, k_4)\}$,
(7) $\lambda^* := \lambda^* \cup \left\{t_{k_1}^{(a)} \pi_j^{(b)} \pi_s^{(a)}\right\}$
(8) endif
(9) else
(10) for j *from* 1 *to* $m^{(b)}$
(11) *if* $k_1 > 0, t_{k_1}^{(a)} > 0, \pi_j^{(b)} > 0$ *then*
(12)$V^* := V^* \cup \{(0, 0, j, k_4)\}$,
(13)$\lambda^* := \lambda^* \cup \left\{t_{k_1}^{(a)} \pi_j^{(b)}\right\}$
(14) *endif*
(15) *endif*
(16) else
(17)if $c(\boldsymbol{v}) < E$ *then*
(18)for s *from* 1 *to* $m^{(a)}$
(19)if $k_1 > 0, t_{k_1}^{(a)} > 0, \pi_s^{(a)} > 0$ *then*
(20)$V^* := V^* \cup \{(s, k_2 + 1, k_3, k_4)\}$,
(21)$\lambda^* := \lambda^* \cup \left\{t_{k_1}^{(a)} \pi_s^{(a)}\right\}$
(22) *endif*
(23)else
(24)if $k_1 > 0, t_{k_1}^{(a)} > 0$ *then*
(25)$V^* := V^* \cup \{(0, k_2 + 1, k_3, k_4)\}$,
(26)$\lambda^* := \lambda^* \cup \left\{t_{k_1}^{(a)}\right\}$
(27)endif
(28) endif

Įvykio e_3 metu sugeneruojamos būsenos surandamos pagal algoritmą:

(1) for i *from* 1 *to* $m^{(b)}$
(2) if $k_3 > 0, t_{k_3 i}^{(b)} > 0$ *then*
(3) $V^* := V^* \cup \{(k_1, k_2, i, k_4)\}$,
(4) $\lambda^* := \lambda^* \cup \left\{t_{k_3 i}^{(b)}\right\}$
(5) endif

Įvykio $e_4$metu sugeneruojamos būsenos surandamos pagal algoritmą:

(1) if $k_2 = 0$ *then*
(2) if $k_3 > 0, t_{k_3}^{(b)} > 0$ *then*
(3) $V^* := V^* \cup \{(k_1, 0,0, k_4 + 1)\}$,
(4) $\lambda^* := \lambda^* \cup \left\{t_{k_3}^{(b)}\right\}$
(5) endif
(6) else
(7) for s from 1 *to* $m^{(b)}$
(8) if $k_3 > 0, t_{k_3}^{(b)} > 0, \pi_s^{(b)} > 0$ *then*
(9) $V^* := V^* \cup \{(k_1, k_2 - 1, s, k_4 + 1)\}$,
(10) $\lambda^* := \lambda^* \cup \left\{t_{k_3}^{(b)} \pi_s^{(b)}\right\}$
(11) endif
(12) endif

Sugeneruotų būsenų V^* aibė (bei juos atitinkantys intensyvumai) yra toliau naudojami konstruojant sistemos Markovo grandinės būsenų grafą pagal anksčiau pateiktą algoritmą (žr. skyr.1.4). Akivaizdu, kad ši sistema laikui bėgant pasiekia sugeriančią būseną $(0, 0,0, Ec)$. Galima nesunkiai įsitikinti, kad laikas per kurį pasiekiama sugeriančioji būsena turi fazinį skirstinį.Beto, pasinaudojant formule (1.9) galėsime surasti kaip laike kinta tikimybės, kad sistema bus tam tikroje būsenoje.

1.5.4 G/G/1/Kaptarnavimo sistemos modeliavimas

Ligi šiol aptarėme kaip modeliuojamos Markovo tipo aptarnavimo sistemos. Aptarnavimo sistema $G/G/1/K$ nėra Markovo tipo, tačiau ją galime aproksimuoti Markovo tipo sistema $PTD/PTD/1/K$. Tam reikia sistemoje esančius paraiškų atėjimo bei aptarnavimo laikų skirstinius aproksimuoti faziniais skirstiniais.

2 TIRIAMOJI DALIS

2.1 Aproksimavimas faziniais skirstiniais

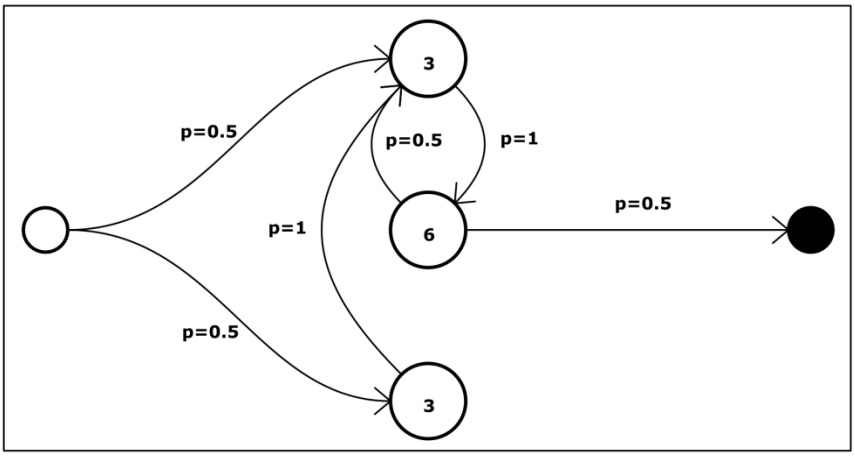
2.1.1Paieškos metodų palyginimas

Tirsime kaip trys paieškos metodai (RND, PSO, LUS) suranda fazinio skirstinio (kuris turi tris fazes) parametrus.Aproksimuosime tris skirstinius, kuriuos toliau darbe nurodysime didžiosiomis raidėmis A, B, C.Parametrai ir charakteristikos yra pateiktos lentelėje 2.1.

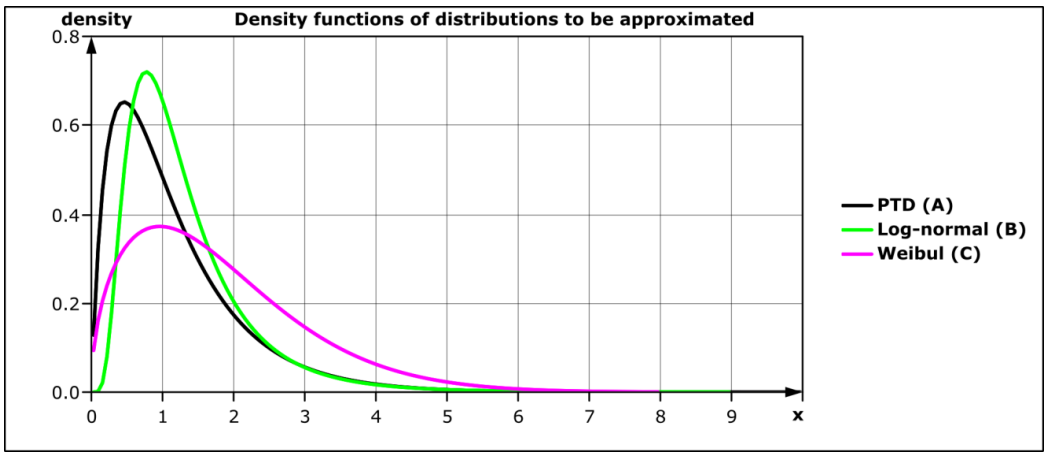
2.1 lentelė. Aproksimuojamų skirstinių parametrai, charakteristikos.

Skirstinio pavadinimas	Tankio funkcija / Parametrai	Skirstinio charakteristikos
Fazinis skirstinys (A)	$f_A(x) = \pi e^{xT} t, x > 0$ $\pi = [0.5, 0, 0.5]$ $T = \begin{bmatrix} -3 & 3 & 0 \\ 3 & -6 & 0 \\ 3 & 0 & -3 \end{bmatrix}, t = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}$	$E(X_A) = 1.167$ $Std(X_A) = 0.928$ $cv(X_A) = 0.795$
Log-normalus skirstinys (B)	$f_B(x) = \frac{1}{\alpha x \sqrt{2\pi}} e^{-\frac{(ln(x)-\lambda)^2}{2\alpha^2}}$, $x > 0$ $\alpha = 0.6;\ \lambda = 0.1$	$E(X_B) = 1.323$ $Std(X_B) = 0.871$ $cv(X_B) = 0.658$
Veibulo skirstinys (C)	$f_C(x) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-\left(\frac{x}{\lambda}\right)^k}$ $\lambda = 2; \mathrm{k} = 1.5$	$E(X_C) = 1.805$ $Std(X_C) = 1.226$ $cv(X_C) = 0.679$

Fazinio skirstinio (A) struktūrą:



2.1 pav. Fazinio skirstinio (A) struktūra (struct-3-6-25)



2.2 pav. Aproksimuojamų skirstinių tankio funkcijos.

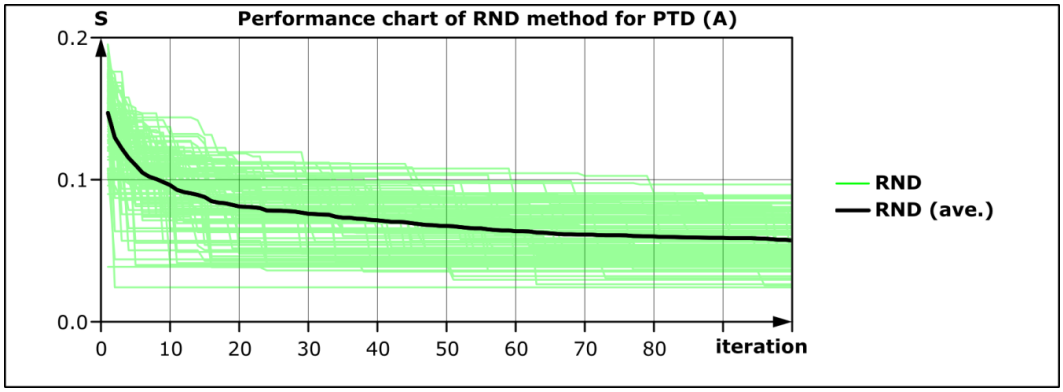
Skirstinius aproksimuosime faziniais skirstiniais. Ieškosime tokių fazinio skirstinio parametrų su kuriais yra minimizuojamas protas tarp skirstinių tankių funkcijų, t.y. minimizuosime statistiką $\hat{S}(\Delta x, x_{end})$. Priklausomai nuo diskretizavimo parametrų $(\Delta x, x_{end})$, diskretizuoto skirstinio vidurkis bei standartinis nuokrypis nutolsta nuo tikrųjų momentų reikšmių. Diskretizavimo parametrus kiekvienam skirstiniui parenkame individualiai, taip, kad vidurkio (standartinio nuokrypio)reikšmė sutaptų su originalia reikšme bent dviem (vienu) skaitmenimis po kablelio.

2.2 lentelė. Aproksimuojamų skirstinių diskretizavimo parametrai

Skirstinio pavadinimas	ParametrasΔx	Parametrasx_{end}	Vidurkis	Standartinis nuokrypis
Fazinis skirstinys (A)	0.125	7	$E(X_A) = 1.167$ $E(\hat{X}_A) = 1.163$	$Std(X_A) = 0.928$ $Std(\hat{X}_A) = 0.918$
Log-normalus skirstinys (B)	0.0625	9	$E(X_B) = 1.323$ $E(\hat{X}_B) = 1.321$	$Std(X_B) = 0.871$ $Std(\hat{X}_B) = 0.859$
Veibulo skirstinys (C)	0.0625	8	$E(X_C) = 1.805$ $E(\hat{X}_C) = 1.803$	$Std(X_C) = 1.226$ $Std(\hat{X}_C) = 1.220$

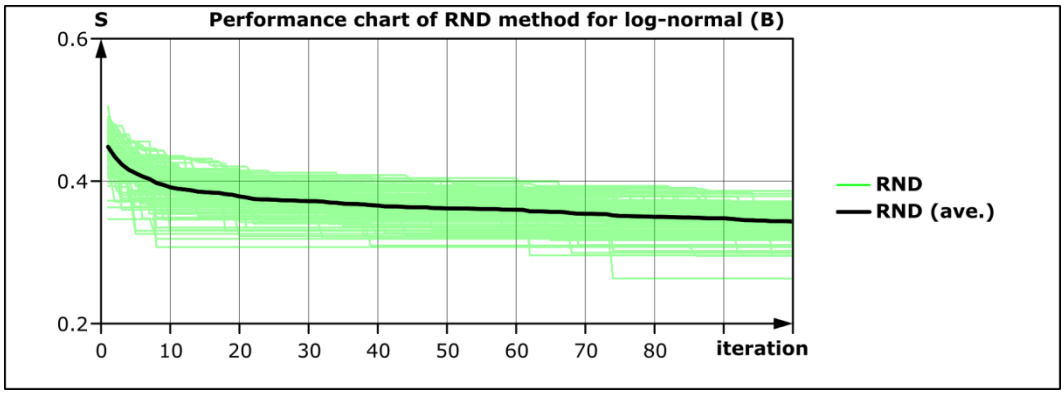
Tiriami paieškos metodai atliks po 100 iteracijų. Kievkienos iteracijos metu bus atlikta 10000 statistikos $\hat{S}(\Delta x, x_{end})$ apskaičiavimų. Kiekvieną paieškos metodą paleisime po 100 kartų, analizuosime gautus rezultatus.

Visų pirma pasižiūrėsime, kaip veikia atsitiktinės paieškos metodas RND.

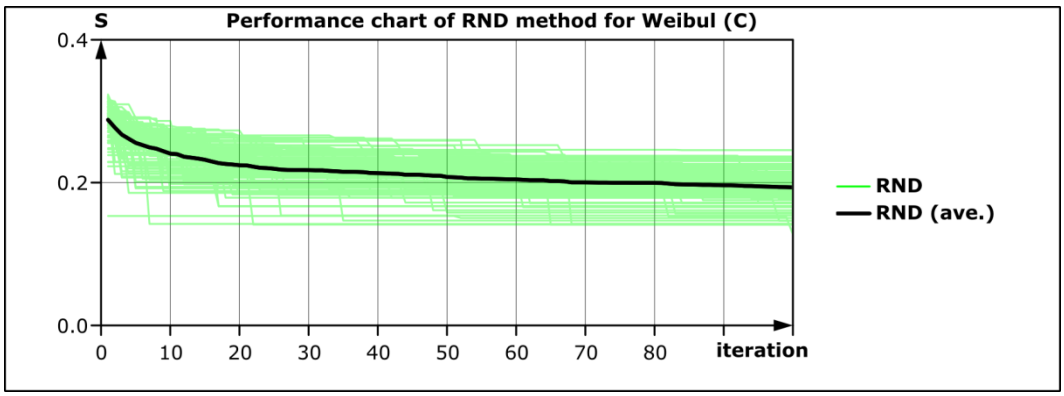


2.3 pav. RND metodo veikimas, aproksimuojant fazinį skirtinį (A).

Pastaba: žalsvos linijos vaizduoja atskirus testus, juoda linija –vidurkį.



2.4 pav. RND metodo veikimas, aproksimuojant log-normalųjį skirtinį (B).



2.5 pav. RND metodo veikimas, aproksimuojant Veibulo skirtinį (C).

2.3 lentelė. Skirstinių aproksimavimo rezultatai, kai naudojamas RND metodas

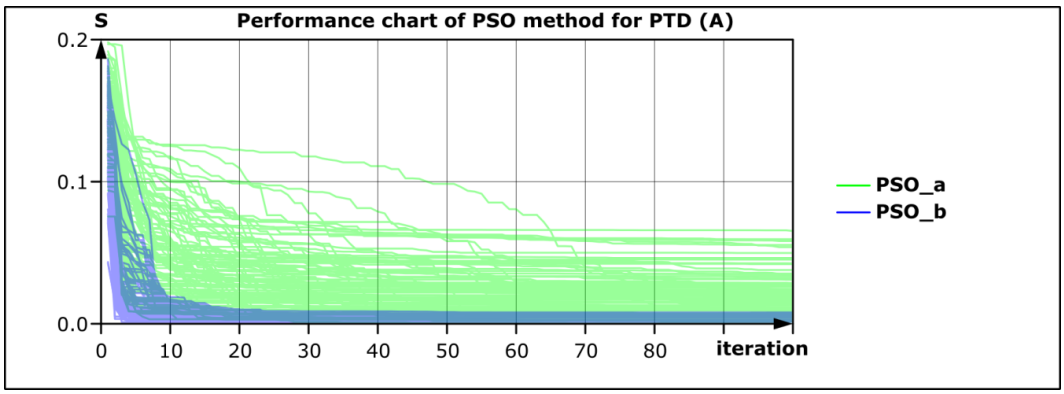
Statistika	Fazinis skirstinys (A)	Log-normalus skirstinys (B)	Veibulo skirstinys (C)
$E[\hat{S}]$	$5.7231 \cdot 10^{-2}$	$3.4329 \cdot 10^{-1}$	$1.9339 \cdot 10^{-1}$
$Std[\hat{S}]$	$2.6345 \cdot 10^{-2}$	$5.1938 \cdot 10^{-2}$	$5.8478 \cdot 10^{-2}$
$min[\hat{S}]$	$2.4211 \cdot 10^{-2}$	$2.633 \cdot 10^{-1}$	$1.2747 \cdot 10^{-1}$
$max[\hat{S}]$	$9.6606 \cdot 10^{-2}$	$3.862 \cdot 10^{-1}$	$2.4546 \cdot 10^{-1}$
Sprendinių struktūros	(struct-3-12-1)x100	(struct-3-12-1)x100	(struct-3-12-1)x100

Toliau fazinio skirstinio parametrų ieškosime su PSO algoritmu. Išmėginsime duparametrų rinkinius, palyginsime rezultatus ir nuspręsime kuris yra geresnis.

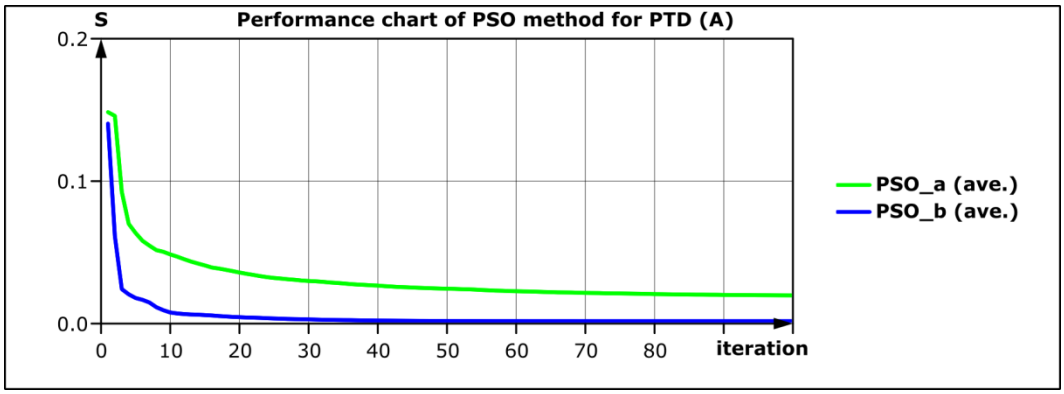
2.4 lentelė. PSO metodų parametrai

Metodas	Parametras W	Parametras C_1	Parametras C_2
PSO_a	0.6	0.4	0.4
PSO_b	0.9→0.4	0.1	0.5

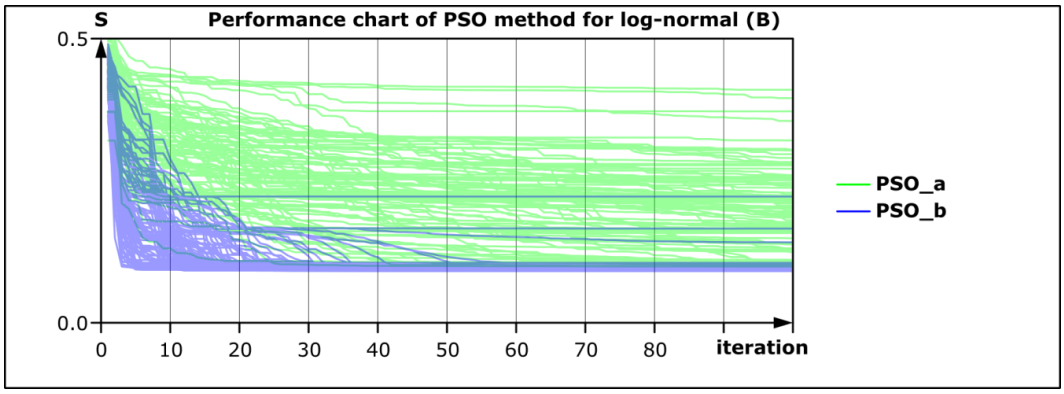
Pastaba: užrašas „0.9→0.4“ reiškia, kad parametras W tolygiai kinta nuo 0.9 iki 0.4 paieškos metodo eigoje.



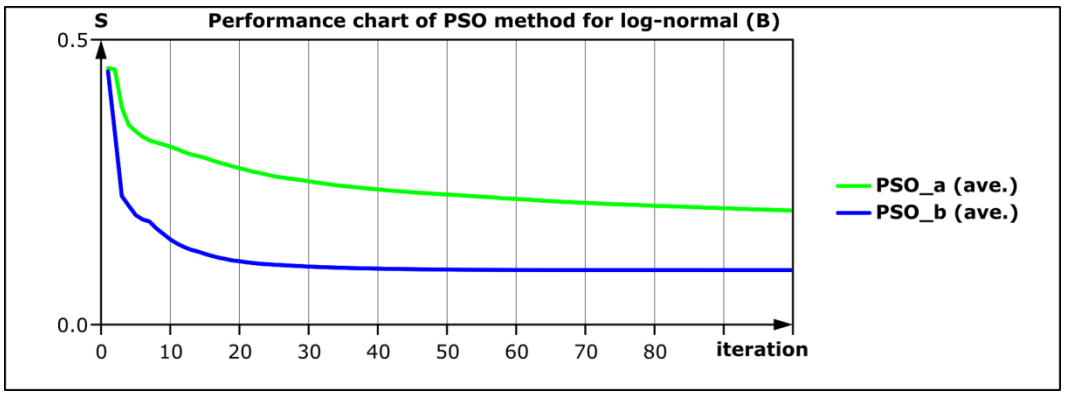
2.6 pav. PSO metodo veikimas, aproksimuojant fazinį skirtinį (A), individualios paieškos.



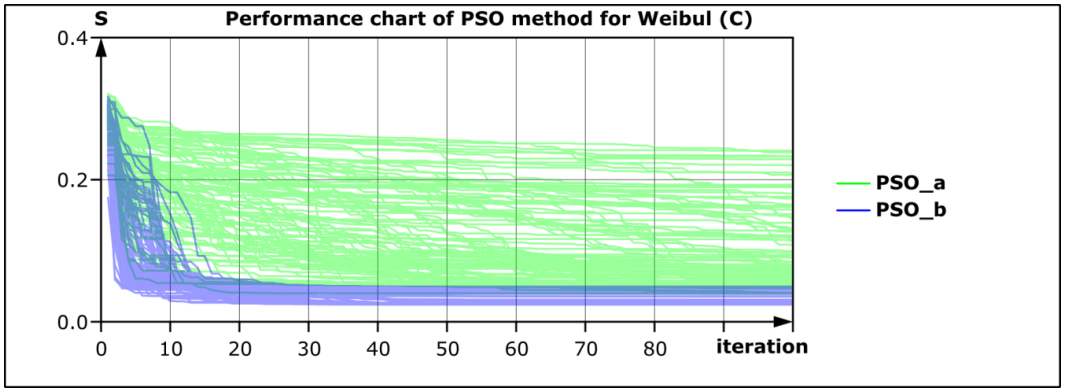
2.7 pav. PSO metodo veikimas, aproksimuojant fazinį skirtinį (A), suvidurkintas rezultatas.



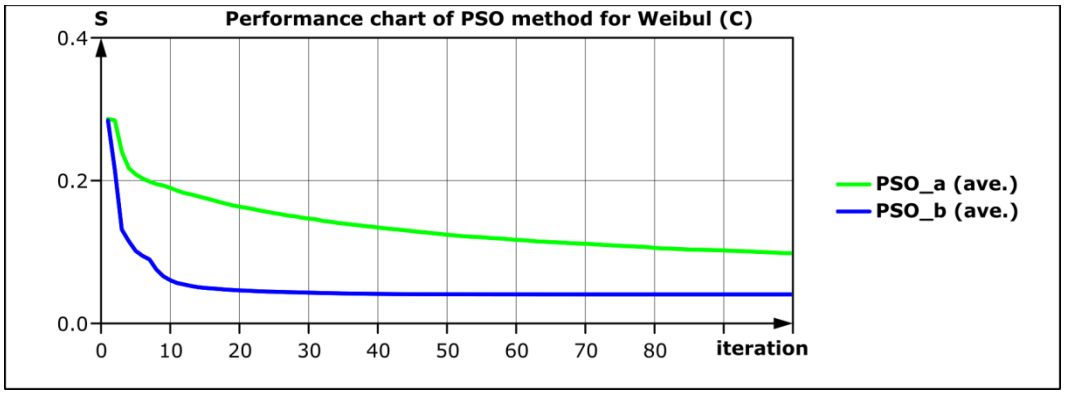
2.8 pav. PSO metodo veikimas, aproksimuojant log-normalųjį skirstinį (B), individualios paieškos.



2.9 pav. PSO metodo veikimas, aproksimuojant log-normalųjį skirstinį (B), suvidurkintas rezultatas.



2.10 pav. PSO metodo veikimas, aproksimuojant Veibulo skirstinį (C), individualios paieškos.



2.11 pav. PSO metodo veikimas, aproksimuojant Veibulo skirstinį (C), suvidurkintas rezultatas.

2.5 lentelė. Skirstinių aproksimavimo rezultatai, kai naudojamas PSO_a metodas

Statistika	Fazinis skirstinys (A)	Log-normalus skirstinys (B)	Veibulo skirstinys (C)
$E[\hat{S}]$	$1.9798 \cdot 10^{-2}$	$2.0027 \cdot 10^{-1}$	$9.8124 \cdot 10^{-2}$
$Std[\hat{S}]$	$2.2195 \cdot 10^{-2}$	$4.5508 \cdot 10^{-1}$	$3.0106 \cdot 10^{-1}$
$min[\hat{S}]$	$3.64 \cdot 10^{-4}$	$9.8806 \cdot 10^{-2}$	$3.9967 \cdot 10^{-2}$
$max[\hat{S}]$	$6.5301 \cdot 10^{-2}$	$4.0994 \cdot 10^{-1}$	$2.4074 \cdot 10^{-1}$
Sprendinių	(struct-3-12-1)x62[1]	(struct-3-4-1)x16	(struct-3-12-1)x25

[1] Čia pateikiama informacija apie tas struktūras, kurios buvo surastos daugiau kaip dešimt kartų

struktūros		(struct-3-5-3)x13 (struct-3-5-9)x11 (struct-3-7-21)x13	

2.6 lentelė. Skirstinių aproksimavimo rezultatai, kai naudojamas PSO_b metodas

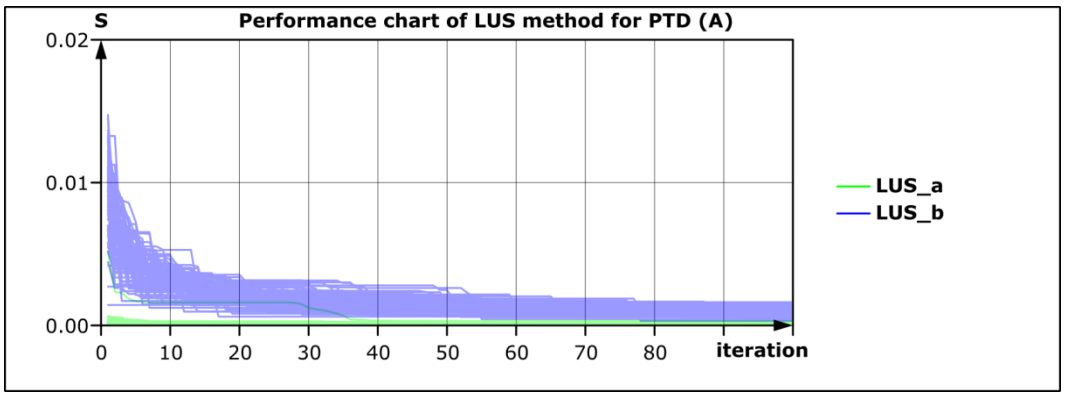
Statistika	Fazinis skirstinys (A)	Log-normalus skirstinys (B)	Veibulo skirstinys (C)
$E[\hat{S}]$	$1.6743 \cdot 10^{-3}$	$9.5556 \cdot 10^{-2}$	$4.058 \cdot 10^{-2}$
$Std[\hat{S}]$	$2.3516 \cdot 10^{-4}$	$2.4452 \cdot 10^{-2}$	$8.7229 \cdot 10^{-3}$
$min[\hat{S}]$	$9.2 \cdot 10^{-5}$	$9.1517 \cdot 10^{-2}$	$2.4276 \cdot 10^{-2}$
$max[\hat{S}]$	$7.999 \cdot 10^{-3}$	$2.2201 \cdot 10^{-1}$	$4.9295 \cdot 10^{-2}$
Sprendinių struktūros	(struct-3-8-49)x10 (struct-3-8-55)x12 (struct-3-9-37)x12 (struct-3-12-1)x26	(struct-3-5-13)x86	(struct-3-9-4)x10 (struct-3-10-1)x14

Iš rezultatų matosi (lent. 2.5, 2.6), kad PSO_b metodas konverguoja greičiau, ir duoda geresnius vidutinius rezultatus. Toliau patyrinėsime kaip veikia lokalia paieška pagrįstas metodas LUS, kurio parametrai pateikti lent. 2.7:

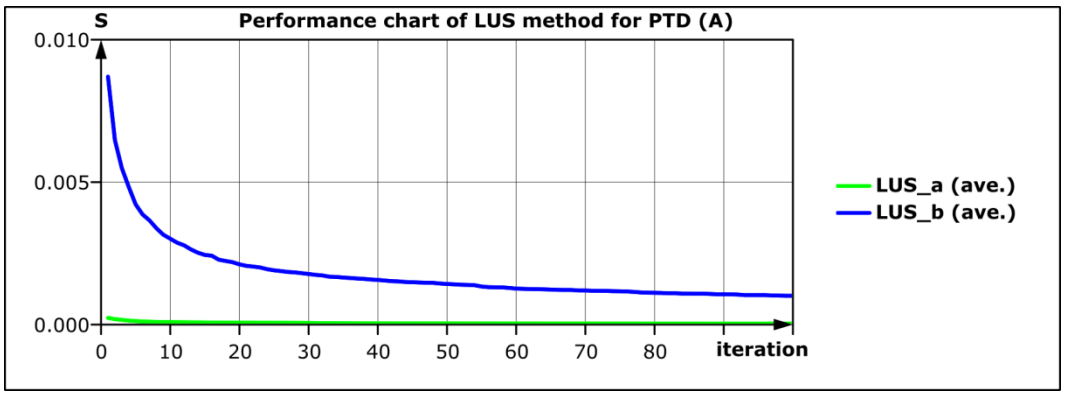
2.7 lentelė. LUS metodų parametrai

Metodas	Parametras β	Dalelių kiekis	Tarpinių iteracijų kiekis
LUS_a	1.0	10	1000
LUS_b	$1/3$	10	1000

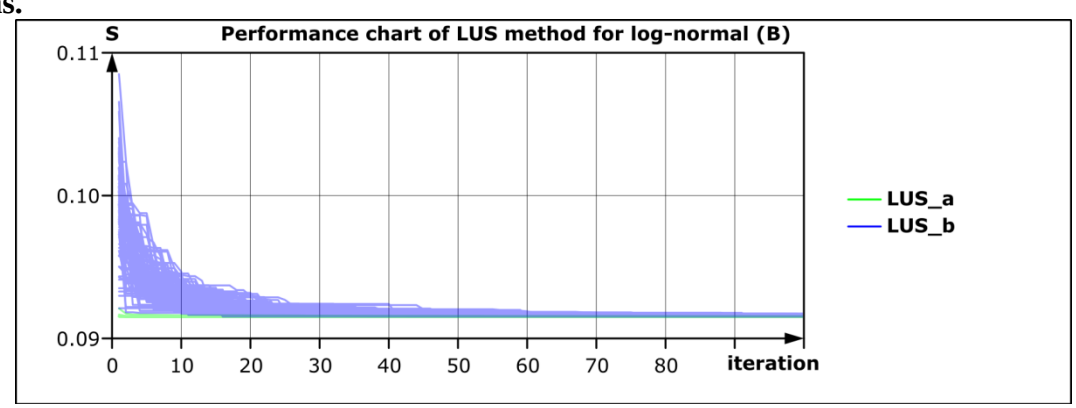
Rezultatai:



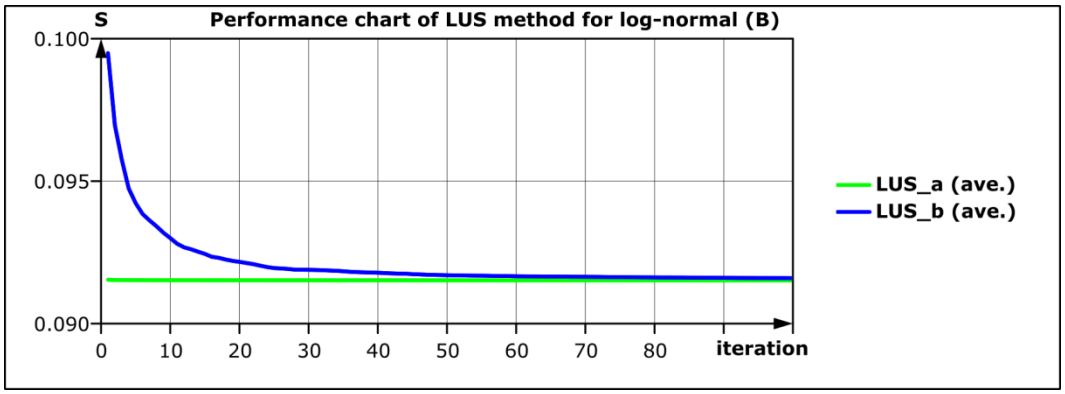
2.12 pav. LUS metodo veikimas, aproksimuojant fazinį skirtinį (A), individualios paieškos.



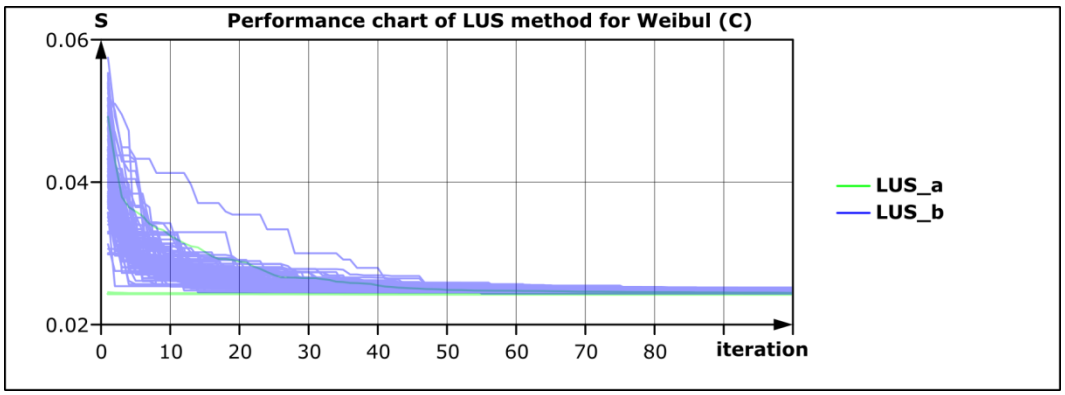
2.13 pav. LUS metodo veikimas, aproksimuojant fazinį skirtinį (A), suvidurkintas rezultatas.



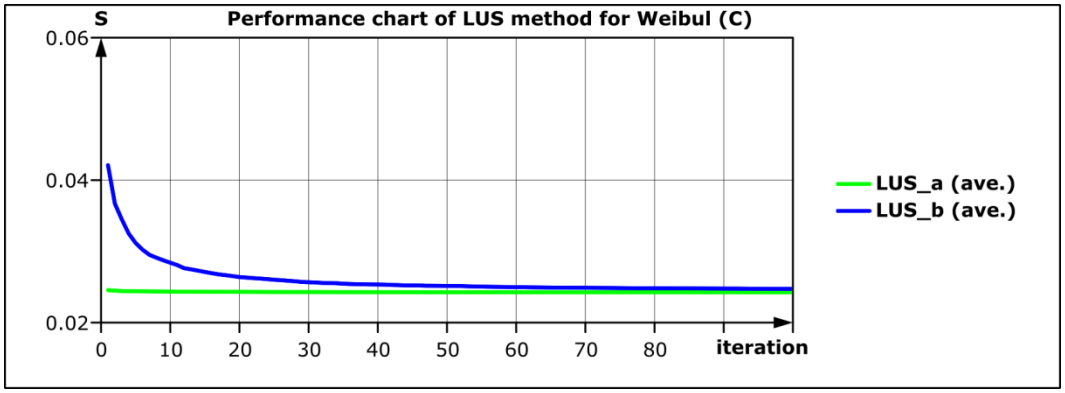
2.14 pav. LUS metodo veikimas, aproksimuojant log-normalųjį skirstinį (B), individualios paieškos.



2.15 pav. LUS metodo veikimas, aproksimuojant log-normalųjį skirstinį (B), suvidurkintas rezultatas.



2.16 pav. LUS metodo veikimas, aproksimuojant Veibulo skirtinį (C), individualios paieškos.



2.17 pav. LUS metodo veikimas, aproksimuojant Veibulo skirtinį (C), suvidurkintas rezultatas.

2.8 lentelė. Skirstinių aproksimavimo rezultatai, kai naudojamas LUS_a metodas

Statistika	Fazinis skirstinys (A)	Log-normalus skirstinys (B)	Veibulo skirstinys (C)
$E[\hat{S}]$	$3.466 \cdot 10^{-5}$	$9.1518 \cdot 10^{-2}$	$2.4264 \cdot 10^{-2}$
$Std[\hat{S}]$	$2.6773 \cdot 10^{-7}$	$1.0628 \cdot 10^{-9}$	$3.4554 \cdot 10^{-8}$
$min[\hat{S}]$	$2 \cdot 10^{-6}$	$9.1517 \cdot 10^{-2}$	$2.4261 \cdot 10^{-2}$
$max[\hat{S}]$	$3.1 \cdot 10^{-4}$	$9.1547 \cdot 10^{-2}$	$2.4447 \cdot 10^{-2}$
Sprendinių struktūros	(struct-3-9-37)x11 (struct-3-9-39)x33 (struct-3-10-13)x29	(struct-3-5-13)x100	(struct-3-8-10)x11 (struct-3-9-2)x79

2.9 lentelė. Skirstinių aproksimavimo rezultatai, kai naudojamas LUS_b metodas

Statistika	Fazinis skirstinys (A)	Log-normalus skirstinys (B)	Veibulo skirstinys (C)
$E[\hat{S}]$	$1.0111 \cdot 10^{-3}$	$9.1595 \cdot 10^{-2}$	$2.4739 \cdot 10^{-2}$
$Std[\hat{S}]$	$6.7244 \cdot 10^{-6}$	$2.1043 \cdot 10^{-7}$	$2.6798 \cdot 10^{-6}$
$min[\hat{S}]$	$3.25 \cdot 10^{-4}$	$9.1523 \cdot 10^{-2}$	$2.435 \cdot 10^{-2}$
$max[\hat{S}]$	$1.622 \cdot 10^{-3}$	$9.1734 \cdot 10^{-2}$	$2.5208 \cdot 10^{-2}$
Sprendinių struktūros	(struct-3-8-49)x23 (struct-3-8-55)x11 (struct-3-9-37)x17 (struct-3-9-39)x17	(struct-3-5-13)x100	(struct-3-6-8)x16 (struct-3-7-6)x15 (struct-3-7-17)x20 (struct-3-8-10)x19

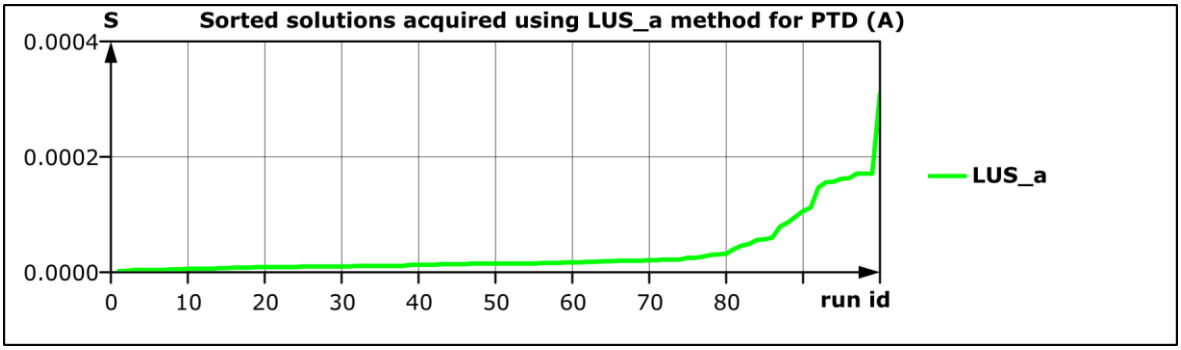
2.10 lentelė. PSO ir LUS metodų palyginimas

Metodas	Fazinis skirstinys (A), $E[\hat{S}], (Std[\hat{S}])$	Log-normalus skirstinys (B), $E[\hat{S}], (Std[\hat{S}])$	Veibulo skirstinys (C), $E[\hat{S}], (Std[\hat{S}])$
PSO_a	$1.9798 \cdot 10^{-2}$, $(2.2195 \cdot 10^{-2})$	$2.0027 \cdot 10^{-1}$, $(4.5508 \cdot 10^{-1})$	$9.8124 \cdot 10^{-2}$, $(3.0106 \cdot 10^{-1})$
PSO_b	$1.6743 \cdot 10^{-3}$, $(2.3516 \cdot 10^{-4})$	$9.5556 \cdot 10^{-2}$, $(2.4452 \cdot 10^{-2})$	$4.0580 \cdot 10^{-2}$, $(8.7229 \cdot 10^{-3})$
LUS_a	$3.466 \cdot 10^{-5}$, $(2.6773 \cdot 10^{-7})$	$9.1518 \cdot 10^{-2}$, $(1.0628 \cdot 10^{-9})$	$2.4264 \cdot 10^{-2}$, $(3.4554 \cdot 10^{-8})$
LUS_b	$1.0111 \cdot 10^{-3}$, $(6.7244 \cdot 10^{-6})$	$9.1595 \cdot 10^{-2}$, $(2.1043 \cdot 10^{-7})$	$2.4739 \cdot 10^{-2}$, $(2.6798 \cdot 10^{-6})$

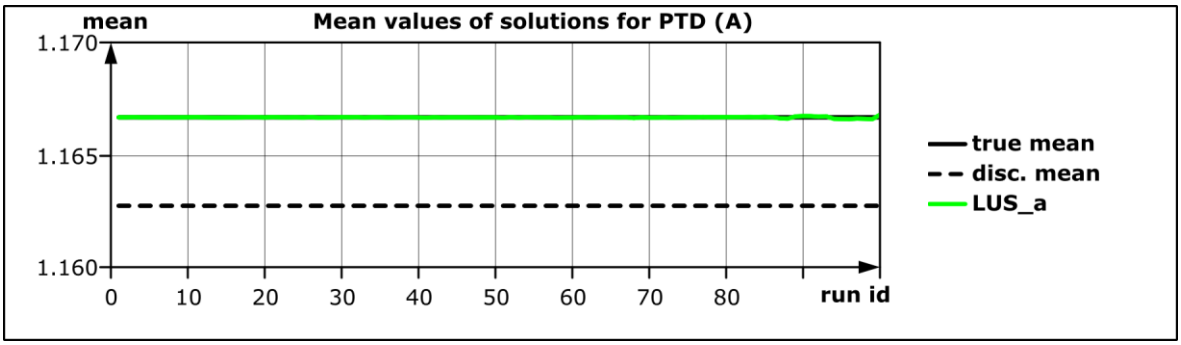
Geriausias metodas yra LUS_a, kuris konverguoja labai greitai, jau po pirmos iteracijos gaunami labai geri rezultatai.

Išvados: Šiame tyrime nustatėme, kad LUS paieškos metodas yra pranašesnis tiek už atsitiktinę paiešką tiek už PSO metodą. Kadangi buvo išbandyti tik du LUS metodo parametrų rinkiniai toliau galima atlikti detalesnį šio metodo tyrimą ir nustatyti optimalius parametrus, bei atsakyti į klausimą, kaip stipriai jie priklauso nuo aproksimuojamos funkcijos.

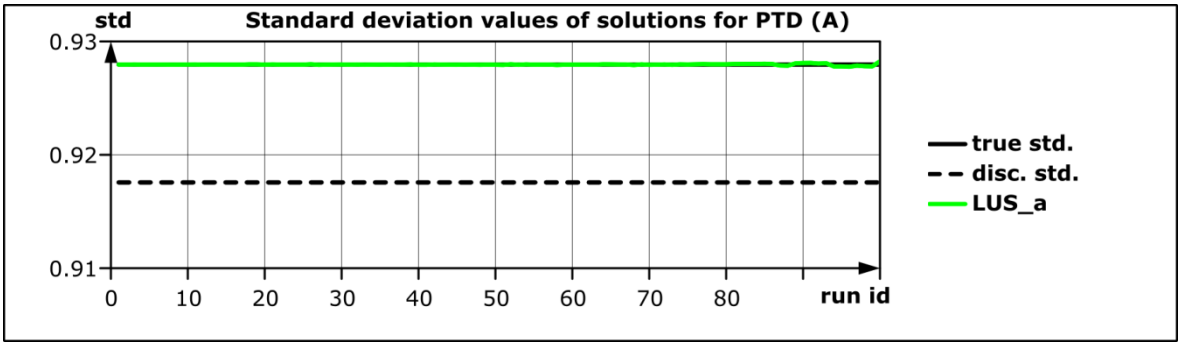
Dabar pasižiūrėsime, kaip atrodo LUS_a metodu gauti sprendiniai.



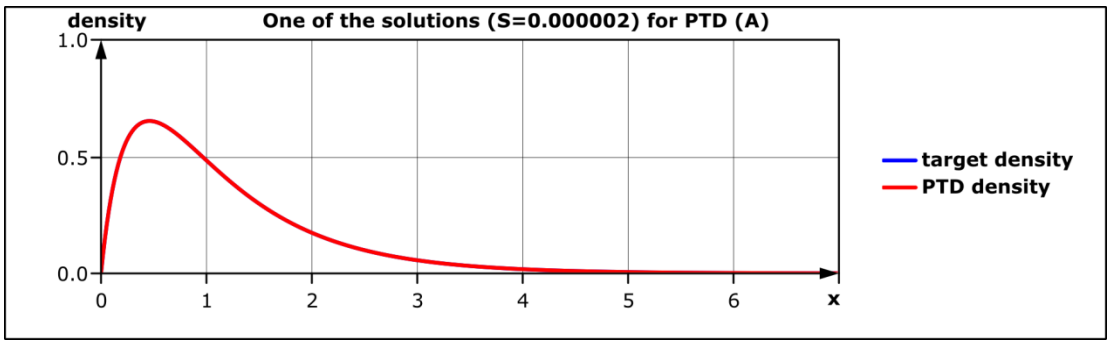
2.18 pav. Sprendiniai gauti naudojant LUS_a metodą, aproksimuojant fazinį skirstinį (A). Sprendiniai išrikiuoti tikslo funkcijos S didėjimo tvarka.



2.19 pav. Teorinio, aproksimuoto ir surastų (naudojant LUS_a metodą, aproksimuojant fazinį skirstinį (A)) skirstinių vidurkiai.



2.20 pav. Teorinio, aproksimuoto ir surastų (naudojant LUS_a metodą, aproksimuojant fazinį skirstinį (A)) skirstinių standartiniai nuokrypiai.



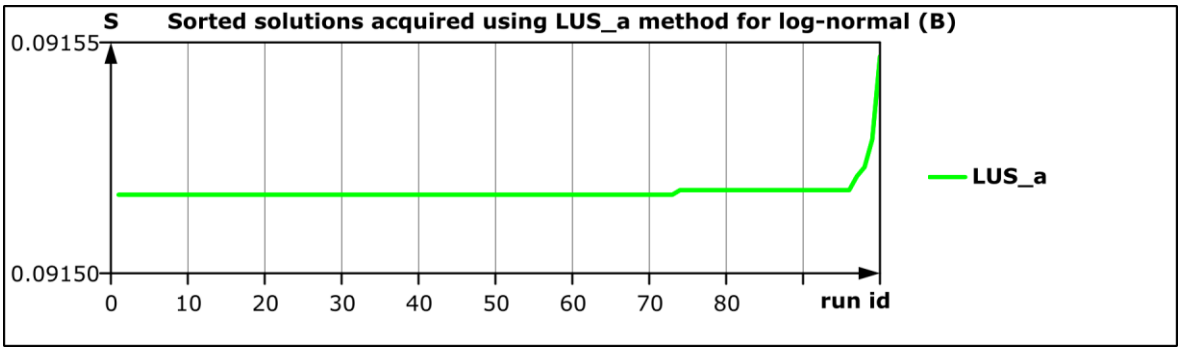
2.21 pav. Fazinio skirstinio (A) ir geriausio sprendinio(gauto, aproksimuojant fazinį skirstinį (A) su LUS_a metodu) tankių funkcijos.

Fazinio skirstinio, kurio tankio funkcija pavaizduota 2.21 pav., parametrai:

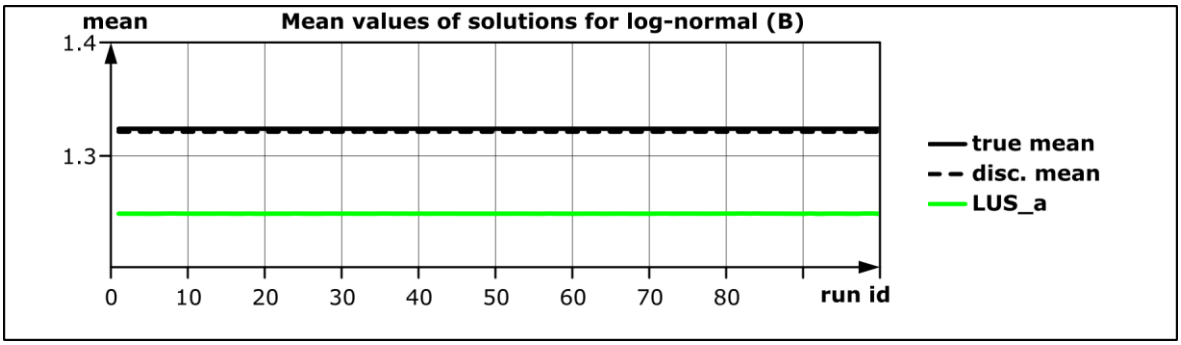
$$\boldsymbol{\pi} \approx [0.597, 0, 0.403], T \approx \begin{bmatrix} -2.175 & 1.138 & 1.037 \\ 0.006 & -2.875 & 0.366 \\ 4.176 & 2.775 & -6.951 \end{bmatrix}$$

Čia juoda ištisinė linija žymi teorinį aproksimuojamo skirstinio vidurkį/standartinį nuokrypį, punktyrinė – diskretizuoto tankio funkcijos vidurkį/standartinį nuokrypį. Žalia linija, rodo sprendinių, surikiuotų statistikos $\widehat{\boldsymbol{S}}$ didėjimo tvarka, atitinkamai vidurkį ir standartinį nuokrypį.

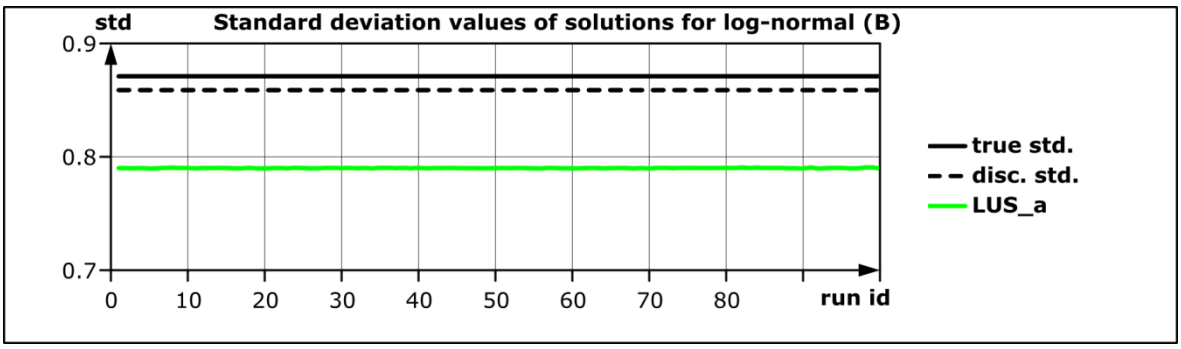
Aiškiai matosi, kad aproksimuodami fazinį skirstinį (A) buvo surasti faziniai skirstiniai, kurių du pirmieji momentai buvo pritraukti prie teorinių aproksimuojamo skirstinio momentų. Dar galima pastebėti, kad daugeliu atveju LUS_a metodu surastų sprendinių struktūros nesutapo su aproksimuojamo fazinio skirstinio struktūra.



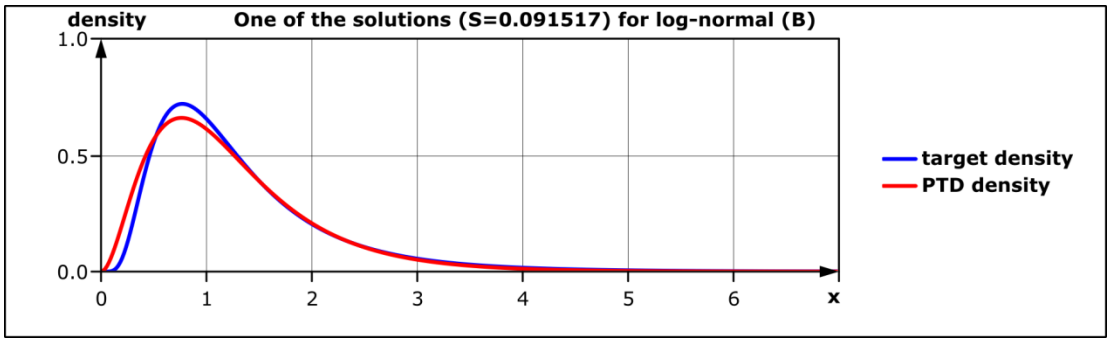
2.22 pav. Sprendiniai gauti naudojant LUS_a metodą, aproksimuojant log-normalųjį skirstinį (B)). Sprendiniai išrikiuoti tikslo funkcijos S didėjimo tvarka.



2.23 pav. Teorinio, aproksimuoto ir surastų (naudojant LUS_a metodą, aproksimuojant log-normalųjį skirstinį (B)). skirstinių vidurkiai.



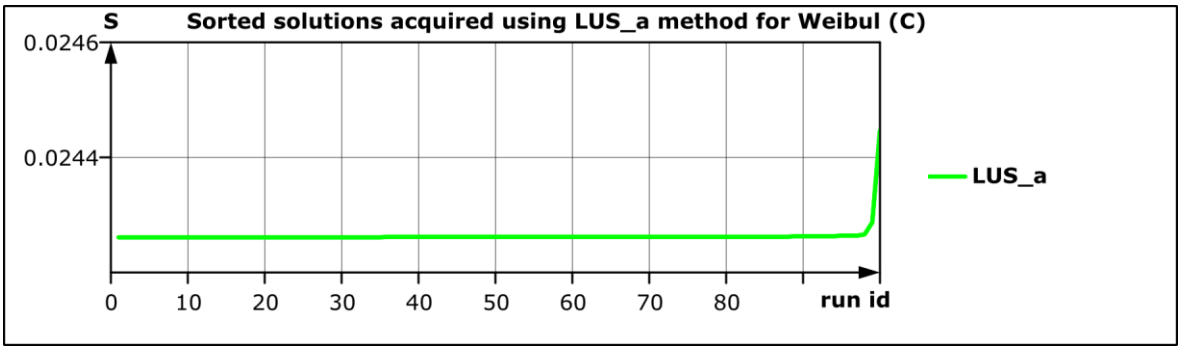
2.24 pav. Teorinio, aproksimuoto ir surastų (naudojant LUS_a metodą, aproksimuojant log-normalųjį skirstinį (B)). skirstinių standartiniai nuokrypiai.



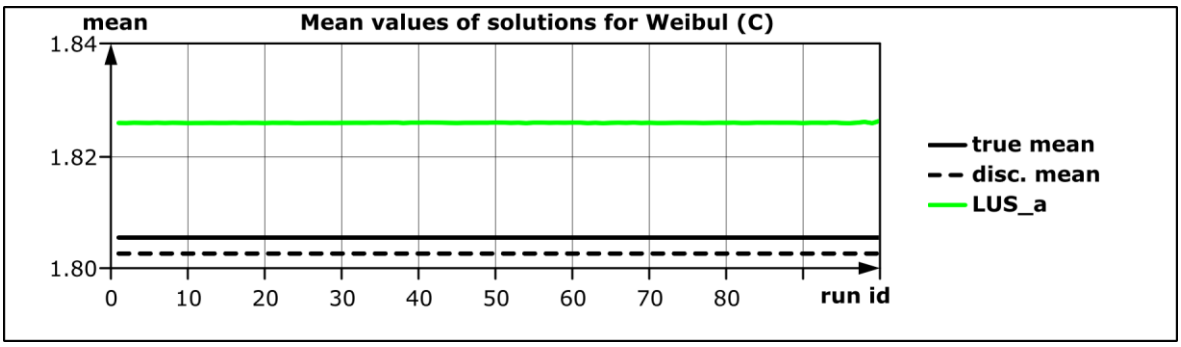
2.25 pav. Log-normaliojo skirstinio (B) ir geriausio surasto fazinio skirstinio (gauto, aproksimuojant log-normalųjį skirstinį (B)) tankių funkcijos.

Fazinio skirstinio, kurio tankio funkcija pavaizduota 2.25 pav., parametrai:

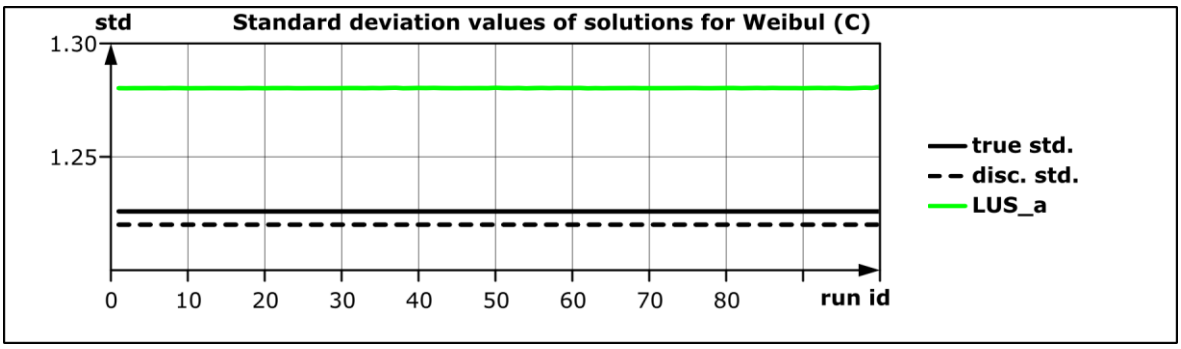
$$\boldsymbol{\pi} \approx [0, 1, 0], T \approx \begin{bmatrix} -2.673 & 0 & 2.673 \\ 2.676 & -2.676 & 0 \\ 0 & 0.272 & -2.679 \end{bmatrix}$$



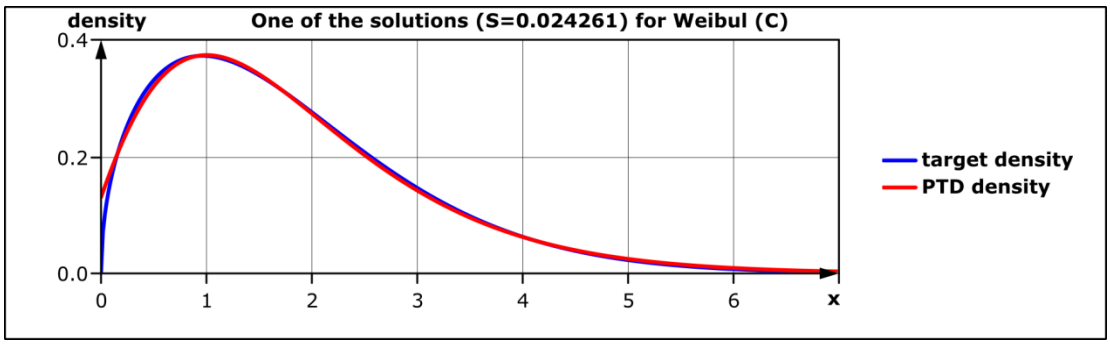
2.26 pav. Sprendiniai gauti naudojant LUS_a metodą, aproksimuojant Veibulo skirstinį (C)). Sprendiniai išrikiuoti tikslo funkcijos S didėjimo tvarka.



2.27 pav. Teorinio, aproksimuoto ir surastų (naudojant LUS_a metodą, aproksimuojant Veibulo skirstinį (C)). skirstinių vidurkiai.



2.28 pav. Teorinio, aproksimuoto ir surastų (naudojant LUS_a metodą, aproksimuojant Veibulo skirstinį (C)). skirstinių standartiniai nuokrypiai.



2.29 pav. Veibulo skirstinio (C) ir geriausio surasto fazinio skirstinio (gauto, aproksimuojant Veibulo skirstinį (C)) tankių funkcijos.

Fazinio skirstinio, kurio tankio funkcija pavaizduota 2.29 pav., parametrai:

$$\boldsymbol{\pi} \approx [0.264, 0.648, 0.087], T \approx \begin{bmatrix} -1.321 & 0 & 1.296 \\ 1.042 & -1.317 & 0.261 \\ 0 & 0 & -1.314 \end{bmatrix}$$

Išvados: Akivaizdu, kad ne kiekviena tankio funkcija gali būti sėkmingai aproksimuojama faziniu skirstiniu. Todėl yra tikslinga ištirti kaip aproksimuojamų skirstinių tankių funkcijų forma įtakoja aproksimavimo tikslumą.

2.1.2 Veibulo skirstinių aproksimavimas trijų būsenų faziniais skirstiniais

Tirsime kaip faziniai skirstiniai (turintys tris fazes) aproksimuoja įvairius Veibulo skirstinius. Veibulo skirstino tankio funkcija:

$$f_W(x;\ \lambda, k) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{\boldsymbol{k-1}} e^{-\left(\frac{x}{\lambda}\right)^k}, x \geq 0 \qquad (2.1)$$

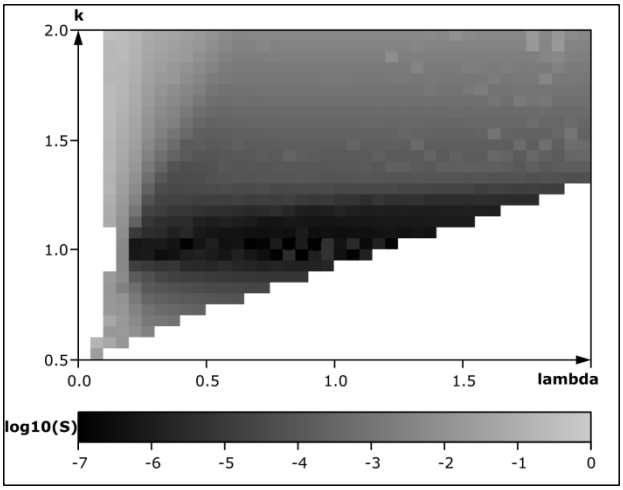
Fazinio skirstinio parametrų ieškosime remdamiesi metodika aprašyta 1.3.3 skyrelyje. Pasirenkame diskretizavimo parametrus: $\Delta x = 0.0625, x_{end} = 8$.

Apibrėžiame Veibulo skirstinių parametrų aibę: $A = [0.025,\ 0.075, \dots, 1.975] \times [0.525,\ 0.575, \dots, 1.975]$, $(\lambda, k) \in A$. Tarkime, kad $\mu_{(\lambda,k)}, \sigma_{(\lambda,k)}$ – teoriniai Veibulo skirstinio vidurkis ir standartinis nuokrypis bei $\hat{\mu}_{(\lambda,k)}, \hat{\sigma}_{(\lambda,k)}$ – diskretizuotų Veibulo skirstinių vidurkis ir standartinis nuokrypis. Sudarome naują parametrų aibę B , į kurią patenka visi parametrų (λ, k) vektoriai iš aibės A, kurie tenkina sąlygas:

$$\frac{|\mu_{(\lambda,k)} - \hat{\mu}_{(\lambda,k)}|}{\mu_{(\lambda,k)}} \leq 0.01, \frac{|\sigma_{(\lambda,k)} - \hat{\sigma}_{(\check{z},k)}|}{\sigma_{(\lambda,k)}} \leq 0.05 \qquad (2.2)$$

Sąlygų (2.2) pagalba atrenkami tie Veibulo skirstiniai, kurių vidurkis diskretizavus pakito ne daugiau kaip 1% , o standartinis nuokrypis – 5%.

Kiekvieną Veibulo skirstinį su parametrais iš aibės B aproksimuojame trijų būsenų faziniu skirstiniu, kurio parametrus $\boldsymbol{\pi}, T$ (maksimalus leistinas perėjimo intensyvumas: $\lambda_{max} = 5$) rasime pasinaudodami paieškos metodu LUS su parametrais: $N = 10, Ic = 100000, \beta = 1$ (žr. skyr. 1.1.3).



2.30 pav. Veibulo skirstinių aproksimavimo (trijų būsenų) faziniu skirstiniu rezultatai.

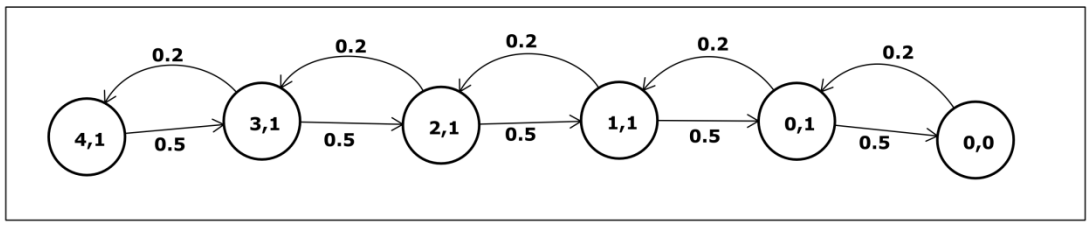
Grafike (2.30 pav.) baltos sritys nurodo nenagrinėtus (t.y. skirstinius, kurie netenkino (2.2) sąlygų) Veibulo skirstinius.

Iš tyrimo rezultatų (2.30 pav.)subjektyviai vertinant galima teigti, kad faziniais skirstiniai geriau aproksimuoja Veibulo skirsitnius, kurių parametras *k* yra arti vieneto, o λ didesnis už 0.25.

2.2 Markovo tipo aptarnavimo sistemų tyrimas

2.2.1 M/M/1/K aptarnavimo sistemos modelis

Reikia sukonstruoti M/M/1/K aptarnavimo sistemos būsenų grafą. Pasirenkame maksimalią laukimo eilės talpą $L_{max} = 4$. Paraiškos ateina su intensyvumu $\lambda_a = 0.2$, o aptarnaujamos su intensyvumu $\mu = \lambda_b = 0.5$. Pagal ankščiau aprašytą algoritmą (žr. skyr. 1.4, 1.5.1) sukonstravome sistemos būsenų grafą:



2.31 pav. M/M/1/K aptarnavimo sistemos būsenų grafas.

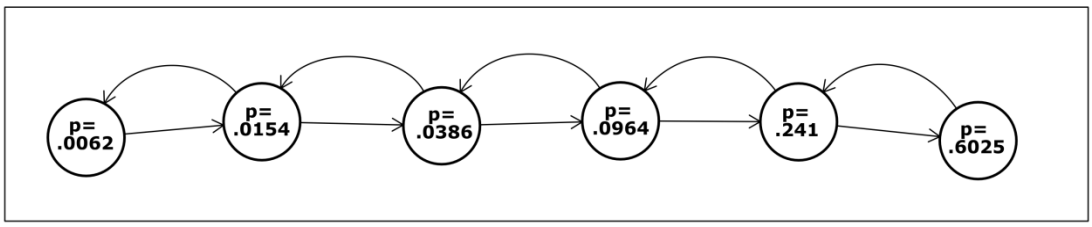
Būsenų eiliškumas: $(0,0), (0,1), (1,1), (2,1), (3,1), (4,1)$. Pradinių tikimybių vektorius:

$$\boldsymbol{\pi}(0) = [1,0,0,0,0,0]$$

Perėjimo intensyvumų matrica:

$$Q = \begin{bmatrix} -0.2 & 0.2 & 0 & 0 & 0 & 0 \\ 0.5 & -0.7 & 0.2 & 0 & 0 & 0 \\ 0 & 0.5 & -0.7 & 0.2 & 0 & 0 \\ 0 & 0 & 0.5 & -0.7 & 0.2 & 0 \\ 0 & 0 & 0 & 0.5 & -0.7 & 0.2 \\ 0 & 0 & 0 & 0 & 0.5 & -0.7 \end{bmatrix}$$

Būsenų finalinės tikimybės:



2.32 pav. M/M/1/K aptarnavimo sistemos būsenų finalinės tikimybės.

Šios sistemos vidutinis laukimo eilės ilgis yra 0.244457. Keisime maksimalią laukimo eilės talpą L_{max} ir žiūrėsime kaip kinta vidutinis laukiančių paraiškų kiekis.

2.11 lentelė. VidutinisM/M/1/K aptarnavimo sistemoje laukiančių paraiškų kiekis.

Laukimo eilės talpa,L_{max}	Vidutinis laukiančių paraiškų kiekis
1	0.102564
2	0.102564
3	0.221145
4	0.244457
5	0.256164
6	0.261814
7	0.264464
8	0.265681
9	0.266230
10	0.266475

Tuo tarpu kai laukimo eilės talpa yra begalinė, analitiniu būdu paskaičiuotas vidutinis laukiančių paraiškų kiekis yra:

$$\rho = \frac{\lambda}{\mu};\ \frac{\lambda^2 + \mu^2\rho^2}{2\mu^2(1-\rho)} = 0.266667$$

Matosi (lent. 2.11), kad vidutinis laukiančių paraiškų kiekis artėja prie teorinio, kai modelyje maksimalilaukimo eilės talpa yra didinama.

2.2.2 PTD/PTD/1/K aptarnavimo sistemos modelis

Patyrinėsime du variantus.

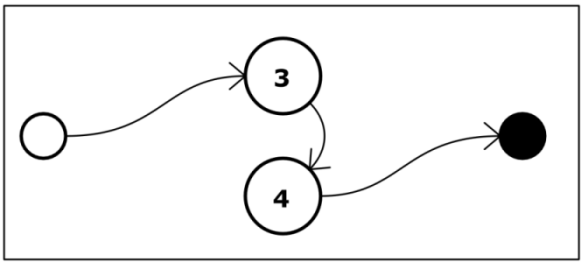
Pirmasis variantas.

Pirmajam atvejui pasirinkome, kad laiko tarpai tarp paraiškų atėjimų turi fazinį skirstinį $PTD^{(a)}\left(\boldsymbol{\pi}^{(a)}, T^{(a)}\right)$ su parametrais:

$$\boldsymbol{\pi}^{(a)} = [1], T^{(a)} = [1.5], \qquad m^{(a)} = 1$$

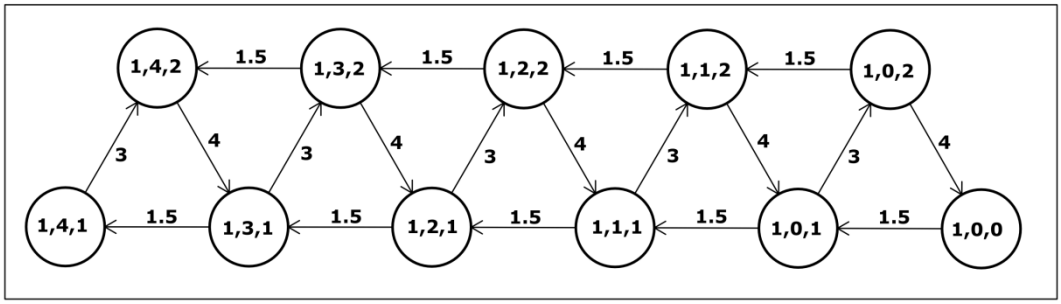
Šis skirstinys yra tapatus eksponentiniam skirstiniui su $\lambda = 1.5$. Todėl šią sistemą galima klasifikuoti kaip M/PTD/1/K tipo sistemą.Paraiškų aptarnavimo laikai turi skirstinį$PTD^{(b)}\left(\boldsymbol{\pi}^{(b)}, T^{(b)}\right)$ su parametrais:

$$\boldsymbol{\pi}^{(b)} = [1, 0], T^{(b)} = \begin{bmatrix} -3 & 3 \\ 0 & -4 \end{bmatrix}, \qquad m^{(b)} = 2$$



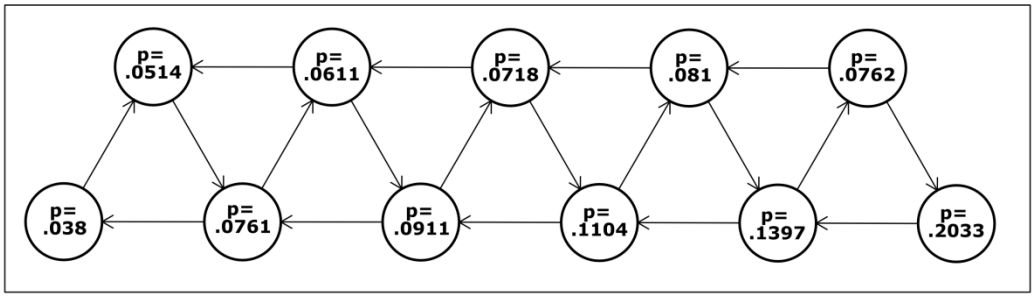
2.33pav. M/PTD/1/K aptarnavimo sistemos paraiškų aptarnavimo laikų fazinio skirstinio struktūra.

Pasirinkę maksimalią laukimo eilės talpą $L_{max} = 4$ gauname tokį sistemos būsenų grafą:



2.34 pav. M/PTD/1/K aptarnavimo sistemos būsenų grafas.

Apskaičiuojame finalines tikimybes:



2.35 pav. M/PTD/1/K aptarnavimo sistemos būsenų grafas.

Vidutinis paraiškų kiekis laukimo eilėje: 1.286176, pasižiūrėsime kaip jis kinta kai didiname maksimalią laukimo eilės talpą.

2.12 lentelė. VidutinisM/PTD/1/K aptarnavimo sistemoje laukiančių paraiškų kiekis.

Laukimo eilės talpa L_{max}	Vidutinis laukiančių paraiškų kiekis
1	0.264624
2	0.603687
4	1.286176
8	2.424174
16	3.772775
32	4.530514
64	4.530514
128	4.625000

Aptarnavimo sistemai M/G/1/K yra išvesta formulė vidutiniam laukiančių paraiškų kiekiui surasti:

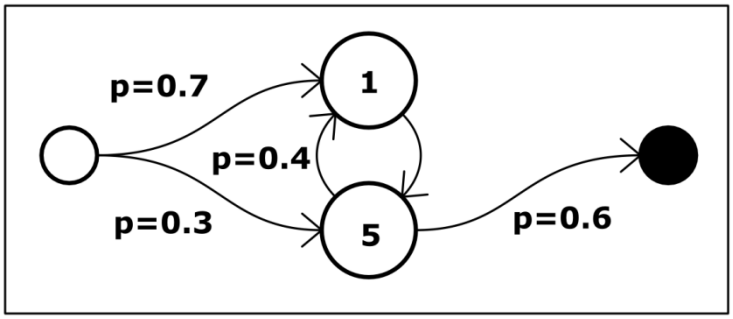
$$X^{(b)} \sim PTD^{(b)};\ \rho = \mathrm{E}\left(X^{(b)}\right)\lambda;\ \frac{\lambda^2 D\left(X^{(b)}\right) + \rho^2}{2\mu^2(1-\rho)} = 4.625000$$

Iš rezultatų matosi (lent. 2.12), kad kai laukimo eilės talpa yra didinama vidutinis laukiančių paraiškų kiekis konverguoja prie teorinio (kai $L_{max} = \infty$).

Antrasis variantas.

Tegu laiko tarpai tarp paraiškų atėjimų turi fazinį skirstinį $PTD^{(a)}(\boldsymbol{\pi}^{(a)}, T^{(a)})$suparametrais:

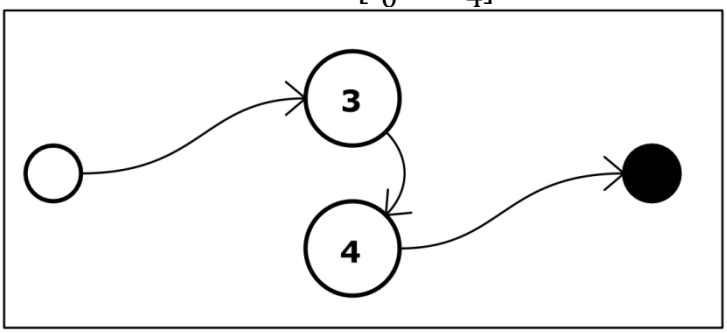
$$\boldsymbol{\pi}^{(a)} = [0.7, 0.3], T^{(a)} = \begin{bmatrix} -1 & 1 \\ 2 & -5 \end{bmatrix}, \qquad m^{(a)} = 2$$



2.36 pav. PTD/PTD/1/K aptarnavimo sistemos laiko tarpų tarp paraiškų atėjimų fazinio skirstinio struktūra.

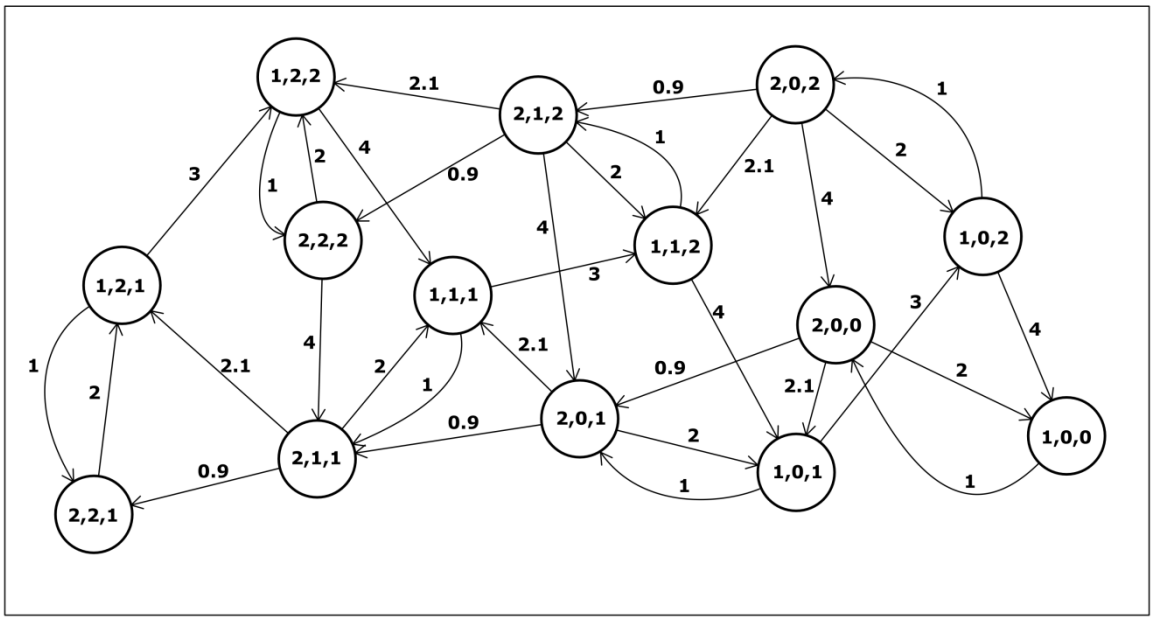
Tuo tarpu, paraiškų aptarnavimo laikų skirstinys $PTD^{(a)}(\boldsymbol{\pi}^{(a)}, T^{(a)})$su parametrais:

$$\boldsymbol{\pi}^{(b)} = [0.7, 0.3], T^{(b)} = \begin{bmatrix} -3 & 3 \\ 0 & -4 \end{bmatrix}, \qquad m^{(b)} = 2$$



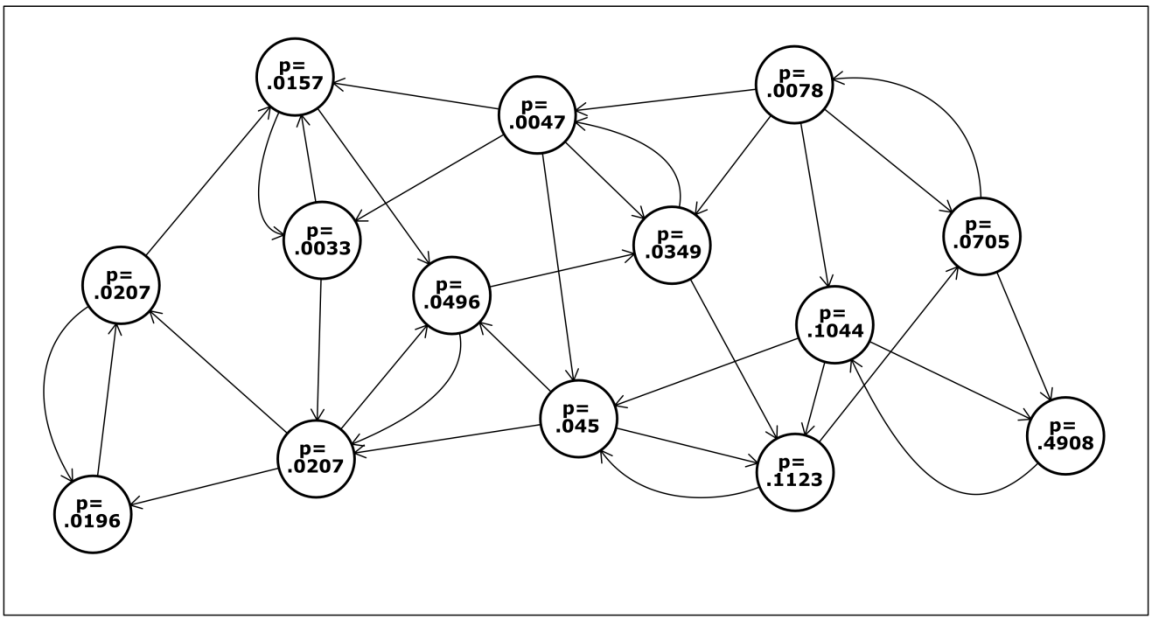
2.37 pav. PTD/PTD/1/K aptarnavimo sistemos paraiškų aptarnavimo laikų fazinio skirstinio struktūra.

Sugeneruojame sistemos būsenų grafą:



2.38 pav. PTD/PTD/1/K aptarnavimo sistemos būsenų grafas

Apskaičiuojame būsenų finalines tikimybes:



2.39 pav. PTD/PTD/1/K aptarnavimo sistemos būsenų finalinės tikimybės

Ištyrėme, kaip kinta vidutinis laukiančių paraiškų kiekis, keičiantis laukimo eilės talpai:

2.13 lentelė. VidutinisPTD/PTD /1/K aptarnavimo sistemoje laukiančių paraiškų kiekis.

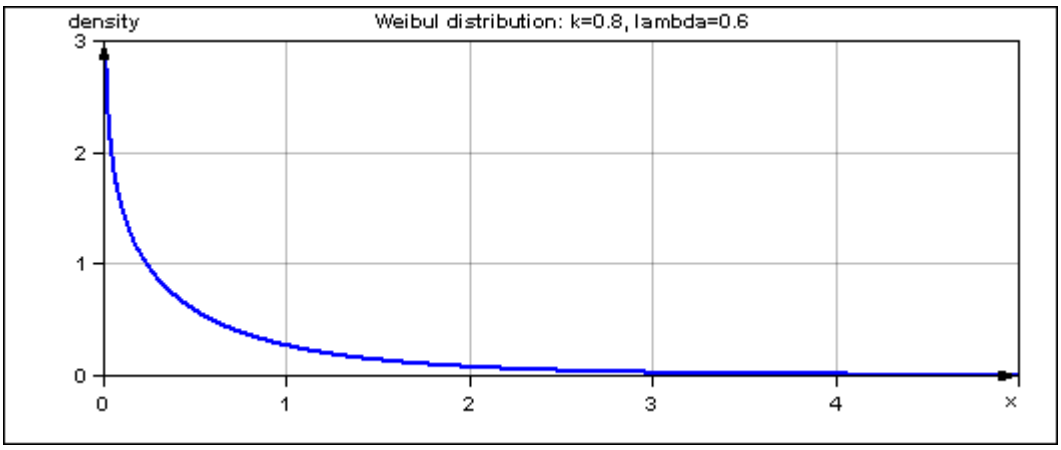
Laukimo eilės talpa,L_{max}	Vidutinis laukiančių paraiškų kiekis
1	0.136895
2	0.228584
3	0.283473
4	0.314499
5	0.331346
10	0.348854
20	0.349395
30	0.349395

Iš rezultatų (lent. 2.13) matosi, kad vidutinis laukiančių paraiškų kiekis didinant L_{max} konverguoja.

2.3 Ne Markovo tipo aptarnavimo sistemų aproksimavimas

Šiame skyriuje parodysime kaip ne Markovo tipo aptarnavimo sistemą galima aproksimuoti Markovo tipo sistema. Pasirenkame du skirtingus skirstinius $F^{(a)}$,$F^{(b)}$kuriuos aproksimuosime atitinkamai faziniais skirstiniais $PTD^{(a)}$, $PTD^{(b)}$ kuriuose vėliau panaudosime sistemų modeliavimui. Pirmojo skirstinio tankio funkcija:

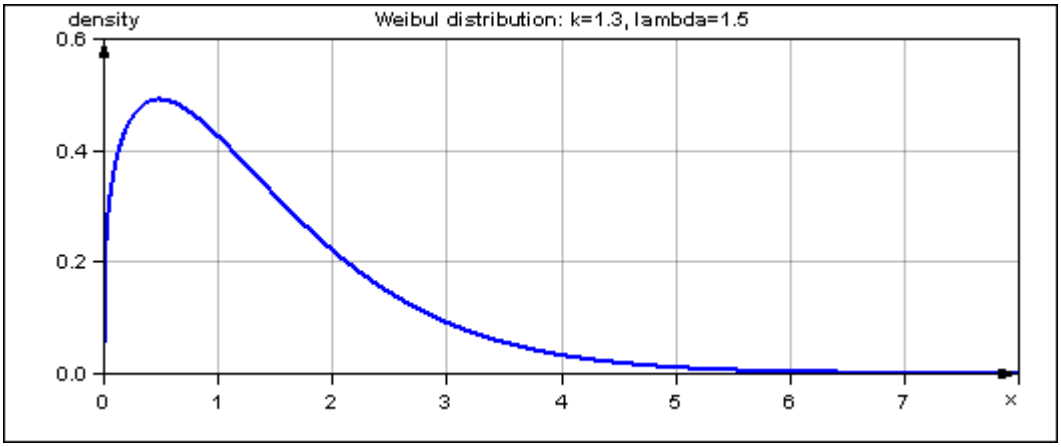
$$f^{(a)}(x) = \frac{4}{3}\left(\frac{4}{3}\right)^{-0.2} e^{-\frac{x}{0.6}},\ x \geq 0 (2.3)$$



2.40 pav. Pirmojo skistinio (a) tankio funkcija

Antrojo skirstinio tankio funkcija:

$$f^{(b)}(x) = \frac{1.3}{1.5}\left(\frac{1.3}{1.5}\right)^{0.3} e^{-\frac{x}{1.3}},\ x \geq 0 (2.4)$$



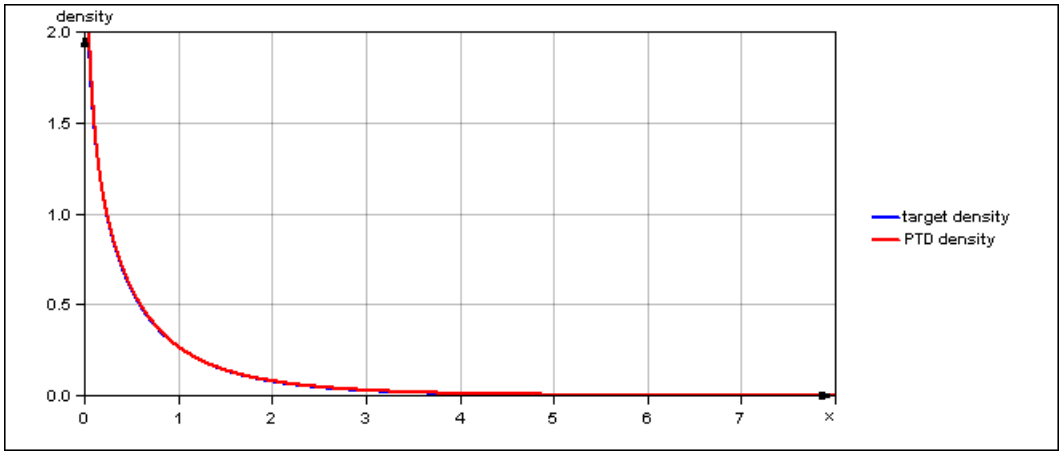
2.41 pav.Antrojo skirstinio (b) tankio funkcija

Šiuos skirstinius (2.3, 2.4) aproksimuosime faziniais skirstiniais (turinčiais tris fazes). Pasirenkame tankio funkcijų diskretizavimo parametrus: $x_{end} = 8, \Delta x = 0.0625$.

Pasinaudodami LUS paieškos metodu (su parametrais $N = 10, Ic = 200000, \beta = 1, \lambda_{max} = 10$) suradome fazinio skirstinio kirstinio $PTD^{(a)}$ parametrus su kuriais aproksimuojamas skirsinys$F_{(a)}(x)$:

$$\boldsymbol{\pi}^{(a)} \approx [3.456 \cdot 10^{-4}, 0.767, 0.233], T^{(a)} \approx \begin{bmatrix} -1.516 & 0.002 & 1.509 \\ 0.584 & -2.348 & 0.005 \\ 1.782 & 7.819 & -15.604 \end{bmatrix}$$

Grafike (pav. 2.42) pavaizduota aproksimuoto skirstinio $F^{(a)}(x)$ tankio funkcija – mėlynas grafikas, fazinio skirstinio $PTD^{(a)}$ – raudonas, plotas tarp tankio funkcijų $\hat{S} = 0.012693$.



2.42 pav. Pirmojo skirstinio (a) aproksimacija faziniu skirstiniu

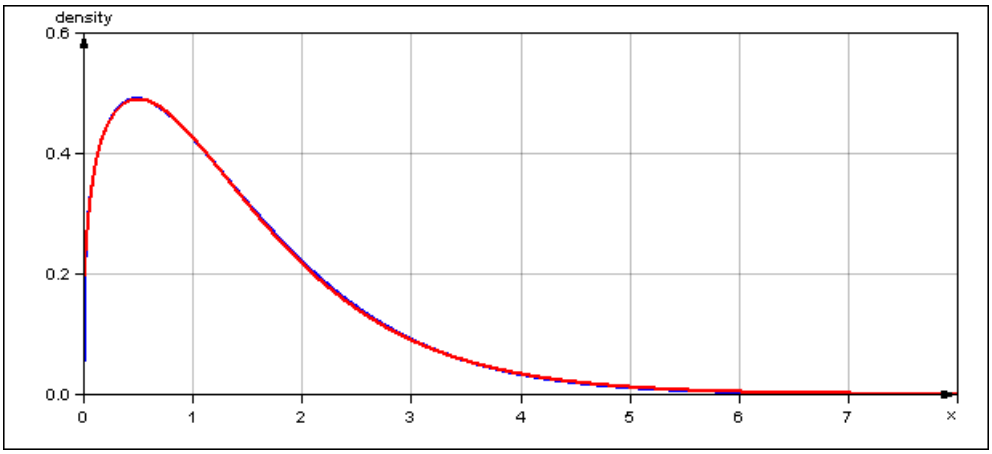
2.14 lentelė.Pirmojo skirstinio (a) vidurkio bei standartinio nuokrypio paklaida po diskretizacijos ir aproksimacijos faziniu skirstiniu

Statistika	Skirstinys $F^{(a)}(x)$	Diskretizuotas skirstinys $\hat{F}^{(a)}(x)$ (statistikos procentinis nuokrypis)	Fazinis skirstinys $PTD^{(a)}$(statistikos procentinis nuokrypis)
Vidurkis	0.6798	0.6769 (0.43%)	0.6958 (2.35%)
Standartinis nuokrypis	0.8569	0.8364 (2.39%)	0.8588 (0.22%)

Tokiu pat būdu aproksimuojame paraiškų aptarnavimo laikų skirstinį $F^{(b)}(x)$ faziniu skirstiniu $PTD^{(b)}$, kurio surasti parametrai yra:

$$\boldsymbol{\pi}^{(b)} \approx [0.008, 1.397 \cdot 10^{-4}, 0.992], T^{(b)} \approx \begin{bmatrix} -1.236 & 0 & 0 \\ 1.913 & -16.338 & 6.770 \\ 0.897 & 0.331 & -1.402 \end{bmatrix}$$

Grafike (pav. 2.43) pavaizduota aproksimuoto skirstinio $F^{(b)}(x)$ tankio funkcija – mėlynas grafikas, fazinio skirstinio $PTD^{(b)}$ – raudonas, plotas tarp tankio funkcijų $\hat{S} = 0.010634$.



2.43 pav. Antrojo skirstinio (b) aproksimacija faziniu skirstiniu

2.15 lentelė.Antrojo skirstinio (a) vidurkio bei standartinio nuokrypio paklaida po diskretizacijos ir aproksimacijos faziniu skirstiniu

Statistika	Skirstinys $F^{(b)}(x)$	Diskretizuotas skirstinys $\hat{F}^{(b)}(x)$ (statistikos procentinis nuokrypis)	Fazinis skirstinys $PTD^{(b)}$ (statistikos procentinis nuokrypis)
Vidurkis	1.3854	1.3841 (0.09%)	1.4000 (1.05%)
Standartinis nuokrypis	1.0747	1.0719 (0.26%)	1.1126 (3.53%)

Su aproksimuotais skirstiniais pademonstruosime kaip yra aproksimuojamos ne Markovo tipo sistemos Markovo tipo sistemomis.

2.3.1 G/G/1/K aptarnavimo sistemos aproksimacija

Tarkime, kad reikia sumodeliuoti aptarnavimo sistema *G/G/1/K*, kurioje laiko trukmės tarp paraiškų atėjimų turi $F^{(b)}$ skirstinį, o aptarnavimo trukmės - $F^{(a)}$ skirstinį. Ši sistema nėra Markovo tipo, tačiau skirstinius $F^{(b)}$, $F^{(a)}$ pakeitę atitinkamais faziniais skirstiniais $PTD^{(b)}$, $PTD^{(a)}$ gauname aptarnavimo sistemą *PTD/PTD/1/K*, kurią jau galime modeliuoti.

Pasižiūrėsime kaip kinta vidutinis laukiančių paraiškų kiekis eilėje kai didinama pastarosios talpa L_{max}.

2.16 lentelė. Vidutinio laukiančių paraiškų kiekio priklausomybė nuo laukimo eilės talpos, aproksimuotoje aptarnavimo sistemoje G/G/1/K.

Laukimo eilės talpa,L_{max}	Vidutinis eilėje laukiančių paraiškų kiekis
1	0.265137
5	1.055698
10	1.535596
20	1.773902
40	1.799246
80	1.799396
150	1.799396

Iš rezultatų (lent. 2.16) matome, kad didinant laukimo eilės talpą, vidutinis laukiančių paraiškų kiekis konverguoja link tam tikros reikšmės. Kitaip tariant, juo talpesnė laukimo eilė – tuo mažiau yra prarandama paraiškų dėl pastarosios perpildymo.

2.3.2 G/G/1/K aptarnavimo sistemos aproksimacija (kai fiksuojamas aptarnautų paraiškų kiekis)

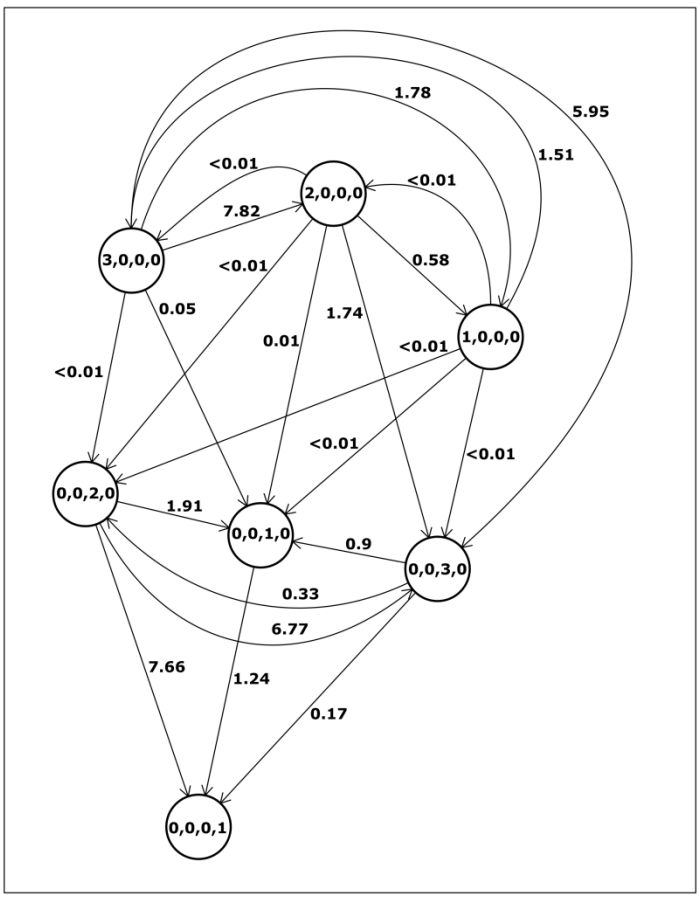
Patyrinėsime aptarnavimo sistemą *G/G/1/K* į kuria patenka baigtinis kiekis paraiškų. Tarkime, kad laiko trukmės tarp paraiškų atėjimų turi $F^{(a)}$ skirstinį, o aptarnavimo trukmės - $F^{(b)}$ skirstinį. Ši sistema nėra Markovo tipo, tačiau skirstinius $F^{(b)}$, $F^{(a)}$ pakeitę atitinkamais faziniais skirstiniais $PTD^{(a)}$, $PTD^{(b)}$ gauname aptarnavimo sistemą *PTD/PTD/1/K*, kurią jau galime modeliuoti.

Ši sistema, kaip ankščiau minėjom, turi vieną sugeriančią būseną (būsena, kai aptarnautos visos paraiškos), todėl skaičiuoti būsenų finalines tikimybes nėra prasmės. Kita vertus, galima pasižiūrėti kaip laike kinta tikimybės, kad sistema bus tam tikroje būsenoje. Beto, mus domina laiko, per kurį yra pasiekiama sugereiančioji būsena, skirstinys.

Panagrinėsime du atvejus, pirmasis kai $L_{max} = 0, Ec = 1$ ir antrasis kai $L_{max} = 9, Ec = 10$.

Pirmasis atvejis

Paraiškų kiekis$Ec = 1$, laukimo eilės nėra. Sukonstruojame sistemos būsenų grafą:

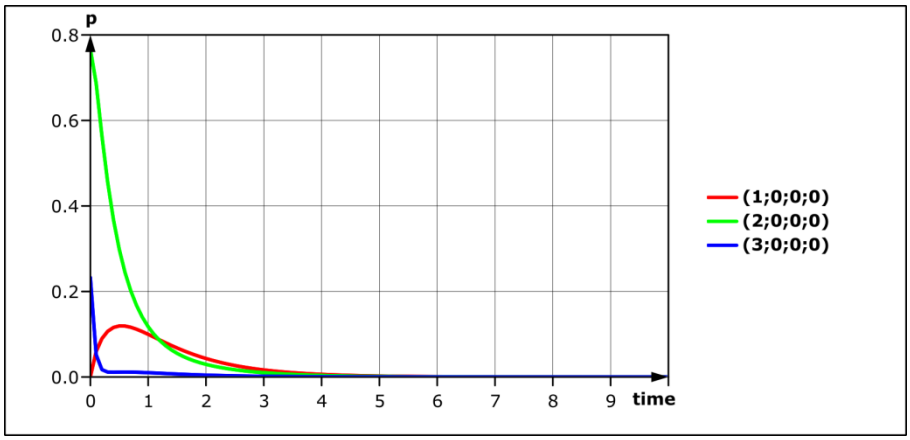


2.44 pav. Aproksimuotos aptarnavimo sistemosG/G/1/K būsenų grafas, kai Ec=1

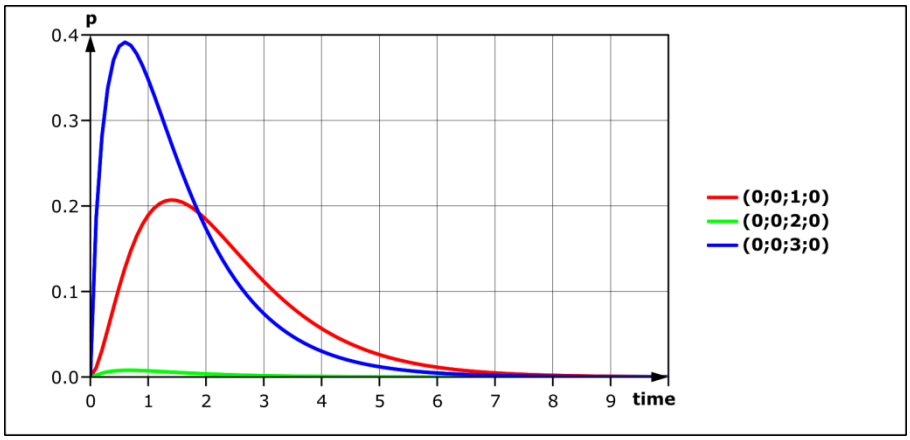
Išsirašome būsenas: **(1,0,0,0), (2,0,0,0), (3,0,0,0),** (0,0,1,0), (0,0,2,0), (0,0,3,0),(0,0,0,1). Būsenų pradinių tikimybių vektorius: $\boldsymbol{\pi}(0) = [3.456 \cdot 10^{-4}, 0.767, 0.233, 0, 0, 0, 0]$ bei perėjimo intensyvumų matrica:

$$Q \approx \begin{bmatrix} -1.516 & 1.731 \cdot 10^{-3} & 1.509 & 3.869 \cdot 10^{-5} & 6.585 \cdot 10^{-7} & 4.676 \cdot 10^{-3} & 0.0 \\ 0.584 & -2.348 & 4.994 \cdot 10^{-3} & 1.443 \cdot 10^{-2} & 2.456 \cdot 10^{-4} & 1.744 & 0.0 \\ 1.782 & 7.819 & -15.604 & 4.926 \cdot 10^{-2} & 8.384 \cdot 10^{-4} & 5.953 & 0.0 \\ 0.0 & 0.0 & 0.0 & -1.236 & 0.0 & 0.0 & 1.236 \\ 0.0 & 0.0 & 0.0 & 1.913 & -16.338 & 6.77 & 7.655 \\ 0.0 & 0.0 & 0.0 & 0.897 & 0.331 & -1.402 & 0.174 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

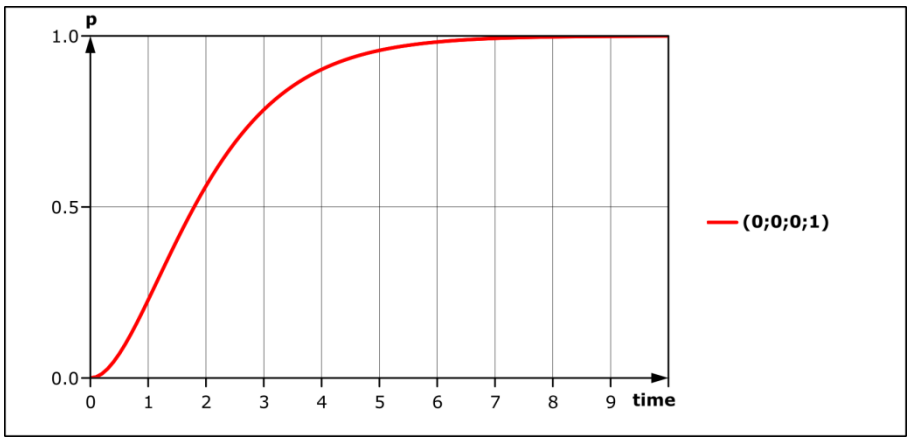
Toliau pateikiame grafikus kurie iliustruoja kaip šioje sistemoje kito kievkienos būsenos tikimybė laike.



2.45 pav. Aproksimuotos aptarnavimo sistemosG/G/1/K būsenų (1,0,0,0),(2,0,0,0),(3,0,0,0) tikimybių kitimas laike(Ec=1)



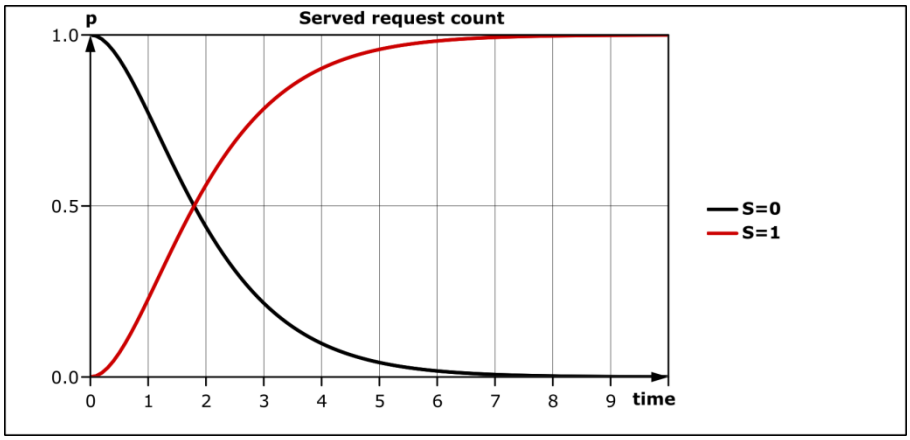
2.46 pav. Aproksimuotos aptarnavimo sistemosG/G/1/K būsenų (0,0,1,0),(0,0,2,0),(0,0,3,0) tikimybių kitimas laike (Ec=1)



2.47 pav. Aproksimuotos aptarnavimo sistemosG/G/1/K būsenos (0,0,0,1) tikimybės kitimas laike (Ec=1)

Žvelgdami į grafikus (2.45, 2.46, 2.47 pav.) galima vizualiai nustatyti kurios būsenos buvo pradinės (jų tikimybės nuliniu laiko momentu didesnės už nulį) bei identifikuoti sugeriančią būseną (laikui bėgant jos tikimybė konverguoja prie vieneto).

Visas būsenas galima sugrupuoti į dvi grupes: pirmajai priklauso būsenos, kuomet dar bebuvo aptarnautų paraiškų (S=0); antrajai priklauso būsenos, kai aptarnauta viena paraiška (S=1). Susumuojame tikimybes kiekvienoje grupėje ir gauname grafiką (2.48 pav.), kuris parodo kaip kito tikimybė, kad sistemoje aptarnautas atitinkamas kiekis paraiškų:



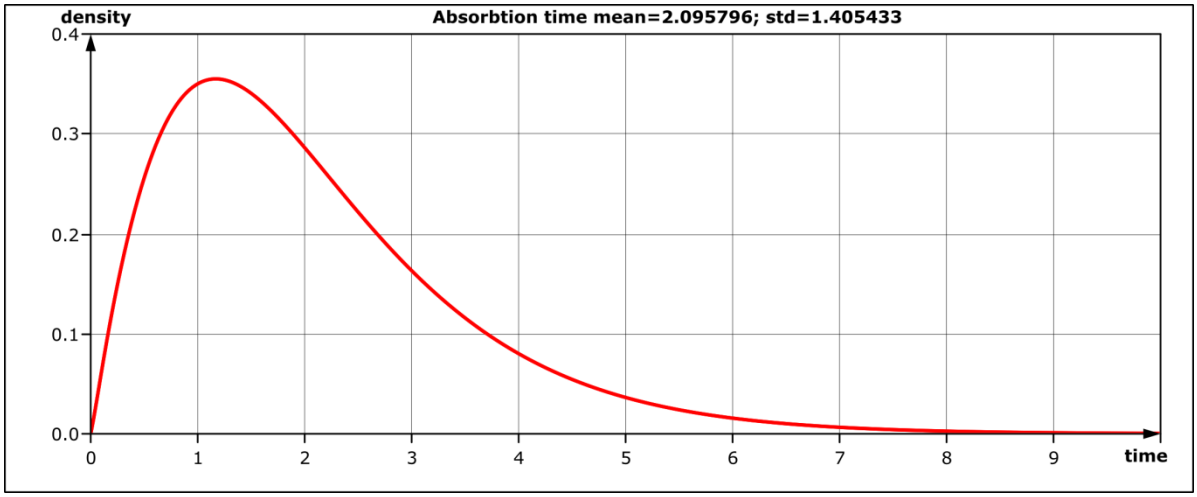
2.48 pav. Aproksimuotos aptarnavimo sistemosG/G/1/K aptarnautų paraiškų kiekio tikimybės kitimas laike (Ec=1)

Kadangi būsenų grafas (2.44 pav.) turi vieną sugeriančią būseną, tai laikas per kurį ji yra pasiekiama, turi fazinį skirstinį $PTD(\boldsymbol{\pi}, T)$ su parametrais:

$$\boldsymbol{\pi} \approx [3.456 \cdot 10^{-4}, 0.767, 0.233, 0, 0]$$

$$T \approx \begin{bmatrix} -1.516 & 1.731 \cdot 10^{-3} & 1.509 & 3.869 \cdot 10^{-5} & 6.585 \cdot 10^{-7} & 4.676 \cdot 10^{-3} \\ 0.584 & -2.348 & 4.994 \cdot 10^{-3} & 1.443 \cdot 10^{-2} & 2.456 \cdot 10^{-4} & 1.744 \\ 1.782 & 7.819 & -15.604 & 4.926 \cdot 10^{-2} & 8.384 \cdot 10^{-4} & 5.953 \\ 0.0 & 0.0 & 0.0 & -1.236 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.913 & -16.338 & 6.77 \\ 0.0 & 0.0 & 0.0 & 0.897 & 0.331 & -1.402 \end{bmatrix}$$

Kurie yra gaunami iš parametrų $\boldsymbol{\pi}(0), Q$ pašalinus informaciją susijusią su sugeriančia būsena.



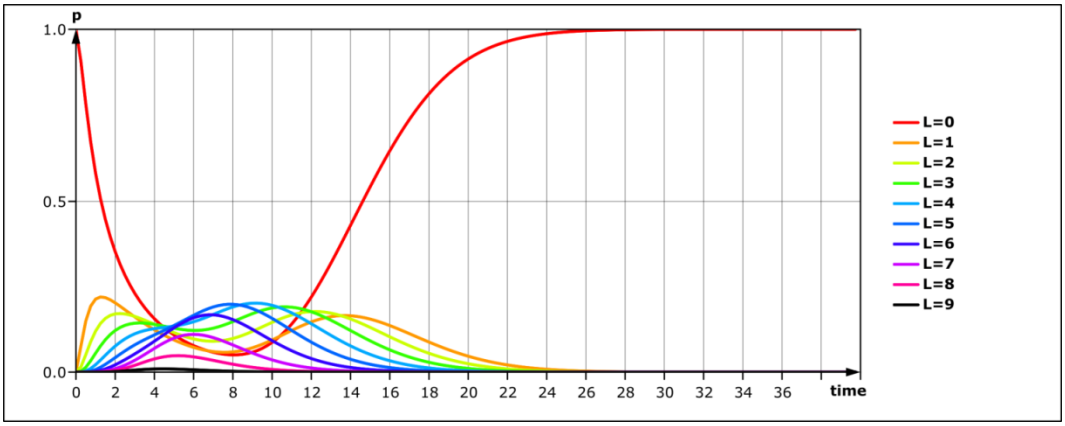
2.49 pav. Aproksimuotos aptarnavimo sistemosG/G/1/K laiko per kurį aptarnaujamos visos paraiškos skirstinio tankio funkcija (Ec=1)

Iš grafike (2.49 pav.) pateiktų duomenų matome, kad vienai paraiškai aptarnauti vidutiniškai prireikia apie 2.1 laiko vienetų.

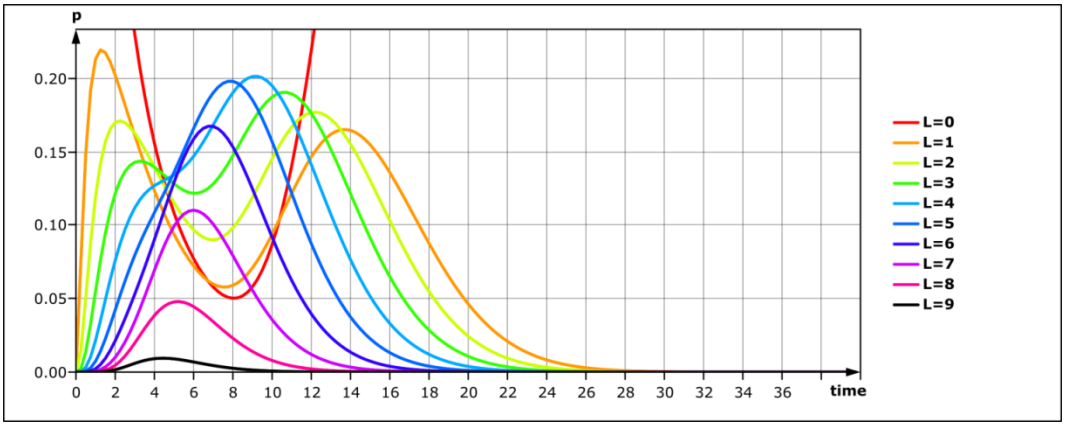
Antrasis variantas.

Paraiškų kiekis $Ec = 10$, laukimo eilės talpa $L_{max} = 9$. Šios sistemos būsenų grafą sudaro 466 būsenos.

Sugrupuojame būsenas pagal laukiančių paraiškų kiekį eilėje ir nubraižome šių grupių tikimybių kitimą laike:

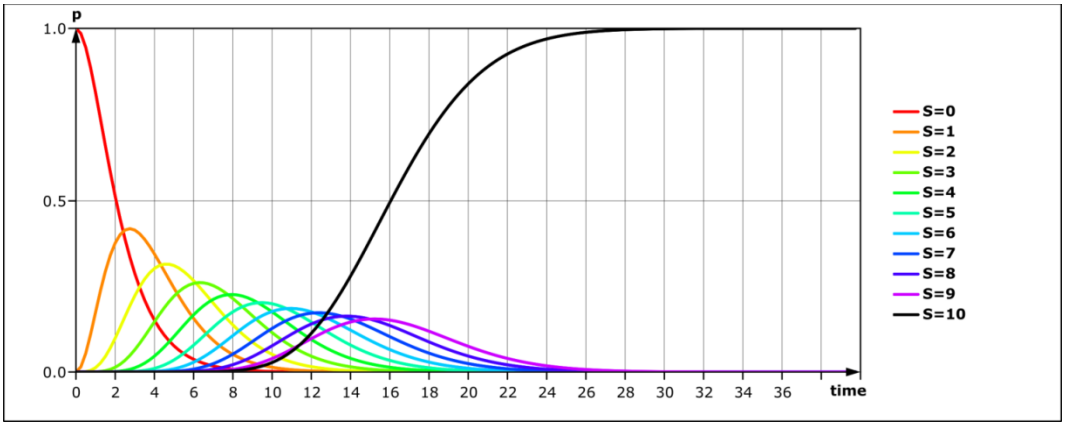


2.50 pav. Aproksimuotos aptarnavimo sistemosG/G/1/K laukiančių paraiškų kiekio tikimybių kitimas laike (Ec=10)

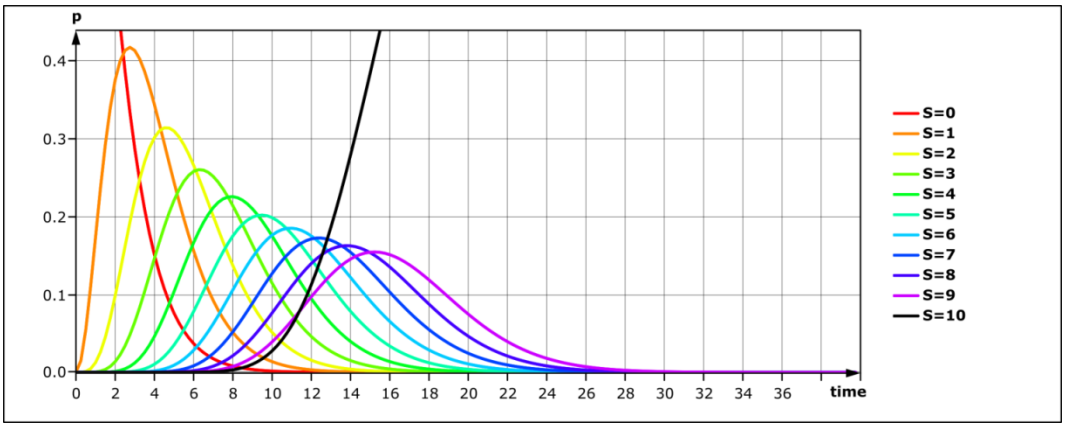


2.51 pav. Aproksimuotos aptarnavimo sistemosG/G/1/K laukiančių paraiškų kiekio tikimybių kitimas laike, priartintas vaizdas (Ec=10)

Pergrupuojame būsenas pagal aptarnautų paraiškų kiekį ir nubraižome šių grupių tikimybių kitimą laike:

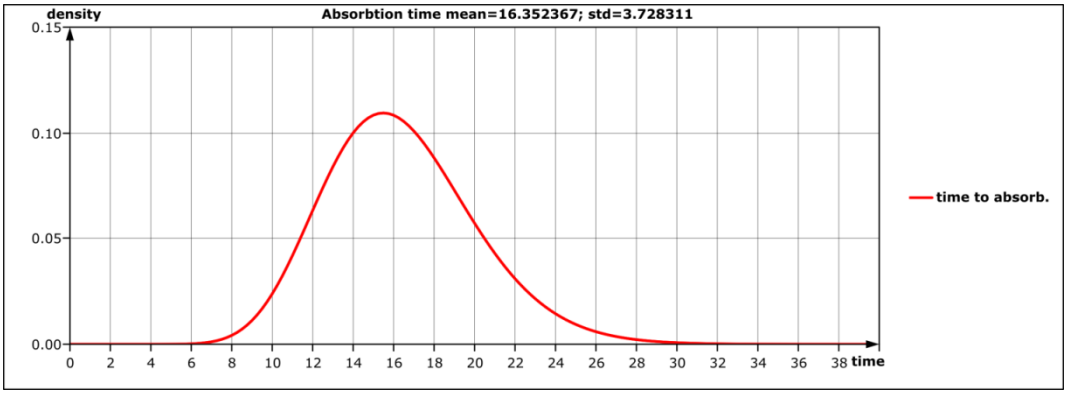


2.52 pav. Aproksimuotos aptarnavimo sistemosG/G/1/K aptarnautų paraiškų kiekio tikimybių kitimas laike (Ec=10)



2.53 pav. Aproksimuotos aptarnavimo sistemosG/G/1/K aptarnautų paraiškų kiekio tikimybių kitimas laike, priartintas vaizdas (Ec=10)

Galiausiai surandame laiko, per kurį yra aptarnaujamos visos paraiškos, fazinį skirstinį, kurio tankio funkcija yra:



2.54 pav. Aproksimuotos aptarnavimo sistemosG/G/1/Klaiko per kurį aptarnaujamos visos paraiškos skirstinio tankio funkcija (Ec=10)

Iš pateiktos informacijos (2.54 pav.) grafike, galima teigti, kad 10 paraiškų šioje sistemoje aptarnaujama vidutiniškai per 16.35 laiko vienetus.

3 PROGRAMINĖ ĮRANGA

Šioje ataskaitoje pateikti rezultatai buvo gauti naudojant savarankiškai sukurtas programas, java ir C++ kalbomis. Java kalba parašytos programos paskirtis yra:

- Piešti grafikus, diagramas (fazinių skirstinių, sistemos būsenų grafo)
- Patogia forma pateikti tyrimų bei modeliavimų rezultatus

C++ kalba parašytos programos buvo naudojamos fazinių skirstinių parametrų radimui.

Taip pat buvo naudojamos šios pagalbinės programos, paketai:

- Vektorinės grafikos redaktorius – Inkscape [11]
- Java programų kūrimo aplinka – Eclipse [12]
- C++ kalbų kūrimo aplinka – CodeBlocks [13]
- CUDA[2] technologijos paketas [14]

Java programos struktūra:

a) Fazinių skirstinių parametrų radimo uždavinys

a.1. Fazinio skirstinio parametrų paieška naudojant RND, PSO ir LUS metodus

a.1.1. Užduoties failo paruošimas (išeksportuojamas skirstinio diskretizuotas tankis)

a.1.2. Užduoties rezultatų peržiūra (pateikiami visi surasti faziniai skirstiniai)

a.2. Veibulo skirstinių aproksimavimas trijų būsenų faziniais skirstiniais

a.2.1. Užduoties failo parengimas

a.2.2. Užduoties rezultatų peržiūra

[2] CUDA technologijos pagalba skaičiavimams galima atlikti panaudojant vaizdo procesorių (GPU)

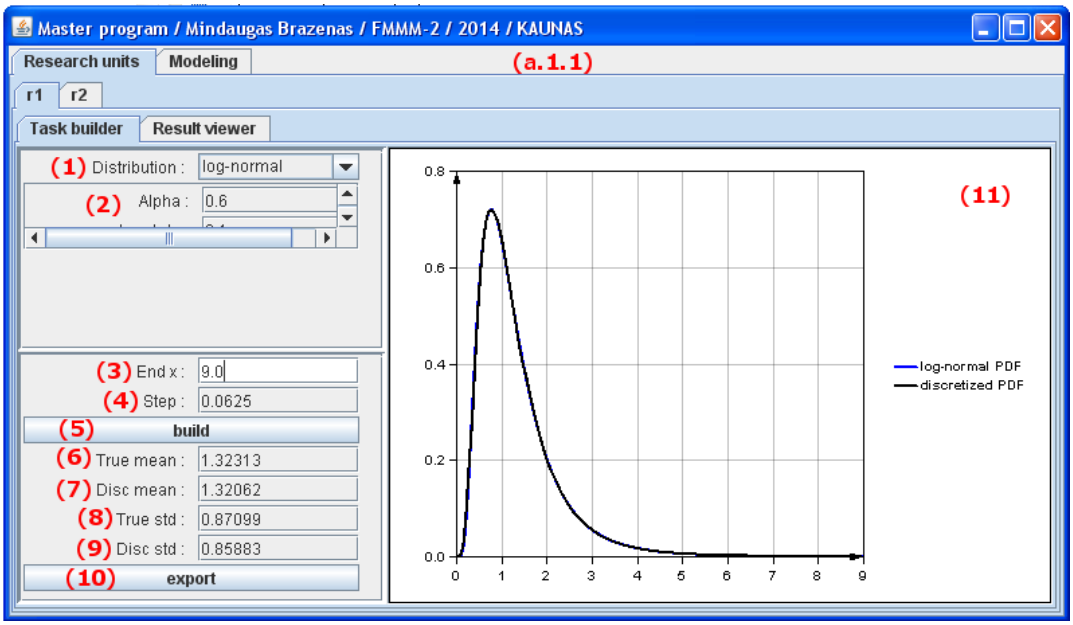
b) Markovo sistemų modeliavimas

b.1. M/M/1/K aptarnavimo sistemos modeliavimas Markovo grandinėmis

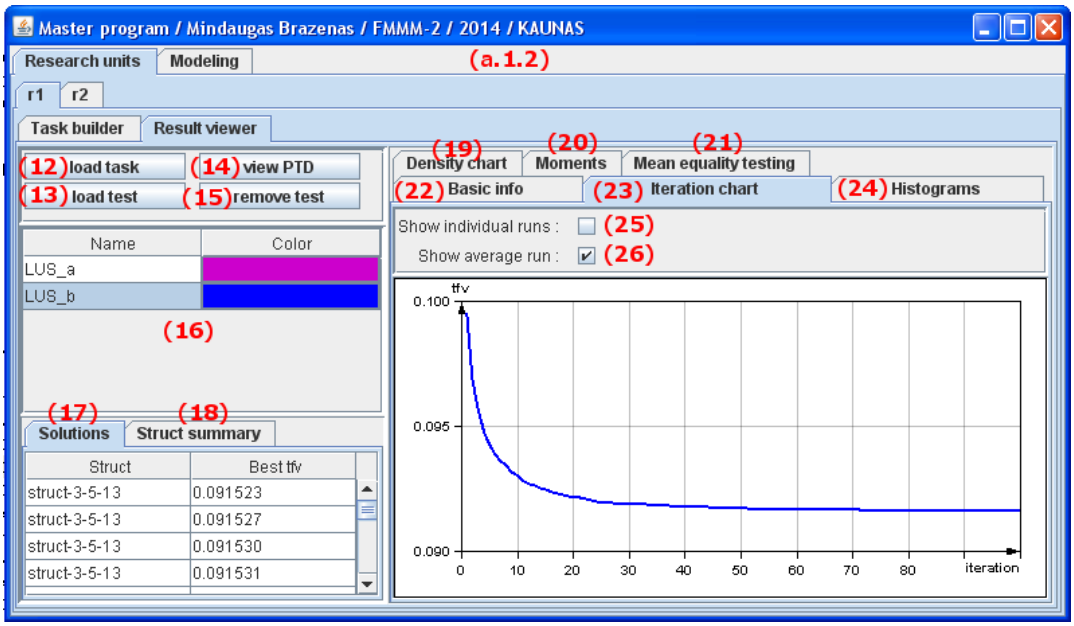
b.2. PTD/PTD/1/K aptarnavimo sistemos modeliavimas Markovo grandinėmis

b.3. PTD/PTD/1/K, kai fiksuojamas aptarnautų paraiškų kiekis, aptarnavimo sistemos modeliavimas Markovo grandinėmis

Dalyse a.1.1, a.2.1, b.1.1 parengti duomenų failai rankiniu būdų nurodomi atitinkamoms C++ programoms, kurių rezultatai užkraunami atitinkamai dalyse a.1.2, a.2.2, b.1.2.



3.1 pav. Java programos a.1.1 dalies vartotojo sąsaja



3.2 pav. Java programos a.1.2 dalies vartotojo sąsaja

Programos dalių a.1.1 (pav. 3.1), a.1.2 (pav. 3.2) vartotojo sąsajos elementų paaiškinimas:

(1) diskretizuojamų skirstinių išskleidžiamas sąrašas

(2) pasirinkto skirstinio parametrai

(3) nurodoma reikšmė iki kurios atliekama tankio funkcijos diskretizacija

(4) nurodomas diskretizavimo žingsnelis

(5) iškviečiama tankio funkcijos diskretizavimo procedūra

(6) skirstinio vidurkis

(7) diskretizuoto skirstinio vidurkis

(8) skirstinio standartinis nuokrypis

(9) diskretizuoto skirstinio standartinis nuokrypis

(10) iškviečiama diskretizuoto tankio funkcijos išeksportavimo į duomenų failą procedūra

(11) grafikas, kuriame pavaizduotos teorinio ir diskretizuoto skirstinių tankio funkcijos

(12) užkraunamas duomenų failas, sukurtas (5) punkto vykdymo metu

(13) užkraunami rezultatų failai su atskirų paieškų metodų rezultatais

(14) peržiūrimas pasirinktas sprendinys (fazinis skirstinys)

(15) pašalinami pasirinkti ((16) lentelė) rezultatai

(16) lentelė, kurioje rodomi atskirų paieškų rezultatai

(17) rodomi lentelėje (16) pasirinktos paieškos visi sprendiniai, išrikiuoti tikslo funkcijos didėjimo tvarka

(18) rodomi lentelėje (16) pasirinktos paieškos fazinių skirstinių struktūrų santrauka

(19) rodomas teorinio skirstinio ir lentelėje (17) pasirinkto fazinio skirstinio tankio funkcijos

(20) informacija apie teorinio ir diskretizuoto skirstinių momentus

(21) pasirinkus dviejų paieškų rezultatus lentelėje (16) galima patikrinti hipotezes apie šios grupėse esančių sprendinių tikslo funkcijų vidurkių statistinį sutapimą

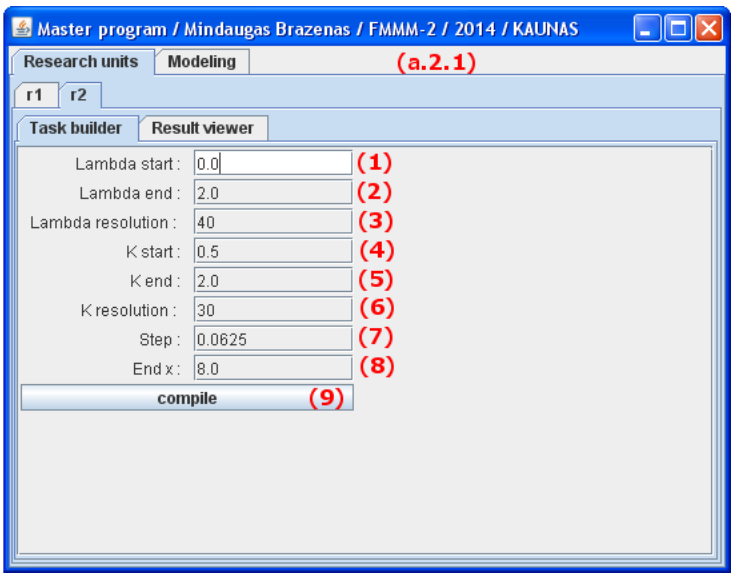
(22) bendra informacija apie pasirinktos paieškos lentelėje (16) sprendinius

(23) rodoma kaip kito sprendinių tikslo funkcija vykdant paieškos metodo iteracijas

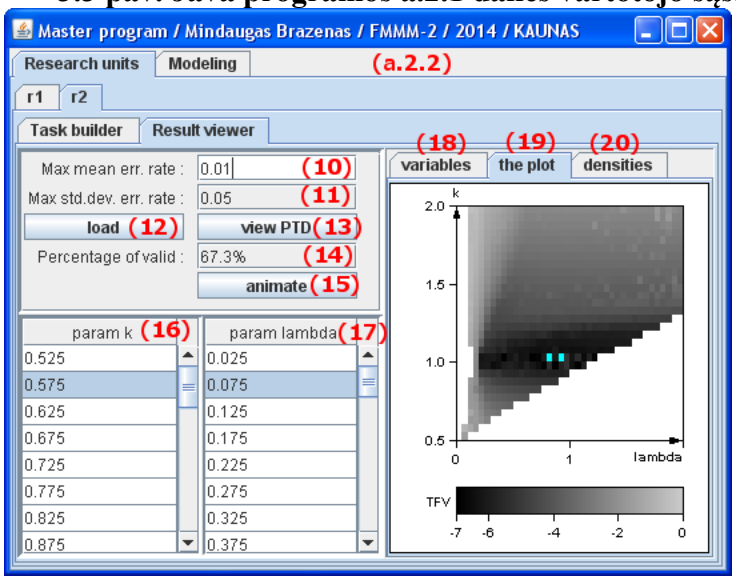
(24) rodomos pasirinktų paieškų lentelėje(16) sprendinių tikslo funkcijų histogramos

(25) pasirinkimas rodyti kaip konvergavo visų bandymų geriausių sprendinių tikslo funkcijos

(26) pasirinkimas rodyti kaip konvergavo suvidurkinta visų bandymų geriausių sprendinių tikslo funkcijos



3.3 pav. Java programos a.2.1 dalies vartotojo sąsaja



3.4 pav. Java programos a.2.2 dalies vartotojo sąsaja

Programos dalių a.2.1 (pav. 3.3), a.2.2 (pav. 3.4) vartotojo sąsajos elementų paaiškinimas:

(1) startinė Veibulo skirstinio *lambda* parametro reikšmė

(2) galinė Veibulo skirstinio *lambda* parametro reikšmė

(3) nurodo kiek reikia paimti parametro *lambda* reikšmių iš intervalo nurodyto punktuose (1),(2)

(4) startinė Veibulo skirstinio *k* parametro reikšmė

(5) galinė Veibulo skirstinio *k* parametro reikšmė

(6) nurodo kiek reikia paimti parametro *k* reikšmių ir intervalo nurodyto punktuose (4),(5)

(7) nurodo Veibulo skirstinio tankio funkcijos diskretizavimo žingsnelio dydį

(8) nurodo iki kurios reikšmės reikia atlikti Veibulo skirstinio tankio funkcijos diskretizavimą

(9) iškviečia procedūrą, kuri duomenis surašo į nurodytą failą

(10) nurodo kiek gali nukrypti diskretizuoto skirstinio vidurkis

(11) nurodo kiek gali nukrypti diskretizuoto skirstinio standartinis nuokrypis

(12) užkraunamas rezultatų failas, kurį sukūrė atitinkama C++ programa pasinaudodama pradiniais duomenimis išvestais (9) punkte

(13) parodo pasirinktą Veibulo skirstinį aproksimuojančio fazinio skirstinio duomenis

(14) nurodo kiek skirstinių tenkina punktuose (10),(11) nustatytus diskretizavimo reikalavimus, procentais

(15) į failų sistemą išveda visų Veibulo skirstinių bei juos aproksimuojančių fazinių skirstinių tankio funkcijų grafikus (*.png formatu)

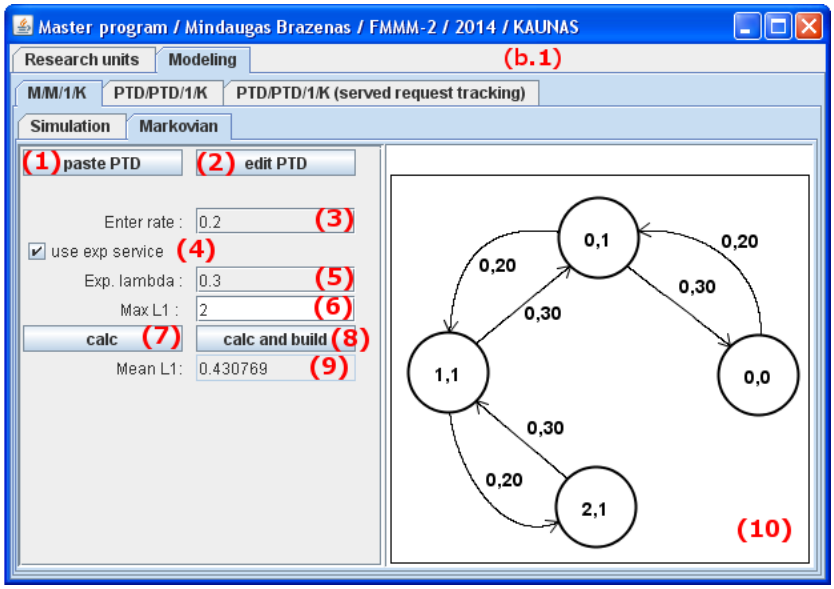
(16) lentelė, kurioje galima pasirinkti Veibulo skirstinio parametro *k* reikšmę

(17) lentelė, kurioje galima pasirinkti Veibulo skirstinio parametro *lambda* reikšmę

(18) galima pasirinkti statistiką, kuri bus vaizduojama (19) grafike

(19) vaizduojama punkte (18) pasirinktos statistikos gradientinis grafikas

(20)rodomas pasirinkto Veibulo skirstinio bei jį aproksimuojančio fazinio skirstinio tankio funkcijos



3.5 pav. Java programos b.1 dalies vartotojo sąsaja

Programos dalių b.1 (pav. 3.5) vartotojo sąsajos elementų paaiškinimas:

(1). iš atmintinės (*clipboard*) nukopijuojami fazinio skirstinio duomenys

(2) redaguojami fazinio skirstinio duomenys

(3) paraiškų atėjimo intensyvumas (laiko tarpai tarp paraiškų atėjimų turi eksponentinį skirstinį)

(4) galim pasirinkti ar paraiškų aptarnavimo laikai turi eksponentinį skirstinį (su intensyvumu nurodytu (5) punkte) ar fazinį skirstinį (nurodytą punktuose (1), (2))

(5) nurodomas paraiškų aptarnavimo intensyvumas, galioja kai pasirenkamas eksponentinis paraiškų aptarnavimas (4) punkte

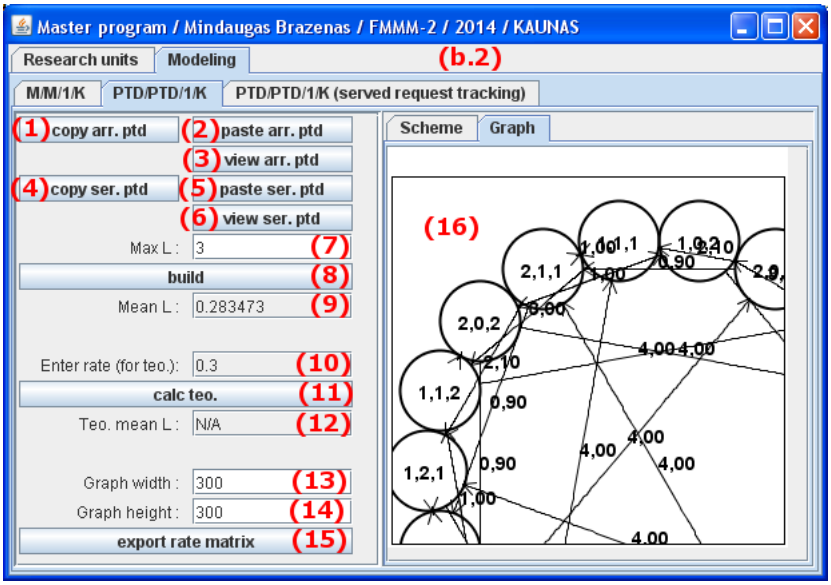
(6) nurodoma maksimali laukimo eilės talpa

(7) iškviečiama procedūra, kuri suranda Markovo proceso grandinę, apskaičiuoja vidutinį laukiančių paraiškų kiekį

(8) iškviečiama procedūra, kuri suranda Markovo proceso grandinę, ją atvaizduoja schematiškai (10) punkte ir apskaičiuoja vidutinį laukiančių paraiškų kiekį

(9) vidutinis laukiančių paraiškų kiekis laukimo eilėje

(10) sukonstruotos Markovo proceso grandinės schema, kurios elementus galima redaguoti tiesiogiai pelytės pagalba



3.6 pav. Java programos b.2 dalies vartotojo sąsaja

Vartotojo sąsajos (pav. 3.6) elementų paaiškinimas:

(1). paraiškų aptarnavimo laikų fazinis skirstinys nukopijuojamas į atmintinę

(2) paraiškų aptarnavimo laikų fazinio skirstinio duomenys nukopijuojami iš atmintinės

(3) redaguojamas paraiškų aptarnavimo laikų fazinio skirstinio duomenys

(4) paraiškų atėjimo trukmių fazinis skirstinys nukopijuojamas į atmintinę

(5) paraiškų atėjimo trukmių fazinio skirstinio duomenys nukopijuojami iš atmintinės

(6) redaguojamas paraiškų atėjimo trukmių fazinio skirstinio duomenys

(7) nurodoma maksimali paraiškų laukimo eilės talpa

(8) iškviečiama procedūra, kuri sukonstruoja Markovo proceso grandinę, ją atvaizduoja schematiškai (13) punkte ir apskaičiuoja vidutinį laukiančių paraiškų kiekį laukimo eilėje

(9) vidutinis laukiančių laukimo eilėje paraiškų kiekis

(10) paraiškų atėjimo intensyvumas, čia laikai per kuriuos ateina naujos paraiškos turi eksponentinį skirstinį

(11) iškviečia procedūra kuri suskaičiuoja aptarnavimo sistemos M/G/1/K teorinį vidutinį laukiančių paraiškų kiekį (kai laukimo eilės ilgis neribojamas), kai paraiškų atėjimų trukmės turi

eksponentinį skirstinį su intensyvumu nurodytu (10) punkte, o paraiškų aptarnavimo laikai turi fazinį skirstinį redaguojamą punktuose (4),(5),(6)

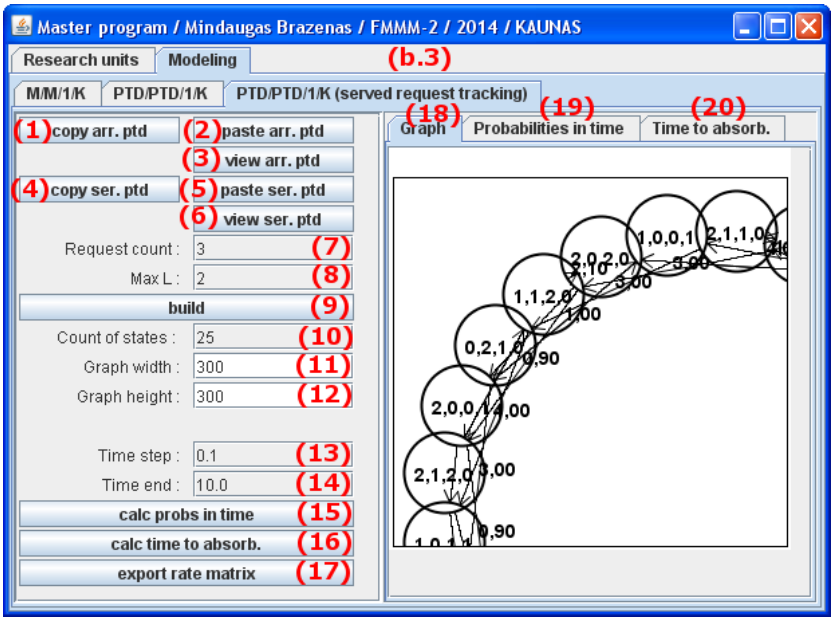
(12) suskaičiuotas teorinis vidutinis laukiančių paraiškų kiekis, žr. (11) punktą

(13) nurodo (16) schemos plotį pikseliais

(14) nurodo (16) schemos aukštį pikseliais

(15) iškviečia procedūrą, kurios metų į nurodytą failą yra išvedami perėjimų tarp būsenų intensyvumų matricos Q duomenys

(16) sukonstruotos Markovo proceso grandinės schema (p.s. čia taip atrodo programos sugeneruotas būsenų grafas, pagal poreikį jį galima redaguoti rankiniu būdu)



3.7 pav. Java programos b.3 dalies vartotojo sąsaja

Vartotojo (pav. 3.7) sąsajos elementų paaiškinimas:

(1). paraiškų aptarnavimo laikų fazinis skirstinys nukopijuojamas į atmintinę

(2) paraiškų aptarnavimo laikų fazinio skirstinio duomenys nukopijuojami iš atmintinės

(3) redaguojamas paraiškų aptarnavimo laikų fazinio skirstinio duomenys

(4) paraiškų atėjimo trukmių fazinis skirstinys nukopijuojamas į atmintinę

(5) paraiškų atėjimo trukmių fazinio skirstinio duomenys nukopijuojami iš atmintinės

(6) redaguojamas paraiškų atėjimo trukmių fazinio skirstinio duomenys

(7) nurodomas paraiškų kiekis

(8) nurodoma maksimali paraiškų laukimo eilės talpa

(9) iškviečiama procedūra, kuri sukonstruoja Markovo proceso grandinę ir ją atvaizduoja schematiškai (18) punkte

(10) sugeneruotos Markovo proceso grandinės būsenų kiekis

(11) nurodo Markovo proceso grandinės schemos plotį, pikseliais
(12) nurodo Markovo proceso grandinės schemos aukštį, pikseliais
(13) nurodo laiko žingsnelio didumą
(14) nurodo galutinį laiko momentą
(15) iškviečia procedūrą, kuri apskaičiuoja kaip kinta sistemos būsenų tikimybės laike, rezultatai parodomi (19) punkte, naudojami punktų (13),(14) nustatymai
(16) iškviečiama procedūra, kuri sukonstruoja fazinį skirstinį pagal kurį yra pasiskirstęs laikas per kurį pasiekiama sistemos sugeriančioji būsena (t.y. aptarnaujamas numatytas kiekis paraiškų); šio skirstinio tankio funkcija pavaizduojama (20) punkte
(17) iškviečiama procedūra, kuri į pasirinktą failą išveda perėjimų tarp būsenų intensyvumų matricos Q duomenis
(18) rodomas sukonstruotos Markovo proceso grandinės schema
(19) rodomas grafikas, kaip kinta sistemos būsenų tikimybės laike
(20) rodoma fazinio skirstinio (pagal kurį yra pasiskirstęs laikas per kurį aptarnaujamas numatytas kiekis paraiškų) tankio funkcija

Fazinio skirstinio parametrų radimas (dėl didelio optimizuojamų kintamųjų kiekio) yra sudėtingas uždavinys. Kadangi nagrinėti paieškos metodai (žr. skyr . 1.1) yra stochastiniai – tiriant jų efektyvumą paieškos procesą reikia pakartoti daug kartų (pvz. 100) tam, kad būtų galima palyginti metodų suvidurkintus rezultatus. Kadangišis tyrimas yra labai reiklus skaičiavimo resursams buvo nuspręsta programą rašyti C++ kalba, kuri leidžia tiesiogiai panaudoti CUDA technologiją. Ieškant fazinio skirstinio parametrų daugiausiai skaičiavimo resursų pareikalauja tikslo funkcijos $\hat{S}$ apskaičiavimas. Šiame darbe tikslo funkcijos $\hat{S}$ reikšmės buvo skaičiuojamos vaizdo procesoriaus pagalba (GPU), tai leido stipriai sumažinti tyrimo laiką. Šį faktą pagrįsime nedidelio tyrimo rezultatais. Buvo atsitiktinai sugeneruota 100000 fazinių skirstinių (su trimis būsenomis) ir apskaičiuotos jų tikslo funkcijų $\hat{S}$ reikšmės, kurios atspindėjo kaip šie faziniai skirstiniai aproksimuoja 2.1.1 skyrelyje pateiktą Veibulo (C) skirstinį. Skaičiavimai buvo atlikti naudojant įprastą procesorių (CPU) ir vaizdo procesorių (GPU). Išmatuotos skaičiavimų trukmės (išmatuotos sekundėmis) pateiktos lentelėje 3.1:

3.1 lentelė. Tikslo funkcijos skaičiavimo laikų palyginimas, kai naudojamas įprastas procesorius (CPU) arba vaizdo procesorius (GPU)

Eksperimento numeris	CPU (Intel core 2 Duo, E4500, 2.2GHz)	GPU (GT 640)
1	52.78	0.73
2	52.91	0.75
3	53.48	0.77
Vidurkis	53.06	0.75

Iš rezultatų (lent. 3.1) matosi, kad skaičiavimai gali būti atlikti iki 70 kartų greičiau pasinaudojant vaizdo procesoriaus skaičiavimo galimybėmis.

IŠVADOS

1. Iš tirtų paieškos metodų (RND, PSO, LUS) fazinio skirstinio parametrams rasti geriausiai tiko LUS metodas.
2. Nustatyta, kad aproksimavimo faziniais skirstiniais tikslumas priklauso nuo aproksimuojamos tankio funkcijos pavidalo.
3. Trijų būsenų faziniai skirstiniai geriausiai aproksimuoja (subjektyviai vertinant) Veibulo skirstinius, kurių *k* parametro reikšmė yra arti vieneto, o *lambda* – daugiau už 0.25.
4. Skaičiavimai atlikti panaudojant vaizdo procesorių (GPU) ir tai leido tyrimo laiką sutrumpinti iki kelių dešimčių kartų.
5. Pasiūlytas algoritmas sistemos būsenų aibei ir perėjimo intensyvumų matricai generuoti.
6. Pasiūlytasmetodasgali būti panaudotas įvairių stochastinių sistemų modeliavimui, kuriose atsitiktinės trukmės tarp gretimų įvykių turi nemarkoviškus skirstinius.

LITERATŪRA

[1]Bo Friis Nielsen, “Lecture notes on phase-type distribution for 02407 Stochastic Processes”, 2012

[2]János Sztrik, “Basic queueing theory”, University of Debrecen, Faculty of Informatics

[3] Ward Whitt, “Continuous-time Markov chains”, Department of Industrial Engineering and Operations Research, Columbia University, New York, 2012

[4] Ramesh Johari, “Lecture 12 notes : Introducing Continuous Time Markov Chains” , MS&E

[5]R. Nelson. Probability, stochastic processes, and queueing theory: themathematics of computer performance modelling. Springer, 1995.

[6] Ben Niu, Yunlong Zhu, Xiaoxian He, Henry Wu, “MCPSO: A multi-swarm cooperative parcticle swam optimizer”, Applied Mathematics and Computation 185 (2007) 1050-1062, ELSEVIER

[7]Y. Shi, R.C. Eberhart, A modified particle swarm optimizer, in: Proceedings of IEEE International Conference on EvolutionaryComputation, Anchorage, AK, May 1998, pp. 69–73.

[8] Magnus Erik Hvass Pedersen, Andrew John Chipperfield Hvass, “Local Unimodal Sampling”, Laboratories Technical Report no. HL0801, 2008

[9] Søren Asmussen, Olle Nerman, Marita Olsson, “Fitting Phase-Type Distributions via the EM Algorithm”, Scandinavian Journal of Statistics, Vol. 23, No. 4 (Dec., 1996), pp. 419-441

[10]Salah E. Elmaghraby, Rachid Benmansour, Abdelhakim Artiba, Hamid Allaoui, “On TheApproximation of Arbitrary Distributions by Phase-Type Distributions”,

[11] Inscape programas internetinė svetainė: http://www.inkscape.org

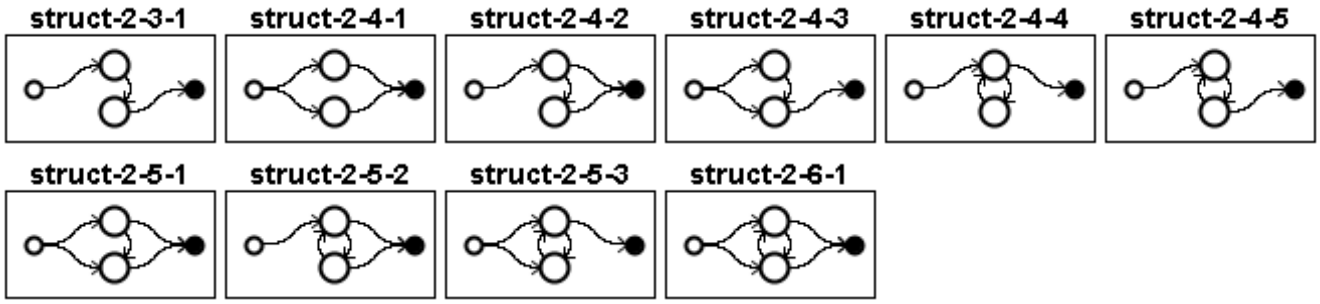
[12] Eclipse programos internetinė svetainė: http://www.eclipse.org/org/

[13] CodeBlocks programos internetinė svetainė:http://www.codeblocks.org/

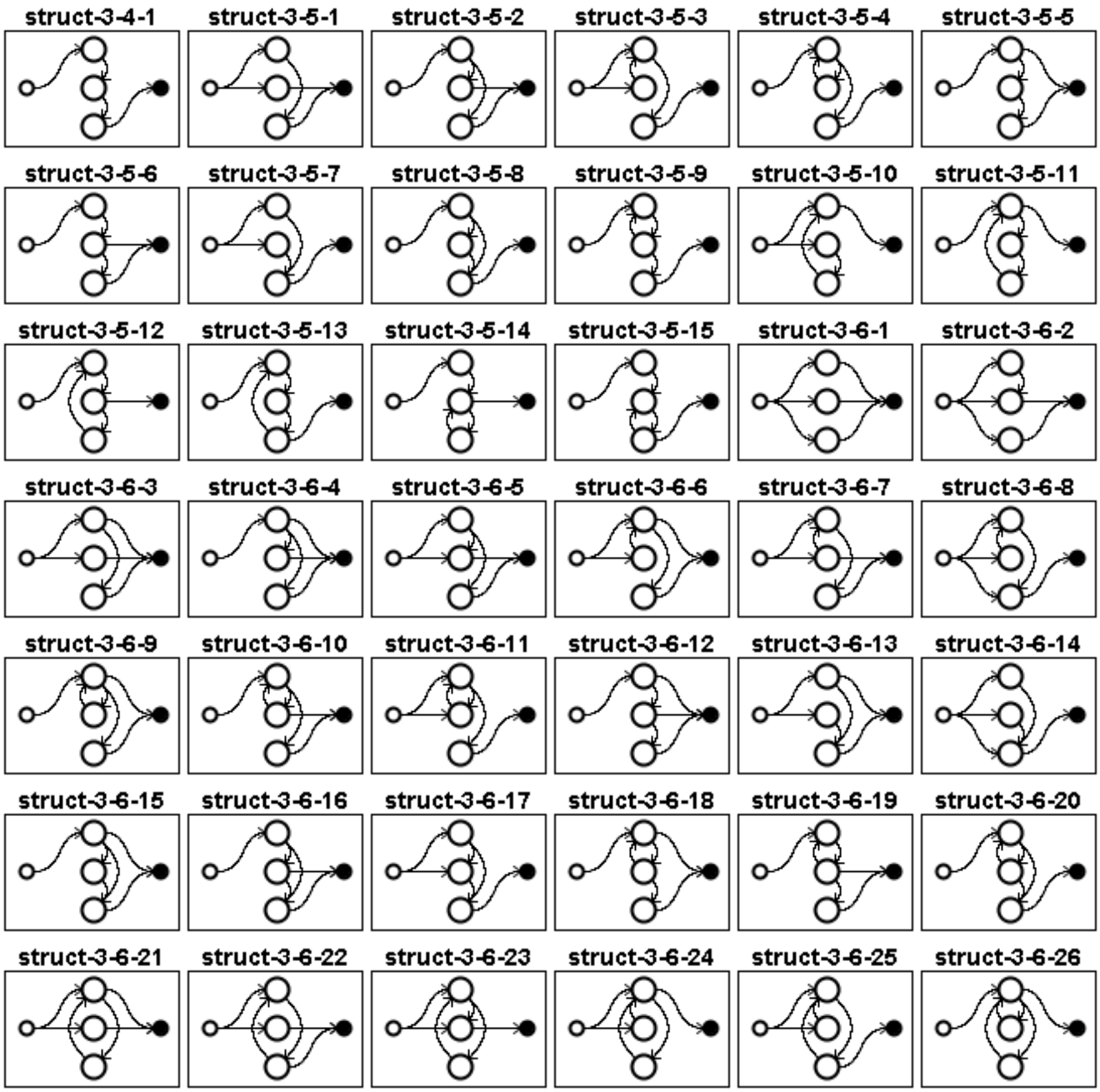
[14] CUDA technologijos paketas: https://developer.nvidia.com/gpu-computing-sdk

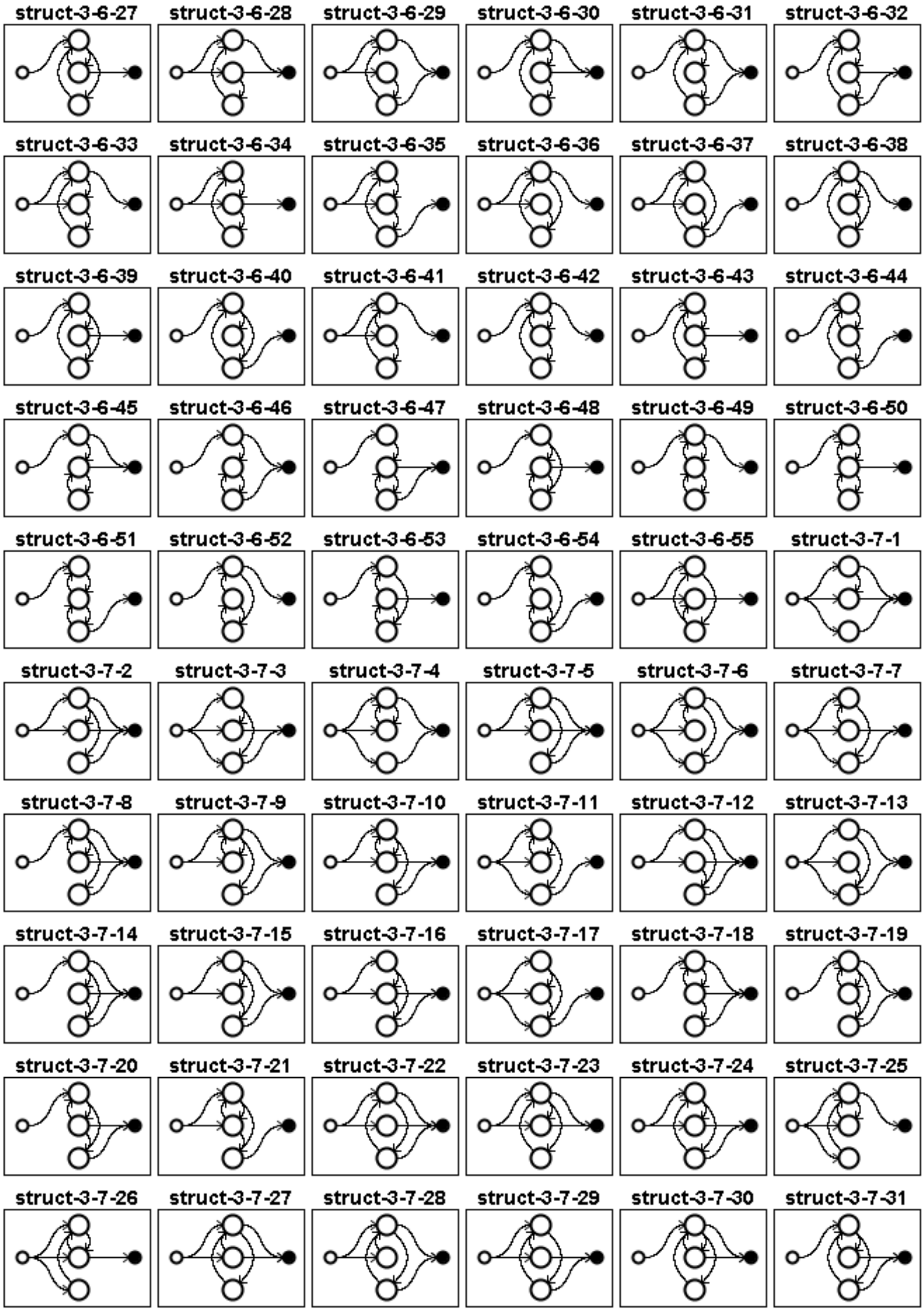
1 PRIEDAS.Fazinių skirstinių, su dvejomis bei trejomis būsenomis, visos galimos struktūros

Fazinių skirstinių su dvejomis būsenomis visos galimos struktūros:

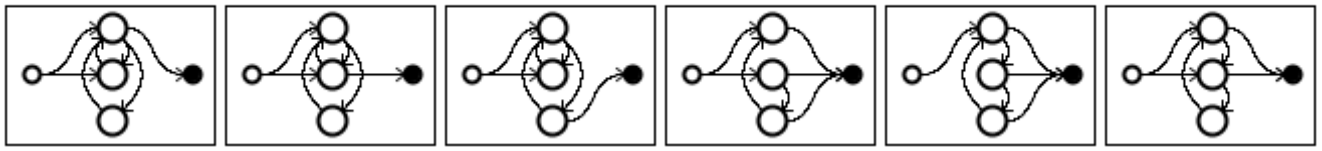


Fazinių skirstinių su trejomis būsenomis visos galimos struktūros:

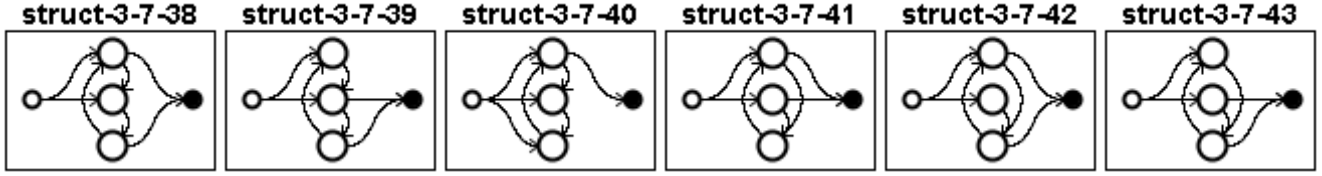



struct-3-6-27
struct-3-6-28
struct-3-6-29
struct-3-6-30
struct-3-6-31
struct-3-6-32
struct-3-6-33
struct-3-6-34
struct-3-6-35
struct-3-6-36
struct-3-6-37
struct-3-6-38
struct-3-6-39
struct-3-6-40
struct-3-6-41
struct-3-6-42
struct-3-6-43
struct-3-6-44
struct-3-6-45
struct-3-6-46
struct-3-6-47
struct-3-6-48
struct-3-6-49
struct-3-6-50
struct-3-6-51
struct-3-6-52
struct-3-6-53
struct-3-6-54
struct-3-6-55
struct-3-7-1
struct-3-7-2
struct-3-7-3
struct-3-7-4
struct-3-7-5
struct-3-7-6
struct-3-7-7
struct-3-7-8
struct-3-7-9
struct-3-7-10
struct-3-7-11
struct-3-7-12
struct-3-7-13
struct-3-7-14
struct-3-7-15
struct-3-7-16
struct-3-7-17
struct-3-7-18
struct-3-7-19
struct-3-7-20
struct-3-7-21
struct-3-7-22
struct-3-7-23
struct-3-7-24
struct-3-7-25
struct-3-7-26
struct-3-7-27
struct-3-7-28
struct-3-7-29
struct-3-7-30
struct-3-7-31

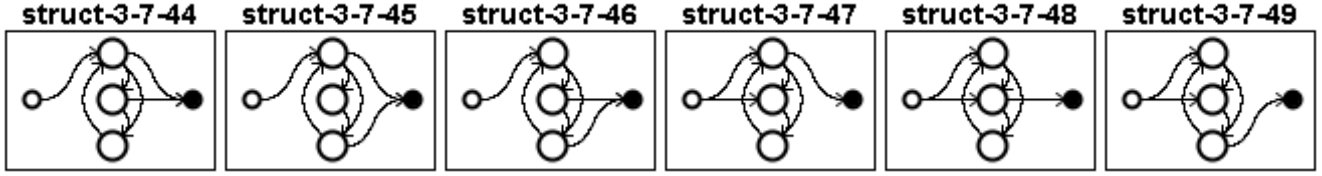
struct-3-7-32 struct-3-7-33 struct-3-7-34 struct-3-7-35 struct-3-7-36 struct-3-7-37



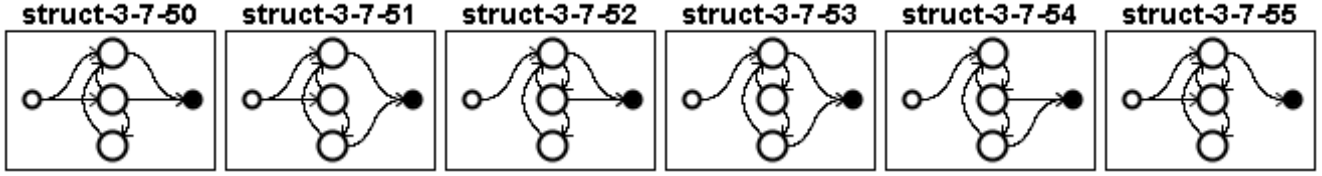
struct-3-7-38 struct-3-7-39 struct-3-7-40 struct-3-7-41 struct-3-7-42 struct-3-7-43



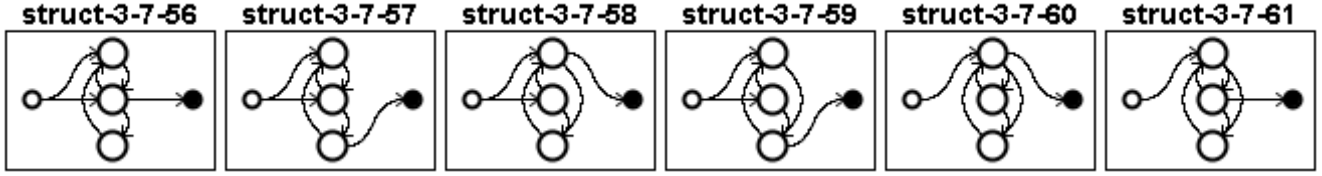
struct-3-7-44 struct-3-7-45 struct-3-7-46 struct-3-7-47 struct-3-7-48 struct-3-7-49



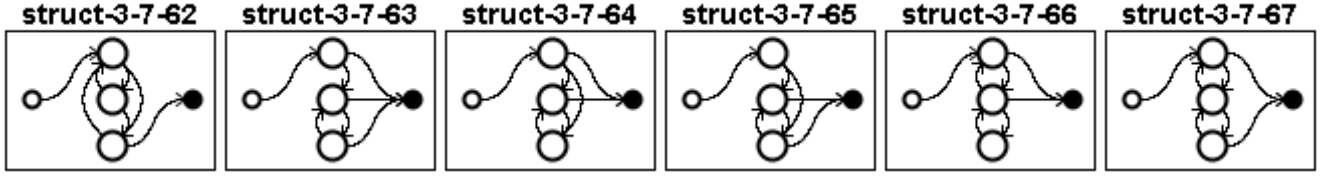
struct-3-7-50 struct-3-7-51 struct-3-7-52 struct-3-7-53 struct-3-7-54 struct-3-7-55



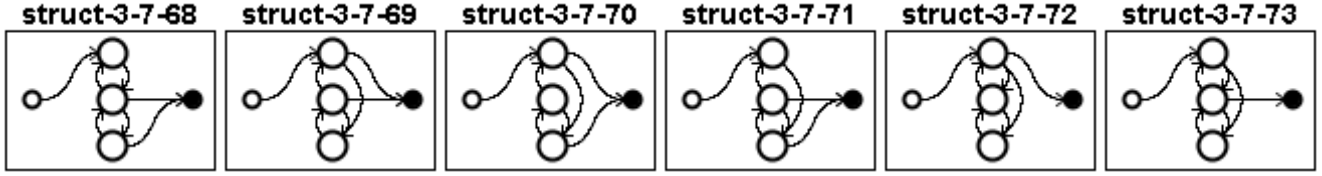
struct-3-7-56 struct-3-7-57 struct-3-7-58 struct-3-7-59 struct-3-7-60 struct-3-7-61



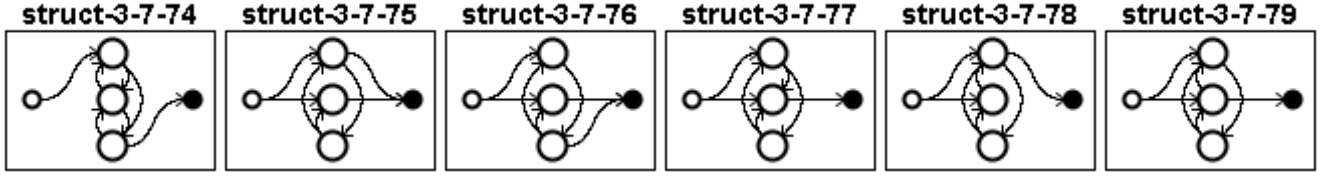
struct-3-7-62 struct-3-7-63 struct-3-7-64 struct-3-7-65 struct-3-7-66 struct-3-7-67



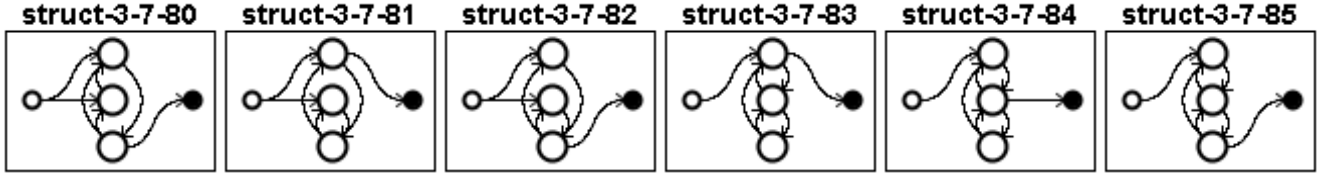
struct-3-7-68 struct-3-7-69 struct-3-7-70 struct-3-7-71 struct-3-7-72 struct-3-7-73



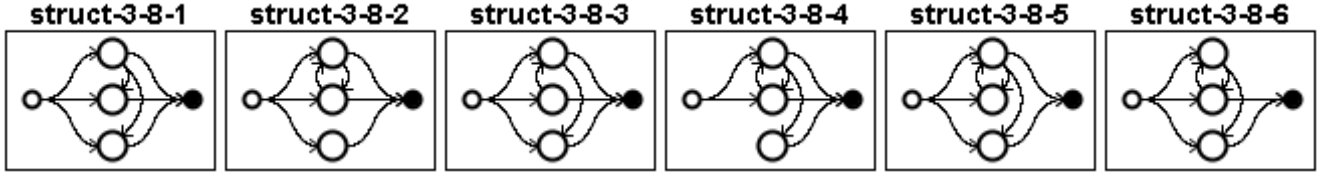
struct-3-7-74 struct-3-7-75 struct-3-7-76 struct-3-7-77 struct-3-7-78 struct-3-7-79



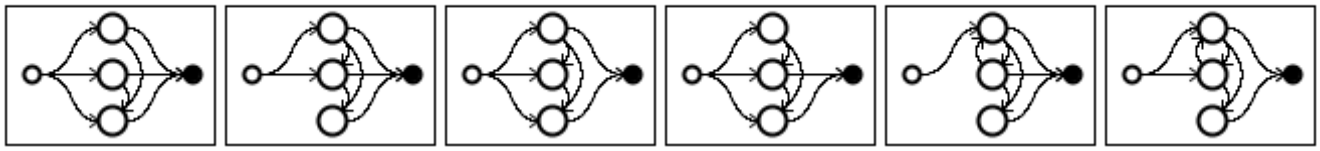
struct-3-7-80 struct-3-7-81 struct-3-7-82 struct-3-7-83 struct-3-7-84 struct-3-7-85



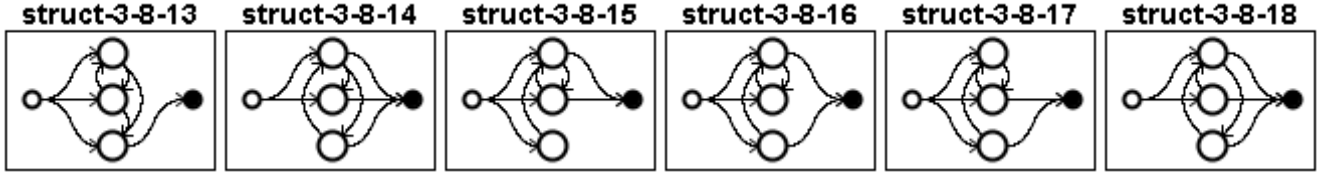
struct-3-8-1 struct-3-8-2 struct-3-8-3 struct-3-8-4 struct-3-8-5 struct-3-8-6



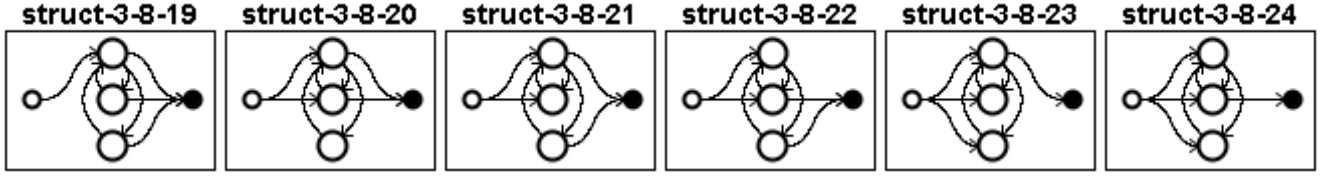
struct-3-8-7 struct-3-8-8 struct-3-8-9 struct-3-8-10 struct-3-8-11 struct-3-8-12



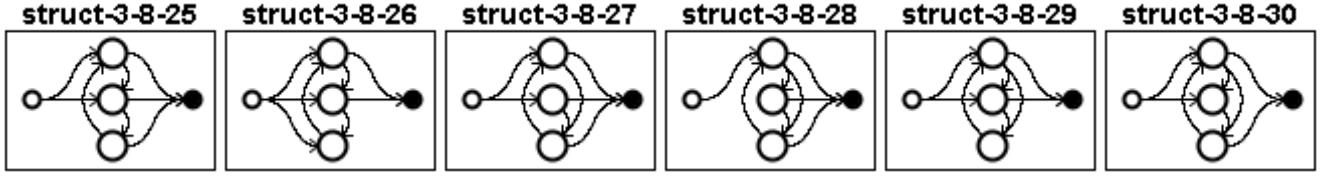
struct-3-8-13 struct-3-8-14 struct-3-8-15 struct-3-8-16 struct-3-8-17 struct-3-8-18



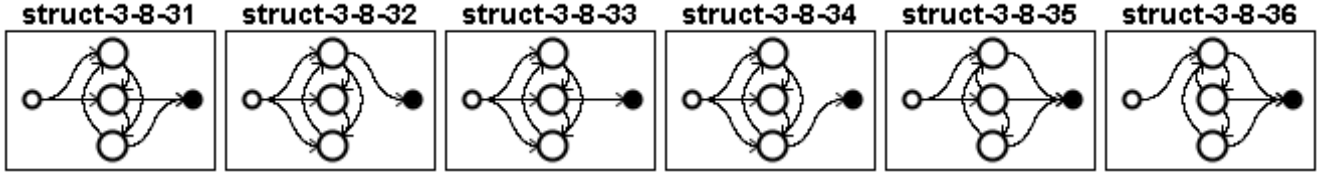
struct-3-8-19 struct-3-8-20 struct-3-8-21 struct-3-8-22 struct-3-8-23 struct-3-8-24



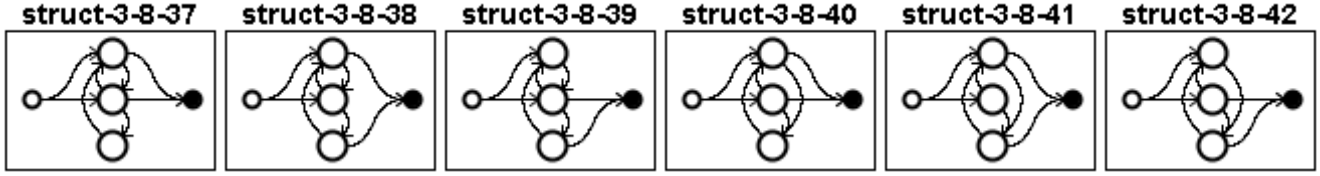
struct-3-8-25 struct-3-8-26 struct-3-8-27 struct-3-8-28 struct-3-8-29 struct-3-8-30



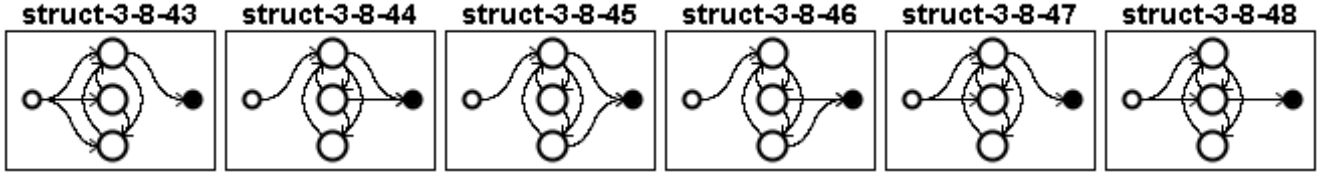
struct-3-8-31 struct-3-8-32 struct-3-8-33 struct-3-8-34 struct-3-8-35 struct-3-8-36



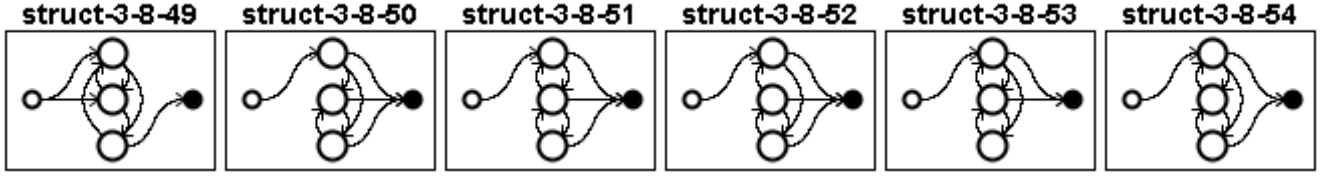
struct-3-8-37 struct-3-8-38 struct-3-8-39 struct-3-8-40 struct-3-8-41 struct-3-8-42



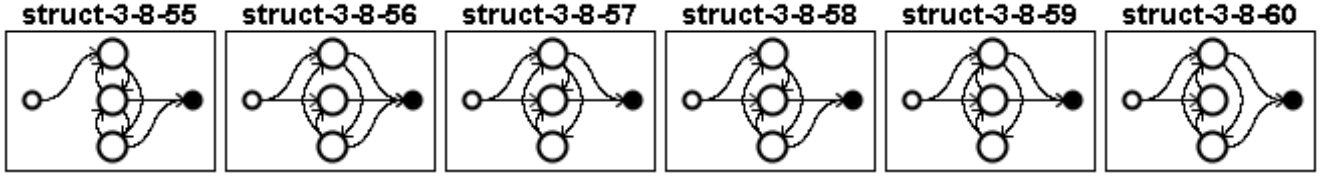
struct-3-8-43 struct-3-8-44 struct-3-8-45 struct-3-8-46 struct-3-8-47 struct-3-8-48



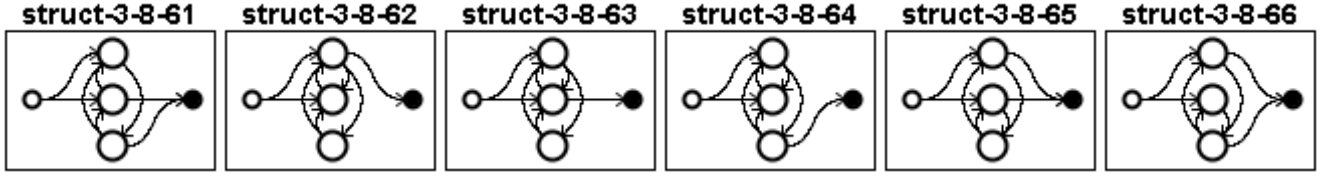
struct-3-8-49 struct-3-8-50 struct-3-8-51 struct-3-8-52 struct-3-8-53 struct-3-8-54

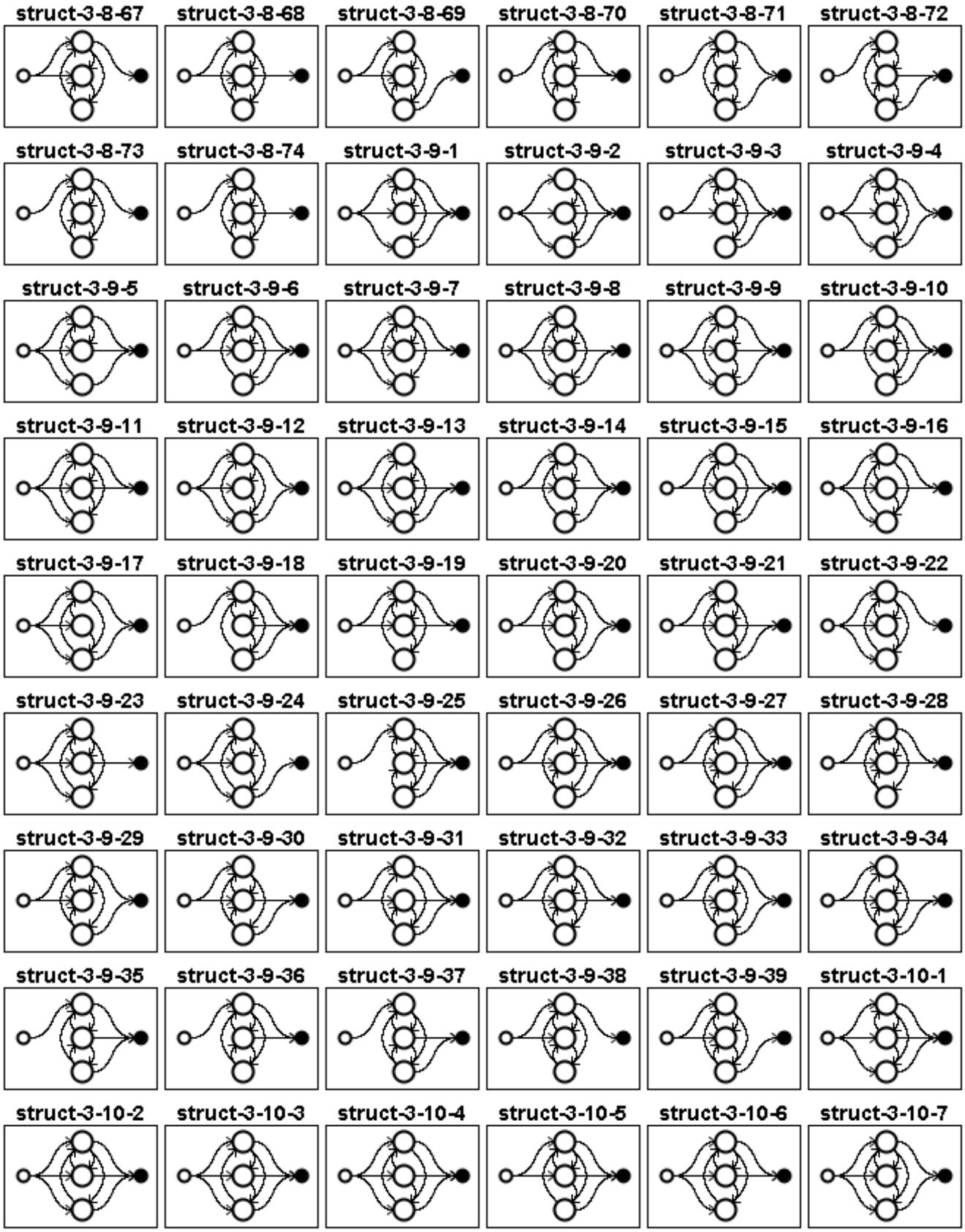


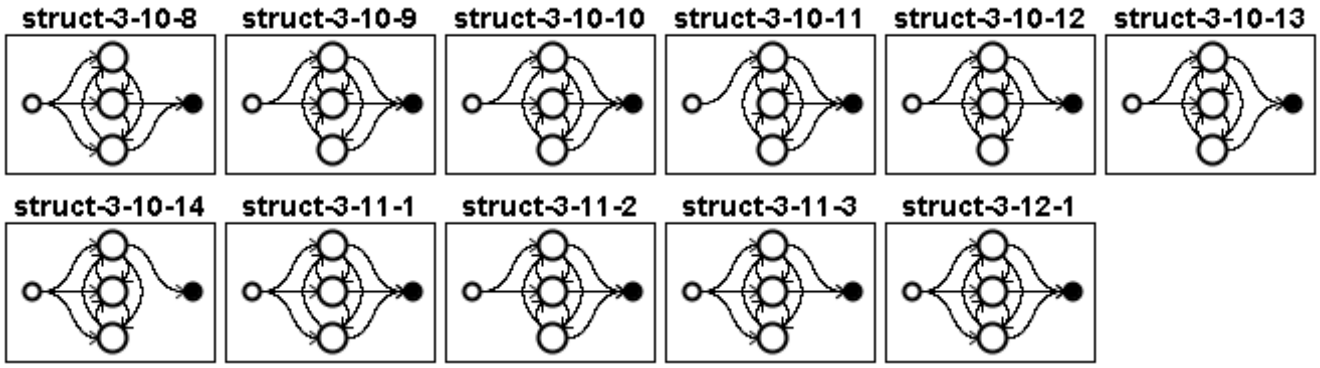
struct-3-8-55 struct-3-8-56 struct-3-8-57 struct-3-8-58 struct-3-8-59 struct-3-8-60



struct-3-8-61 struct-3-8-62 struct-3-8-63 struct-3-8-64 struct-3-8-65 struct-3-8-66




struct-3-8-67
struct-3-8-68
struct-3-8-69
struct-3-8-70
struct-3-8-71
struct-3-8-72
struct-3-8-73
struct-3-8-74
struct-3-9-1
struct-3-9-2
struct-3-9-3
struct-3-9-4
struct-3-9-5
struct-3-9-6
struct-3-9-7
struct-3-9-8
struct-3-9-9
struct-3-9-10
struct-3-9-11
struct-3-9-12
struct-3-9-13
struct-3-9-14
struct-3-9-15
struct-3-9-16
struct-3-9-17
struct-3-9-18
struct-3-9-19
struct-3-9-20
struct-3-9-21
struct-3-9-22
struct-3-9-23
struct-3-9-24
struct-3-9-25
struct-3-9-26
struct-3-9-27
struct-3-9-28
struct-3-9-29
struct-3-9-30
struct-3-9-31
struct-3-9-32
struct-3-9-33
struct-3-9-34
struct-3-9-35
struct-3-9-36
struct-3-9-37
struct-3-9-38
struct-3-9-39
struct-3-10-1
struct-3-10-2
struct-3-10-3
struct-3-10-4
struct-3-10-5
struct-3-10-6
struct-3-10-7



2 PRIEDAS. Programų kodų ištraukos

Pateiksime svarbesnių skaičiavimų programų kodus.

Paieškos metodo LUS programos kodas (C++):

```
class SiAlgLus : public SwarmIntel {
private:

    double mBeta;
    int mSampleCount;

    float *mModifParticleLocArray;
    int *mFailureCounterArray;
    double *mRangeArray;
    float *mTfvArray;
    float *mModifTfvArray;

    float *mLowerBounds;
    float *mUpperBounds;
public:
    SiAlgLus(int particleCount, SiProblem *siProblem, int sampleCount, double beta);
    void calc(int iterCount);
    void init();
    void peformIteration(int iter, int iterCount);
    void finish();
};

SiAlgLus::SiAlgLus(int particleCount, SiProblem *siProblem, int sampleCount , double beta)
    : SwarmIntel(particleCount, siProblem)
{
    mSampleCount = sampleCount;
    mBeta = beta;
}

void SiAlgLus::calc(int iterCount)
{
    init();

    if(cSiProblem->sip_outputIterData())
        cSiProblem->sip_iterDataOutput(0, mParticleCount, mParticleLocArray, mGlobalBestParLoc,
mGlobalBestParTfv);

    for(int iter = 1; iter <= iterCount; iter++)
    {
        peformIteration(iter, iterCount);

        if(cSiProblem->sip_outputIterData())
            cSiProblem->sip_iterDataOutput(iter, mParticleCount, mParticleLocArray,
mGlobalBestParLoc, mGlobalBestParTfv);

    }

    cSiProblem->sip_result(mGlobalBestParLoc, mGlobalBestParTfv);
```

```
    finish();

}

void SiAlgLus::init()
{

    mParticleLocArray = (float*)malloc(sizeof(float)*mNN);
    mModifParticleLocArray = (float*)malloc(sizeof(float)*mNN);
    mGlobalBestParLoc = (float*)malloc(sizeof(float)*mVariableCount);
    mGlobalBestParTfv = FLT_MAX;

    mIterBestParLoc = (float*)malloc(sizeof(float)*mVariableCount);
    mIterBestParTfv = FLT_MAX;
    //
    mTfvArray = (float*) malloc(sizeof(float)*mParticleCount);
    mModifTfvArray = (float*) malloc(sizeof(float)*mParticleCount);

    mFailureCounterArray = (int*)malloc(sizeof(int)*mParticleCount);
    mRangeArray = (double*)malloc(sizeof(double)*mNN);

    mLowerBounds = (float*)malloc(sizeof(float)*mVariableCount);
    mUpperBounds = (float*)malloc(sizeof(float)*mVariableCount);
    ///
    cSiProblem->sip_variableBounds(mLowerBounds, mUpperBounds);
    for(int parId = 0; parId < mParticleCount; parId++)
    {
        mFailureCounterArray[parId] = 0;
        for(int varId = 0; varId < mVariableCount; varId++)
            mRangeArray[parId*mVariableCount + varId] = mUpperBounds[varId]-mLowerBounds[varId];
    }


    cSiProblem->sip_evaluateParticles(mParticleCount, mParticleLocArray, mTfvArray);

/// update best particle info
    int bestIterIndex = 0;

    for(int parId = 0; parId < mParticleCount; parId++)
        if(mTfvArray[parId] < mTfvArray[bestIterIndex])
            bestIterIndex = parId;

    for(int varId = 0; varId < mVariableCount; varId++)
        mIterBestParLoc[varId] = mParticleLocArray[bestIterIndex*mVariableCount+varId];
    mIterBestParTfv = mTfvArray[bestIterIndex];

    if(mIterBestParTfv<mGlobalBestParTfv)
    {
        for(int varId = 0; varId < mVariableCount; varId++)
            mGlobalBestParLoc[varId] = mIterBestParLoc[varId];
        mGlobalBestParTfv = mIterBestParTfv;
    }

}
void SiAlgLus::peformIteration(int iter, int iterCount)
{

    float delta;
    double shrinkFactor;
    for(int sampleId = 0; sampleId < mSampleCount; sampleId++)
    {
        for(int parId = 0; parId < mParticleCount; parId++)
        {

            for(int varId = 0; varId < mVariableCount; varId++)
            {
                delta = (((float)rand() / (float)RAND_MAX)*2-
1)*mRangeArray[parId*mVariableCount+varId];
                mModifParticleLocArray[parId*mVariableCount+varId] =
mParticleLocArray[parId*mVariableCount+varId]+delta;
                if(mModifParticleLocArray[parId*mVariableCount+varId] < mLowerBounds[varId])
                    mModifParticleLocArray[parId*mVariableCount+varId] = mLowerBounds[varId];
                if(mModifParticleLocArray[parId*mVariableCount+varId] > mUpperBounds[varId])
                    mModifParticleLocArray[parId*mVariableCount+varId] = mUpperBounds[varId];
            }
        }

        cSiProblem->sip_evaluateParticles(mParticleCount, mModifParticleLocArray, mModifTfvArray);
```

```
        for(int parId = 0; parId < mParticleCount; parId++)
        {
            if(mModifTfvArray[parId]> mTfvArray[parId])
            {
                mFailureCounterArray[parId]++;
                shrinkFactor = pow(0.5, mBeta/(double)mFailureCounterArray[parId]);
                for(int varId = 0; varId < mVariableCount; varId++)
                    mRangeArray[parId*mVariableCount+varId] *= shrinkFactor;

            }
            else
            {
                mTfvArray[parId] = mModifTfvArray[parId];
                for(int varId = 0; varId < mVariableCount; varId++)
                    mParticleLocArray[parId*mVariableCount+varId] =
mModifParticleLocArray[parId*mVariableCount+varId];

            }
        }
    }

/// update best particle info
    int bestIterIndex = 0;

    for(int parId = 0; parId < mParticleCount; parId++)
        if(mTfvArray[parId] < mTfvArray[bestIterIndex])
            bestIterIndex = parId;

    for(int varId = 0; varId < mVariableCount; varId++)
        mIterBestParLoc[varId] = mParticleLocArray[bestIterIndex*mVariableCount+varId];
    mIterBestParTfv = mTfvArray[bestIterIndex];

    if(mIterBestParTfv<mGlobalBestParTfv)
    {
        for(int varId = 0; varId < mVariableCount; varId++)
            mGlobalBestParLoc[varId] = mIterBestParLoc[varId];
        mGlobalBestParTfv = mIterBestParTfv;
    }
}
void SiAlgLus::finish()
{

    free(mParticleLocArray);
    free(mModifParticleLocArray);

    free(mGlobalBestParLoc);
    free(mIterBestParLoc);
    free(mTfvArray);
    free(mModifTfvArray);

    free(mFailureCounterArray);
    free(mRangeArray);

    free(mUpperBounds);
    free(mLowerBounds);


}
```

Programos kodas, skirtas fazinių skirstnių (su trejomis būsenomis) tikslo funkcijoms $\hat{S}$ apskaičiuoti pasinaudojant CUDA technologija (C++):

```
__global__ void kernel_3_multi(int groupCount, int groupSize, int passCount, int N, float step, const
float* dvc_ptdData, const float* xArray, const float *denArray, float* dvc_tfv)
{
    extern __shared__ float s[];
    int groupId =  (blockIdx.x - blockIdx.x % groupSize) / groupSize;
    float ra, rb, rc;
    float *pi = &s[blockDim.x];
    float *t = &s[blockDim.x+3];
    float *T = &s[blockDim.x+3+3+threadIdx.x*3*3];
    float *memory = &s[blockDim.x+3+3+blockDim.x*3*3 + threadIdx.x*3*3*2];
    float *ptr = (float*)&dvc_ptdData[blockIdx.x*(3*3+3)];
   if(threadIdx.x == 0)
    {
```

```
        cu_ptd_float_get_pi(3, (float*)&dvc_ptdData[blockIdx.x*(3*3+3)], pi);
        ra = ptr[0];
        rb = ptr[1];
        rc = ptr[2];

        pi[0] = ra /(ra+rb+rc);
        pi[1] = rb /(ra+rb+rc);
        pi[2] = rc /(ra+rb+rc);

        t[0] = ptr[3+3*0+0];
        t[1] = ptr[3+3*1+1];
        t[2] = ptr[3+3*2+2];
    }
    __syncthreads();
    s[threadIdx.x]=0.0;
    for(int subId = threadIdx.x*passCount; subId < threadIdx.x*passCount+passCount; subId++)
    {
        if(subId >= N) return;
        ra = ptr[3];
        rb = ptr[4];
        rc = ptr[5];

        T[0*3+0] = -(ra + rb+rc);
        T[0*3+1] = rb;
        T[0*3+2] = rc;

        ra = ptr[6];
        rb = ptr[7];
        rc = ptr[8];
        T[1*3+0] = ra;
        T[1*3+1] = -(ra+rb+rc);
        T[1*3+2] = rc;

        ra = ptr[9];
        rb = ptr[10];
        rc= ptr[11];
        T[2*3+0] = ra;
        T[2*3+1] = rb;
        T[2*3+2] = -(ra+rb+rc);
        ra = xArray[groupId*N + subId];
        T[0] = T[0]*ra;
        T[1] = T[1]*ra;
        T[2] = T[2]*ra;
        T[3] = T[3]*ra;
        T[4] = T[4]*ra;
        T[5] = T[5]*ra;
        T[6] = T[6]*ra;
        T[7] = T[7]*ra;
        T[8] = T[8]*ra;

        cu_expm_f3(T, memory);
        rb = 0.0;

        rc = T[0*3+0]*t[0];
        rc += T[0*3+1]*t[1];
        rc +=T[0*3+2]*t[2];
        rb += pi[0]*rc;

        rc = T[1*3+0]*t[0];
        rc += T[1*3+1]*t[1];
        rc +=T[1*3+2]*t[2];
        rb += pi[1]*rc;

        rc = T[2*3+0]*t[0];
        rc += T[2*3+1]*t[1];
        rc +=T[2*3+2]*t[2];
        rb += pi[2]*rc;

        s[threadIdx.x]+=abs(rb - denArray[groupId*N + subId]);
    }

    __syncthreads();
    if(threadIdx.x == 0)
    {
        float sum = 0.0;
        for(int i = 0; i < blockDim.x; i++)
            sum += s[i];
        dvc_tfv[blockIdx.x] = sum*step;
    }
```

```
}
void cuda_ptd_float_eval_array_3_multi(int groupCount, int groupSize, const float *ptdData, int N,
float step, const float* xArray, const float *denArray, float *tfv){

    int count = groupCount*groupSize;

    int p = 3;
    float *dvc_ptdData = 0;
    float *dvc_xArray = 0;
    float *dvc_denArray = 0;
    float *dvc_tfv = 0;

    int bc_ptdData = sizeof(float)*count*(p*p+p);
    int bc_xArray = sizeof(float)*N*groupCount;
    int bc_denArray = sizeof(float)*N*groupCount;
    int bc_tfv = sizeof(float)*count;

    // allocate device memory
    gpuErrchk(cudaMalloc((void**)&dvc_ptdData, bc_ptdData));
    gpuErrchk(cudaMalloc((void**)&dvc_xArray, bc_xArray));
    gpuErrchk(cudaMalloc((void**)&dvc_denArray, bc_denArray));
    gpuErrchk(cudaMalloc((void**)&dvc_tfv, bc_tfv));

    // copy data from host to device
    gpuErrchk(cudaMemcpy(dvc_ptdData, ptdData, bc_ptdData, cudaMemcpyHostToDevice));
    gpuErrchk(cudaMemcpy(dvc_xArray, xArray, bc_xArray, cudaMemcpyHostToDevice));
    gpuErrchk(cudaMemcpy(dvc_denArray, denArray, bc_denArray, cudaMemcpyHostToDevice));

    int blockCount = count;
    int passCount = ceil((float)N/32.0);
    int threadCount = 32;

int shared_memory = (threadCount+p+p+p*p*threadCount + p*p*2*threadCount)*sizeof(float);
    kernel_3_multi<<<blockCount,threadCount, shared_memory>>>(groupCount, groupSize,  passCount,  N,
step, dvc_ptdData, dvc_xArray, dvc_denArray,  dvc_tfv );

    gpuErrchk(cudaPeekAtLastError());
    gpuErrchk( cudaDeviceSynchronize() );
    gpuErrchk(cudaMemcpy(tfv, dvc_tfv, bc_tfv, cudaMemcpyDeviceToHost));
    gpuErrchk(cudaFree(dvc_ptdData));
    gpuErrchk(cudaFree(dvc_xArray));
    gpuErrchk(cudaFree(dvc_denArray));
    gpuErrchk(cudaFree(dvc_tfv));
}
```

Programos kodo dalis skirta PTD/PTD/1/K aptarnavimo sistemos (kai fiksuojamas aptarnautų paraiškų kiekis) būsenų generavimui, (Java):

```
publicvoid generateRateMatrix() {
mIndexOfObsorbingState = -1;
double aT[] = mArrivalPtd.get_T();
double at[] = mArrivalPtd.get_t();
double api[] = mArrivalPtd.get_pi();

double sT[] = mServicePtd.get_T();
double st[] = mServicePtd.get_t();
double spi[] = mServicePtd.get_pi();

boolean usedStatesA[] = newboolean[mMaxStateCode + 1];
double initialProb[] = newdouble[mMaxStateCode + 1];

List<Integer> allStateList = new ArrayList<>();
List<Integer> stateCodeList = new ArrayList<>();
List<Integer> stateCodeListB = new ArrayList<>();

int code;
int code2;

for (int ai = 0; ai < mAP; ai++) {
code = encode(ai + 1, 0, 0, 0);
stateCodeListB.add(code);
allStateList.add(code);
usedStatesA[code] = true;
initialProb[code] = api[ai];
}
```

```
double rateMatrixRaw[][] = newdouble[mMaxStateCode + 1][mMaxStateCode + 1];
for (int row = 0; row < mMaxStateCode + 1; row++) {
for (int col = 0; col < mMaxStateCode + 1; col++) {
rateMatrixRaw[row][col] = 0.0;
}
}
int queueLength_L;
int queueLength_L_2;
int a_state_id;
int s_state_id;
int a_state_id_2;
int s_state_id_2;
int rC, rC_2;
double rate;
double rate2;
double rate3;
int total_rC;
intcounter = 0;
while (stateCodeListB.size() > 0) {
counter++;
stateCodeList.clear();
stateCodeList.addAll(stateCodeListB);
stateCodeListB.clear();
for (int i = 0; i < stateCodeList.size(); i++) {
code = stateCodeList.get(i);
a_state_id = decode_aid(code);
queueLength_L = decode_L(code);
s_state_id = decode_sid(code);
rC = decode_rC(code);
total_rC = total_requestCount(code);

// arrival->L
if (a_state_id > 0) {
rate = at[a_state_id - 1];
if (rate > 0) {
if (s_state_id > 0) { // put to L
s_state_id_2 = s_state_id;
queueLength_L_2 = queueLength_L + 1;
if (queueLength_L_2 > mMaxL)
queueLength_L_2 = mMaxL;
if (total_rC < mRequestCount) { // -> A
for (int k = 0; k < mAP; k++) {
if (api[k] > 0) {
rate2 = rate * api[k];
a_state_id_2 = k + 1;
rC_2 = rC;
// add
code2 = encode(a_state_id_2, queueLength_L_2, s_state_id_2, rC_2);
if (usedStatesA[code2] == false) {
stateCodeListB.add(code2);
usedStatesA[code2] = true;
allStateList.add(code2);
}
rateMatrixRaw[code][code2] = rate2;
}
}
} else {
a_state_id_2 = 0;
rC_2 = rC;
// add
code2 = encode(a_state_id_2, queueLength_L_2, s_state_id_2, rC_2);
if (usedStatesA[code2] == false) {
stateCodeListB.add(code2);
usedStatesA[code2] = true;
allStateList.add(code2);
}
rateMatrixRaw[code][code2] = rate;
}
} else { // put to S
queueLength_L_2 = 0;
if (total_rC < mRequestCount) { // -> A
for (int k = 0; k < mAP; k++) {
if (api[k] > 0) {
rate2 = rate * api[k];
a_state_id_2 = k + 1;
for (int kk = 0; kk < mSP; kk++) {
if (spi[kk] > 0) {
rate3 = rate2 * spi[kk];
s_state_id_2 = kk + 1;
```

```
rC_2 = rC;
// add
code2 = encode(a_state_id_2, queueLength_L_2, s_state_id_2, rC_2);
if (usedStatesA[code2] == false) {
stateCodeListB.add(code2);
usedStatesA[code2] = true;
allStateList.add(code2);
}
rateMatrixRaw[code][code2] = rate3;
}
}
}
}
} else {
a_state_id_2 = 0;
for (int kk = 0; kk < mSP; kk++) {
if (spi[kk] > 0) {
rate3 = rate * spi[kk];
s_state_id_2 = kk + 1;
rC_2 = rC;
// add
code2 = encode(a_state_id_2, queueLength_L_2, s_state_id_2, rC_2);
if (usedStatesA[code2] == false) {
stateCodeListB.add(code2);
usedStatesA[code2] = true;
allStateList.add(code2);
}
rateMatrixRaw[code][code2] = rate3;
}}}}}
}// end of / arrival->L
               // service->exit
if (s_state_id > 0) {

if (st[s_state_id - 1] > 0) {// exit from S
rate = st[s_state_id - 1];
if (queueLength_L > 0) {

for (int kk = 0; kk < mSP; kk++) { // take one from
// queue

if (spi[kk] > 0) {
rate2 = spi[kk] * rate;
a_state_id_2 = a_state_id;
queueLength_L_2 = queueLength_L - 1;
s_state_id_2 = kk + 1;
rC_2 = rC + 1;
// add
code2 = encode(a_state_id_2, queueLength_L_2, s_state_id_2, rC_2);
if (usedStatesA[code2] == false) {
stateCodeListB.add(code2);
usedStatesA[code2] = true;
allStateList.add(code2);
}
rateMatrixRaw[code][code2] = rate2;
}
}
} else {
a_state_id_2 = a_state_id;

queueLength_L_2 = 0;
s_state_id_2 = 0;
rC_2 = rC + 1;

// add
code2 = encode(a_state_id_2, queueLength_L_2, s_state_id_2, rC_2);
if (usedStatesA[code2] == false) {
stateCodeListB.add(code2);
usedStatesA[code2] = true;
allStateList.add(code2);
}
rateMatrixRaw[code][code2] = rate;
}
}
}
                              // transition inside A
if (a_state_id > 0) {
for (int k = 0; k < mAP; k++) {
if (k == a_state_id - 1)
continue;
```

```
if (aT[(a_state_id - 1) * mAP + k] > 0) {
rate = aT[(a_state_id - 1) * mAP + k];

a_state_id_2 = k + 1;
queueLength_L_2 = queueLength_L;
s_state_id_2 = s_state_id;
rC_2 = rC;
// add
code2 = encode(a_state_id_2, queueLength_L_2, s_state_id_2, rC_2);
if (usedStatesA[code2] == false) {
stateCodeListB.add(code2);
usedStatesA[code2] = true;
allStateList.add(code2);
}
rateMatrixRaw[code][code2] = rate;
}
}
}
// transition inside S
if (s_state_id > 0) {
for (int k = 0; k < mSP; k++) {
if (k == a_state_id - 1)
continue;
if (sT[(s_state_id - 1) * mSP + k] > 0) {
rate = sT[(s_state_id - 1) * mSP + k];
a_state_id_2 = a_state_id;
queueLength_L_2 = queueLength_L;
s_state_id_2 = k + 1;
rC_2 = rC;
// add
code2 = encode(a_state_id_2, queueLength_L_2, s_state_id_2, rC_2);
if (usedStatesA[code2] == false) {
stateCodeListB.add(code2);
usedStatesA[code2] = true;
allStateList.add(code2);
}
rateMatrixRaw[code][code2] = rate;
}
}
}
usedStatesA[code] = true;
}
}
mStateCodeArray = newint[allStateList.size()];
for (int i = 0; i < allStateList.size(); i++) {
mStateCodeArray[i] = allStateList.get(i);
}
mInitialProbArray = newdouble[allStateList.size()];
for (int i = 0; i < allStateList.size(); i++) {
mInitialProbArray[i] = initialProb[allStateList.get(i)];
}
mRateMatrix = newdouble[allStateList.size()][allStateList.size()];
for (int row = 0; row < mStateCodeArray.length; row++) {
for (int col = 0; col < mStateCodeArray.length; col++) {
mRateMatrix[row][col] = rateMatrixRaw[mStateCodeArray[row]][mStateCodeArray[col]];
}
}
mIndexOfObsorbingState = -1;
int absorbingStateCode = encode(0, 0, 0, mRequestCount);
for (int i = 0; i < mStateCodeArray.length; i++) {
if (absorbingStateCode == mStateCodeArray[i]) {
mIndexOfObsorbingState = i;
break;
}
}
for (int row = 0; row < mStateCodeArray.length; row++) {
double sum = 0.0;
for (int col = 0; col < mStateCodeArray.length; col++) {
if (col == row)
continue;
sum += mRateMatrix[row][col];
}
mRateMatrix[row][row] = -sum;
}
}
```