LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

ZEMU POLINOMA PAKĀPJU BŪLA FUNKCIJU VAICĀJUMU SAREŽĢĪTĪBA

BAKALAURA DARBS

Autors: **Igors Stepanovs**
Stud. apl. is06030
Darba vadītājs: profesors Dr.habil.mat. Rūsiņš Mārtiņš Freivalds

RĪGA 2010

ANOTĀCIJA

Šajā darbā tiek pētīti vaicājošie algoritmi. Saņemot Būla funkciju ar nezināmām argumentu vērtībām, vaicājošais algoritms katrā darbības solī drīkst uzzināt par jebkuru vienu no šīm vērtībām. Nepieciešams noteikt šīs funkcijas rezultātu, izmantojot mazāko iespējamo soļu skaitu. Šim mērķim nepieciešamo soļu skaitu sauksim par algoritma sarežģītību un darbā tam pievērsīsim īpašu uzmanību. Algoritmu sarežģītības novērtējumam apskatīti arī funkcijas jutīgums un polinoma simetrizācijas metode. Tiek pētītas Būla funkcijas, kurām kvantu vaicājošā algoritma sarežģītība ir mazāka par determinētā vaicājošā algoritma sarežģītību. Darba rezultātā piedāvātas vairākas idejas šāda veida funkciju iegūšanai. Tai skaitā parādīts, kā funkciju atrašanu noreducēt līdz konstantās maksas maksimālās plūsmas meklēšanai grafā.

Atslēgvārdi: vaicājošais algoritms, kvantu mehānika, Būla funkcija, polinoma simetrizācijas metode, funkcijas jutīgums, plūsmas grafs.

ABSTRACT

This work provides a research on query algorithms. Given a Boolean function with unknown argument values the query algorithm can acquire any single one of values per execution step. The goal is to determine the result of the function using the least possible amount of execution steps. We define algorithm complexity as the amount of required execution steps and proceed to examine this concept thoughtfully. In order to approximate the complexity we provide an insight into function sensitivity and polynom symmetrization method. We research functions for which the quantum query algorithm requires smaller amount of questions then deterministic query algorithm. As a result of this work several approaches are offered for search of such functions, including the way to convert this task into solving constant cost maximum flow in a network.

Keywords: query algorithm, quantum computing, Boolean function, polynom symmetrization method, function sensitivity, flow network.

SATURS

APZĪMĒJUMU SARAKSTS

Dom(f) – funkcijas f definīcijas apgabals.

Ran(f) – funkcijas f vērtību apgabals.

Visur definēta funkcija – jebkura funkcija f, kurai izpildās īpašība $\forall x \in Dom(f)\ \exists y \in Ran(f) \bullet f(x) = y$.

Kortežs – sakārtota viena veida elementu virkne.

Monoms – vairāku mainīgo reizinājums.

Disjunktīvā normālforma – disjunkcija, kura sastāv no dažādu mainīgo vai to negāciju konjunkcijām (elementārajām konstrukcijām).

Grafs – pāris $G = (V, E)$, kas sastāv no virsotņu kopas V un šķautņu kopas E. Katra no šķautnēm ir kopas V divu elementu liela apakškopa un apzīmē to, ka šīs virsotnes savā starpā ir savienotas.

Orientēts grafs – pāris $G = (V, E)$, kas sastāv no virsotņu kopas V un šķautņu kopas E. Katra no šķautnēm ir kopas V divu elementu sakārtots kortežs un apzīmē to, ka no pirmās korteža virsotnes iet vienvirziena ceļš uz otro.

Svērts grafs – grafs, kurā katrai no šķautnēm papildus ir noteikts skaitlisks raksturlielums, piemēram – cena, garums vai caurlaidība.

Rijīgs algoritms – algoritmu sauksim par rijīgu (*greedy*), ja katrā solī tiek izdarīta šajā laika momentā lokāli optimāla izvēle.

IEVADS

Algoritmu sarežģītība ir teorētisko datorzinātņu nodaļa, kas pēta dažādu skaitļošanas uzdevumu atrisināšanai nepieciešamo resursu patēriņu. Vairums algoritmu darbojas ar mainīgā garuma ievaddatiem, tāpēc visbiežāk šo algoritmu sarežģītību var izteikt kā uzdevuma risinājuma patērētā laika vai atmiņas atkarību no ievaddatu apjoma. Tomēr ne visiem algoritmiem ir iespējams sniegt šādu novērtējumu. Piemēram, eksistē skaitļošanas uzdevumi, kuri tiek klasificēti kā „NP-pilni" – tiem nav zināma risinājuma, kurš uzdevuma atbildi mācētu atrast polinomiālā laikā.

Lai prastu atrisināt un analizēt lielāku uzdevumu klāstu, bieži vien ir noderīgi izpētīt vieglāku uzdevumu risinājumus. Šajā darbā apskatīsim sekojošu uzdevumu: dota Būla funkcija $f:\{0,1\}^n \to \{0,1\}$, kuras argumentu vērtības sākumā nav zināmas. Katrā algoritma izpildes solī mēs drīkstam uzzināt viena funkcijas argumenta vērtību. Ir nepieciešams noskaidrot funkcijas rezultātu, uzdodot pēc iespējas mazāku jautājumu skaitu.

Šī uzdevuma iespējamo risinājumu analīzei izmantosim determinēto un kvantu vaicājošos algoritmus. Darba mērķis ir atrast un izpētīt tādas Būla funkcijas, kurām kvantu vaicājošā algoritma sarežģītība ir labāka par determinētā vaicājošā algoritma sarežģītību. Savukārt par vaicājošā algoritma sarežģītību sauksim mazāko iespējamo soļu skaitu funkcijas vērtības noteikšanai sliktākajā gadījumā, tas ir – ja uz mūsu jautājumiem speciāli dos tādas atbildes, kas novilcinās brīdi, kad mēs mācēsim viennozīmīgi noteikt funkcijas rezultātu.

Darba pirmajā nodaļā pievērsīsim uzmanību Būla funkcijām un iemācīsimies tās pārveidot par tām atbilstošiem polinomiem. Apskatīsim arī tādus jēdzienus kā funkcijas jutīgums un polinomu simetrizācijas metode.

Otrajā nodaļā iepazīsimies ar vaicājošiem algoritmiem, klasificējot tos divās grupās – determinēti vaicājošie algoritmi un kvantu vaicājošie algoritmi. Sniegsim priekšstatu par kvantu mehāniku un analizēsim pastāvošās sakarības starp determinēto vaicājošo algoritmu sarežģītību, kvantu vaicājošo algoritmu sarežģītību un funkcijas jutīgumu.

Trešajā nodaļā apskatīsim jau zināmas funkcijas, kuras atbilst mūs interesējošu funkciju aprakstam, bet ceturtajā daļā piedāvāsim vairākus paņēmienus šāda veida sarežģītāku funkciju meklēšanai.

Darba gaitā iegūtos rezultātus apkoposim ceturtajā nodaļā. Savukārt tai sekos secinājumu nodaļa, kurā novērtēsim darbā paveikto un mēģināsim izvirzīt nākotnes pētījumu mērķus.

1. BŪLA FUNKCIJAS UN POLINOMI

1.1. Būla funkcijas jēdziens

Par Būla funkciju sauc visur definētu n argumentu funkciju $f:\{0,1\}^n \to \{0,1\}$, kur n ir vesels nenegatīvs skaitlis. Būla funkciju var aprakstīt ar vērtību tabulas palīdzību vai Būla loģikas formulas veidā, kas sastāv no loģiskiem mainīgajiem un loģiskām operācijām.

Piemēram, apskatīsim Būla funkciju $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$. Šo formulu varam aprakstīt arī ar vērtību tabulas palīdzību (skat. 1.1.1. tab.), kur katram funkcijas vērtību kortežam ir dota funkcijas rezultāta vērtība.

1.1.1. tabula

Būla funkcijas $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$ vērtību tabula

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Pastāv $a = 2^n$ dažādi veidi izvēlēties funkcijas argumentu kortežu no n vērtībām $\{0,1\}$, savukārt katram argumentu kortežam ir $b = 2$ iespējas izvēlēties tam attiecīgo funkcijas vērtību – 0 vai 1. Tātad katram n eksistē tieši $b^a = 2^{2^n}$ dažādas n argumentu Būla funkcijas.

Turpmāk darbā Būla funkcijas $f(x_1, x_2, ..., x_n)$ argumentus $x_1, x_2, ..., x_n$ sauksim arī par funkcijas ievaddatu bitu virknes x bitiem ar kārtas numuriem $1, 2, ..., n$.

1.2. Fiktīvi mainīgie Būla funkcijās

Definīcija. Būla funkcijas $f(x_1, x_2, ..., x_n)$ mainīgais x_i ir būtisks, ja eksistē tādas argumentu $x_1, ..., x_{i-1}, x_{i+1}, ..., x_n$ vērtības, kurām $f(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) \neq$ $\neq f(x_1, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)$. Citādi mainīgo sauksim par fiktīvu.

Apskatīsim n argumentu Būla funkciju $f(x_1, x_2, ..., x_n) = \neg x_1 \wedge (x_2 \vee \neg x_2) \wedge$ $\wedge (x_3 \vee \neg x_3) \wedge ... \wedge (x_n \vee x_n)$. Acīmredzams, ka $x_2, x_3, ..., x_n$ ir fiktīvi mainīgie, jo to vērtības nekādā veidā neietekmē funkcijas rezultātu. Tāpēc šo funkciju var saīsināt līdz pierakstam $f(x_1, x_2, ..., x_n) = \neg x_1$ vai pat $f(x_1) = \neg x_1$. Līdzīgi varam rīkoties ar jebkuru iepriekš zināmu funkciju – izmest no formulas visus fiktīvos mainīgos jeb izsvītrot šo mainīgo stabiņus Būla funkcijas vērtību tabulā.

Turpmāk darbā pieļausim, ka visas apskatītās Būla funkcijas satur tikai būtiskus mainīgos.

1.3. Funkcijas jutīgums

Dota bitu virkne $x \in \{0,1\}^n$ un kopa $B \subseteq Z_n$ (kopu B mēdz saukt arī par *bloku*). Ar x^B apzīmēsim bitu virkni, kuras bits ar kārtas numuru i ir vienāds ar $\overline{x_i}$, ja $i \in B$, citādi šis bits ir vienāds ar x_i. Savukārt ar x^i apzīmēsim bitu virkni x, kurai tikai i-tais ir pamainīts uz pretējo vērtību.

Definīcija. Attiecībā uz patvaļīgu Būla funkciju $f : \{0,1\}^n \to \{0,1\}$ un $x \in \{0,1\}^n$ saka, ka $B \subseteq Z_n$ ir bitu virknes x jutīguma bloks, ja $f(x^B) \neq f(x)$. Definēsim $bs_x(f)$ kā lielāko d, kuram eksistē d savstarpēji nešķeļošies bloki $B_1, ..., B_d \in Z_n$, visi no kuriem ir bitu virknes x jutīguma bloki [1].

Definīcija. Par funkcijas f bloka jutīgumu $bs(f)$ sauksim lielāko no visu funkcijas f ievaddatu virkņu x bloka jūtīgumiem: $bs(f) = \max_x bs_x(f)$ [1].

Definīcija. Definēsim funkcijas f jutīgumu $s(f)$ kā šīs funkcijas f bloka jutīgumu $bs(f)$, kuram papildus ir ierobežojums, ka visu izmantoto bloku izmēri ir 1, tas ir – $\forall i \bullet |B_i| = 1$ [1].

Piemērs. Būla funkcija f kā argumentu saņem $n = 4k^2$ garu bitu virkni, kura sastāv no $2k$ blokiem pa $2k$ mainīgajiem katrā. Bloks ar kārtas numuru i sastāv no mainīgo virknes $x_{1+(i-1)2k}, ..., x_{i2k}$. Funkcija f ir definēta tā, ka $f(x) = 1$ tad un tikai tad, ja eksistē vismaz viens bloks B_i, kurā diviem secīgiem mainīgajiem ir vērtība 1, bet visu pārējo $2k-2$ mainīgo vērtība ir 0. Apskatīsim funkcijas f jutīgumu $s(f)$ un bloka jūtīgumu $bs(f)$ [2]:

- ✓ funkcijas f jutīgums ir $s(f) = 2k$. Šis novērtējums ir iegūstams gadījumā, kad katrs no $2k$ blokiem apmierina funkcijas prasības – tas ir, katrā no blokiem ir

tieši divi blakus stāvoši mainīgie ar vērtību 1, bet visu pārējo mainīgo vērtība ir 0. Lai nomainītu funkcijas vērtību no 1 uz 0, katrā no $2k$ blokiem jebkura mainīgā vērtību ir jāpamaina uz pretējo;

- ✓ funkcijas f bloka jutīgums ir $bs(f) = 2k^2$. Šādu novērtējumu iegūstam, kad visi $4k^2$ mainīgie ir vienādi ar 0. Varam izvēlēties $2k^2$ blokus $B_i \in \{2i-1, 2i\}$ pie $i \in \{1, 2, ..., 2k^2\}$, kuri savā starpā nešķeļas un katrs no kuriem apraksta divus secīgus elementus. Piešķirot vieninieka bitus jebkuru divu elementu vērtībām ar šādiem kārtas numuriem, funkcijas vērtība mainās no 0 uz 1.

Teorēma 1. Jebkurai funkcijai $f : \{0,1\}^n \to \{0,1\}$ izpildās likumsakarība $s(f) \le bs(f) \le n$ [2].

Pierādījums. Tā kā funkcijas jutīguma bloku definīcijas ietvaros aprakstītās kopas B_i nedrīkst savstarpēji šķelties, tātad – katrs no n mainīgajiem ir sastopams ne vairāk kā vienā blokā, tad vienmēr izpildīsies īpašība $bs(f) \le n$. Savukārt no funkcijas jutīguma definīcijas acīmredzami seko īpašība $s(f) \le bs(f)$, jo $s(f)$ iegūstams ir kā viens no $bs(f)$ speciālgadījumiem.

Teorēma 2. Jebkurai funkcijai $f : \{0,1\}^n \to \{0,1\}$, ja $f(0) = 0$ un $f(x) = 1$ pie $|x| = 1$, tad izpildās īpašība $s(f) = n$ [2].

Pierādījums. Šo novērojumu iegūstam no funkcijas bloka jutīguma definīcijas gadījumā, kad $x = \{0\}^n$, $d = n$ un $\forall i \in \{1, 2, ..., d\} \bullet B_i = \{i\}$. Šīs ievaddatu bitu virknes bloka jutīgums pie norādītā bloku izkārtojuma ir vienāds ar n, un tas ir lielākais iespējamais bloka jutīgums.

1.4. Būla funkcijai atbilstošais vairāku mainīgo polinoms

Šīs nodaļas mērķis ir aprakstīt polinoma būvēšanas algoritmus priekš patvaļīgas Būla funkcijai, lai pie vienādiem ievaddatiem uzbūvētā polinoma un sākotnējās Būla funkcijas rezultāti būtu vienādi. Tātad iegūtais polinoms līdzīgi sākotnējai funkcijai izmantos mainīgos ar vērtībām no definīcijas apgabala $\{0,1\}$ un atgriezīs rezultātu no vērtību apgabala $\{0,1\}$. Toties atšķirībā no Būla funkcijas formulas, kas operē ar loģiskiem mainīgajiem un operācijām, polinoms strādās ar aritmētiskiem mainīgajiem un operācijām.

Sāksim ar funkcijas pakāpes un polinoma pakāpes definīcijām, tad apskatīsim divas dažādas metodes polinoma iegūšanai no dotās funkcijas.

1.4.1. Būla funkcijas un polinoma pakāpju jēdzieni

Definīcija. Monoma pakāpe ir šo monomu sastādošo reizinātāju pakāpju summa.

Piemērs. Monoma xy jeb x^1y^1 pakāpe ir $1+1=2$, bet monoma z^4tw^2 pakāpe ir $4+1+2=7$.

Definīcija. Polinoma pakāpe ir vienāda ar lielāko no monomu pakāpēm, no kuriem sastāv šis polinoms.

Piemērs. Polinoma $xy+z^3$ pakāpe ir 3, bet polinoma z^4tw^2+xy pakāpe ir 7.

Lemma 1. $p,q:R^n \to R$ ir vairāku mainīgo polinomi ar pakāpi, kas nepārsniedz d. Ja $p(x)=q(x)$ visiem $x \in \{0,1\}^n$ ar $|x| \le d$, tad $p=q$ [2].

Ne šīs lemmas seko, ka katrai funkcijai var atbilst tikai viens vairāku mainīgo polinoms.

Definīcija. Polinoms $p:R^n \to R$ reprezentē funkciju $f:\{0,1\}^n \to \{0,1\}$, ja $p(x)=f(x)$ priekš visām ievaddatu virknēm $x \in \{0,1\}^n$.

Definīcija. Par funkcijas f pakāpi $\deg(f)$ sauc pakāpi tam vairāku mainīgo polinomam, kurš reprezentē šo funkciju.

1.4.2. Polinoma iegūšana, izmantojot Būla funkcijas vērtību tabulu

Apskatīsim funkcijai atbilstošo vērtību tabulu. Vērtību tabula apraksta visus iespējamos n argumentu kortežus, un katram no tiem atbilst funkcijas rezultāts – 0 vai 1.

Veidojot polinomu no vērtību tabulas, apskatīsim katru no tabulas argumentu kortežiem:

- ✓ ja kortežam atbilstošais funkcijas rezultāts ir 1, tad no šī korteža uzbūvējam un pieskaitām polinomam tādu n mainīgo monomu, kas būs vienāds ar 1 tad un tikai tad, ja katra no monoma mainīgo vērtībām būs vienāda ar attiecīgo vērtību argumentu kortežā, – pretējā gadījumā šī monoma vērtībai jābūt vienādai ar 0;
- ✓ ja kortežam atbilstošais funkcijas rezultāts ir 0, tad polinomā to neiekļaujam.

Lai uzbūvētu monomu no argumentu korteža ar funkcijas rezultātu 1, rīkojamies sekojoši:

- ✓ monoma sākuma vērtība ir 1;
- ✓ katram korteža elementam $x_i = 0$ – monomam piereizinām izteiksmi $(1-x_i)$;
- ✓ katram korteža elementam $x_i = 1$ – monomam piereizinām mainīgo x_i.

Rezultātā iegūstam meklēto polinomu. Tā kā visi argumentu korteži ir dažādi, šajā polinomā nevar būt divu tādu monomu, kuru vērtības varētu vienlaicīgi būt vienādas ar 1.

Piemērs. 1.1. nodaļā minētās Būla funkcijas $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$ polinoms izskatās sekojoši:

$$p_f(x_1, x_2, x_3) = (1-x_1)\times(1-x_2)\times x_3 + (1-x_1)\times x_2 \times x_3 + x_1 \times x_2 \times (1-x_3) + x_1 \times x_2 \times x_3 =$$
$$= (x_3 - x_2x_3 - x_1x_3 + x_1x_2x_3) + (x_2x_3 - x_1x_2x_3) + (x_1x_2 - x_1x_2x_3) + x_1x_2x_3 =$$
$$= x_3 + x_1x_2 - x_1x_3.$$

1.4.3. Polinoma iegūšana, vadoties pēc tā vispārīgā veida formulas

Mēģināsim iepriekšējās nodaļas rezultātu iegūt ar citu polinoma būvēšanas paņēmienu. Izmantojot funkciju no 1.1. nodaļas, ieguvām 3 mainīgo polinomu ar pakāpi 2. Vispārīgā veidā n mainīgo polinoms ar pakāpi k var tikt izteikts veidā:

$$p_f(x_1, x_2, .., x_n) = c_{\varnothing} + c_{\{1\}}x_1 + c_{\{2\}}x_2 + ... + c_{\{n-1\}}x_{n-1} + c_{\{n\}}x_n + c_{\{1,2\}}x_1x_2 + c_{\{1,3\}}x_1x_3 + ... +$$
$$+ c_{\{n-1,n\}}x_{n-1}x_n + ... + c_{\{1,2,..,k\}}x_1x_2..x_k + c_{\{1,2,..,k+1\}}x_1x_2..x_{k+1} + ... + c_{\{n-k+1,n-k+2,..,n-1,n\}}x_{n-k+1}x_{n-k+2}..x_{n-1}x_n.$$

Lai atrastu šī polinoma koeficientus, secīgi apskatīsim funkcijas argumentu kortežus vieninieka bitu skaita pieaugšanas secībā:

- ✓ no sākuma n vērtību kortežu, kas sastāv tikai no nullēm;
- ✓ tad visus n vērtību kortežus, kuri satur tieši vienu vērtību 1;
- ✓ tad visus n vērtību kortežus, kuri satur tieši divas vērtības 1, utt.

Atradīsim visus polinoma koeficientus pēc indukcijas:

- ✓ *bāze*: ja n nuļļu kortežam atbilst funkcijas vērtība 0, tad $c_{\varnothing} = 0$, citādāk $c_{\varnothing} = 1$;
- ✓ *pāreja*: apskatām n vērtību kortežu ar precīzi i vieniniekiem; meklējot polinoma koeficientu i-tās pakāpes monomam, mēs jau zināsim visus polinoma koeficientus priekš monomiem ar pakāpi mazāku par i; mēs vienmēr zinām arī pašam kortežam atbilstošo funkcijas vērtību; visi monomi ar pakāpi lielāku par i būs acīmredzami vienādi ar nulli, kā arī tikai viens monoms ar pakāpi i nelīdzināsies nullei – tādējādi viennozīmīgi varam uzzināt šim i vieninieku kortežam atbilstošo polinoma koeficientu.

Piemērs. Apskatīsim šī paņēmiena pielietojumu 1.1. nodaļā minētai Būla funkcijai $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$:

- ✓ $f(0, 0, 0) = 0$, tāpēc $c_{\varnothing} = 0$;
- ✓ $f(1, 0, 0) = 0$ un $p_f(1, 0, 0) = c_{\varnothing} + c_{\{1\}}x_1 = 0 + c_{\{1\}} = c_{\{1\}}$, un $f(1, 0, 0) = p_f(1, 0, 0)$, tāpēc $c_{\{1\}} = 0$;
- ✓ $f(0, 1, 0) = 0$ un $p_f(0, 1, 0) = c_{\varnothing} + c_{\{2\}}x_2 = c_{\{2\}}$, un $f(0, 1, 0) = p_f(0, 1, 0)$, tāpēc $c_{\{2\}} = 0$;

- ✓ $f(0,0,1)=1$ un $p_f(0,0,1)=c_{\varnothing}+c_{\{3\}}x_3=c_{\{3\}}$ un $f(0,0,1)=p_f(0,0,1)$, tāpēc $c_{\{3\}}=1$;
- ✓ $f(1,1,0)=1$ un $p_f(1,1,0)=c_{\varnothing}+c_{\{1\}}x_1+c_{\{2\}}x_2+c_{\{3\}}x_3+c_{\{1,2\}}x_1x_2=$ $=1\times x_3+c_{\{1,2\}}x_1x_2=c_{\{1,2\}}$, un $f(1,1,0)=p_f(1,1,0)$, tāpēc $c_{\{1,2\}}=1$;
- ✓ $f(1,0,1)=0$ un $p(1,0,1)=c_{\varnothing}+c_{\{1\}}x_1+c_{\{2\}}x_2+c_{\{3\}}x_3+c_{\{1,2\}}x_1x_2=$ $=1\times x_3+c_{\{1,3\}}x_1x_3=1+c_{\{1,3\}}$, un $f(1,0,1)=p_f(1,0,1)$, tāpēc $c_{\{1,3\}}=-1$;
- ✓ $f(0,1,1)=1$ un $p(0,1,1)=c_{\varnothing}+c_{\{1\}}x_1+c_{\{2\}}x_2+c_{\{3\}}x_3+c_{\{2,3\}}x_2x_3=$ $=1\times x_3+c_{\{2,3\}}x_2x_3=1+c_{\{2,3\}}$ un $f(0,1,1)=p_f(0,1,1)$, tāpēc $c_{\{2,3\}}=0$.

Ieguvām atbildi
$p_f(x_1,x_2,x_3)=0+0x_1+0x_2+1x_3+1x_1x_2-1x_1x_3+0x_2x_3=x_3+x_1x_2-x_1x_3$. Tātad ar abām apskatītām polinoma iegūšanas metodēm tika uzbūvēts viens un tas pats polinoms, kas atbilst sākotnējai funkcijai.

1.5. Simetriskas Būla funkcijas un polinomu simetrizācijas metode

Definīcija. Simetriska Būla funkcija ir Būla funkcija, kuras vērtība ir atkarīga tikai no vienīnieka bitu skaita tās ievaddatu bitu virknē.

Turpmāk vienīnieka bitu skaitu virknē x apzīmēsim ar $|x|$.

Piemērs. Dažas no simetriskām funkcijām [2]:

- ✓ $OR_n(x)=1$ tad un tikai tad, ja $|x|\geq 1$;
- ✓ $AND_n(x)=1$ tad un tikai tad, ja $|x|=n$;
- ✓ $PARITY_n(x)=1$ tad un tikai tad, ja $|x|$ ir nepāra skaitlis.

Par simetrisku polinomu sauksim polinomu, kura ievaddatu bitu virknei piemīt simetriskām Būla funkcijām analoģiskā īpašība. Tālāk aprakstīsim, kā jebkuras Būla funkcijas polinomu (iespējams, ar informācijas zudumu) pārveidot par simetrisku polinomu. Iemācoties būvēt simetrisku polinomu, pārveidosim to par viena mainīgā polinomu, kurš sākotnējā polinoma sniegto informāciju aprakstīs mums ērtākā veidā.

1.5.1. Simetrizācijas polinomi

Definīcija. Par polinoma $p : R^n \to R$ simetrizācijas polinomu p^{sym} sauc $p^{sym}(x) = \frac{\sum_{\pi \in S_n} p(\pi(x))}{n!}$, kur $\pi(x)$ ir kāda permutācija $\pi(x) = (x_{\pi(1)}, ..., x_{\pi(n)})$ no tās ievaddatiem $x = x_1 ... x_n$, bet ar S_n tiek apzīmēta kopa, kas sastāv no visām $n!$ permutācijām garumā n [2].

1.5.1.1. tabula

Sākotnējie polinomi p un tiem atbilstošie simetrizācijas polinomi p^{sym}

Sākotnējais polinoms p	Simetrizācijas polinoms p^{sym}
$p(x) = x_1 + x_2$	$p^{sym}(x) = x_1 + x_2$
$p(x) = x_1 - x_2 - x_1 x_2$	$p^{sym}(x) = -2x_1 x_2$
$p(x) = x_1 - x_2 + x_3 + x_1 x_2 x_3$	$p^{sym}(x) = \frac{2(x_1 + x_2 + x_3) + 6x_1 x_2 x_3}{6}$

Tātad simetrizācijas polinoms (skat. 1.5.1.1. tab.) tiek definēts kā sākotnējā polinoma vidējā vērtība, ievaddatu bitu virknes veidā padodot visas iespējamās permutācijas no x. Simetrizācijas polinoma pakāpe $\deg(p^{sym})$ nepārsniedz sākotnējā polinoma pakāpi $\deg(p)$ un var būt pat mazāka par to. Piemēram, polinoma $p = x_1 - x_2$ simetrizācijas polinoms ir $p^{sym} = 0$.

1.5.2. Simetrizācijas polinoma pārveidošana par viena mainīgā polinomu

Definīcija. Ja $p : R^n \to R$ ir vairāku mainīgo polinoms, tad eksistē viena mainīgā polinoms $q : R^n \to R$, ka $\deg(q) \le \deg(p)$ un $p^{sym}(x) = q(|x|)$ priekš visiem $x \in \{0, 1\}^n$ [2].

Pierādījums. Ar d apzīmēsim simetrizācijas polinoma p^{sym} pakāpi, kura nepārsniedz polinoma p pakāpi. Ar V_j apzīmēsim summu no j dažādu mainīgo $\binom{n}{j}$ reizinājumiem, tātad $V_1 = x_1 + ... + x_n$, $V_2 = x_1 x_2 + x_1 x_3 + ... + x_{n-1} x_n$. Tā kā p^{sym} ir simetrisks polinoms, pēc indukcijas viegli pierādīt, ka šo polinomu var uzrakstīt $p^{sym}(x) = c_0 + c_1 V_1 + c_2 V_2 + ... + c_d V_d$ veidā, kur $c_i \in R$. Tā kā visas x_i vērtības ir vienādas ar 0 vai 1, summa V_j vienmēr pieņem

vērtību $\binom{|x|}{j} = \frac{|x|(|x|-1)(|x|-2)...(|x|-j+1)}{j!}$. Tāpēc viena mainīgā polinomu q (skat. 1.5.2.1. tab.) var definēt kā $q(|x|) = c_0 + c_1\binom{|x|}{1} + c_2\binom{|x|}{2} + ... + c_d\binom{|x|}{d}$ [2].

1.5.2.1. tabula

Simetrizācijas polinomi p^{sym} un tiem atbilstošie viena mainīgā polinomi $q(|x|)$

Simetrizācijas polinoms p^{sym}	Viena mainīgā polinoms $q(\|x\|)$
$p^{sym}(x) = x_1 + x_2 = 0 + 1 \times V_1$	$q(\|x\|) = 0 + 1 \times \binom{\|x\|}{1} = \|x\|$
$p^{sym}(x) = -2x_1x_2 = 0 + 0 \times V_1 - 2V_2$	$q(\|x\|) = 0 + 0 \times \binom{\|x\|}{1} - 2 \times \binom{\|x\|}{2} = -\|x-1\| \times \|x\|$
$p^{sym}(x) = \frac{2(x_1 + x_2 + x_3) + 6x_1x_2x_3}{6} =$ $= 0 + \frac{1}{3} \times V_1 + 0 \times V_2 + 1 \times V_3$	$q(\|x\|) = 0 + \frac{1}{3} \times \binom{\|x\|}{1} + 0 \times \binom{\|x\|}{2} + 1 \times \binom{\|x\|}{3} =$ $= \frac{1}{3} \times \|x\| + \frac{\|x-2\| \times \|x-1\| \times \|x\|}{6}$

Turpmāk šajā darbā, pieminot viena mainīgā polinomu, vienmēr uzskatīsim, ka tas tika iegūts no tam attiecīgā simetrizācijas polinoma.

1.5.3. No simetrizācijas polinoma iegūstamā informācija par sākotnējo Būla funkciju

Pēc viena mainīgā polinoma definīcijas – visiem $x \in \{0,1\}^n$ izpildās $p^{sym}(x) = q(|x|)$. Tātad visām ievaddatu bitu virknēm garumā $|x|$ ir viena un tā pati simetrizācijas polinoma p^{sym} vērtība, kas ir vienāda ar $q(|x|)$. Šī vērtība tika iegūta, pārveidojot funkcijas polinomu $p(x)$ par simetrizācijas polinomu $p^{sym}(x) = \frac{\sum_{x \in S_n} p(\pi(x))}{n!}$.

Ja $|x| = t$, tad $q(t) = q(|x|) = p^{sym}(x)$ ir vienāds ar n mainīgo sakārtojumu skaitu no precīzi t vieniniekiem, kuriem atbilstošais funkcijas f rezultāts ir 1. Turklāt šis sakārtojumu skaits simetrizācijas polinoma vērtībā ietilpst $t!(n-t)!$ reizes (tik veidos var savā starpā pārkārtot t vieniniekus un $n-t$ nulles, nemainot pašu pozīciju) un $p^{sym}(x)$ formulā ir dalīts ar $n!$.

Uzskatāmības pēc viena mainīgā polinoma $q(|x| = t)$ vērtību vietā turpmāk centīsimies uzrādīt vērtības $q(t) \times \left(\frac{n}{t} \right)$ – cik kopā ir tādu funkcijas ievaddatu virkņu x, kas satur tieši t vieniniekus un kuriem izpildās $f(x) = 1$.

Ievērosim, ka $q(|x|)$ ir definēts visām ievaddatu vērtībām no kopas $\{0, 1, ..., n\}$, jo šie ir visi iespējamie vieninieka bitu skaiti virknē x. Spriežot pēc augstāk minētiem novērojumiem par $q(t)$ vērtības kombinatorisko interpretāciju, jebkuram $t \in \{0, 1, ..., n\}$ jāizpildās likumsakarībai $q(t) \times \left(\frac{n}{t} \right) \in \left\{ 0, ..., \left(\frac{n}{t} \right) \right\}$.

2. VAICĀJOŠIE ALGORITMI

2.1. Vaicājošā algoritma jēdziens

Vaicājošie algoritmi ir specifisks automātu paveids. Šajā darbā tos apskatīsim tikai patvaļīgas Būla funkcijas vērtības aprēķināšanas kontekstā.

Pieņemsim, ka Būla funkcijas $f(x_1, x_2, ..., x_n)$ disjunktīvā normālforma ir zināma. Nepieciešams iegūt funkcijas rezultātu, tomēr sākumā neviena no argumentu x_i vērtībām nav zināma. Katrā solī drīkst izvēlēties jebkuru kārtas numuru i un uzdot jautājumu „Kāda ir argumenta x_i vērtība?". Tā kā mēs darbojamies ar Būla funkciju, tad ir tikai divas iespējamās atbildes: „$x_i = 0$" vai „$x_i = 1$". Uzdodot katru nākamo jautājumu, mēs ņemsim vērā visas iepriekš saņemtās atbildes, lai kopējo jautājumu skaits būtu pēc iespējams mazāks. Tikko esam pārliecināti, ka zinām funkcijas vērtību neatkarīgi no pāri palikušo nezināmo argumentu vērtībām – algoritma izpilde apstājas.

Piemērs. Ir jānosaka funkcijas $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$ (skat. 1.1. nod.) vērtība. Vaicājošā algoritma norise varētu būt sekojoša:

- ✓ Jautājums: „Kāda ir argumenta x_1 vērtība?"
- ✓ Atbilde: „$x_1 = 0$"
- ✓ Jautājums: „Kāda ir argumenta x_3 vērtība?"
- ✓ Atbilde: „$x_3 = 0$"

Šajā brīdī algoritms apstājas, jo, iegūtās argumentu vērtības ieliekot funkcijas formulā, viennozīmīgi iegūstam tās rezultātu: $(0 \wedge x_2) \vee (1 \wedge 0) = 0 \vee 0 = 0$. Apskatītai Būla funkcijai šis ir viens no diviem optimālā vaicājošā algoritma darbības iznākumiem. Vaicājošais algoritms rīkojas sekojoši:

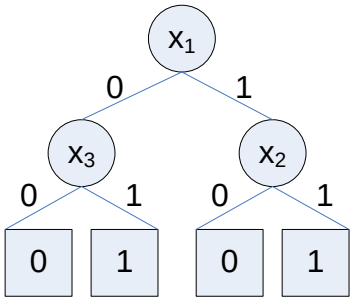
- ✓ pirmo jautājumu vienmēr prasa par argumentu x_1;
- ✓ ja $x_1 = 0$, uzdod jautājumu par argumenta x_3 vērtību;
- ✓ ja $x_1 = 1$, uzdod jautājumu par argumenta x_2 vērtību.

Neatkarīgi no uz uzstādītiem jautājumiem saņemtajām atbildēm, šīs 3 argumentu funkcijas rezultāta atrašanai būs vienmēr nepieciešami tikai 2 jautājumi.

Definīcija. Vaicājošam algoritmam sliktākajā gadījumā nepieciešamo jautājumu skaitu konkrētās funkcijas rezultāta noteikšanai sauc par vaicājošā algoritma sarežģītību.

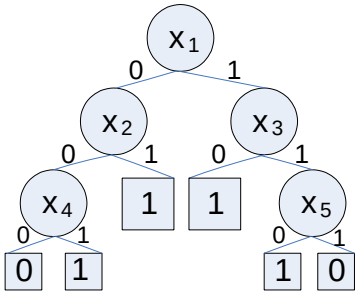
2.2. Vaicājošā algoritma lēmumu koks

Katrā solī vaicājošā algoritma darbība ir atkarīga no atbildēm uz iepriekš uzdotiem jautājumiem, tāpēc šo algoritmu var attēlot arī grafiski – koka veidā. Katrā koka virsotnē vaicājošais algoritms prasīs šai virsotnei atbilstošā funkcijas argumenta x_i vērtību. Atkarībā no iegūtās atbildes nolaidīsimies uz nākamo virsotni pa vienu no pašreizējās virsotnes zariem, katram no kuriem atbilst noteikta uz mūsu jautājumu saņemta atbilde. Citādākas ir koka lapas – tās satur iespējamos Būla funkcijas rezultātus, ar kuriem šī funkcija ir vienāda, algoritmam izpildes gaitā nonākot šajās lapas no koka saknes. Šādu koku sauc par lēmumu koku.



2.2.1. att. **Būla funkcijas** $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$ **lēmumu koks**

Lēmumu koka augstums reprezentē attiecīgā vaicājošā algoritma sarežģītību. Šajā darbā par koka augstumu sauksim garāko ceļu no koka saknes līdz jebkurai no lapām, kur ceļa garuma kritērijs ir uzdoto jautājumu skaits, lai noietu pa šo ceļu. Piemēram, uzzīmējot nodaļā 2.1. aprakstītā funkcijas $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$ vaicājošā algoritma lēmumu koku, var pārliecināties, ka šī algoritma sarežģītība ir 2 jautājumi (skat. 2.2.1. att.).



2.2.2. att. **Lēmumu koks, kurš apraksta optimālo vaicājošo algoritmu Būla funkcijai** $f(x_1, x_2, x_3, x_4, x_5) = (\neg x_1 \wedge \neg x_2 \wedge x_4) \vee (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_3) \vee (x_1 \wedge x_3 \wedge \neg x_5)$

Acīmredzams, ka katram lēmumu kokam atbilst vismaz viena Būla funkcija, kurai šis koks apraksta optimālu vaicājošo algoritmu. Lai no patvaļīga koka iegūtu šādu funkciju (skat. 2.2.2. att.), katrai no koka lapām, kurās funkcijas vērtība ir 1, vairāku mainīgo (un to

negāciju) konjunkcijas veidā aprakstām ceļu no koka saknes līdz konkrētai lapai. Meklētā Būla funkcija ir disjunkcija no visām koka vienīnieka vērtību lapām atbilstošām formulām.

2.3. Vaicājošā algoritma sarežģītības novērtējums

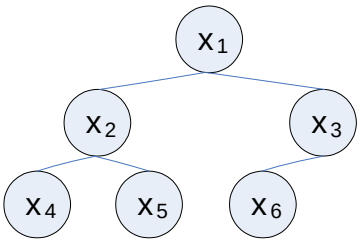
Šajā nodaļā iegūsim sarežģītības novērtējumus Būla funkcijām: kāds ir lielākais un mazākais nepieciešamo jautājumu skaits, lai atrastu patvaļīgas n argumentu Būla funkcijas vērtību sliktākajā gadījumā.

Lielākais nepieciešamais jautājumu skaits priekš n argumentu Būla funkcijas ir vienāds ar n. Vispārīgā veidā šādas funkcijas var nodefinēt kā $f(x_1, x_2, ..., x_n) = x_1 \vee x_2 \vee ... \vee x_n$. Sliktākajā gadījumā mēs paprasīsim $n-1$ mainīgo vērtības un tās visas būs vienādas ar 0. Tad, lai atrastu funkcijas vērtību, nāksies prasīt arī pēdējā nezināmā argumenta vērtību.

Novērtēsim, kāds ir mazākais nepieciešamo jautājumu skaits. Izmantosim n mainīgos, lai uzbūvētu lēmumu koku ar pēc iespējas mazāku sarežģītību. Nodaļā 2.2. ievērojām, ka katram lēmumu kokam atbilst vismaz viena Būla funkcija, kurai šis koks apraksta optimālo vaicājošo algoritmu. Tāpēc pietiek atrast, kāds ir mazākais lēmumu koka augstums, ja tas sastāv no n virsotnēm.

Definīcija. Perfekts binārs koks – binārs koks, kuram visas lapas atrodas vienādā attālumā no šī koka saknes.

Definīcija. Pilnīgs binārs koks – binārs koks, ko var iegūt, perfektam bināram kokam no labās puses pēc kārtas atmetot kādas lapas.



2.3.1. att. **Pilnīgs binārs koks no 6 pieprasījuma virsotnēm ar augstumu 3** $(\lfloor \log_2 6 \rfloor + 1)$ **(koka lapas ar funkcijas rezultātiem nav norādītas)**

No pieprasījumu virsotnēm vēlamies uzbūvēt pilnīgu bināru koku. Savukārt pilnīga bināra koka augstums pie n virsotnēm ir vienāds ar $\lfloor \log_2 n \rfloor + 1$ (skat. 2.3.1. att.). Šī izteiksme ir mūsu meklētais apakšējais novērtējums vaicājošā algoritma sarežģītībai.

2.4. Determinēts lēmumu koks

Definīcija. Par determinētu lēmumu koku sauc lēmumu koku, kura darbība jebkurā brīdī ir pilnīgi viennozīmīga: saņemot atbildi uz kārtējo jautājumu, no pašreizējās virsotnes vienmēr iziet tieši viena šķautne, kura atbilst saņemtai atbildei; prasot un saņemot tos pašus jautājumus un atbildes, konkrēts lēmumu koks vienmēr savu darbību pabeigs vienā un tajā pašā koka lapā.

Lai nodrošinātu determinētā lēmuma koka viennozīmību mūsu uzdevuma ietvaros, tajā no katras iekšējās virsotnes jāiziet tieši diviem zariem – kreisais zars atbilst atbildei „$x_i = 0$" un labais zars atbilst atbildei „$x_i = 1$".

Definīcija. Determinēts vaicājošais algoritms aprēķina n argumentu Būla funkciju f tad un tikai tad, ja šī algoritma rezultāts ir vienāds ar $f(x)$ visiem ievaddatiem $x \in \{0,1\}^n$.

Definīcija. Ar $D(f)$ apzīmēsim funkcijas f optimālā determinētā vaicājošā algoritma sarežģītību. $D(f)$ ir vienāds ar tādu funkcijas f lēmuma koka augstumu, kura augstums ir mazākais iespējamais starp visiem dotās funkcijas lēmuma kokiem.

Teorēma 3. $bs(f) \le D(f)$ [2].

Pierādījums. Priekš ievaddatiem ar savstarpēji nešķeļošiem jūtīguma blokiem $B_1, ..., B_{bs(f)}$ determinētam vaicājošam algoritmam katrā no šiem blokiem ir jāpaprasa vismaz par vienu mainīgo. Ja par kādu bloku nekas netiks jautāts, mēs varētu nomainīt vērtības šajā blokā tā, lai funkcijas vērtība arī mainās uz pretējo, bet vaicājošais algoritms to nepamanītu. Tātad vaicājošam algoritmam par ievaddatu bitu virkni jāuzdod vismaz $bs(f)$ jautājumi.

2.5. Kvantu lēmumu koks

Iepriekš aprakstījām determinētu lēmumu koku, bet eksistē arī nedeterminētā lēmumu koka jēdziens, kā arī varbūtiskā lēmumu koka jēdziens [2]. Atšķirībā no šiem lēmumu kokiem, kvantu lēmuma koka jēdziens neatbilst kvantu vaicājošā algoritma darbībai. Tomēr šo jēdzienu mēs turpināsim izmantot, jo šajā darbā tas tiek lietots pretstatā citiem lēmumu kokiem, kā arī tāpēc, ka ar kvantu vaicājošā algoritma palīdzību ir iespējams nosimulēt jebkuru no iepriekš pieminētiem lēmumu kokiem.

Šo nodaļu iesāksim ar kvantu mehānikas aksiomu izklāstu, tad sekos kvantu vaicājošā algoritma apraksts, kā arī tā izmantošanas piemērs determinētā lēmumu koka simulācijai. Nodaļas nobeigumā sniegsim kvantu vaicājošā algoritma sarežģītības novērtējumus.

2.5.1. Kvantu mehānika

Klasiskā informācijas daudzuma mērvienība ir bits, un tā vērtība var būt 0 vai 1. Savukārt kvantu mehānikā analoģisko mērvienību sauc par kvantu bitu jeb kubitu, kas ir divu klasisko stāvokļu kombinācija jeb superpozīcija $\alpha_0|0\rangle + \alpha_1|1\rangle$.

Vispārinot, m kvantu bitu stāvoklis $|\phi\rangle$ ir visu klasisko m-bitu virkņu superpozīcija $|\phi\rangle = \sum_{i\in\{0,1\}^m} \alpha_i|i\rangle$. Šeit α_i ir komplekss skaitlis, ko sauc par bāzes stāvokļa $|i\rangle$ amplitūdu. Tiek prasīts, lai vienmēr izpildītos īpašība $\sum_i |\alpha_i|^2 = 1$.

Attiecībā uz m kvantu bitu stāvokli spējam veikt divu veidu operācijas:

- ✓ mērījums – mērot m kvantu bitu stāvokli $|\phi\rangle$, iegūst bāzes stāvokli $|i\rangle$ ar varbūtību $|\alpha_i|^2$.Pēc mērījuma stāvoklis $|\phi\rangle$ kļūst vienāds ar mērījumā iegūto stāvokli $|i\rangle$, tāpēc visa informācija par citiem $|\phi\rangle$ bāzes stāvokļiem tiek zaudēta;
- ✓ unitāra transformācija – lineāras transformācijas pielietojums, kas pārveido stāvokli $|\phi\rangle$ par citu stāvokli $|\psi\rangle = U(|\phi\rangle)$, bet nemaina bāzes amplitūdu kvadrātu summas vērtību ($\sum_i |\alpha_i|^2 = 1$).

Definīcija. Transformācija ir unitāra tad un tikai tad, ja, pareizinot unitārās transformācijas matricu $U = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ ar tai kompleksi saistīto matricu $U^t = \begin{pmatrix} a^* & b^* \\ c^* & d^* \end{pmatrix}$, iegūst vienības matricu $I = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.

2.5.2. Kvantu vaicājošais algoritms [2]

Šajā nodaļā turpināsim strādāt ar n argumentu Būla funkciju, tātad – aizvien darbojamies ar ievaddatiem $x \in \{0,1\}^n$.

Definēsim m kvantu bitu bāzes stāvokļa pierakstu kā $|i, b, z\rangle$, kur:

- ✓ i – tā elementa indekss, par kuru vaicājošais algoritms uzdod kārtējo jautājumu; bāzes stāvokļa pierakstā šim lielumam tiek izmantoti $\lceil \log n \rceil$ biti;
- ✓ b – elementa x_i vērtība, kuru vaicājošais algoritms saņem kā atbildi uz kārtējo jautājumu; bāzes stāvokļa pierakstā aizņem 1 bitu;

- ✓ z – kvantu datora algoritma izpildei nepieciešamā iekšējā atmiņa, kas sīkāk netiek aprakstīta; bāzes stāvokļa pierakstā aizņem $m - \lceil \log n \rceil - 1$ bitus.

Definēsim pieprasījumu O kā unitāru transformāciju, kas pārveido $|i, b, z\rangle$ par $|i, b \otimes x_i, z\rangle$. Kvantu vaicājošā algoritma izpilde sākas ar m kvantu bitu stāvokli $|\vec{0}\rangle$, kur katrs no bitiem ir vienāds ar 0. Šim stāvoklim secīgi pielietojam unitāru transformāciju U_0, pieprasījumu O, unitāru transformāciju U_1, utt. Tādējādi T pieprasījumu vaicājošais algoritms atbilst garajai unitārai transformācijai $A = U_T O U_{T-1} ... O U_1 O U_0$. Šeit U_i ir unitāra transformācija, kas nav atkarīga no ievaddatiem x. Tas nozīmē, ka gala stāvoklis $A|\vec{0}\rangle$ ir atkarīgs tikai no T reižu transformācijas O pielietošanas. Vaicājošā algoritma atbilde tiek iegūta, izmērot gala stāvokļa pēdējo bitu.

Definīcija. Kvantu vaicājošai algoritms precīzi aprēķina f, ja algoritma rezultāts ar varbūtību 1 ir vienāds ar funkcijas $f(x)$ vērtību.

Definīcija. Ar $Q_E(f)$ apzīmē pieprasījumu skaitu jeb sarežģītību optimālam kvantu vaicājošam algoritmam, kas precīzi aprēķina f.

2.5.3. Determinētā vaicājošā algoritma simulācija ar kvantu vaicājošo algoritmu [2]

Determinētā vaicājošā algoritma simulācijas izklāstam nedaudz detalizēsim m kvantu bitu bāzes stāvokļa pierakstu $|i, b, z\rangle$. Sadalīsim elementu z divās daļās, tādējādi turpmāk strādājot ar stāvokļiem formāta $|i, b, h, a\rangle$, kur:

- ✓ h – satur informāciju par visiem iepriekšējiem vaicājošā algoritma pieprasījumiem un to rezultātiem; bāzes stāvokļa pierakstā aizņem $m - \lceil \log n \rceil - 2$ bitus;
- ✓ a – kvantu stāvokļa pēdējais bits, kur pēc algoritma izpildes pabeigšanas ir jāglabājas iegūtajam rezultātam.

Unitārā transformācija U_0 noteiks mainīgo i, par kuru tiks veikts sākotnējais pieprasījums – tā pārveidos sākuma stāvokli $|\vec{0}, 0, \vec{0}, 0\rangle$ par $|i, 0, \vec{0}, 0\rangle$. Tā kā x_i ir Būla funkcijas argumenta vērtība, par kuru tiek veikts sākotnējais vaicājums, tad nākamajā solī pieprasījums O pārveidos stāvokli $|i, 0, \vec{0}, 0\rangle$ par $|i, x_i, \vec{0}, 0\rangle$.

Nākamā unitārā transformācija U_1 pārveidos $\left|i, x_i, \vec{0}, 0\right\rangle$ par $\left|j, 0, h, 0\right\rangle$, kur h ir algoritma atjaunotā darbības vēsture (kas sevī iekļauj informāciju par vērtībām i un x_i). Savukārt j ir Būla funkcijas argumenta numurs, par kuru determinēts vaicājošais algoritms būtu prasījis nākamajā solī. Pēc U_1 izpildes var atkal pielietot O, no $\left|j, 0, h, 0\right\rangle$ iegūstot $\left|j, x_j, h, 0\right\rangle$.

Secīgi pielietojot divu veidu aprakstītās transformācijas T reizes, paliek tikai veikt transformāciju U_T, kas uzstādīs atbildes bitu atbilstoši pēc iepriekšējās kvantu bitu operācijas iegūtajam iekšējam stāvoklim h.

Nodemonstrējot veidu, kā ar kvantu vaicājošā algoritma palīdzību nosimulēt jebkura determinētā vaicājošā algoritma darbību, mēs ar to pašu esam parādījuši, ka kvantu vaicājošā algoritma sarežģītība nav lielāka par determinētā vaicājošā algoritma sarežģītību jeb $Q_E(f) \le D(f)$.

2.5.4. Kvantu vaicājošā algoritma sarežģītības novērtējums

Lemma 2. Ar A apzīmēsim kvantu lēmumu koku, kas izdara T pieprasījumus. Tad kompleksos skaitļos eksistē tādi n mainīgo polinomi α_i, kuru pakāpe ir ne lielāka kā T un ar kuru palīdzību priekš visiem $x \in \{0,1\}^n$ lēmumu koka A gala stāvokli var aprakstīt kā $\sum_{i \in \{0,1\}^m} \alpha_i(x)\left|i\right\rangle$ [2].

Teorēma 4. $\deg(f) \le 2Q_E(f)$ [2].

Pieradījums. Apskatīsim funkcijas f kvantu vaicājošo algoritmu ar $Q_E(f)$ pieprasījumiem. Ar S apzīmējam to bāzes stāvokļu kopu, kuriem atbilst algoritma izvads ar vērtību 1. Tad atbildes 1 varbūtība ir $P(x) = \sum_{k \in S} \left|\alpha_k(x)\right|^2$. Pēc iepriekšējās lemmas, α_i ir polinomi ar pakāpi $\le 2Q_E(f)$. Tā kā P apraksta funkciju f, tad tā pakāpe ir $\deg(f)$. Attiecīgi izpildās $\deg(f) \le 2Q_E(f)$.

3. ZINĀMĀS FUNKCIJAS

Apskatīsim paritātes Būla funkciju $PARITY_n(x_1, x_2, ..., x_n) = x_1 \otimes x_2 \otimes ... \otimes x_n$. Tā ir vienāda ar 1 tad un tikai tad, kad starp funkcijas n argumentiem ir nepāra skaits vieninieka bitu. Citādi šīs funkcijas vērtība ir vienāda ar 0.

Priekš $f = PARITY_n$ izpildās īpašības $\deg(f) = n$ un $Q_E(f) = \lceil n/2 \rceil$ [2]. Acīmredzami patiess ir arī apgalvojums $D(f) = n$, jo ikviens nepārbaudīts funkcijas ievaddatu bits potenciāli var izmainīt funkcijas rezultātu uz pretējo. Tādējādi šai funkcijai izpildās $Q_E(f) = D(f)/2$.

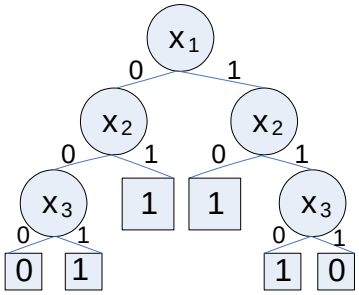
Šī darba mērķis ir atrast tādas funkcijas, kurām kvantu vaicājošais algoritms strādā pēc iespējas labāk par determinēto vaicājošo algoritmu. Tāpēc turpmāk ir jāmēģina uzlabot funkcijas $PARITY_n$ rezultāts, atrodot citas funkcijas, kurām $Q_E(f) < D(f)/2$.

Atcerēsimies nodaļā 2.5.4. pierādīto likumsakarību $\frac{\deg(f)}{2} \le Q_E(f)$ – tā kopā ar tikko aprakstītās īpašības $Q_E(f) < D(f)/2$ nepieciešamību ļauj izvirzīt jaunu kritēriju: $\frac{\deg(f)}{2} \le Q_E(f) < \frac{D(f)}{2}$ jeb $\deg(f) < D(f)$. Šis kritērijs ir labāks, jo abus lielumus $\deg(f)$ un $D(f)$ mēs mākam atrast vieglāk par kvantu vaicājošā algoritma sarežģītību $Q_E(f)$. Tātad turpmāk mūs interesēs tieši tās funkcijas, kurām polinoma (jeb pašas funkcijas) pakāpe ir mazāka par to determinētā vaicājošā algoritma sarežģītību.

Katrai no turpmāk apskatītām jau zināmām funkcijām ir raksturīga lielākā iespējamā determinētā vaicājošā algoritma sarežģītība, salīdzinot ar citām attiecīgās pakāpes funkcijām.

3.1. Nisan un Szegedy funkcija: deg(f) = 2, D(f) = 3

Apskatīsim funkciju $f = E_3(x_1, x_2, x_3) = \neg(x_1 \wedge x_2 \wedge x_3) \wedge \neg(\neg x_1 \wedge \neg x_2 \wedge \neg x_3)$ (skat. 3.1.1. tab.), kas ir vienāda ar 1 tad un tikai tad, ja viens vai divi no trīs ievaddatu bitiem ir vienādi ar 1 [3].



3.1.1. att. **Būla funkcijas** $E_3(x_1, x_2, x_3) = \neg(x_1 \wedge x_2 \wedge x_3) \wedge \neg(\neg x_1 \wedge \neg x_2 \wedge \neg x_3)$ **lēmumu koks**

Uzbūvējot funkcijai $f = E_3(x_1, x_2, x_3)$ atbilstošo polinomu, iegūstam $p_f(x_1, x_2, x_3) = x_1 + x_2 + x_3 - x_1x_2 - x_1x_3 - x_2x_3$. Tātad funkcijas pakāpe ir $\deg(f) = 2$. Savukārt $D(f) = 3$ (skat. 3.1.1. att.), jo, lai kādā secībā mēs bijām uzdevuši pirmos divus jautājumus, ja abas atbildes bija vienādas, trešā jautājuma iznākums ietekmē funkcijas rezultātu. Šī situācija radīsies, piemēram, ja visi trīs funkcijas argumenti ir vienādi.

3.1.1. tabula

Būla funkcijas $E_3(x_1, x_2, x_3) = \neg(x_1 \wedge x_2 \wedge x_3) \wedge \neg(\neg x_1 \wedge \neg x_2 \wedge \neg x_3)$ vērtību tabula

x_1	x_2	x_3	$E_3(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

3.2. Kushilevitz funkcija: deg(f) = 3, D(f) = 6

Funkciju $f = KUSHILEVITZ_6(x)$, kura pieņem 6 ievaddatu bitus, aprakstīsim ar polinomu $P_f = \sum_i x_i - \sum_{ij} x_i x_j + x_1x_3x_4 + x_1x_2x_5 + x_1x_4x_5 + x_2x_3x_4 + x_2x_3x_5 + x_1x_2x_6 + x_1x_3x_6 + x_2x_4x_6 + x_3x_5x_6 + x_4x_5x_6$ [3].

Funkcijas rezultāts ir 0, ja $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \in \{0, 4, 5\}$.

Funkcijas rezultāts ir 1, ja $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \in \{1, 2, 6\}$.

Pie $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 3$ funkcijas rezultāts ir 1 tad un tikai tad, ja jebkurš no šiem monomiem ir vienāds ar 1: $x_1x_3x_4$, $x_1x_2x_5$, $x_1x_4x_5$, $x_2x_3x_4$, $x_2x_3x_5$, $x_1x_2x_6$, $x_1x_3x_6$, $x_2x_4x_6$, $x_3x_5x_6$, $x_4x_5x_6$.

Viegli pierādīt, ka šai funkcijai $D(f) = 6$. Ja uz pirmiem pieciem jautājumiem tika saņemtas tikai atbildes 1, tad, lai kādā kārtībā mēs šos jautājumus bijām prasījuši, jebkādā gadījumā nāksies prasīt arī par pēdējo nezināmo bitu, jo tas izšķirs funkcijas gala vērtību. Tātad funkcijas pakāpe ir $\deg(f) = 3$, bet funkcijas determinētā vaicājošā algoritma sarežģītība ir $D(f) = 6$.

3.3. Nisan un Szegedy funkcijas vispārinājums: deg(f) = 4, D(f) = 9

Izmantojot nodaļā 3.2. apskatīto funkciju $E_3(x_1, x_2, x_3)$, rekursīvi definēsim $E_3^0(z) = z$ un $E_3^k(\cdot) = E_3(E_3^{k-1}(\cdot), E_3^{k-1}(\cdot), E_3^{k-1}(\cdot))$, kur katrs no E_3^{k-1} izmanto dažādus 3^{k-1} ievaddatu bitus [3]. Acīmredzams, ka $\deg(E_3^k) = 2^k$. Savukārt $D(E_3^k) = 3^k$, kas ir pierādāms pēc analoģijas ar nodaļas 3.1. pamatojumu par $D(E_3^1) = 3^1$.

Izmantojot $k = 2$, iegūstam funkciju ar pakāpi $\deg(E_3^2) = 2^2 = 4$ un determinētā vaicājošā algoritma sarežģītību $D(E_3^2) = 3^2 = 9$.

Šīs funkcijas polinomu no augstāk sniegtās definīcijas būvēsim sekojoši:

$$E_3^2 = E_3(E_3^1, E_3^1, E_3^1) = E_3(E_3(E_3^0, E_3^0, E_3^0), E_3(E_3^0, E_3^0, E_3^0), E_3(E_3^0, E_3^0, E_3^0)) =$$
$$= E_3(E_3(x_1, x_2, x_3), E_3(x_4, x_5, x_6), E_3(x_7, x_8, x_9)) =$$
$$= E_3(x_1 + x_2 + x_3 - x_1x_2 - x_1x_3 - x_2x_3, x_4 + x_5 + x_6 - x_4x_5 - x_4x_6 - x_5x_6,$$
$$x_7 + x_8 + x_9 - x_7x_8 - x_7x_9 - x_8x_9) =$$
$$= (x_1 + x_2 + x_3 - x_1x_2 - x_1x_3 - x_2x_3) + (x_4 + x_5 + x_6 - x_4x_5 - x_4x_6 - x_5x_6) + \quad .$$
$$+ (x_7 + x_8 + x_9 - x_7x_8 - x_7x_9 - x_8x_9) - (x_1 + x_2 + x_3 - x_1x_2 - x_1x_3 - x_2x_3) \times$$
$$\times (x_4 + x_5 + x_6 - x_4x_5 - x_4x_6 - x_5x_6) - (x_1 + x_2 + x_3 - x_1x_2 - x_1x_3 - x_2x_3) \times$$
$$\times (x_7 + x_8 + x_9 - x_7x_8 - x_7x_9 - x_8x_9) - (x_4 + x_5 + x_6 - x_4x_5 - x_4x_6 - x_5x_6) \times$$
$$\times (x_7 + x_8 + x_9 - x_7x_8 - x_7x_9 - x_8x_9) = \ldots$$

4. PIEKTĀS PAKĀPES POLINOMU MEKLĒŠANA

Iepriekšējā nodaļā aplūkojām tādas otrās, trešās un ceturtās pakāpes funkcijas, kuru determinētā sarežģītība ir lielākā iespējamā starp funkcijām ar tādu pašu pakāpi. Tikmēr autoram nav zināmu 5. pakāpes funkciju piemēru, kurām determinētā sarežģītība ir labāka par funkcijas pakāpi.

A.Lučko savā darbā atrada astoņus viena mainīgā polinomus, kas atbilst 5. pakāpes 15 mainīgo Būla funkcijām ar determinētā vaicājošā algoritma sarežģītību $D(f)=15$ [4]. Šajā nodaļā mēģināsim uzbūvēt vienu no šīm Būla funkcijām, kā arī piedāvāsim dažādas detalizācijas vispārīgas idejas Būla funkciju meklēšanai no viena mainīgā polinomiem.

4.1. Piektās pakāpes Būla funkciju viena mainīgā polinomu ģenerēšana [4]

Ir nepieciešams atrast Būla funkciju f ar pakāpi $\deg(f)=5$ un pēc iespējas lielāku determinēto sarežģītību $D(f)$. Pašlaik vienīgais gadījums, kad pietiekami droši un ātri mākam noteikt funkcijas f raksturlielumu $D(f)$, ir tikai tad, ja tas ir vienāds ar funkcijas mainīgo skaitu n.

No teorēmām 1 ($s(f)\le bs(f)\le n$) un 3 ($bs(f)\le D(f)$) iegūstam, ka pie $s(f)=n$ viennozīmīgi izpildās arī $D(f)=n$. Savukārt teorēma 2 ($f(0)=0\wedge(|x|=1\to f(x)=1)\to s(f)=n$) parāda, ka $s(f)=n$ ir spēkā visām Būla funkcijām, kuru viena mainīgā polinomiem $q(|x|)$ izpildās $q(0)=0$ un $q(1)=1$.

Tātad, ja meklētajai Būla funkcijai atbilst polinoms $p(x)$, apskatām šī polinoma simetrizācijas polinomu $p^{sym}(x)$ un tam attiecīgo viena argumenta polinomu $q(|x|)$. Lai izpildītos $D(f)=n$, mūs interesē tikai tādi viena argumenta polinomi, kuriem $q(0)=0$ un $q(1)=1$.

Nodaļā 1.5.3. izsecinājām, ka viena mainīgā polinoma definīcijas apgabals ir $\{0,1,...,n\}$ un tā vērtību apgabals ir raksturojams ar izteiksmi $q(t)\times\left(\frac{n}{t}\right)\in\left\{0,...,\left(\frac{n}{t}\right)\right\}$. Lai iegūtu visus iespējamos 5. pakāpes viena mainīgā polinomus, pielietosim Lagranža interpolācijas algoritmu. Zinot viena mainīgā polinoma rezultātus sešos ($\deg(q)+1$) no piecpadsmit (n) ievaddatu gadījumiem, šis algoritms māk izrēķināt polinoma rezultātu jebkurai citai ievaddatu vērtībai.

Divos punktos viena mainīgā polinoma rezultāts jau ir fiksēts ($q(0) = 0$ un $q(1) = 1$). Tāpēc jāizvēlas vēl četras ievaddatu parametra vērtības k no kopas $\{2, 3, ..., 15\}$, kurām ir pēc iespējas mazāk dažādu funkcijas rezultāta vērtību $\left\{0, ..., \left(\frac{n}{k}\right)\right\}$. Visizdevīgāk šim nolūkam ir ņemt $q(2)$, $q(13)$, $q(14)$ un $q(15)$. Tad varēsim pārlasīt visas kombinācijas no pieļaujamiem funkcijas rezultātiem šajos punktos un pārbaudīt, vai, interpolējot pārējos viena mainīgā polinoma definīcijas apgabalam atbilstošos rezultātus, visi no tiem atbilst polinoma vērtību apgabalam.

4.2. Lagranža interpolācijas algoritms [5]

Doti tādi $k+1$ punkti $(x_0, y_0), (x_1, y_1), ..., (x_k, y_k)$, ka $(x_i = x_j) \Rightarrow (i = j)$ un $\forall i \bullet p(x_i) = y_i$. Šie $k+1$ vērtību pāri viennozīmīgi apraksta polinomu ar pakāpi k. Lagranža algoritms, zinot šos $k+1$ vērtību pārus, ļauj patvaļīgam polinoma argumentam x_p atrast attiecīgo polinoma vērtību y_p:

$$y_p = L(x_p) = \sum_{j=0}^{k} y_j l_j(x_p), \text{ kur}$$

$$l_j(x) = \prod_{f=0, f\neq j}^{k} \frac{x - x_f}{x_j - x_f} = \frac{(x - x_0)}{(x_j - x_0)} ... \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} ... \frac{(x - x_k)}{(x_j - x_k)}$$

4.3. Atrastie 5. pakāpes viena mainīgā polinomi

Pārlasot visus iespējamos viena mainīgā polinomus, kuriem kvantu vaicājošais algoritms iespējami darbojas labāk par determinēto vaicājošo algoritmu, ieguvām rezultātus katram sākotnējās funkcijas mainīgo skaitam no 5 līdz 15, ieskaitot (skat. 4.3.1. tab.).

Jāpiebilst, ka daļai no šādā veidā atrastajiem viena mainīgā polinomiem atbilstošo 5. pakāpes Būla funkciju var arī neeksistēt. Viena mainīgā polinoma definīcijā (skat. 1.5.2. nod.) minēts, ka $\deg(q) \leq \deg(p)$. Tātad daži no atrastajiem polinomiem atbilst Būla funkcijām ar pakāpi, kas ir lielāka par piekto.

4.3.1. tabula

Atrasto 5. pakāpes viena mainīgā polinomu skaits atkarībā no Būla funkcijas mainīgo skaita

5	6	7	8	9	10	11	12	13	14	15	16
1452	2934	3994	4605	4791	4413	3475	2319	1041	247	8	0

Kā jau iepriekš pamanījām, viena mainīgā polinomus $q(t)$ visērtāk ir apskatīt veidā $q(t)\times\left(\frac{n}{t}\right)$, kas sniedz informāciju par sākotnējās Būla funkcijas kombinatorām īpašībām. Tāpēc turpmāk visus atrastos viena mainīgā polinomus minēsim tieši šādā pierakstā.

Tabulā 4.3.2. parādīti 8 viena mainīgā polinomi, kas varētu atbilst 5. pakāpes 15 mainīgo funkcijām. Šīs tabulas i-tās rindiņas j-tajā kolonnā (kolonnas numurējam no nulles) esošā vērtība c nozīmē, ka i-tajam polinomam eksistē tieši c tādas ievaddatu bitu virknes, kuras sastāv no j vieniniekiem un $15-j$ nullēm un kurām meklētās Būla funkcijas rezultāts ir 1.

4.3.2. tabula

15 mainīgo 5. pakāpes viena mainīgā polinomi pierakstā $q(t)\times\left(\frac{n}{t}\right)$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1.	0	15	104	270	270	6	280	1953	4020	4395	2832	1040	174	0	0	1
2.	**0**	**15**	**104**	**271**	**280**	**51**	**400**	**2163**	**4272**	**4605**	**2952**	**1085**	**184**	**1**	**0**	**1**
3.	**0**	**15**	**104**	**280**	**315**	**126**	**490**	**2205**	**4230**	**4515**	**2877**	**1050**	**175**	**0**	**0**	**1**
4.	0	15	104	281	325	171	610	2415	4482	4725	2997	1095	185	1	0	1
5.	0	15	104	277	286	1	175	1695	3684	4137	2727	1035	190	7	1	1
6.	0	15	104	278	296	46	295	1905	3936	4347	2847	1080	200	8	1	1
7.	0	15	104	279	306	91	415	2115	4188	4557	2967	1125	210	9	1	1
8.	0	15	104	277	287	11	220	1815	3894	4389	2937	1155	235	17	2	1

Tabulā 4.3.2. iegūtās vērtības atbilst tām, kuras savā darbā aprakstīja A.Lučko. Savukārt no četrām A.Lučko darbā minētajām tabulas 4.3.1. vērtībām (priekš 10, 14, 15 un 16 mainīgo skaita) radusies nesakritība pie 14 mainīgo Būla funkcijām atbilstošo viena mainīgo polinomu skaita, kur par spīti A.Lučko atrastajiem 252 polinomiem šī darba autors ieguva tikai 247 polinomus.

Īpaši ievērojami ir 2. un 3. polinoms, jo tā vērtībām kolonnās j un $15-j$ (nosauksim šīs vērtības par v_j un v_{15-j}) izpildās vienādība $v_j + v_{15-j} = C_{15}^{j}$, kas norāda uz zināmu simetriskumu un var palīdzēt šo polinoma atrašanai. Katram mainīgo skaitam no 5 līdz 15 visus šāda veida simetriskos viena mainīgā polinomus var apskatīt 1. pielikumā.

Turpmāk mēģināsim uzbūvēt abus no šiem polinomiem. Trešo polinomu izvēlēsimies detalizētai kombinatoriskai analīzei nākamajās nodaļās, bet otro polinomu pētīsim tikai ar vēlāk aprakstītu datorizētu skaitļošanas algoritmu palīdzību.

4.4. Polinoma reprezentācijas veidi

Vispārīgā gadījumā 5. pakāpes 15 mainīgo polinoms izskatās šādi:

$$p(x_1, x_2, .., x_n) = c_{\emptyset} + c_{\{1\}}x_1 + c_{\{2\}}x_2 + ... + c_{\{n-1\}}x_{n-1} + c_{\{n\}}x_n + c_{\{1,2\}}x_1x_2 + c_{\{1,3\}}x_1x_3 + ... +$$
$$+ c_{\{n-1,n\}}x_{n-1}x_n + ... + c_{\{1,2,..,k\}}x_1x_2..x_k + c_{\{1,2,..,k+1\}}x_1x_2..x_{k+1} + ... + c_{\{n-k+1,n-k+2,..,n-1,n\}}x_{n-k+1}x_{n-k+2}..x_{n-1}x_n.$$

Ir nepieciešams, lai vismaz viens no 5. pakāpes monomu koeficientiem nebūtu vienāds ar 0. Šādu reprezentāciju, kas polinoma aprakstam izmanto tā monomu koeficientus, turpmāk sauksim par polinoma koeficientu reprezentāciju (jeb pozīciju).

4.4.1. tabula

15 mainīgo 5. pakāpes trešais viena mainīgā polinoms pierakstā $q(t) \times \left(\frac{n}{t}\right)$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	104	280	315	126	490	2205	4230	4515	2877	1050	175	0	0	1

Lai gan polinoma koeficientu reprezentācija ir intuitīvi saprotama, tomēr nodefinēsim un biežāk lietosim citu polinoma reprezentācijas veidu – sauksim to par polinoma kombinatorisko reprezentāciju. Trešais viena mainīgā polinoms (skat. 4.4.1. tab.) katram no vieninieka bitu skaitiem Būla funkcijas ievaddatu bitu virknē apraksta, cik gadījumos pie šādiem ievaddatiem Būla funkcijas rezultāts ir vienāds ar 1. Piemēram, ievaddatu virknē esot precīzi 3 vieninieka bitiem, 280 no iespējamiem $455\,(C_{15}^3)$ gadījumiem funkcijas rezultātam ir jābūt vienādam ar 1. No 455 iespējamām ievaddatu bitu virknēm uzrādot 280, kurām šīs funkcijas rezultātam ir jābūt vienādam ar 1, mēs iegūstam meklētās funkcijas jeb polinoma kombinatorisko reprezentāciju. Protams, lai iegūtu precīzu reprezentāciju, šīs izvēles jānorāda visām ievaddatu bitu virknēm, kuras satur precīzi 0, 1, 2, 4 un 5 vieninieka bitus. Nodaļā 4.5. parādīsim, ka no šīs reprezentācijas var iegūt arī koeficientu reprezentāciju.

Acīmredzami, ikviena polinoma kombinatoriskā reprezentācija pēc definīcijas apmierina viena mainīgā polinoma tabulas nosacījumus priekš precīzi 0, 1, 2, 3, 4 un 5 vieninieku skaita ievaddatu virknē.

Turpmāk polinoma kombinatorisko reprezentāciju bieži vien pierakstīsim kā monomu kopu, katrs no kuriem atbilst vieninieka bitu kārtas numuru kopai polinoma ievaddatu bitu virknē. Piemēram, ja $p(001000000100100) = 1$, tad teiksim, ka monoms $x_3x_{10}x_{13}$ atbilst polinoma vieninieka vērtībai.

Sniegsim dažus novērtējumus, kas ļauj izvēlēties, kādu no divām reprezentācijām ir izdevīgāk izmantot atkarībā no darbināmā algoritma prasībām pēc dažāda veida resursiem.

Apskatot 3. pakāpes monomu izvēli priekš kombinatoriskās reprezentācijas, ir jāizvēlas 280 monomus, kuriem attiecīgais funkcijas rezultāts ir 1, un 175 monomus, kuriem funkcijas rezultāts ir 0. Zinot visus funkcijas argumentu kortežus priekš vieninieka rezultātiem vai kortežu priekš nuļļu rezultātiem, mēs zinām, ka visi pāri palikušie korteži atbilst pretējai funkcijas vērtībai. Tātad gadījumā ar monoma pakāpi 3 mums ir izdevīgāk izvēlēties nevis 280 monomus priekš funkcijas vieninieka rezultāta, bet gan 175 kortežus priekš funkcijas rezultāta 0. Tātad priekš 3. pakāpes monomiem mēs izvēlamies $\min(280,455-280)=175$ monomus.

Polinoma koeficientu reprezentācijas pierakstam nepieciešams norādīt $C_{15}^0+C_{15}^1+C_{15}^2+C_{15}^3+C_{15}^4+C_{15}^5=4944$ koeficientus, savukārt polinoma kombinatoriskās reprezentācijas pierakstam nepieciešams minēt tikai $\min(0,1)+\min(15,0)+\min(104,0)+$ $+\min(280,175)+\min(315,1050)+\min(126,2877)=736$ monomus, kuru attiecīgiem funkcijas rezultātiem jābūt vienādiem ar 1. Tātad kombinatorisko reprezentāciju izdevīgāk izmantot tad, ja nepieciešams patērēt mazāk atmiņas pozīcijas glabāšanai vai apstrādei.

Nomainot vienu monomu pret citu tādas pašas pakāpes monomu polinoma kombinatoriskā pierakstā, jauniegūtais polinoms saglabās sākotnējā polinoma iespējamo atbilstību viena mainīgā polinoma tabulas pirmām 6 vērtībām. Savukārt polinoma koeficientu reprezentācijā viena koeficienta nomaiņa par citu vērtību visdrīzāk sabojās sākotnējās reprezentācijas atbilstību katrai no viena mainīgā polinoma tabulas vērtībām. Šī īpašība jāpatur prātā, izvēloties reprezentācijas veidu polinoma meklēšanas algoritmu programmēšanas realizācijām.

4.5. Meklētā 5. pakāpes 15 mainīgo polinoma koeficienti

Šajā nodaļā, izmantojot polinoma kombinatorisko reprezentāciju, iegūsim vairāku veidu ierobežojumus attiecībā uz konkrēto polinoma koeficientu vērtībām vai visu polinoma koeficientu dažādi veidotu apakškopu summām.

4.5.1. Aprēķināmie koeficienti pie zemu pakāpju monomiem

Tabulas 4.4.1. vērtības priekš precīzi 0, 1 un 2 vieninieka bitiem meklētā polinoma ievaddatu virknē ir attiecīgi vienādas ar 0, 15 un 104. Secīgi apskatīsim, par ko liecina katra no šīm vērtībām:

- ✓ gadījumā, kad neviens meklētā polinoma arguments nav vienāds ar 1, šī polinoma rezultāts ir vienāds ar 0. Tāpēc meklētā polinoma koeficients $c_{\varnothing}=0$;

- ✓ ja tieši vienam no patvaļīga 15 mainīgo polinoma argumentiem ir jābūt vienādam ar 1, pastāv 15 (C_{15}^{1}) dažādi veidi izveidot atbilstošu argumentu kortežu. Arī tabulā norādīts, ka meklētajam polinomam eksistē 15 dažādi mainīgo korteži, kuriem šī polinoma rezultāts ir vienāds ar 1. Tātad visi meklētā polinoma koeficienti pie pirmās pakāpes monomiem arī ir vienādi ar 1 jeb $\forall i \in \{1, 2, ..., 15\} \bullet c_{\{i\}} = 1$;
- ✓ ja precīzi diviem bitiem no polinoma ievaddatu bitu virknes piemīt vērtība 1 – tabulā minētais skaitlis 105 ir vienāds ar C_{15}^{2} – visu iespējamo kombināciju skaitu, kā no 15 mainīgajiem izvēlēties divus. Tātad visu divu vieninieka bitu ievaddatu virkņu gadījumos – polinoma rezultātam jābūt vienādam ar 1. Tā kā katram no diviem vieninieka bitiem meklētajā polinomā eksistē attiecīgs pirmās pakāpes monoms, kas palielina rezultāta vērtību par 1 (summā – par 2), tad priekš gala rezultāta 1 sasniegšanas šī polinoma visiem otrās pakāpes monomu koeficientiem jābūt vienādiem ar -1 jeb $\forall i, j \in \{1, 2, ..., 15\} \wedge i \neq j \bullet c_{\{i,j\}} = -1$.

No 455 (C_{15}^{3}) trešās pakāpes monomiem ir nepieciešams izvēlēties 280 tādus, kuru funkcijas rezultātus pielīdzināsim vieniniekam. Pārējiem 175 trešās pakāpes monomiem funkcijas rezultāts būs 0. Kopumā ir $C_{455}^{280} \approx 1{,}12 \times 10^{318}$ veidi, kā no 455 monomiem izvēlēties nepieciešamos 280 monomus. Līdzīga situācija izveidojas ar 4. pakāpes monomiem, kur no 1365 iespējamiem ir jāizvēlas tādi 315 monomi, kuru funkcijas rezultāti būs vienādi ar 1. Priekš 5. pakāpes ir jāizvēlas 126 monomi no 3003 iespējamiem. Tātad pavisam ir iespējami $C_{455}^{280} \times C_{1365}^{315} \times C_{3003}^{126} \approx 3{,}14 \times 10^{674}$ veidi, kā uzbūvēt tabulas 4.4.1. pirmām sešām norādītām vērtībām atbilstošu polinomu. Vēlāk katram no šiem polinomiem ir jāveic pārbaude, vai tie atbilst pārējiem 10 tabulā norādītajiem kritērijiem par dažāda veida ievaddatiem atbilstošiem rezultātiem. Acīmredzami, ar pilno pārlasi to izdarīt nesanāks, tāpēc ir jāatrod papildus kritēriji, kuri varētu stipri palīdzēt samazināt polinoma meklēšanas darbietilpību.

4.5.2. Neviennozīmīgo polinoma koeficientu pieļaujamās vērtības

Iepriekšējā nodaļā jau noskaidrojām nulltās, pirmās un otrās pakāpes monomu koeficientus, kas meklētajam polinomam būs vienādi ar 0, 1 un -1 attiecīgi. Pamanījām arī to, ka nav viennozīmīga veida sadalīt trešās, ceturtās un piektās pakāpes monomus divās grupās pēc attiecīgās funkcijas rezultāta: 0 vai 1.

Mēģināsim noteikt, kādas vērtības var pieņemt meklētā polinoma 3., 4. un 5. pakāpes monomu koeficienti atkarībā no konkrētā monoma piederības vienai no divām izdalītām funkcijas vērtību grupām.

Ja apskatām 15 bitu funkcijas ievaddatu virkni, no kuriem precīzi $p > 2$ biti ir vienādi ar 1, tad:

- ✓ neatkarīgi no p vērtības polinoma vērtībai tiek pieskaitīts $c_{\varnothing} = 0$ jeb nekas;
- ✓ katrs no atsevišķiem p vieninieka bitiem palielina polinoma vērtību par 1, jo visu pirmās pakāpes monomu koeficienti ir 1;
- ✓ katrs no visiem C_p^2 divu dažādu vieninieka bitu pāriem samazina funkcijas vērtību par 1, jo visu otrās pakāpes monomu koeficienti ir -1.

Tātad atkarībā no p jeb vieninieku bita skaita ievaddatu virknē, polinoma rezultātā ir iekļauts saskaitāmais $0 + p - C_p^2$.

Jebkurai $p = 3$ vieninieka bitu virknei (apzīmēsim šos trīs bitus ar x_a, x_b, x_c) polinoma sākotnējais rezultāts ir vienāds ar $0 + p - C_p^2 = p - \frac{(p-1)*p}{2} = 3 - 3 = 0$. Tā kā ir nepieciešams, lai 280 gadījumos polinoma rezultāts būtu vienāds ar 1, tad no 455 monomiem ir jāizvēlas tos 280, kuriem šī īpašība izpildīsies. Šo monomu koeficienti būs vienādi ar 1, bet pārējo 175 monomu koeficienti vienādi ar 0:

- ✓ $f(x_a x_b x_c) = 0 \Leftrightarrow c_{\{x_a, x_b, x_c\}} = 0$ (175 gadījumos);
- ✓ $f(x_a x_b x_c) = 1 \Leftrightarrow c_{\{x_a, x_b, x_c\}} = 1$ (280 gadījumos).

Gadījumā ar $p = 4$ vieninieka bitiem (apzīmēsim šos četrus bitus ar x_a, x_b, x_c, x_d), polinoma sākotnējais rezultāts ir vienāds ar $0 + p - C_p^2 = 0 + 4 - 6 = -2$. Lai iegūtu polinoma gala rezultātu, jāpieskaita 4 (C_p^3) dažādu trešās pakāpes monomu koeficientu vērtības $c_{\{x_b,x_c,x_d\}} + c_{\{x_a,x_c,x_d\}} + c_{\{x_a,x_b,x_d\}} + c_{\{x_a,x_b,x_c\}}$ un paša monoma $x_a x_b x_c x_d$ koeficientu. Secinām, ka:

- ✓ $f(x_a x_b x_c x_d) = 0 \Leftrightarrow 0 = -2 + 4 \times \{0,1\} + c_{\{x_a, x_b, x_c, x_d\}} \Leftrightarrow c_{\{x_a, x_b, x_c, x_d\}} \in \{-2,-1,0,1,2\}$ (1050 gadījumos);
- ✓ $f(x_a x_b x_c x_d) = 1 \Leftrightarrow 1 = -2 + 4 \times \{0,1\} + c_{\{x_a, x_b, x_c\ x_d\}} \Leftrightarrow c_{\{x_a, x_b, x_c\ x_d\}} \in \{-1,0,1,2,3\}$ (315 gadījumos).

Gadījumā ar $p = 5$ vieninieka bitiem no kopumā 15 ievaddatu bitiem (apzīmēsim šos piecus bitus ar x_a, x_b, x_c, x_d, x_e), polinoma sākuma rezultāts vienāds ar $0 + p - C_p^2 = 0 + 5 - 10 = -5$. Lai iegūtu polinoma gala vērtību, jāpieskaita pieciem vieninieka

bitiem atbilstošo 10 (C_p^3) trešās pakāpes monomu koeficientus un 5 (C_p^4) ceturtās pakāpes monomu koeficientus. Tātad:

- ✓ $f(x_a, x_b, x_c, x_d, x_e) = 0 \Leftrightarrow 0 = -5 + 10 \times \{0,1\} + 5 \times \{-2, ..., 3\} + c_{\{x_a, x_b, x_c, x_d, x_e\}} \Leftrightarrow$
  $\Leftrightarrow c_{\{x_a, x_b, x_c, x_d, x_e\}} \in \{-20, ..., 15\}$ (2877 gadījumos);
- ✓ $f(x_a, x_b, x_c, x_d, x_e) = 0 \Leftrightarrow 1 = -5 + 10 \times \{0,1\} + 5 \times \{-2, ..., 3\} + c_{\{x_a, x_b, x_c, x_d, x_e\}} \Leftrightarrow$
  $\Leftrightarrow c_{\{x_a, x_b, x_c, x_d, x_e\}} \in \{-19, ..., 16\}$ (126 gadījumos).

Līdzīgus ierobežojumus var izvest attiecībā uz jebkura mainīgo skaita 5. pakāpes polinomu.

4.5.3. Ierobežojumi uz visu monomu koeficientu apakškopu summām

Šajā nodaļā ar $F(K, s)$ apzīmēsim kopas K visas iespējamās apakškopas, kuru izmērs nepārsniedz skaitli s. Tātad $F(K, s) = \forall P \bullet P \in K \wedge |P| \leq s$. Savukārt ar N_m apzīmēsim kopu $\{1, ..., m\}$.

No viena mainīgā polinoma vērtībām pie dažāda vieninieku skaita ievaddatu virknē iegūsim vispārīgākus ierobežojumus priekš polinoma monomu koeficientu dažādi veidotām summām:

- ✓ Ja visi 15 ievaddatu virknes biti ir vienādi ar 1, tad meklētās funkcijas rezultāts arī vienāds ar 1.

Šī īpašība acīmredzami norāda uz to, ka visu polinoma koeficientu summa ir vienāda ar 1 jeb $\sum_{L \in F(N_{15}, 5)} c_L = 1.$

Tātad visu 3., 4. un 5. pakāpes monomu koeficientu summa ir

$$\sum_{L \in F(N_{15}, 5) - F(N_{15}, 2)} c_L = \sum_{L \in F(N_{15}, 5)} c_L - c_{\emptyset} - \sum_{i \in N_{15}} c_{\{i\}} - \sum_{i, j \in N_{15} \wedge i \neq j} c_{\{i, j\}} = 1 - 0 - C_{15}^1 \times 1 - C_{15}^2 \times (-1) = 91.$$

- ✓ Ja precīzi 14 no ievaddatu virknes bitiem ir vienādi ar 1, tad meklētās funkcijas rezultāts vienmēr vienāds ar 0.

Ja apzīmēsim ievaddatu vienīgā nulles bita kārtas numuru ar i, tad

$$\forall i \in N_{15} \bullet \sum_{L \in F(N_{15}, 5)} c_L - \sum_{L \in F(N_{15}, 5) \wedge i \in L} c_L = 0$$

$$\Rightarrow \forall i \in N_{15} \bullet \sum_{L \in F(N_{15}, 5)} c_L - \sum_{L \in F(N_{15} - \{i\}, 4)} c_{\{i\} \cup L} = 0$$

$$\Rightarrow \forall i \in N_{15} \bullet \sum_{L \in F(N_{15} - \{i\}, 4)} c_{\{i\} \cup L} = 1$$

Atņemsim visu tādu monomu koeficientus, kuru pakāpe ir mazāka par 3.

$$\forall i \in N_{15} \bullet \sum_{L \in F(N_{15}-\{i\},4)-F(N_{15}-\{i\},1)} c_{\{i\}\cup L} = \sum_{L \in F(N_{15}-\{i\},4)} c_{\{i\}\cup L} - \sum_{L \in F(N_{15}-\{i\},1)} c_{\{i\}\cup L} =$$

$$= \sum_{L \in F(N_{15}-\{i\},4)} c_{\{i\}\cup L} - \left(c_{\{i\}} + \sum_{L \in N_{15}-\{i\}} c_{\{i\}\cup L} \right) = 1-(1+14\times(-1)) = 14$$

- ✓ Ja precīzi 13 no ievaddatu bitiem ir vienādi ar 1, tad meklētās funkcijas rezultāts ir vienmēr vienāds ar 0.

Ja apzīmēsim ievaddatu nulles bitu kārtas numurus ar i un j $(i \neq j)$, tad

$$\forall i, j \in N_{15} \wedge i \neq j \bullet \sum_{L \in F(N_{15},5)} c_L - \sum_{L \in F(N_{15},5) \wedge i \in L} c_L - \sum_{L \in F(N_{15},5) \wedge j \in L} c_L + \sum_{L \in F(N_{15},5) \wedge \{i,j\} \subseteq L} c_L = 0$$

$$\Rightarrow \forall i, j \in N_{15} \wedge i \neq j \bullet \sum_{L \in F(N_{15},5)} c_L - \sum_{L \in F(N_{15}-\{i\},4)} c_{\{i\}\cup L} - \sum_{L \in F(N_{15}-\{j\},4)} c_{\{j\}\cup L} + \sum_{L \in F(N_{15}-\{i,j\},3)} c_{\{i,j\}\cup L} = 0$$

$$\Rightarrow \forall i, j \in N_{15} \wedge i \neq j \bullet 1-1-1+ \sum_{L \in F(N_{15}-\{i,j\},3)} c_{\{i,j\}\cup L} = 0$$

$$\Rightarrow \forall i, j \in N_{15} \wedge i \neq j \bullet \sum_{L \in F(N_{15}-\{i,j\},3)} c_{\{i,j\}\cup L} = 1$$

Atņemsim visu tādu monomu koeficientus, kuru pakāpe ir mazāka par 3.

$$\forall i, j \in N_{15} \wedge i \neq j \bullet \sum_{L \in F(N_{15}-\{i,j\},3)-F(N_{15}-\{i,j\},0)} c_{\{i,j\}\cup L} = \sum_{L \in F(N_{15}-\{i,j\},3)} c_{\{i,j\}\cup L} - c_{\{i,j\}} = 1-(-1) = 2$$

- ✓ Ja precīzi 12 no ievaddatu bitiem ir vienādi ar 1, tad meklētās funkcijas rezultāts (apzīmēsim to ar t) 175 gadījumos ir 1, bet pārējos 280 gadījumos vienāds ar 0.

Ja apzīmēsim ievaddatu nuļļu bitu kārtas numurus ar i, j un k $(i \neq j \neq k)$, tad

$$\forall i, j, k \in N_{15} \wedge i \neq j \neq k \bullet \sum_{L \in F(N_{15},5)} c_L - C_3^1 \times \sum_{L \in F(N_{15},5) \wedge i \in L} c_L + C_3^2 \times \sum_{L \in F(N_{15},5) \wedge \{i,j\} \subseteq L} c_L - C_3^3 \sum_{L \in F(N_{15},5) \wedge \{i,j,k\} \subseteq L} c_L = t$$

$$\Rightarrow \forall i, j, k \in N_{15} \wedge i \neq j \neq k \bullet \sum_{L \in F(N_{15},5)} c_L - 3\times \sum_{L \in F(N_{15}-\{i\},4)} c_{\{i\}\cup L} + 3\times \sum_{L \in F(N_{15}-\{i,j\},3)} c_{\{i,j\}\cup L} - \sum_{L \in F(N_{15}-\{i,j,k\},2)} c_{\{i,j,k\}\cup L} = t$$

$$\Rightarrow \forall i, j, k \in N_{15} \wedge i \neq j \neq k \bullet 1-3\times 1+3\times 1- \sum_{L \in F(N_{15}-\{i,j,k\},2)} c_{\{i,j,k\}\cup L} = t$$

$$\Rightarrow \forall i, j, k \in N_{15} \wedge i \neq j \neq k \bullet \sum_{L \in F(N_{15}-\{i,j,k\},2)} c_{\{i,j,k\}\cup L} = 1-t$$

$$\Rightarrow \forall i, j, k \in N_{15} \wedge i \neq j \neq k \bullet \sum_{L \in F(N_{15}-\{i,j,k\},2)} c_{\{i,j,k\}\cup L} \in \{0,1\}$$

- ✓ Ja precīzi 11 no ievaddatu bitiem ir vienādi ar 1, tad funkcijas rezultāts (apzīmēsim to ar t) 1050 gadījumos ir 1, bet pārējos 315 gadījumos tas ir vienāds ar 0.

Ja apzīmēsim ievaddatu nuļļu bitu kārtas numurus ar i, j, k un l $(i \neq j \neq k \neq l)$, tad

$$\forall i,j,k,l \in N_{15} \wedge i \neq j \neq k \neq l \bullet$$
$$\sum_{L \in F(N_{15},5)} c_L - C_4^1 \times \sum_{L \in F(N_{15},5) \wedge i \in L} c_L + C_4^2 \times \sum_{L \in F(N_{15},5) \wedge \{i,j\} \subseteq L} c_L - C_4^3 \sum_{L \in F(N_{15},5) \wedge \{i,j,k\} \subseteq L} c_L + C_4^4 \sum_{L \in F(N_{15},5) \wedge \{i,j,k,l\} \subseteq L} c_L = t$$
$$\Rightarrow \forall i,j,k,l \in N_{15} \wedge i \neq j \neq k \neq l \bullet$$
$$\sum_{L \in F(N_{15},5)} c_L - 4 \times \sum_{L \in F(N_{15}-\{i\},4)} c_{\{i\} \cup L} + 6 \times \sum_{L \in F(N_{15}-\{i,j\},3)} c_{\{i,j\} \cup L} - 4 \times \sum_{L \in F(N_{15}-\{i,j,k\},2)} c_{\{i,j,k\} \cup L} + 1 \times \sum_{L \in F(N_{15}-\{i,j,k,l\},1)} c_{\{i,j,k,l\} \cup L} = t$$
$$\Rightarrow \forall i,j,k,l \in N_{15} \wedge i \neq j \neq k \neq l \bullet$$
$$1 - 4 \times 1 + 6 \times 1 - 4 \times \{0,1\} + \sum_{L \in F(N_{15}-\{i,j,k,l\},1)} c_{\{i,j,k,l\} \cup L} = t$$
$$\Rightarrow \forall i,j,k,l \in N_{15} \wedge i \neq j \neq k \neq l \bullet \sum_{L \in F(N_{15}-\{i,j,k,l\},1)} c_{\{i,j,k,l\} \cup L} = t - 3 + 4 \times \{0,1\}$$
$$\Rightarrow \forall i,j,k,l \in N_{15} \wedge i \neq j \neq k \neq l \bullet \sum_{L \in F(N_{15}-\{i,j,k,l\},1)} c_{\{i,j,k,l\} \cup L} \in \{-3,...,2\}$$

- ✓ Ja precīzi 10 no ievaddatu bitiem ir vienādi ar 1, tad funkcijas rezultāts (apzīmēsim to ar t) 2877 gadījumos ir 1, bet pārējos 126 gadījumos tas ir 0.

Ja apzīmēsim ievaddatu nuļļu bitu kārtas numurus ar i, j, k, l un r ($i \neq j \neq k \neq l \neq r$), tad

$$\forall i,j,k,l,r \in N_{15} \wedge i \neq j \neq k \neq l \neq r \bullet \sum_{L \in F(N_{15},5)} c_L - C_5^1 \times \sum_{L \in F(N_{15},5) \wedge i \in L} c_L + C_5^2 \times \sum_{L \in F(N_{15},5) \wedge \{i,j\} \subseteq L} c_L -$$
$$- C_5^3 \sum_{L \in F(N_{15},5) \wedge \{i,j,k\} \subseteq L} c_L + C_5^4 \sum_{L \in F(N_{15},5) \wedge \{i,j,k,l\} \subseteq L} c_L - C_5^5 \sum_{L \in F(N_{15},5) \wedge \{i,j,k,l,r\} \subseteq L} c_L = t$$
$$\Rightarrow \forall i,j,k,l,r \in N_{15} \wedge i \neq j \neq k \neq l \neq r \bullet \sum_{L \in F(N_{15},5)} c_L - 5 \times \sum_{L \in F(N_{15}-\{i\},4)} c_{\{i\} \cup L} + 10 \times \sum_{L \in F(N_{15}-\{i,j\},3)} c_{\{i,j\} \cup L} -$$
$$- 10 \times \sum_{L \in F(N_{15}-\{i,j,k\},2)} c_{\{i,j,k\} \cup L} + 5 \times \sum_{L \in F(N_{15}-\{i,j,k,l\},1)} c_{\{i,j,k,l\} \cup L} - 1 \sum_{L \in F(N_{15}-\{i,j,k,l,r\},0)} c_{\{i,j,k,l,r\} \cup L} = t$$
$$\Rightarrow \forall i,j,k,l,r \in N_{15} \wedge i \neq j \neq k \neq l \neq r \bullet 1 - 5 \times 1 + 10 \times 1 - 10 \times \{0,1\} + 5 \times \{-3,...,2\} -$$
$$- \sum_{L \in F(N_{15}-\{i,j,k,l,r\},0)} c_{\{i,j,k,l,r\} \cup L} = t$$
$$\Rightarrow \forall i,j,k,l,r \in N_{15} \wedge i \neq j \neq k \neq l \neq r \bullet \sum_{L \in F(N_{15}-\{i,j,k,l,r\},0)} c_{\{i,j,k,l,r\} \cup L} = -t + 6 - 10 \times \{0,1\} + 5 \times \{-3,...,2\}$$
$$\Rightarrow \forall i,j,k,l,r \in N_{15} \wedge i \neq j \neq k \neq l \neq r \bullet c_{\{i,j,k,l,r\}} \in \{-20,...,16\}$$

- ✓ Ja precīzi $SR \leq 9$ no ievaddatu bitiem ir vienādi ar 1, tad no iepriekš iegūtiem ierobežojumiem uz koeficientu kopu summām iegūstam jaunus ierobežojumus. Apzīmēsim $R = 15 - SR$ (tātad $R \in [6;15]$).

Ja visos iespējamos veidos ar i, j, k, l un r ($i \neq j \neq k \neq l \neq r$) apzīmēsim dažus no ievaddatu bitu virknes nuļļu bitiem, tad iegūstam

$$\forall i,j,k,l,r \in N_{15} \wedge i \neq j \neq k \neq l \neq r \bullet \sum_{L \in F(N_{15},5)} c_L - C_R^1 \times \sum_{L \in F(N_{15},5) \wedge i \in L} c_L + C_R^2 \times \sum_{L \in F(N_{15},5) \wedge \{i,j\} \subseteq L} c_L -$$
$$- C_R^3 \sum_{L \in F(N_{15},5) \wedge \{i,j,k\} \subseteq L} c_L + C_R^4 \sum_{L \in F(N_{15},5) \wedge \{i,j,k,l\} \subseteq L} c_L - C_R^5 \sum_{L \in F(N_{15},5) \wedge \{i,j,k,l,r\} \subseteq L} c_L = t$$

$$\Rightarrow \forall i,j,k,l,r \in N_{15} \wedge i \neq j \neq k \neq l \neq r \bullet \sum_{L \in F(N_{15},5)} c_L - R \times \sum_{L \in F(N_{15}-\{i\},4)} c_{\{i\} \cup L} + \frac{R(R-1)}{2} \times \sum_{L \in F(N_{15}-\{i,j\},3)} c_{\{i,j\} \cup L} -$$

$$- \frac{R(R-1)(R-2)}{6} \times \sum_{L \in F(N_{15}-\{i,j,k\},2)} c_{\{i,j,k\} \cup L} + \frac{R(R-1)(R-2)(R-4)}{24} \times \sum_{L \in F(N_{15}-\{i,j,k,l\},1)} c_{\{i,j,k,l\} \cup L} -$$

$$- \frac{R(R-1)(R-2)(R-4)(R-5)}{120} \sum_{L \in F(N_{15}-\{i,j,k,l,r\},0)} c_{\{i,j,k,l,r\} \cup L} = t$$

$$\Rightarrow \forall i,j,k,l,r \in N_{15} \wedge i \neq j \neq k \neq l \neq r \bullet 1 - R \times 1 + \frac{R(R-1)}{2} \times 1 - \frac{R(R-1)(R-2)}{6} \times \{0;1\} +$$

$$+ \frac{R(R-1)(R-2)(R-3)}{24} \times \{-3,...,2\} - \frac{R(R-1)(R-2)(R-3)(R-4)}{120} \times \{-20,...,16\} = t$$

$$\Rightarrow \forall i,j,k,l,r \in N_{15} \wedge i \neq j \neq k \neq l \neq r \bullet 1 - R + \frac{R^2 - R}{2} - \frac{R^3 - 3R^2 + 2R}{6} \times \{0,1\} +$$

$$+ \frac{R^4 - 6R^3 + 11R^2 - 6R}{24} \times \{-3,...,2\} - \frac{R^5 - 10R^4 + 35R^3 - 50R^2 + 24R}{120} \times \{-20,...,16\} = t$$

$$\Rightarrow \forall i,j,k,l,r \in N_{15} \wedge i \neq j \neq k \neq l \neq r \bullet \frac{120}{120} - \frac{120R}{120} + \frac{60R^2 - 60R}{120} - \frac{20R^3 - 60R^2 + 40R}{120} \times \{0,1\} +$$

$$+ \frac{5R^4 - 30R^3 + 55R^2 - 30R}{120} \times \{-3,...,2\} - \frac{R^5 - 10R^4 + 35R^3 - 50R^2 + 24R}{120} \times \{-20,...,16\} = \frac{120t}{120}$$

$$\Rightarrow \forall i,j,k,l \in N_{15} \wedge i \neq j \neq k \neq l \bullet$$

$$120 - 180R + 60R^2$$

$$+ \{0,1\} \times (-40R + 60R^2 - 20R^3)$$

$$+ \{-3,...,2\} \times (-30R + 55R^2 - 30R^3 + 5R^4)$$

$$+ \{-20,...,16\} \times (-24R + 50R^2 - 35R^3 + 10R^4 - R^5)$$

$$= 120t$$

Apskatīsim, kādi nosacījumi ir iegūstami mainīgā $R \in [6;15]$ pieļaujamos vērtību galapunktos:

$$R = 6 \Rightarrow 10 - 20 \times \{0,1\} + 15 \times \{-3,...,2\} - 6 \times \{-20,...,16\} = t$$

$$R = 15 \Rightarrow 91 - 455 \times \{0,1\} + 1365 \times \{-3,...,2\} - 3003 \times \{-20,...,16\} = t$$

Diemžēl šāda novērojumi nepalīdz atrast meklēto polinomu, tomēr vēlāk tie varētu noderēt kādu polinoma meklēšanas algoritmu sastāvā.

4.6. Polinoma meklēšanas reducēšana līdz uzdevumam par fiksētās maksas maksimālās plūsmas meklēšanu grafā

4.6.1. Jēdziens par plūsmām grafos

Definīcija. Plūsmas grafs (*flow network*) – orientēts grafs $G = (V, E)$, katrai no kura šķautnēm $(u,v) \in E$ ir piekārtota vērtība $c(u,v) \geq 0$, kuru sauc par šķautnes caurlaidspēju

(*capacity*). Ja $(u,v) \notin E$, tad pieņemsim, ka $c(u,v) = 0$. Grafā izdalīsim divas virsotnes: izteku s (*source*) un ieteku t (*sink*) [6].

Definīcija. Plūsma – dots plūsmas grafs $G = (V,E)$ ar izteku s un ieteku t, kura šķautņu caurlaidspēja tiek definēta ar funkcijas c palīdzību. Par plūsmu (*flow*) šajā grafā sauksim funkciju $f : V \times V \to R$, kas apmierina trīs nosacījumus [6]:

- ✓ pieļaujamās caurlaidības ierobežojums (*capacity constraint*): $f(u,v) \le c(u,v)$ visiem u,v no V;
- ✓ pretējā simetrija (*skew symmetry*): $f(u,v) = -f(v,u)$ visiem u,v no V;
- ✓ plūsmas saglabāšanās (*flow conservation*): $\sum_{v \in V} f(u,v) = 0$ visiem u,v no $V - \{s,t\}$.

Plūsmas apjoms $f(u,v)$ var būt kā pozitīvs, tā arī negatīvs. Tas nosaka, kāds ir vielas daudzums, kas virzās no virsotnes u uz virsotni v (negatīvs lielums nozīmē pārvietošanos pretējā virzienā).

Definīcija. Plūsmas daudzums grafā – no sākuma virsotnes uz visām pārējām virsotnēm izejošās plūsmas daudzuma summa $\sum_{v \in V} f(s,v)$ [6].

Uzdevums par maksimālās plūsmas atrašanu grafā – atrast lielāko iespējamo plūsmas daudzumu grafā G ar izteku s un ieteku t.

Pamainīsim plūsmu grafa definīciju tā, lai visas $G = (V,E)$ šķautnes $(u,v) \in E$ saturētu vēl vienu raksturlielumu – cenu $cst(u,v)$, kura tiek ņemta par katru atsevišķu plūsmas vienību, kas iet šajā šķautnē. Tad šķautnes $(u,v) \in E$ kopējā cena būs vienāda ar $cst(u,v) * f(u,v)$, bet visā grafā tekošās plūsmas cena attiecīgi vienāda ar $\sum_{v,u \in V} f(v,u) * c(v,u)$.

Pamainīsim arī plūsmas grafa caurlaidspējas definīciju. Mūsu būvētajā grafā šķautnēm caurlaidspēju ierobežosim ne tikai no augšas, bet arī no apakšas – norādīsim, ka caur katru no šķautnēm $(u,v) \in E$ ir noteikti jātek vismaz $c^*(u,v) \ge 0$ plūsmas vienībām.

Uzdevums par minimālās cenas maksimālās plūsmas atrašanu grafā – atrast lielāko iespējamo plūsmas daudzumu grafā G ar izteku s un ieteku t, kuras kopējā cena starp tāda paša lieluma plūsmām grafā ir mazākā iespējamā.

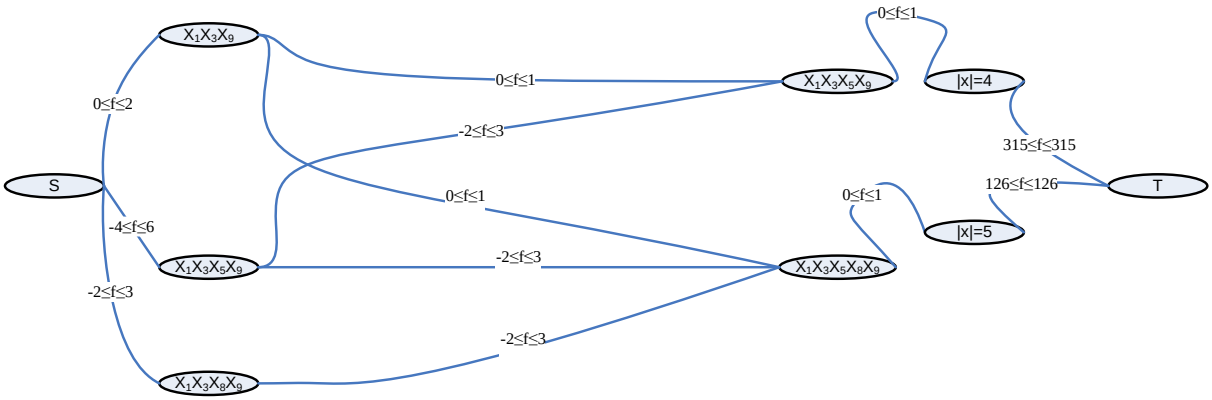
4.6.2. Uzdevuma reducēšana

Šajā nodaļā izmantosim polinoma koeficientu reprezentāciju. Atcerēsimies, ka meklētajam 5. pakāpes 15 mainīgo polinomam viennozīmīgi zināmi koeficienti visiem 0., 1.

un 2. pakāpes monomiem, bet aizvien nepieciešams atrast koeficientus monomiem ar pakāpēm 3, 4 un 5.

Kā jau pieminējām iepriekš, jebkura funkcijas ievaddatu bitu virkne ar precīzi k vieniniekiem var tikt attēlota kā konkrēts monoms, kura pakāpe ir k. Savukārt katra 3., 4. vai 5. pakāpes monoma koeficienta izvēle ietekmē funkciju rezultātus priekš tādiem ievaddatiem, kuros visi biti ar konkrēto monomu sastādošo mainīgo kārtas numuriem ir vienādi ar 1. Katrs no pieminētiem funkcijas rezultātiem tiek palielināts vai samazināts par vienu un to pašu skaitli, kas ir vienāds ar monoma koeficienta vērtību.

Pastāvošās likumsakarības varam uzzīmēt plūsmas grafa veidā (skat. 4.6.2.1. att.). Visās šķautnēs šajā grafā plūsma virzās no kreisās puses uz labo. Lai iegūtu meklētā polinoma koeficientus, nepieciešams atrast maksimālo plūsmu šajā grafā.



4.6.2.1. att. **Plūsmas grafa būvēšanas pirmais posms, attēlotas tikai dažas no grafa virsotnēm**

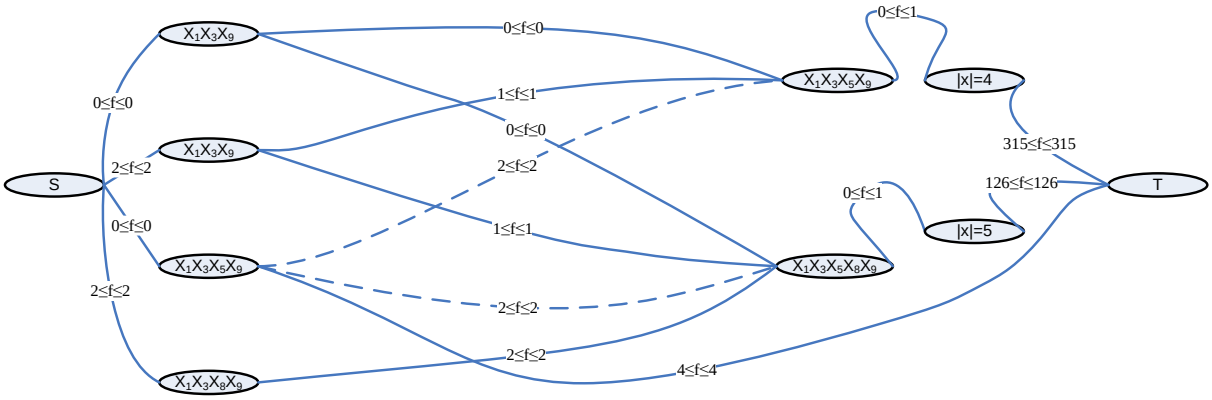
Grafa kreisajā pusē atrodas izteka s un virsotnes, kas atbilst visiem 3. līdz 5. pakāpes monomiem, katram no kuriem ir jāatrod koeficienti (attēlā 4.6.2.1. iekļautas tikai virsotnes $x_1x_3x_9$, $x_1x_3x_5x_9$, $x_1x_3x_8x_9$). Savukārt grafa labajā pusē atrodas funkcijas ievaddatus aprakstošie 3. līdz 15. pakāpes monomi (attēlā 4.6.2.1. iekļauti tikai $x_1x_3x_5x_9$, $x_1x_3x_5x_8x_9$).

Katram ievaddatu virknes vienīnieka bitu skaitam $p \in \{3,..,15\}$ grafā eksistēs virsotne ar nosaukumu $|x| = p$. Šajā virsotnē no visiem labās puses monomiem ar pakāpi p ieies šķautnes ar plūsmas ierobežojumu $0 \le f \le 1$, kas nozīmē, ka visiem funkcijas rezultātiem ir jābūt vienādiem ar 0 vai 1. Savukārt no šķautnēm $|x| = p$ uz ieteku t ies šķautnes ar ierobežojumu $t_p \le f \le t_p$, kur t_p ir skaitlis no viena mainīgā polinoma tabulas kolonnas ar numuru p – summārais gadījumu skaits, kuros ievaddatu virknei ar precīzi p vienīnieka bitiem atbilst funkcijas rezultāts 1.

Ierobežojumi šķautnēm, kas iet no grafa (skat. 4.6.2.1. att.) kreisās puses koeficientu virsotnēm uz grafa labās puses ievaddatu virsotnēm, atbilst neviennozīmīgo koeficientu pieļaujamām vērtībām, kuras esam noskaidrojuši nodaļā 4.5.2. Tā kā jebkura koeficienta vērtība vienādā apmērā ietekmē visus ar šo koeficientu saistītos funkcijas rezultātus, tad no katras grafa kreisās puses koeficientu virsotnes visu izejošo šķautņu plūsmas lielumiem ir jābūt vienādiem savā starpā. Savukārt no iztekas s uz katru no grafa kreisās puses koeficientu virsotnēm atļauts virzīt tieši tādu plūsmas daudzumu, cik summāri var būt nepieciešams, lai vēlāk šo plūsmu jebkurai no iespējamām koeficientu vērtībām varētu sadalīt vienādās daļās tālākai virzīšanai uz grafa labās puses ievaddatu virsotnēm.

Diemžēl augstāk aprakstītos ierobežojumus attiecībā uz vairākās šķautnēs vienlaicīgi tekošo vienādo (bet stingri neierobežoto) plūsmu apjomu ieviest nav iespējams. Tāpēc modificēsim šo grafu, katru no kreisās puses koeficientu virsotnēm sadalot tik atsevišķās virsotnēs, ar cik dažādām vērtībām var būt vienāds attiecīgās monoma pakāpes koeficients.

Piemēram, 4. pakāpes monoma koeficienta iespējamās vērtības ir no -2 līdz 3, ieskaitot. Tāpēc katru no 4. pakāpes koeficientu virsotnēm sadalīsim 6 dažādās virsotnēs. Pirmā no šīm virsotnēm pa katru no šķautnēm uz grafa labo pusi virzīs precīzi 3 plūsmas vienības; otrā no šīm virsotnēm – precīzi 2 plūsmas vienības, utt.

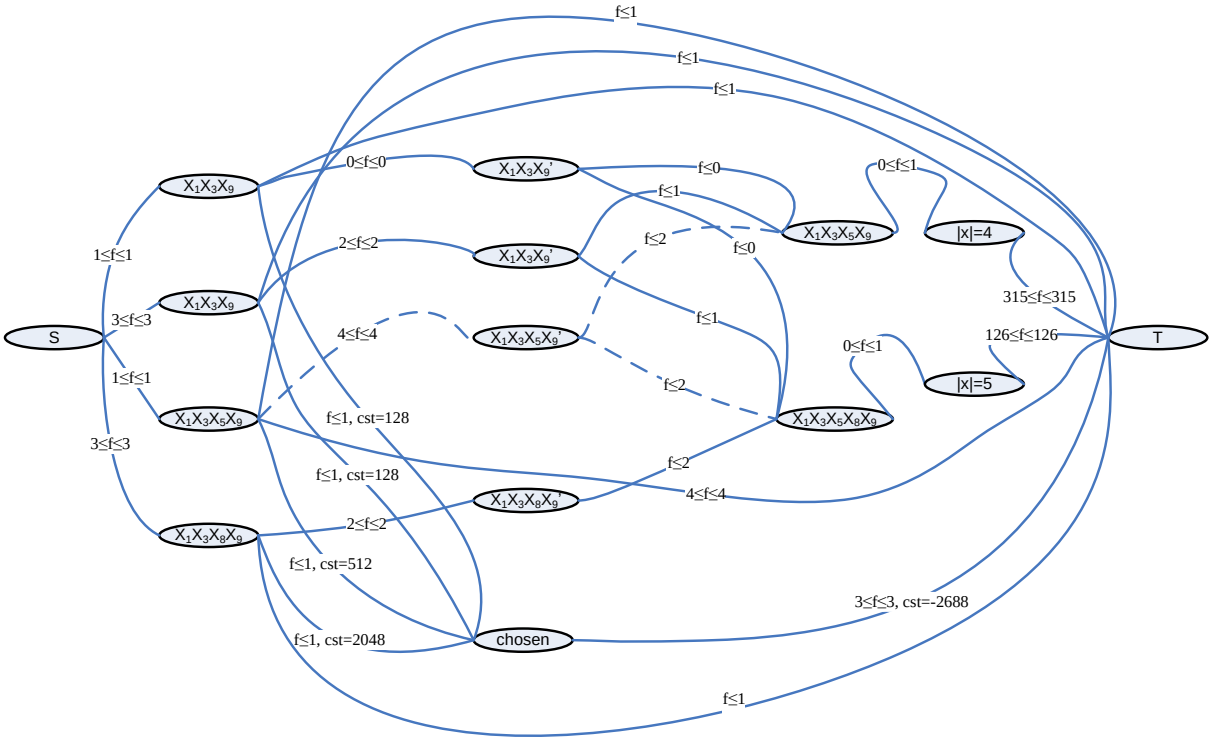


4.6.2.2. att. **Plūsmas grafa būvēšanas otrais posms, attēlotas tikai dažas no grafa virsotnēm**

Gadījumā, kad grafā vēlamies attēlot negatīvu koeficientu $-e$, izveidojam tādas pašas šķautnes un to ierobežojumus kā koeficientam e, bet pamainām šo šķautņu virzienu uz pretējo. Šķautnes, kas iet virzienā no labās grafa puses uz kreiso, zīmēsim ar raustītu līniju. Negatīvi koeficienti atņemot daļu no topošā funkcijas rezultāta. Šāda veida no grafa labās puses ievaddatu virsotnēm „atņemtai" plūsmai ir kaut kur jāpazūd, tāpēc šo plūsmu no grafa kreisās puses vēlāk pilnībā novadīsim uz ieteku t (skat. 4.6.2.2. att.).

Tomēr arī attēlā 4.6.2.2. attēlotais grafs nav pabeigts. Pašlaik plūsma tiek laista caur visām grafa kreisās puses virsotnēm neatkarīgi no to koeficienta vērtībām. Gribētos panākt,

lai no vairākām viena monoma dažādu koeficientu virsotnēm tieši viena tiktu lietderīgi izmantota grafa labās puses ievaddatu virsotnēs, bet visas pārējās šim monomam atbilstošās virsotnes ar citiem koeficientiem no tām izejošās plūsmas „izšķērdētu". Tāpēc no iztekas s katrā no virsotnēm laidīsim vienu papildus plūsmas vienību, kā arī vecām šķautnēm starp grafa kreisām un labām virsotnēm ar papildus pievienotās virsotnes palīdzību ieviesīsim stingrus ierobežojumus no apakšas un augšas, lai pa tām turpinātu tecēt tāds pats plūsmas daudzums, kāds tecēja līdz šim. Tādējādi papildus ieviestajai plūsmas vienībai būs divi iespējamie ceļi – vai nu tecēt uz grafa ieteku t uzreiz, vai nu tecēt uz t caur jaunpievienoto virsotni *chosen* (skat. 4.6.2.3. att.).



4.6.2.3. att. **Plūsmas grafa būvēšanas trešais posms, attēlotas tikai dažas no grafa virsotnēm**

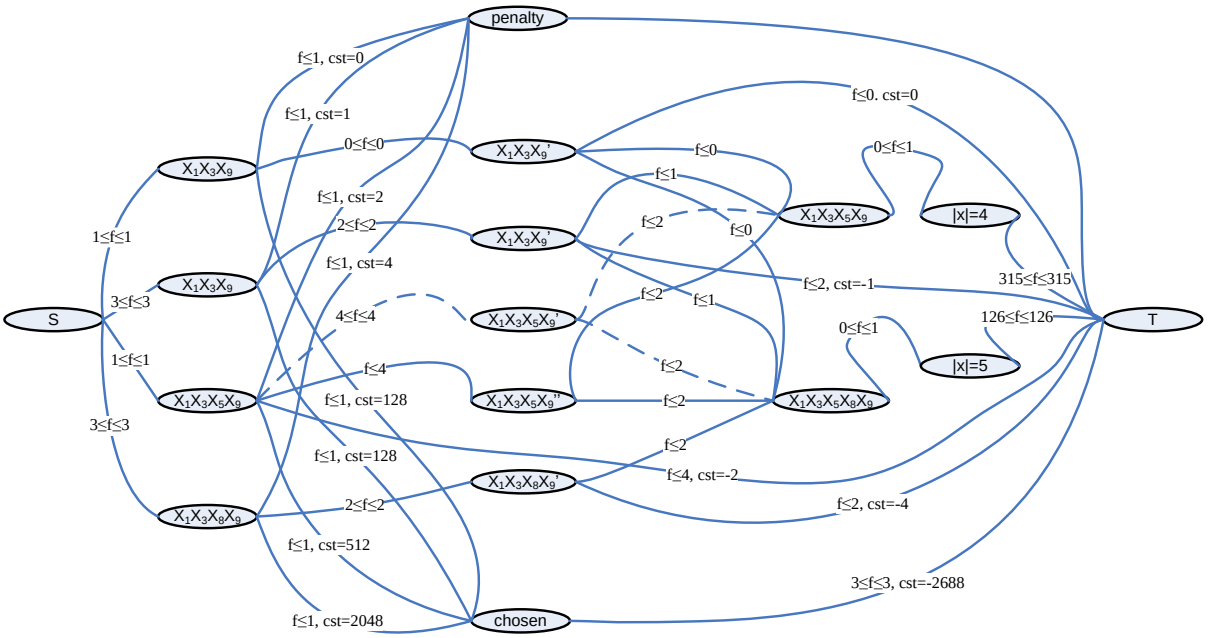
No virsotnes *chosen* uz grafa ieteku t novilkta šķautne ar tādu ierobežojumu, lai caur to vajadzētu iet tieši tādam plūsmas daudzumam, no cik 3., 4. un 5. pakāpes monomiem sastāv meklētais polinoms (attēla 4.6.2.3. kreisajā pusē attēloti tikai 3 dažādi monomi, tāpēc arī šīs šķautnes caurlaišanas spēja ir $3 \le f \le 3$).

Ar virsotni *chosen* (un turpmāk arī – *penalty*) savienoto šķautņu plūsmu aprakstā turpmāk izmantosim cenu, kura atbilst šķautnes izmantošanai kopumā, nevis par katru no plūsmas vienībām, kas tek šajā šķautnē. Šo nepilnību var viegli novērst, vienas divnieka pakāpes cenas vietā izmantojot p dažādu divnieka pakāpju summu priekš vieninieka plūsmas šķautnēm. Savukārt šķautni, kas no augšas un apakšas ierobežota ar plūsmas apjomu p,

varam sadalīt p atsevišķās vieninieka plūsmas šķautnēs, katrai no kurām būs unikāla divnieka pakāpes cena.

Lai nodrošinātu, ka priekš katra unikāla monoma caur virsotni *chosen* tecētu vienības plūsma tieši vienai no vairāku šī monoma koeficientu virsotnēm, visām no šīm koeficientu virsotnēm uz virsotni *chosen* izejošām vienības šķautnēm pieliksim vienādu cenu 2^k. Savukārt starp dažādu monomu koeficientu virsotnēm vērtībām k arī jābūt dažādām. Šķautnei, kas novilkta no *chosen* uz ieteku t, uzliksim cenu $\sum_{i \in K^*} -2^i$, kur ar K^* tiek apzīmēta visu izmantoto vērtību k kopa. Kopējai plūsmai grafā izveidosim nosacījumu, ka tās cenai noteikti jābūt vienādai ar 0.

Tā kā jebkuram skaitlim $\sum_{i \in K^*} -2^i$ atbilst tieši viens veids to izteikt kā dažādu divnieka pakāpju summu, tad līdz ar to esam izveidojuši ierobežojumu, ka caur virsotni *chosen* tecēs tieši viena plūsmas vienība no katra unikāla kreisajā grafa pusē izvietotā monoma. Meklētajā polinomā šī monoma koeficients būs vienāds ar to, no kuram atbilstošās virsotnes tek šī vienības plūsma. Savukārt plūsmas grafā tagad jāatrod iespēju anulēt visas plūsmas no tām koeficientu virsotnēm, kurām papildus vieninieka plūsma iet pa tiešo uz ieteku t, nevis cauri virsotnei *chosen*.



4.6.2.4. att. **Polinoma meklēšanas uzdevumam atbilstošā plūsmas grafa piemērs, attēlotas tikai dažas no grafa virsotnēm**

Visām virsotnēm, kuras vieninieka plūsmu nevar izstumt caur virsotni *chosen*, pa ceļam uz izteku uztaisīsim virsotni *penalty* (skat. 4.6.2.4. att.). Vienas plūsmas vienības

pārvietošana no monoma koeficienta virsotnes uz virsotni *penalty* maksās 2^r. Katrai koeficienta virsotnei skaitļa r lielums būs unikāls – pat viena monoma atbilstošām dažādu koeficientu virsotnēm. Aizvien atstāsim arī nosacījumu, ka plūsmas cenai visā grafā kopumā ir jālīdzinās precīzi 0. Līdzīgi kā ar virsotni *chosen*, izvēlēsimies tādas r vērtības, lai priekš jebkāda kopējās plūsmas apjoma šo cenu summu visā grafā $\sum_{i \in R^*} 2^i$ (kur ar R^* tiek apzīmēta visu izmantoto r vērtību kopa) varētu izteikt tikai vienā veidā.

Ja no koeficienta virsotnes vieninieka plūsma virzās uz virsotni *penalty*, tas nozīmē, ka koeficients ar šādu vērtību attiecīgajam monomam neietilps meklētā polinoma reprezentācijā. No šīs plūsmas vajadzētu kaut kā atbrīvoties – tāpēc no vidus virsotnēm izveidosim šķautnes, kas par cenu -2^r (izmantojot vienīgo veidu, kā kompensēt attiecīgās *penalty* šķautnes samaksāto cenu 2^r) var novadīt visu šīs virsotnes plūsmu pa tiešo uz grafa ieteku t, ignorējot grafa labās puses ievaddatu virsotnes. Šādā veidā atbrīvojamies no neērtām plūsmām, kuras neatbilst polinomā izmantotajiem koeficientiem.

Tātad esam polinoma meklēšanas uzdevumu noreducējuši uz tādas maksimālās plūsmas meklēšanu grafā, kuras summārā cena būtu vienāda ar 0. Atrodot šādu plūsmu, ir zināms, ka katram no polinoma 3. līdz 5. pakāpes monomiem atbilst tas koeficients, no kura virsotnes grafā iet vieninieka plūsma uz virsotni *chosen*. Ja nav iespējams visu iztekas s plūsmu novadīt līdz ietekai t, vai arī nav iespējams uzbūvēt šo plūsmu tā, lai tās vērtība būtu 0 – tad polinoms neeksistē.

4.6.3. Zināmie algoritmi plūsmu uzdevumu risināšanai

Uzbūvētā grafa daudzām šķautnēm eksistē plūsmas apjoma ierobežojumi no apakšas, kuru vērtība ir lielāka par nulli. Lai atrastu lielāko plūsmu šādā grafā, tas no sākuma ir jāpārveido tā, lai nevienai šķautnei ierobežojumu no apakšas vairs nebūtu. Pārveidojums ir iespējams, izveidojot grafā papildus iztekas un ietekas un starp tām atrodot maksimālo plūsmu [7].

Uzbūvētais grafs sastāv no šādām virsotnēm:

- ✓ 4 unikālas virsotnes s, t, *chosen*, *penalty*;
- ✓ 12 virsotnes $|x| = p$ katram no ievaddatu virkņu garumiem $p \in \{3,..,15\}$;
- ✓ 32647 (iegūstams kā $\sum_{3 \le p \le 15} C_n^p$) virsotnes, kas atbilst katrai no ievaddatu virknēm garumā $p \in \{3,..,15\}$;

- ✓ 113932 (iegūstams kā $2\times\left(1\times C_{15}^{3}+4\times C_{15}^{4}+17\times C_{15}^{5}\right)$) monomu nenegatīvu koeficientu virsotnes grafa kreisajā pusē, kā arī katrai no pamatvirsotnēm viena palīgvirsotne grafa vidū;
- ✓ 189735 (iegūstams kā $3\times\left(1\times C_{15}^{3}+2\times C_{15}^{4}+20\times C_{15}^{5}\right)$) monomu negatīvu koeficientu virsotnes grafa kreisajā pusē, kā arī katrai no pamatvirsotnēm divas palīgvirsotnes grafa vidū.

Tātad grafa kopējais virsotņu skaits N ir vienāds ar 336330, savukārt šķautņu skaits M ir vienāds ar virsotņu skaitu N, kas pareizināts ar nelielu konstanti.

Maksimālās plūsmas atrašana grafā no N virsotnēm un M šķautnēm ir iespējama [6]

- ✓ laikā $O(N\times M^{2})$ ar algoritmu Ford-Fulkerson;
- ✓ laikā $O(N^{2}\times M)$ ar algoritmu Preflow-Push;
- ✓ laikā $O(N^{3})$ ar algoritmu Lift-To-Front.

Atrodot jebkādas cenas maksimālo plūsmu, iespējams no tās iegūt minimālās cenas maksimālo plūsmu ar ciklu svītrošanas algoritma palīdzību (*cycle-canceling algorithm*). Šim nolūkam grafā secīgi jāatrod visus tādus ciklus ar negatīvām cenām, katrā no kuru šķautnēm iespējams pamainīt plūsmas tecēšanas apjomu tā, ka kopējā plūsma grafā nemainās, savukārt kopējās plūsmas cena samazinās. To ir iespējams izdarīt laikā $O(N^{3}\times C\times P)$, kur P ir lielākā iespējamā šķautnes caurlaidspēja grafā un C ir pēc absolūtās vērtības lielākā šķautnes cena grafā [8].

Ievērosim arī to, ka katrā no grafa šķautnēm ir atļauts virzīt tikai tādus plūsmas apjomus, kuri ir izsakāmi veselos skaitļos. Šīs īpašības dēļ uzdevums paliek sarežģītāks.

Diemžēl autoram nav izdevies atrast informāciju par to, vai šajā nodaļā iegūtajam uzdevumam par konstantās cenas lielākās plūsmas atrašanu grafā eksistē polinomiālā laika risinājums.

4.7. Polinoma izteikšana kā vairāku polinomu summu

Atkarībā no mainīgā x_{15} vērtības, meklētā polinoma uzvedību var sadalīt divos atsevišķos gadījumos: $x_{15}=0$ vai $x_{15}=1$. Tātad šo polinomu varam izteikt kā $p=A+x_{15}\times B$, kur A un B ir polinomi, abi no kuriem nesatur mainīgo x_{15}. A ir 5. pakāpes 14 mainīgo polinoms, bet B ir 4. pakāpes 14 mainīgo polinoms.

Gadījumā $x_{15}=0$ polinoma p vērtība ir vienāda ar polinomu A. Savukārt gadījumā $x_{15}=1$ polinoma p vērtība ir vienāda ar $A+B$ (tātad $p=A+x_{15}\times B$ var pārveidot arī par

$p = (1 - x_{15}) \times A + x_{15} \times (A + B)$). Mēģināsim izpētīt polinomu p, A, B un $A + B$ savstarpējās likumsakarības. Turpmāk šajā nodaļā polinomu $A + B$ sauksim arī par AB. Ievērosim, ka arī polinoms AB ir 5. pakāpes 14 mainīgo polinoms.

Acīmredzams, ka katram no polinomiem p, A un AB – visu argumentu definīcijas apgabali ir $\{0,1\}$. Polinomu A un AB vērtību apgabali sakrīt ar polinoma p vērtību apgabalu $\{0,1\}$. Savukārt, lai gadījumā $x_{15} = 1$ saglabātos nosacījums par polinoma p vērtību apgabala atbilstību kopai $\{0,1\}$, polinoma B vērtību apgabals ir ierobežots vērtībās $\{-1,0,1\}$. Piemēram, ir pieļaujams gadījums, kad $A(x) = 1$ un $B(x) = -1$, jo polinoms $AB(x) = 0$ aizvien atbilst sava vērtību apgabala ierobežojumiem.

Meklēsim polinomu A un AB viena mainīgā polinomus (skat. 4.7.1. tab.). Polinoma p rezultāts pie jebkuras x_{15} vērtības daļēji vai pilnīgi sastāvēs no polinoma A. Bet polinoms AB mūs interesē tikai gadījumā, kad $x_{15} = 1$.

Tā kā polinoma B vērtību apgabals var būt $\{-1,0,1\}$, tad darbā izmantotie kombinatoriskie viena mainīgā polinomu skaidrojumi šajā gadījumā nav spēkā un pieminēt sākotnējā polinoma B viena mainīgā polinomu nav jēgas.

Izmantojot nodaļā 4.5.3. iegūtos ierobežojumus, zinām, ka $\forall i \in N_{15} \bullet \sum_{L \in F(N_{15} - \{i\}, 4)} c_{\{i\} \cup L} = 1$, tāpēc saskaitāmā $x_{15} \times B$ ietilpstošā polinoma B visu koeficientu summa ir vienāda ar 1.

4.7.1. tabula

Polinomiem p, A un AB atbilstošie viena mainīgā polinomi, zināmās vērtības aizpildītas

	0	1	2	...	13	14	15
p	0	15	104	...	0	0	1
A	0	14	...	...	...	0	-
AB	1	...	...	...	...	1	-

Ja visi 15 ievaddatu virknes biti vienādi ar 1, tad

- ✓ vienīgais iespējamais polinoma p rezultāts arī vienāds ar 1;
- ✓ $x_{15} = 1$ un, tā kā visi polinoma B 14 ievaddatu biti ir vienādi ar 1, polinoma B rezultāts ir 1.

Nonākam pie secinājuma, ka pie 14 ar vieninieku vienādiem ievaddatu bitiem polinoma A rezultāts ir vienāds ar 0, savukārt šajā pašā gadījumā polinoma $A + B$ rezultāts ir 1.

Ja kopējais ievaddatu vieninieku bitu skaits ir 0, tad acīmredzot arī $x_{15} = 0$ un polinoma p rezultāts ir vienāds ar polinoma A rezultātu. Abi šiem polinomiem rezultāti ir vienādi ar 0 – tātad, polinoma A nulltās pakāpes monoma koeficients arī vienāds ar 0.

Turpmāk šajā nodaļā ar M^b apzīmēsim polinoma M rezultātu skaitu, kas ir vienādi ar 1, starp visām iespējamām ievaddatu bitu virknēm ar precīzi b vieninieka bitiem

Šķirojot pēc mainīgā x_{15} vērtības, iegūstam vienādību $p^1 = A^1 + AB^0$: A^1 gadījumā $x_{15} = 0$ un AB^0 gadījumā $x_{15} = 1$. Acīmredzami, ka $A^1 \le 14$ sasniedz lielāko vērtību, ja visi polinoma A pirmās pakāpes monomu koeficienti ir vienādi ar 1. Savukārt $AB^0 \le 1$ sasniedz lielāko vērtību, ja polinoma AB nulltās pakāpes koeficients ir 1. Tā kā $p^1 = 15$, tad abos gadījumos lielākās A^1 un AB^0 vērtības tiek viennozīmīgi sasniegtas un pēc indukcijas skaidrs, ka arī $B^0 = 1$. Tātad pašlaik mums ir zināms, ka $A = 0 + \sum_{1 \le i \le 14} x_i + \ldots$ un $B = 1 + \ldots$.

Iepriekšējā rindkopā pielietotais novērojums var būt vispārināts līdz $\forall k \bullet p^k = A^k + AB^{k-1}$. Bet AB^{k-1} ir vienāds ar tādu ievaddatu virkņu skaitu, kas satur precīzi $k-1$ vieninieka bitus un kurām izpildās $(A(x) = 0 \wedge B(x) = 1) \vee (A(x) = 1 \wedge B(x) = 0)$. Diemžēl šis veids izteikt polinoma AB^{k-1} vērtību nedod iespēju kaut ko vienkāršot. Mēs nezinām nevienu no polinomiem A un B, bet šajā gadījumā būtu nepieciešams šķirot ne tikai polinomu dažādo rezultātu skaitu, bet arī to, kādus no šiem rezultātiem katrs no polinomiem atgriež pie vienādas ievaddatu virknes x.

Jebkuram 14 mainīgo polinomam M, nezinot nekādu papildus informāciju, spēkā ir acīmredzams ierobežojums $0 \le M^b \le C_{14}^b$. Vairākām k vērtībām varētu pārlasīt visas pieļaujamās A^k un AB^{k-1} vērtības, kas summā vienādas ar mums zināmo p^k. Iegūstot šādas vērtības pieciem dažādiem k gadījumiem, pārējās vērtības varētu pārlasīt ar interpolācijas palīdzību, tāpat, kā to izdarījām, meklējot polinomu p. Tomēr pēc autora domām ir ļoti maza varbūtība, ka, darbojoties šādi, sanāktu vienkāršot apskatāmo uzdevumu.

4.8. Rijīgais algoritms meklētā polinoma atrašanai

Ņemot vērā nodaļā 4.5.1. pieminētos novērtējumus par dažādu polinoma reprezentāciju skaitu, kas atbilst 5. pakāpes viena mainīgā polinomam, – pilnā pārlase meklētā polinoma atrašanai nestrādās pietiekami efektīvi. Tāpēc polinoma meklēšanai izmantosim rijīgo algoritmu.

Šis algoritms no patvaļīgi uzģenerētām polinoma kombinatoriskām reprezentācijām (turpmāk sauksim arī par pozīcijām) izvēlēsies tādu, kuras novērtējums ir pēc iespējas mazāks. Tad no pašreizējās polinoma reprezentācijas katrā solī tiks mēģināts nomainīt kādu no reprezentācijā iekļautajiem monomiem pret citu tādas pašas pakāpes monomu, kas

reprezentācijā nav iekļauts. No visām iespējamām šādā veidā nomaiņām tiks izvēlēta tā, kura noved pie jaunas polinoma reprezentācijas ar pēc iespējas mazāku novērtējumu.

Tādējādi viena mainīgā polinoma tabulas nosacījumi attiecībā uz meklēta polinoma rezultātiem priekš 0, 1, 2, 3, 4 un 5 vieniniekiem ievaddatu bitu virknē vienmēr saglabāsies, savukārt pārbaudīt nāksies tos nosacījumus, kas atbilst visiem pārējiem vieninieku skaitiem no 6 līdz 15.

4.8.1. Pozīcijas novērtējuma noteikšana rijīgā algoritmā

Pielietojot rijīgo algoritmu, ir jāiemācās novērtēt jebkuru atsevišķi apskatāmu patvaļīgu pozīciju. Vērtējot pozīciju, skaitīsim tajā „nekārtības" – jo vairāk nekārtību, jo lielāks ir pozīcijas skaitliskais novērtējums. Nulles novērtējums atbilst atrastajai atrisinājuma pozīcijai. Novērtējums varētu sastāvēt no divām daļām:

- ✓ cik kopā ir tādu ievaddatu virkņu, kurām funkcijas rezultāts nav vienāds ar 0 vai 1 – tas ir svarīgākais rādītājs, katra no šādām nekārtībām novērtējumu palielinās par skaitli R;
- ✓ katram iespējamam vieninieka skaitam $p = \{6, ..., 15\}$ ievaddatu bitu virknē – cik funkcijas rezultātu, kas vienādi ar 1, trūkst (vai pārpalikumā) līdz viena mainīgā polinoma tabulā norādītās vērtības sasniegšanai – katra vieninieka rezultātu (to pašu skaitīsim arī priekš nulles) skaita novirze no vēlamā par vienu vienību palielina pozīcijas novērtējumu par S.

Intuitīvi gribētos prioritizēt pirmā veida nekārtību novēršanu, bet vairāku izvēļu gadījumā ļaut otrā veida nekārtībām būt par izšķirošām. Tātad neatkarīgi no otrā veida nekārtību skaita, pat vienas pirmā veida nekārtības novēršanai/nepieļaušanai ir jābūt par prioritāti.

Nodaļā 4.5.1. parādījām, ka pie precīzi 0, 1 vai 2 vieninieka bitiem ievaddatu virknē attiecīgie polinoma koeficienti ir viennozīmīgi atrodami. Tāpēc pašlaik apskatāmās pozīcijas reprezentācija sastāv no 175 trešās pakāpes monomiem, 315 ceturtās pakāpes monomiem un 126 piektās pakāpes monomiem – un arī šiem monomiem atbilstošais funkcijas vieninieka rezultātu skaits pēc polinoma kombinatoriskās reprezentācijas definīcijas ir pareizs. Lielākais otrā veida nekārtību skaits 5. pakāpes n mainīgo polinoma pozīcijai ir $C_n^6 + C_n^7 + ... + C_n^{n-1} + C_n^n$ – gadījumā, kad priekš visām ievaddatu virknēm ar vieninieka bitu skaitu virs 5 jebkurš funkcijas rezultāts nav vienādas ar 0 vai 1.

Lai viena pirmā veida nekārtība būtu svarīgāka par jebkuru otrā veida nekārtību skaitu, nepieciešams, lai izpildītos $R > S \times \sum_{6 \le i \le 15} C_n^i$. Priekš $n = 15$ vērtība $\sum_{6 \le i \le 15} C_n^i$ ir vienāda ar 27824, tāpēc varam izvēlēties šādas vērtības: $R = 27824 + 1$ un $S = 1$.

4.8.2. Vienas polinoma reprezentācijas ietvaros pastāvošās simetrijas

Katru no polinoma reprezentācijām pastāvošo simetriju dēļ var izteikt vairākos veidos. Lai nepieļautu iespēju, ka algoritms atkārtoti apskata jau apstrādāto pozīciju vai pat ieciklojās, apskatīsim šīs simetrijas un mēģināsim ieviest papildus nosacījumus, lai katru no pozīcijām varētu attēlot tikai vienā veidā:

- ✓ jebkādu no k-tās pakāpes monomiem var uzrakstīt $k!$ veidos, pārkārtojot to sastādošo mainīgo secību. Tomēr pozīcijas būtību šādi pārkārtojumi nemaina. Tāpēc monomus pierakstīsim tā, lai to sastādošie mainīgie x_i reizinājumā ietu indeksa i pieaugšanas secībā;
- ✓ apskatot visus reprezentācijā ietvertos k-tās pakāpes monomus (apzīmēsim to skaitu ar p), arī tos varam uzrakstīt jebkurā savstarpējā secībā, tas ir – attiecīgi $p!$ veidos. Tāpēc katrai no pakāpēm $k = \{3,4,5\}$ visus atbilstošos monomus polinoma reprezentācijas pierakstā varētu sakārtot to sastādošo k mainīgo indeksu secības pieaugšanas secībā;
- ✓ pastāv vēl viena veida simetrija – jebkurā pozīcijas pierakstā varam $n!$ jeb $15! \approx 1{,}3 \times 10^9$ veidos savā starpā samainīt 15 mainīgo nosaukumus (x_i) vietām. Acīmredzams, ka pozīcija nemainās, bet tās pieraksts, pēc iepriekšējo divu simetriju novēršanas, ar lielu varbūtību paliks citādāks.

Pirmā veida simetrijas ir nepieciešams novērst, jo ar divu monomu salīdzināšanas nepieciešamību rijīgā algoritmā izpildes laikā sastapsimies ļoti bieži. Diemžēl nav zināms efektīvs veids novērst trešā veida simetrijas, un arī otrā veida simetriju novēršanai jātērē daudz algoritma darbības laika resursu. Tāpēc novērtēsim, kuros gadījumos otrā un trešā veida simetrijas var rasties.

Rijīgais algoritms katrā solī centīsies uzlabot apskatāmās pozīcijas novērtējumu. Ja simetriju dēļ jau otro reizi tiek apskatīta tā pati pozīcija, tad kārtējā solī pašreizējās pozīcijas novērtējumu uzlabot nav izdevies – izvēle krita uz simetrisko pozīciju ar tādu pašu novērtējumu.

Lai pasliktinātu simetriskās pozīcijas izvēles varbūtību, katrā solī rīkosimies sekojoši:

- ✓ ja atradām pozīciju ar labāku novērtējumu par tekošo pozīciju – izvēlamies to;

- ✓ ja labāks novērtējums vēl nav atrasts, un atnāk i-tā šī soļa laikā sastaptā pozīcija ar novērtējumu vienādu ar pašreizējo pozīciju – ar varbūtību $p = \frac{1}{i}$ atceramies jauno pozīciju kā pašlaik labāko sastapto.

Tādējādi katru no pozīcijām, kuru novērtējums ir vienāds ar pašreizējās pozīcijas novērtējumu, neveiksmīga soļa laikā tālākai apstrādei izvēlēsimies ar vienādu varbūtību. Tas aizvien atstāj risku, ka mēs ieciklēsimies savstarpēji simetriskās pozīcijās, tomēr šīs idejas pielietojums varētu sniegt labākus rezultātus par determinētu nākamās pozīcijas izvēli.

4.8.3. Sākuma pozīcijas Kirkamana skolnieču uzdevuma risinājuma veidā

Nonākot pie koeficientu izvēles polinoma reprezentācijas 3. pakāpes monomiem, savā darbā E.Kalniņa izteica ideju, ka atbildes atrašanai varētu palīdzēt funkcijas vieninieka vērtībām atbilstošo ievaddatu kortežu simetriskais izvietojums [9]. Pārbaudīsim šo pieņēmumu. Mūsu mērķis ir izvēlēties ievaddatiem atbilstošos monomus tā, lai katri divi no n mainīgajiem vienā un tajā pašā monomā (starp visiem funkcijas rezultātam 1 atbilstošiem monomiem) būtu sastopami precīzi k gadījumos.

Šis uzdevums ir Kirkmana skolnieču uzdevuma modificējums. Kirkmana skolnieču sākotnējais uzdevums prasa sadalīt 15 skolnieces grupās pa 3 skolniecēm 7 secīgu dienu laikā (kopā $\frac{15}{3} \times 7 = 35$ grupas), lai nekādas divas no tām vienā un tajā pašā grupā nenokļūtu divas reizes.

Mūsu 15 mainīgo polinoma interpretācijā šis uzdevums skanēs šādi – jāsadala 15 skolnieces grupās pa s skolniecēm, lai visu dienu laikā kopā pastaigāties sanāktu g grupām, un šajā laika sprīdī katras no divām skolniecēm tiktos precīzi k reizes. Tā kā visiem no funkcijas vieninieka rezultātam atbilstošiem monomiem jābūt dažādiem, tad arī attiecībā uz Kirkmana modificēto uzdevumu jāizvirza nosacījums, ka nekādas divas no grupām nedrīkst pilnībā sakrist.

4.8.3.1. tabula

Kirkmana skolnieču modificētā uzdevumā izmantotie parametri

Grupas izmērs (s)	Grupas kopā (g)	Dienas	Tikšanās reizes (k)
3	175	35	5
4	315	105	18
5	126	42	12

Nezināmās mainīgo vērtības iegūstam šādi:

- ✓ s ir vienāds ar 3, 4 vai 5 – vieninieku skaitu ievaddatu kortežā (jeb attiecīgā monoma pakāpi), visus no kuriem vēlamies iegūt, atrisinot Kirkmana uzdevumu;
- ✓ g ir vienāds ar viena mainīgā polinoma tabulā norādīto gadījumu skaitu, kad funkcijas rezultāts ir vienāds ar 1 priekš ievaddatu bitu virknes ar precīzi s vieniniekiem;
- ✓ dienu skaits iegūstams, dalot kopumā nepieciešamo grupu skaitu g ar to grupu skaitu, ko vēlamies izveidot katrā no dienām (vienas dienas grupu skaits acīmredzami nepārsniedz $\left\lfloor \frac{15}{s} \right\rfloor$);
- ✓ katru divu iespējamo skolnieču tikšanās reižu skaits k iegūstams kā $k = \frac{g \times C_s^2}{C_{15}^2}$ – kopējais grupu skaits reizināts ar dažādu skolnieču pāru skaitu, ko var izveidot katrā no grupām, dalīts ar kopējo iespējamo pāru skaitu (no 15 skolniecēm var izveidot $C_{15}^2 = 105$ dažādus pārus).

Kā norādījām Kirkmana uzdevuma risināšanai izmantoto mainīgo vērtību tabulā (skat. 4.8.3.1. tab.), priekš trīs skolnieču grupas izmēra meklēsim 175 grupas, nevis 280 grupas – šī izvēle iepriekš tika pamatota nodaļā 4.4.

4.8.4. Risinājuma atrašana Kirkmana modificētajam uzdevumam

Kirkmana skolnieču modificētā uzdevuma risinājuma iegūšanai pamainīsim avotā [10] aprakstīto klasiskā Kirkmana uzdevuma risinājumu. Šim uzdevumam izmantosim rijīgo algoritmu, kas glabās nesen izdarīto pārveidojumu aprakstu nākamās pozīcijas iegūšanai, kā arī neļaus šādus pašus pārveidojumus īsā laika sprīdī pielietot atkārtoti, lai novērstu risku iecikloties starp nedaudzajām pozīcijām uzdevuma lokālā minimuma punktos.

Algoritms sastāvēs no sekojošām daļām:

- ✓ sākuma pozīcijas izveidošana – izmantojam gadījumskaitļu ģeneratoru, lai izveidotu sākuma pozīciju, kas sastāvēs no pareizā grupu un skolnieču skaita, bet nepievērsīs uzmanību, cik reizes tiekas jebkādas divas no skolniecēm;
- ✓ pozīcijas novērtējuma funkcija – līdzīgi polinoma meklēšanas rijīgajam algoritmam, arī Kirkmana uzdevumā „nekārtības" palielinās pozīcijas novērtējumu, bet atrastajam risinājumam atbildīs pozīcija ar novērtējumu 0. Katram no divu skolnieču pāriem tiks saskaitīts, cik reizes visu dienu laikā šīs skolnieces kopumā tiekas. Novērtējums katram pārim tiks palielināts par tādu vērtību, cik tikšanās reižu šīm skolniecēm ir par daudz vai par maz, salīdzinot ar

nepieciešamo rādītāju k. Savukārt katra no skolnieču grupām, kas visu dienu laikā nemainīgā sastāvā ir sastopamas vairāk par vienu reizi, novērtējumu palielina par 5 vienībām;

- ✓ pozīcijas izmaiņas viena soļa ietvaros – katrai no dienām mēģinām visos iespējamos veidos savstarpēji samainīt divas skolnieces no dažādām grupām. Izvēlamies tādas divas skolnieces, lai jaunizveidotajai pozīcijai būtu pēc iespējas mazāks novērtējums;
- ✓ nesen apmainīto skolnieču statistika – no brīža, kad divas skolnieces no dažādām grupām tiek samainītas kādā no dienām, 30 nākamo soļu laikā ir jāaizliedz atkārtoti mainīt šīs skolnieces šīs pašas dienas ietvaros. Piemēram, šādu divu skolnieču maiņu vērtēsim sliktāk nekā jebkādu citu iespējamo maiņu nākamo 30 soļu laikā.

No Kirkmana skolnieču uzdevuma atrastajiem izkārtojumiem vairākos veidos mēģināsim sastādīt rijīgā polinoma meklēšanas algoritma sākuma pozīcijas un salīdzināt to novērtējumus ar varbūtiski uzģenerētām sākuma pozīcijām.

5. REZULTĀTI

5.1. Būla funkcijas ar mūs interesējošām vaicājošo algoritmu sarežģītībām

Atcerēsimies, ka šī darba izvirzītais mērķis ir atrast un izpētīt tādas Būla funkcijas, kurām kvantu vaicājošā algoritma sarežģītība ir labāka par determinētā vaicājošā algoritma sarežģītību.

Darba gaitā tika apkopota informācija par vaicājošiem algoritmiem, kuri var tikt izmantoti Būla funkciju rezultāta atrašanai. No vairāku veidu vaicājošiem algoritmiem ir apskatīti determinēts vaicājošais algoritms un kvantu vaicājošais algoritms. Izmantojot funkcijas jutīguma mērus, parādījām pastāvošās sakarības starp dažāda tipa vaicājošo algoritmu sarežģītībām. Noskaidrojās, ka meklējam tādas Būla funkcijas, kuru pakāpe ir mazāka par to determinēto sarežģītību.

Būla funkcijām, kuru pakāpe ir mazāka par 5, jau atrasti piemēri ar labāko iespējamo determinēto sarežģītību, tāpēc tika apskatīti tieši 5. pakāpes polinomi.

Ar polinoma simetrizācijas metodes palīdzību izdevās atrast viena mainīgā polinomus, kas varētu atbilst mūs interesējošām 5. pakāpes 15 mainīgo funkcijām ar determinētā algoritma sarežģītību 15. Pateicoties šim novērojumam darba mērķis tika noformulēts saprotamākā veidā. Ir nepieciešams atrast tādu 5. pakāpes 15 mainīgo funkciju, kura atbilst kāda no iegūto viena mainīgā polinomu aprakstītiem nosacījumiem: atkarībā no funkcijas ievaddatu virknē esošo vieninieka bitu skaita ir zināms, cik gadījumos meklētās funkcijas rezultātam ir jābūt vienādam ar 1.

5.2. Piektās pakāpes 15 mainīgo polinomu meklēšana

Iepriekšējās nodaļās tika piedāvātas vairākas idejas 5. pakāpes 15 mainīgo polinomu meklēšanai:

- ✓ apskatītas kombinatoriski iegūstamās likumsakarības, kas apraksta ierobežojumus uz meklētā polinoma koeficientiem vai arī kopām, kas sastāv no šiem koeficientiem (nodaļa 4.5);
- ✓ aprakstīts veids, kā reducēt jebkura p. pakāpes m mainīgo polinoma meklēšanu līdz klasiskā grafu uzdevuma nestandarta modifikācijai, kurā prasīts atrast konstantās maksas lielāko plūsmu grafā (nodaļa 4.6);
- ✓ mēģināts polinomu izteikt kā divu mazāku polinomu summu, šķirojot to vērtības atkarībā no divām iespējamām mainīgā x_{15} vērtībām (nodaļa 4.7).

Diemžēl neviena no šīm idejām nepalīdzēja atrast meklētos polinomus, tāpēc daudz uzmanības tika veltīts arī polinoma meklēšanai ar rijīgā algoritma palīdzību (nodaļa 4.8). No astoņiem iespējami eksistējošiem 5. pakāpes 15 mainīgo polinomiem tika meklēti tādi divi, kuru atbilstošiem viena mainīgā polinomiem piemīt nodaļā 4.3. aprakstīts simetriskums (skat. 5.2.1. tab.).

5.2.1. tabula

Meklētiem 5. pakāpes 15 mainīgo polinomiem atbilstošie viena mainīgā polinomi

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A.	0	15	104	271	280	51	400	2163	4272	4605	2952	1085	184	1	0	1
B.	0	15	104	280	315	126	490	2205	4230	4515	2877	1050	175	0	0	1

Rijīgā algoritma sākuma pozīciju veidā mēģināts izmantot gan pilnīgi varbūtiski uzģenerētas polinoma reprezentācijas, gan ar Kirkmana skolnieču modificētā uzdevuma risinājuma palīdzību speciāli uzģenerētas pozīcijas.

5.2.1. Kirkmana skolnieču modificētā uzdevuma risinājumi

Tika atrasti Kirkmana skolnieču modificētā uzdevuma atrisinājumi priekš dažādām parametru vērtībām (skat. 5.2.1.1. tab.). Risinātais uzdevums skanēja sekojoši: nepieciešams atrast veidu, kā s skolnieces var pastaigāties g grupās pa k skolniecēm d dienu laikā, lai jebkuras divas no skolniecēm precīzi t dienās atrastos vienā un tajā pašā grupā, kā arī divās dažādās dienās nekādas divas grupas nedrīkst pilnīgi sakrist.

Kirkmana skolnieču modificētā uzdevuma risinājuma galvenās funkcijas pirmkods iekļauts 2. pielikumā. Nākamajos pielikumos doti arī daži no atrastajiem skolnieču izkārtojumiem.

Izpildes laiks tika mērīts, darbinot risinājuma programmu ar šādas konfigurācijas datoru: Intel Core 2 Duo E7300 @ 2.66GHz, 3.25 GB RAM, Win XP SP3.

5.2.1.1. tabula

Atrisinātie Kirkmana skolnieču modificētā uzdevuma gadījumi

Skolnieces	Grupas izmērs	Grupu skaits dienā	Dienas	Tikšanās reizes	Izpildes laiks	Risinājums pielikumā
15	3	5	35	5	0.359 sek.	3. pielikums
15	4	3	105	18	0.485 sek.	4. pielikums
15	5	2	63	12	5.906 sek.	5. pielikums
15	6	2	245	70	9.000 sek.	-.
15	7	2	2115	846	174.532 sek.	-

Atrastie izkārtojumi priekš grupu izmēriem 3, 4 un 5 izmantoti, lai veidotu rijīgā algoritma sākuma pozīcijas 5. pakāpes 15 mainīgo polinomu meklēšanai. Atgādināsim, ka

mūsu polinoma kombinatorā reprezentācija sastāv no 175 trešās pakāpes monomiem, 315 ceturtās pakāpes monomiem un 126 piektās pakāpes monomiem. Visos iespējamos veidos mēģināts dažām no šīm polinoma pakāpēm visus atbilstošos monomus izvēlēties, izmantojot iegūtos Kirkmana skolnieču izkārtojumus. Tām monomu pakāpēm, kuru ģenerēšanai konkrētajā gadījuma netika izmantots Kirkmana skolnieču modificētā uzdevuma risinājums – nepieciešamais monomu skaits tika uzģenerēts varbūtiski.

5.2.1.2. tabula

Dažādi veidi ģenerēt rijīgā algoritma sākuma pozīcijas, un tiem atbilstošie novērtējumiem

Monomu pakāpes, kuru ģenerēšanai izmantots Kirkmana uzdevums	Labākais pozīcijas novērtējums	Vidējais pozīcijas novērtējums	Sliktākais pozīcijai novērtējums
-	20039 (20659)	20608 (21033)	21093 (21277)
3.	20009 (20627)	22699 (23443)	22964 (23602)
4.	20028 (20692)	20464 (20962)	22946 (23618)
5.	20112 (20746)	20602 (21029)	21148 (21320)
3., 4.	22069 (22989)	22235 (23114)	22417 (23255)
3., 5.	20009 (20687)	22669 (23422)	22947 (23597)
4., 5.	20013 (20667)	20468 (20967)	22861 (23655)
3., 4., 5.	22161 (23031)	22161 (23031)	22161 (23031)

Tabulā 5.2.1.2 doti augstāk aprakstītā veidā priekš tabulas 5.2.1. polinoma „A” iegūto sākuma pozīciju labākie, sliktākie un vidējie novērtējumi – ārpus iekavām minēts pirmā veida nekārtību skaits, bet iekavās minēts otrā veida nekārtību skaits. Katra no tabulā minēto veidu pozīcijām šajā pētījumā tika uzģenerēta 50000 reizes (izņemot pozīciju „3., 4., 5.”, kura pilnībā sastāv no Kirkmana skolnieču risinājuma izkārtojumiem un tādējādi tiek iegūta viennozīmīgi).

Šāda veida pētījums tika veikts tikai polinomam „A”, jo polinoma „B” sākuma pozīciju nav iespējams izteikt Kirkmana skolnieču modificētā uzdevuma risinājuma izkārtojumu veidā tam attiecīgā viena mainīgā polinoma īpašību dēļ.

5.2.2. Labākie iegūtie pozīciju novērtējumi rijīgā algoritma darbināšanas gaitā

Rijīgam algoritmam katra soļa ietvaros ir nepieciešams apskatīt $\approx 0{,}74 \times 10^6$ $(175 \times (C_{15}^3 - 175) + 315 \times (C_{15}^4 - 315) + 126 \times (C_{15}^5 - 126))$ iespējamos veidus, kā kādu no pašreizējās polinoma reprezentācijas monomiem nomainīt uz citu pašlaik reprezentācijā neiekļautu monomu. Savukārt katrai no iespējamām monomu savstarpējām maiņām seko jauniegūtās pozīcijas novērtējuma iegūšana, kurai nepieciešami $\approx 7{,}6 \times 10^6$ $(\sum_{6 \le k \le 15} C_{15}^k \times (C_k^3 + C_k^4 + C_k^3))$ skaitļošanas soļi.

Šādu novērtējumu dēļ īpaša uzmanība tika pievērsta algoritma laika resursu patēriņa uzlabošanai, rūpīgi testējot iespējamās ātrdarbību uzlabojošās izmaiņas. Lielākā daļa no kombinatoriskām funkcijām tika izrēķinātas iepriekš un glabātas atmiņā, tādējādi taupot laika resursus uz atmiņas resursu rēķina. Veicot šāda veida optimizāciju programmas skaitļošanas ziņā kritiskajā pirmkoda daļā, ātrdarbība tika uzlabota apmēram 30 reizes. Piemēram, šāds paņēmiens izmantots noteikta garuma kombināciju kārtas numuru iegūšanai, kā arī nākamo pēc kārtas kombināciju ģenerēšanai.

Par spīti uzlabojumiem, izmantojot datoru ar augstāk minēto konfigurāciju, vienas pozīcijas novērtējums ilgst $\approx 0.05 - 0.09$ sekundes, savukārt viena soļa izdarīšanai nepieciešamas ≈ 16 stundas. Lai samazinātu katra soļa ietvaros nepieciešamo laika resursu apjomu, rijīgā algoritma realizācija, atrodot jebkuru pozīciju, kura ir labāka par pašreizējo – uzreiz pamaina pašreizējo pozīciju uz atrasto un sāk rijīgā algoritma soļa izpildi no jauna. Tādējādi tika panākts, ka 16 stundas viena soļa izpildei nepieciešamas tikai sliktākajā gadījumā, piemēram, kad šī soļa ietvaros pozīcijas novērtējums netika uzlabots.

Lai vēl vairāk paātrinātu rijīgā algoritma darbību, dažos veiktā pētījuma laika posmos pozīcijas tika novērtētas, ņemot vērā tikai tās nekārtības, kas rodas, apskatot polinoma ievaddatu virknes, kurās sastopami precīzi 6 vai 12 ar vieninieku vienādi biti. Autors izvirzīja pieņēmumu, ka, atrodot polinomu, kuram šajos gadījumos neeksistē nekārtības, ir liela varbūtība, ka arī pārējām ievaddatu virknēm nekārtību skaits būs tuvs nullei. Diemžēl neizdevās atrast pat tādu polinomu, kuram 6 vai 12 bitu vieninieku ievaddatu virkņu novērtējums būtu tuvs nullei.

5.2.2.1. tabula

Labākie no rijīgā algoritma atrastajiem polinomiem, kas atbilst apskatītiem viena mainīgā polinomiem

Viena mainīgā polinoms (skat. 5.2.1. tab.)	Labākais polinoms, vērtējot visas nekārtības	Labākais polinoms, vērtējot 6 vieninieka bitu nekārtības	Labākais polinoms, vērtējot 11 vieninieka bitu nekārtības
A	19247 (21069)	1124 (2814)	1189 (1189)
B	20527 (22285)	1355 (3013)	1262 (1262)

Katram no diviem tabulā 5.2.1. minētajiem viena mainīgā polinomiem apskatīsim labākos atrastos polinomus atbilstoši pilnajam novērtējumam, kā arī divu veidu daļējiem novērtējumiem (skat. 5.2.2.1. tab.).

Katru reizi, atrodot pozīciju ar novērtējumu labāku par tekošo pozīciju, – jaunatrastā pozīcija tika saglabāta uz datu nesēja. Kopumā saglabātas 4281 dažādas pozīcijas.

Rijīgā algoritma galvenās funkcijas pirmkodu var redzēt 6. pielikumā, bet novērtējuma funkcijas pirmkods atrodams 7. pielikumā. Augstāk minēto algoritmu realizācijai izmantota programmēšanas valoda C++, un kopumā uzrakstītais pirmkods aizņem 1275 rindiņas.

SECINĀJUMI

Secīgi apskatīsim un novērtēsim darba gaitā piedāvātās idejas tādas Būla funkcijas jeb tai atbilstošā polinoma meklēšanai, kas atbilstu kādam no uzģenerētajiem viena mainīgā polinomiem.

Apskatot konkrētu viena mainīgā polinomu, tika iegūti vairāku veidu ierobežojumi attiecībā uz meklētā polinoma atsevišķiem koeficientiem un visu koeficientu dažādu apakškopu summām. Starp atrastajiem ierobežojumiem pretrunas par polinoma eksistenci atrastas neizdevās, un turpmāk tie kļuva par citu polinoma atrašanas ideju neatņemamu sastāvdaļu. Tomēr šos ierobežojumus varētu būt iespējams pielietot neatkarīgas risinājuma metodes izvirzīšanai, piemēram, risinot no tiem izveidotu vienādojumu sistēmu. Varētu tos izmantot arī tāda lineārās programmēšanas uzdevuma ietvaros, kas censtos minimizēt starpību kvadrātu summu starp viena mainīgā polinoma vērtībām un kādā brīdi labākās iegūtās funkcijas rezultātu statistiku.

Tika atrasts veids, kā noreducēt polinoma meklēšanas uzdevumu līdz uzdevumam par konstantās cenas maksimālo plūsmu grafā. Lai gan minimālās cenas maksimālās plūsmas atrašana grafā ir ļoti plaši zināms klasisks grafu teorijas uzdevums, tomēr netika atrasta nekāda informācija par to, vai tas ir risināms gadījumā, ja vēlamies stingri ierobežot pieļaujamo maksimālās plūsmas cenu. Ņemot vērā arī to, ka šī grafa katrā šķautnē ir atļauts tikai tāds plūsmas daudzums, kas var būt izteikts veselā skaitļa veidā – uzdevumam varētu neeksistēt polinomiālā laika risinājuma. Tomēr šo plūsmas grafa uzdevumu nepieciešams izpētīt rūpīgāk. Neatkarībā no tā, vai tas ir atrisināms, vērts apskatīt arī to, vai kāda veida plūsmu pilnā pārlase šajā grafā nevarētu mums nepieciešamo rezultātu dot ar mazāku laika resursu patēriņu, nekā ar visu polinomu pilno pārlasi vai rijīgā algoritma darbināšanu.

Vislielāko neveiksmi cieta mēģinājums izteikt meklēto polinomu kā divu citu polinomu summu. Diemžēl šīs idejas rezultātā par polinomu jau zināmo informāciju sanācis izteikt tikai nedaudz savādākā veidā, bet netika iegūts priekšstats, kā tas varētu kādā veidā palīdzēt uzdevuma atrisinājuma atrašanai.

Daudz laika tika ieguldīts rijīgā algoritma izstrādei un optimizācijai. Tomēr šim algoritmam aizvien nepieciešami tādi laika resursu patēriņi, kas liedz gūt pietiekami labu priekšstatu par tā efektivitāti. Algoritma iegūto pozīciju nekārtību skaits salīdzinājumā ar pašā sākumā uzģenerētajām sākuma pozīcijām nav stipri krities un aizvien atrodas ļoti tālu no vēlamā nulles novērtējuma. Rijīgā algoritma ātrdarbības uzlabošanai nepieciešams dziļāk

izpētīt polinoma kombinatorisko reprezentāciju simetrijas veidus un uzlabot to noteikšanas un novēršanas paņēmienus.

Darbā tika apskatīts iepriekš izvirzīts pieņēmums, ka Kirkmana skolnieču modificētā uzdevuma risinājums varētu palīdzēt izdevīgas rijīgā algoritma sākuma pozīcijas atrašanai. Diemžēl šis pieņēmums neizrādījās patiess, jo varbūtiski uzģenerētu sākuma pozīciju novērtējumi ir salīdzināmi ar Kirkmana skolnieču risinājuma novērtējumiem. Pēc autora domām šajā jomā uzlabojumus iegūt nav iespējams.

Darbu iespējams attīstīt, vispārinot visas attiecībā uz 15 mainīgo polinomiem izvirzītās idejas mazāka mainīgo skaita polinomu meklēšanai. Attiecībā uz rijīgo algoritmu īpašu uzmanību jāpievērš pozīcijas novērtējuma noteikšanai gadījumos, kad tas atrodas tuvu nullei un tādējādi tiek sasniegta liela varbūtība iecikloties kādā no pozīciju novērtējumu lokālajiem minimumiem. Atrastos mazāka mainīgo skaita polinomus varētu izmantot padziļinātai 15 mainīgo polinoma atrašanas izpētei.

Nodaļā 1.5.2 tika minēts, ka viena mainīgā polinoma q pakāpe nepārsniedz sākotnējā polinoma p pakāpi jeb $\deg(q) \leq \deg(p)$. Šajā darbā, meklējot 5. pakāpes sākotnējos polinomus, tika apskatīti viena mainīgā polinomi ar pakāpi $\deg(q) = 5$. Nākotnē ir vērts apskatīt mazāku pakāpju viena mainīgā polinomus, jo arī tie var atbilst kādiem no sākotnējiem 5. pakāpes polinomiem.

Darba gaitā tika piedāvātas daudzveidīgas iespējas problēmas izpētei, un var apgalvot, ka darba mērķis ir sasniegts. Darbu var attīstīt arī turpmāk, dziļāk izpētot jebkādu no apspriestām polinomu meklēšanas idejām.

IZMANTOTĀ LITERATŪRA UN AVOTI

1. Drucker, A. Block Sensitivity of Minterm-Transitive Functions [tiešsaiste]. – [atsauce 16.05.2010.]. Pieejams: http://arxiv.org/abs/1001.2052.

2. Buhrman, H., Wolf, R. Complexity Measures and Decision Tree Complexity: A Survey. *Theoretical Computer Science, 2002, N 1, vol. 288, p. 21-43.*

3. Nisan, N., Wigderson, A. On Rank vs. Communication Complexity. *Combinatorica*, 2005, N 4, vol. 15, p. 557-565.

4. Lučko, A. Būla funkcijas ar zemu polinoma pakāpi pielietojumi kvantu skaitļošanā : bakalaura darbs. LU Fizikas un matemātikas fakultāte. Rīga : Latvijas Universitāte, 2002. 34 lp.

5. Saniee, K. A Simple Expression for Multivariate Lagrange Interpolation [tiešsaiste].– [atsauce 27.05.2010.]. Pieejams: http://www.siam.org/students/siuro/vol1issue1/S01002.pdf.

6. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. *Introduction to Algorithms (3rd Edition).* Massachusetts : The MIT Press, 2009. 1312 p.

7. Christofides, N. *Graph Theory - an Algorithmic Approach.* London : Academic Press, 1975. 415 p.

8. Sedgewick, R. *Algorithms in C++ Part 5: Graph Algorithms (3rd Edition).* Massachusetts : Addison-Wesley Professional, 2002. 528 p.

9. Kalniņa, E. Polinomiālā un vaicājumu sarežģītība Būla funkcijām : bakalaura darbs. LU Fizikas un matemātikas fakultāte. Rīga : Latvijas Universitāte, 2005. 78 lp.

10. Deng, D., Ma, J., Shen, H. A Simple and Efficient Tabu Search Heuristics for Kirkman Schoolgirl Problem [tiešsaiste]. University of Turku, June 2005 – [atsauce 11.05.2010.]. Pieejams: http://tucs.fi/publications/attachment.php?fname=TR704.pdf.

PIELIKUMI

1. pielikums. Ievērojamākie no atrastiem viena mainīgā polinomiem

Šajā pielikumā apkopoti visi 5. pakāpes viena mainīgā polinomi $q(|x|)$, kas atbilst $k \in \{5, ..., 15\}$ mainīgo sākotnējai Būla funkcijai un kuriem izpildās $\forall j \bullet q(j) + q(k - j) = C_k^j$.

5. pakāpes 5 mainīgo polinomi:

0 5 0 10 0 1

0 5 1 9 0 1

0 5 2 8 0 1

0 5 3 7 0 1

0 5 4 6 0 1

0 5 5 5 0 1

0 5 6 4 0 1

0 5 7 3 0 1

0 5 8 2 0 1

0 5 9 1 0 1

0 5 10 0 0 1

5. pakāpes 6 mainīgo polinomi:

0 6 0 10 15 0 1

0 6 1 10 14 0 1

0 6 2 10 13 0 1

0 6 3 10 12 0 1

0 6 4 10 11 0 1

0 6 5 10 10 0 1

0 6 6 10 9 0 1

0 6 7 10 8 0 1

0 6 8 10 7 0 1

0 6 9 10 6 0 1

0 6 10 10 5 0 1

0 6 11 10 4 0 1

0 6 12 10 3 0 1

0 6 13 10 2 0 1

0 6 14 10 1 0 1

0 6 15 10 0 0 1

5. pakāpes 7 mainīgo polinomi:

0 7 0 3 32 21 0 1

0 7 1 4 31 20 0 1

0 7 2 5 30 19 0 1

0 7 3 6 29 18 0 1

0 7 4 7 28 17 0 1

0 7 5 8 27 16 0 1

0 7 6 9 26 15 0 1

0 7 7 10 25 14 0 1

0 7 8 11 24 13 0 1

0 7 9 12 23 12 0 1

0 7 10 13 22 11 0 1

0 7 11 14 21 10 0 1

0 7 12 15 20 9 0 1

0 7 13 16 19 8 0 1

0 7 14 17 18 7 0 1

0 7 15 18 17 6 0 1

0 7 16 19 16 5 0 1

0 7 17 20 15 4 0 1

0 7 18 21 14 3 0 1

0 7 19 22 13 2 0 1

0 7 20 23 12 1 0 1

0 7 21 24 11 0 0 1

5. pakāpes 8 mainīgo polinomi:

0 8 7 0 35 56 21 0 1

0 8 8 2 35 54 20 0 1

0 8 9 4 35 52 19 0 1

0 8 10 6 35 50 18 0 1

0 8 11 8 35 48 17 0 1

0 8 12 10 35 46 16 0 1

0 8 13 12 35 44 15 0 1

0 8 14 14 35 42 14 0 1

0 8 15 16 35 40 13 0 1

0 8 16 18 35 38 12 0 1

0 8 17 20 35 36 11 0 1

0 8 18 22 35 34 10 0 1

0 8 19 24 35 32 9 0 1

0 8 20 26 35 30 8 0 1
0 8 21 28 35 28 7 0 1
0 8 22 30 35 26 6 0 1
0 8 23 32 35 24 5 0 1
0 8 24 34 35 22 4 0 1
0 8 25 36 35 20 3 0 1
0 8 26 38 35 18 2 0 1
0 8 27 40 35 16 1 0 1
0 8 28 42 35 14 0 0 1

5. pakāpes 9 mainīgo polinomi:

0 9 15 1 30 96 83 21 0 1
0 9 16 4 32 94 80 20 0 1
0 9 17 7 34 92 77 19 0 1
0 9 18 10 36 90 74 18 0 1
0 9 19 13 38 88 71 17 0 1
0 9 20 16 40 86 68 16 0 1
0 9 21 19 42 84 65 15 0 1
0 9 22 22 44 82 62 14 0 1
0 9 23 25 46 80 59 13 0 1
0 9 24 28 48 78 56 12 0 1
0 9 25 31 50 76 53 11 0 1
0 9 26 34 52 74 50 10 0 1
0 9 27 37 54 72 47 9 0 1
0 9 28 40 56 70 44 8 0 1
0 9 29 43 58 68 41 7 0 1
0 9 30 46 60 66 38 6 0 1
0 9 31 49 62 64 35 5 0 1
0 9 32 52 64 62 32 4 0 1
0 9 33 55 66 60 29 3 0 1
0 9 34 58 68 58 26 2 0 1
0 9 35 61 70 56 23 1 0 1
0 9 36 64 72 54 20 0 0 1

5. pakāpes 10 mainīgo polinomi:

0 10 23 2 10 126 200 118 22 0 1
0 10 24 6 15 126 195 114 21 0 1
0 10 25 10 20 126 190 110 20 0 1
0 10 26 14 25 126 185 106 19 0 1
0 10 27 18 30 126 180 102 18 0 1
0 10 28 22 35 126 175 98 17 0 1
0 10 29 26 40 126 170 94 16 0 1
0 10 30 30 45 126 165 90 15 0 1
0 10 31 34 50 126 160 86 14 0 1
0 10 32 38 55 126 155 82 13 0 1
0 10 33 42 60 126 150 78 12 0 1
0 10 34 46 65 126 145 74 11 0 1
0 10 35 50 70 126 140 70 10 0 1
0 10 36 54 75 126 135 66 9 0 1
0 10 37 58 80 126 130 62 8 0 1
0 10 38 62 85 126 125 58 7 0 1
0 10 39 66 90 126 120 54 6 0 1
0 10 40 70 95 126 115 50 5 0 1
0 10 41 74 100 126 110 46 4 0 1
0 10 42 78 105 126 105 42 3 0 1
0 10 43 82 110 126 100 38 2 0 1
0 10 44 86 115 126 95 34 1 0 1
0 10 45 90 120 126 90 30 0 0 1

5. pakāpes 11 mainīgo polinomi:

0 11 36 25 4 130 332 326 140 19 0 1
0 11 37 30 13 135 327 317 135 18 0 1
0 11 38 35 22 140 322 308 130 17 0 1
0 11 39 40 31 145 317 299 125 16 0 1
0 11 40 45 40 150 312 290 120 15 0 1
0 11 41 50 49 155 307 281 115 14 0 1
0 11 42 55 58 160 302 272 110 13 0 1
0 11 43 60 67 165 297 263 105 12 0 1
0 11 44 65 76 170 292 254 100 11 0 1
0 11 45 70 85 175 287 245 95 10 0 1
0 11 46 75 94 180 282 236 90 9 0 1
0 11 47 80 103 185 277 227 85 8 0 1
0 11 48 85 112 190 272 218 80 7 0 1
0 11 49 90 121 195 267 209 75 6 0 1
0 11 50 95 130 200 262 200 70 5 0 1
0 11 51 100 139 205 257 191 65 4 0 1
0 11 52 105 148 210 252 182 60 3 0 1
0 11 53 110 157 215 247 173 55 2 0 1
0 11 54 115 166 220 242 164 50 1 0 1
0 11 55 120 175 225 237 155 45 0 0 1

5. pakāpes 12 mainīgo polinomi:

0 12 50 58 7 106 462 686 488 162 16 0 1
0 12 51 64 21 120 462 672 474 156 15 0 1
0 12 52 70 35 134 462 658 460 150 14 0 1
0 12 53 76 49 148 462 644 446 144 13 0 1
0 12 54 82 63 162 462 630 432 138 12 0 1
0 12 55 88 77 176 462 616 418 132 11 0 1

0 12 56 94 91 190 462 602 404 126 10 0 1
0 12 57 100 105 204 462 588 390 120 9 0 1
0 12 58 106 119 218 462 574 376 114 8 0 1
0 12 59 112 133 232 462 560 362 108 7 0 1
0 12 60 118 147 246 462 546 348 102 6 0 1
0 12 61 124 161 260 462 532 334 96 5 0 1
0 12 62 130 175 274 462 518 320 90 4 0 1
0 12 63 136 189 288 462 504 306 84 3 0 1
0 12 64 142 203 302 462 490 292 78 2 0 1
0 12 65 148 217 316 462 476 278 72 1 0 1
0 12 66 154 231 330 462 462 264 66 0 0 1

5. pakāpes 13 mainīgo polinomi:

0 13 64 94 0 7 512 1204 1280 715 192 14 0 1
0 13 65 101 20 35 526 1190 1252 695 185 13 0 1
0 13 66 108 40 63 540 1176 1224 675 178 12 0 1
0 13 67 115 60 91 554 1162 1196 655 171 11 0 1
0 13 68 122 80 119 568 1148 1168 635 164 10 0 1
0 13 69 129 100 147 582 1134 1140 615 157 9 0 1
0 13 70 136 120 175 596 1120 1112 595 150 8 0 1
0 13 71 143 140 203 610 1106 1084 575 143 7 0 1
0 13 72 150 160 231 624 1092 1056 555 136 6 0 1
0 13 73 157 180 259 638 1078 1028 535 129 5 0 1
0 13 74 164 200 287 652 1064 1000 515 122 4 0 1
0 13 75 171 220 315 666 1050 972 495 115 3 0 1
0 13 76 178 240 343 680 1036 944 475 108 2 0 1
0 13 77 185 260 371 694 1022 916 455 101 1 0 1
0 13 78 192 280 399 708 1008 888 435 94 0 0 1

5. pakāpes 14 mainīgo polinomi:

0 14 84 178 123 28 525 1716 2478 1974 878 186 7 0 1
0 14 85 186 150 76 567 1716 2436 1926 851 178 6 0 1
0 14 86 194 177 124 609 1716 2394 1878 824 170 5 0 1
0 14 87 202 204 172 651 1716 2352 1830 797 162 4 0 1
0 14 88 210 231 220 693 1716 2310 1782 770 154 3 0 1
0 14 89 218 258 268 735 1716 2268 1734 743 146 2 0 1
0 14 90 226 285 316 777 1716 2226 1686 716 138 1 0 1
0 14 91 234 312 364 819 1716 2184 1638 689 130 0 0 1

5. pakāpes 15 mainīgo polinomi:

0 15 104 271 280 51 400 2163 4272 4605 2952 1085 184 1 0 1
0 15 105 280 315 126 490 2205 4230 4515 2877 1050 175 0 0 1

2. pielikums. Kirkmana skolnieču modificētā uzdevuma atrisinājuma galvenās funkcijas pirmkods

```
/*
Kirkmana skolnieču problēmas atrisinājuma atrašana, sākuma pozīciju
ģenerējot ar gadījumskaitļu režģi seed.
Ir nepieciešams izveidot group grupas no groupSize skolniecēm katrā no days
dienām,lai katra no girls skolniecēm ar katru citu skolnieci tiktos precīzi
frequency reizes.
Samainot vietām divas meitenes no vienas dienas, šo maiņu nedrīkst atkārtot
nākamo tabuTenture gājienu laikā.
expandedOutput parāda, vai ir niepieciešams atrastā risinājuma aprakstu
saglabāt īpašā veidā priekš talākas izmantošanas 15 mainīgo 5. pakāpes
polinoma meklēšanai.
Rezultātus ieraksta failā filename.
*/
bool tabuSearch(int seed, int days, int girls, int groups, int groupSize,
int frequency, int tabuTenture, char* filename, bool expandedOutput)
{
    // Nomēram risinājuma meklēšanas sākuma laiku.
    clock_t end, start = clock();
    // Turpmāk mainīgo vārdi nozīmēs:
    // "schoolgirl A" - skolniece ar vārdu A,
    // "girl A" - skolniece ar kārtas numuru A.
    int differentGroups = cnk[girls][groupSize];

    int* uniqueGroups = new int[differentGroups];
    int* tabu = new int[days * girls * girls];
    int* pairs = new int[girls * girls];
    memset(uniqueGroups, 0, differentGroups * sizeof(int));
    memset(tabu, 0, days * girls * girls * sizeof(int));
    memset(pairs, 0, girls * girls * sizeof(int));
    int* solution = new int[days * girls];
    int* temporaryGroup = new int[groupSize];

    // Izveido skolnieču izvietojuma sākuma stāvokli.
    srand(seed);
    for (int i = 0; i < days; ++i)
        generateRandomSequence(&solution[i * girls], girls);

    // Katrām divām no skolniecēm vienādā grupā katrā no dienām atzīmē, ka
tās atrodas vienā grupā.
    int currentScore = 0, bestScore, schoolgirlA, schoolgirlB;
    for (int day = 0; day < days; ++day)
        for (int group = 0; group < groups; ++group)
            for (int girlA = 0; girlA < groupSize; ++girlA)
                for (int girlB = girlA + 1; girlB < groupSize; ++girlB)
                {
                    schoolgirlA = solution[day * girls + group * groupSize
+ girlA];
                    schoolgirlB = solution[day * girls + group * groupSize
+ girlB];
                    if (schoolgirlA > schoolgirlB) swap(schoolgirlA,
schoolgirlB);
                    ++pairs[schoolgirlA * girls +  schoolgirlB];
                }
    // Izmantojot informāciju par to, cik reizes tiekas katras divas no
skolniecēm, izrēķina sākotnējo stāvokļa novērtējumu.
    for (int schoolgirlA = 0; schoolgirlA < girls; ++schoolgirlA)
```

```
        for (int schoolgirlB = schoolgirlA + 1; schoolgirlB < girls;
++schoolgirlB)
        {
            pairs[schoolgirlA * girls + schoolgirlB] -= frequency;
            currentScore += abs(pairs[schoolgirlA * girls + schoolgirlB]);
        }
    // Katrai grupai katrā no dienām izrēķina šīs skolnieču kombinācijas
kārtas numuru starp visām kombinācijām no girls elementiem pa groupSize
izvēlēm.
    for (int day = 0; day < days; ++day)
        for (int group = 0; group < groups; ++group)
        {
            for (int id = 0; id < groupSize; ++id)
            {
                temporaryGroup[id] = solution[day * girls + group *
groupSize + id];
            }
            sort(temporaryGroup, temporaryGroup + groupSize);
            int num = computeIDFromCombination(temporaryGroup, groupSize,
girls);
            // Atzīmē, ka šāda grupa ir sastopama.
            ++uniqueGroups[num];
            // Ja tā ir sastopama vairāk par vienu reizi, palielina
pozīcijas novērtējumu.
            if (uniqueGroups[num] > 1)
                currentScore += SAME_GROUP_PENALTY;
        }

    // Statistikas nolūkos atceras visu laiku labāko novērtējumu.
    bestScore = currentScore;
    int candidateScore, swapGirlA, swapGirlB, swapDay, swapScore,
tabuStatus;
    // Pirms algoritma izpilde apstājas ar neveiksmi, izpildas
MAX_ITERATIONS (10000) soļus.
    for (int iterations = 1; iterations < MAX_ITERATIONS; ++iterations)
    {
        // Pārbauda, vai nav sasniegts līdz šim labākais rezultāts.
        bestScore = min(currentScore, bestScore);
        // Ja atrasta atbilde - apstājas.
        if (currentScore == 0) break;

        // Šajā solī labākā atrastā pozīcija ir (-1) jeb nav atrasta
nekāda.
        candidateScore = EPS;
        // Katrām divām skolniecēm ar kārtas numuriem girlA un girlB -
mēģina tās samainīt.
        for (int girlA = 0; girlA < girls; ++girlA)
            for (int girlB = girlA + 1; girlB < girls; ++girlB)
                // Katru skolnieču pāri iespējams samainīt jebkurā no
dienām.
                for (int day = 0; day < days; ++day)
                {
                    // Ja skolnieces atrodas tajā pašā grupā, tad mainīt
tās ir bezjēdzīgi.
                    if (girlA / groupSize == girlB / groupSize) continue;
                    schoolgirlA = solution[day * girls + girlA];
                    schoolgirlB = solution[day * girls + girlB];
                    // Atrod šo skolnieču vārdus un pārbauda, vai tās
pašlaik šajā dienā drīkst mainīt.
                    if (schoolgirlA > schoolgirlB) swap(schoolgirlA,
schoolgirlB);
                    tabuStatus = tabu[day * girls * girls + schoolgirlA *
girls + schoolgirlB];
```

```
                    // Ja nedrīkst - tālāk izpildi neturpina.
                    if (tabuStatus != 0 && tabuStatus + tabuTenture >
iterations) continue;
                    // Izsauc funkciju, kas aprēķina tekošā novērtējuma
izmaiņas, ja apmaina šīs divas skolnieces.
                    swapScore = evaluateChange(&solution[day * girls],
pairs, groups, groupSize, girls, girlA, girlB, uniqueGroups,
temporaryGroup);
                    // Ja atrasta pozīcija, kas ir labāka par līdz šim šajā
solī labāko iespējamo - saglabā informāciju par jauno pozīciju.
                    if (currentScore + swapScore < candidateScore ||
candidateScore == EPS)
                    {
                        candidateScore = currentScore + swapScore;
                        swapGirlA = girlA;
                        swapGirlB = girlB;
                        swapDay = day;
                    }
                }
        // Atrod informāciju par šajā solī visperspektīvāko skolnieču
maiņu.
        schoolgirlA = solution[swapDay * girls + swapGirlA];
        schoolgirlB = solution[swapDay * girls + swapGirlB];
        if (schoolgirlA > schoolgirlB) swap(schoolgirlA, schoolgirlB);
        // Saglabā informāciju, ka tās šajā dienā nedrīkstēt mainīt
turpmākos tabuTenture gājienos.
        tabu[swapDay * girls * girls + schoolgirlA * girls + schoolgirlB] =
iterations;
        // Izsauc funkciju, kas samaina šīs skolnices.
        swapGirls(&solution[swapDay * girls], pairs, groups, groupSize,
girls, swapGirlA, swapGirlB, uniqueGroups, temporaryGroup);
        // Atjauno tekošās pozīcijas novērtējumu.
        currentScore = candidateScore;
    }

    bool res;
    if (currentScore != 0)
    {
        // Risinājums netika atrasts.
        res = false;
        printf("Risinajums netika atrasts (bestScore = %d).\n", bestScore);
    } else {
        res = true;
        printf("Risinajums atrasts.\n");
        // Nomēram risinājuma atrašanas beigu laiku un parādām starpību ar
sākuma laiku.
        end = clock();
        printf("Pateretais laiks: %.3f\n", ((double)(end - start)) /
CLOCKS_PER_SEC);
        // Saglabājam skolnieču izkārtojumu failā cilvēciskā formātā.
        FILE* fout = fopen(filename, "w");
        fprintf(fout, "Seed = %d; Groups = %d; GroupSize = %d; Tabu =
%d.\n", seed, groups, groupSize, tabuTenture);
        for (int day = 0; day < days; ++day)
        {
            fprintf(fout, "%d. diena: ", day + 1);
            for (int group = 0; group < groups; ++group)
            {
                fprintf(fout,"{");
                for (int id = 0; id < groupSize; ++id)
                {
                    fprintf(fout, "%d", solution[day * girls + group *
groupSize + id] + 1);
```

```
                    if (id != groupSize - 1) fprintf(fout, " ");
                }
                fprintf(fout, "}");
            }
            fprintf(fout, "\n");
            if (day == days - 1) fprintf(fout, "\n");
        }

        if (expandedOutput)
        {
            // Saglabājam skolnieču izkārtojumu failā C++ masīva veidā, kas
apraksta šo pozīciju.
            vector<vector<int> > kirkman;
            vector<int> singleGroup;
            for (int day = 0; day < days; ++day)
            {
                for (int group = 0; group < groups; ++group)
                {
                    // Katras grupas ietvaros skolnieces sarindojam to
kārtas numuru pieaugšanas secībā.
                    sort(solution + day * girls + group * groupSize,
solution + day * girls + group * groupSize + groupSize);
                    singleGroup.clear();
                    for (int id = 0; id < groupSize; ++id)
                        singleGroup.push_back(solution[day * girls + group
* groupSize + id]);
                    kirkman.push_back(singleGroup);
                }
            }

            // Katru no skolnieču grupām arī sakārtojam to pieaugšanas
secībā.
            sort(kirkman.begin(), kirkman.end());
            // Kā arī saglabājam atrasto pozīciju kā skolnieču grupu
kombināciju numurus.
            int combinationID;
            fprintf(fout, "%d", days * groups);
            kirkmanSufficientCombinations[groupSize - 3] = days * groups;
            memset(kirkmanCombinationIsSufficient[groupSize - 3], false,
3003 * sizeof(bool));
            int* singleGroupForID = new int[groupSize];
            for (int i = 0; i < kirkman.size(); ++i)
            {
                for (int j = 0; j < kirkman[i].size(); ++j)
singleGroupForID[j] = kirkman[i][j];
                combinationID = computeIDFromCombination(singleGroupForID,
groupSize, 15);
                fprintf(fout, " %d", combinationID);
                kirkmanCombinationIsSufficient[groupSize -
3][combinationID] = true;
            }
            delete[] singleGroupForID;
        }
        fclose(fout);
    }
    // Atbrīvojam izdalīto atmiņu.
    delete[] pairs;
    delete[] tabu;
    delete[] solution;
    delete[] temporaryGroup;
    delete[] uniqueGroups;
    return res;
}
```

3. pielikums. Kirkmana skolnieču modificētā uzdevuma atrisinājums 15 skolniecēm, kas pastaigājas grupās pa 3

15 skolnieces 35 dienas pēc kārtas pastaigājas 5 grupās pa 3 tā, lai katras divas no tām atrastos kopējā grupā tieši 5 dažādās dienās. Visu dienu laikā nekādas divas grupas nedrīkst būt pilnībā vienādas.

1. diena: {13 4 1}{10 7 15}{8 3 5}{6 9 2}{12 14 11}
2. diena: {7 5 10}{6 2 3}{14 11 1}{9 13 15}{12 8 4}
3. diena: {13 8 10}{15 11 1}{2 9 4}{5 3 14}{7 6 12}
4. diena: {11 7 2}{4 5 14}{9 10 8}{12 1 3}{13 15 6}
5. diena: {1 9 5}{8 13 3}{11 4 2}{14 15 6}{7 12 10}
6. diena: {11 3 7}{4 14 13}{10 1 6}{2 12 8}{5 15 9}
7. diena: {10 12 15}{5 11 3}{14 4 6}{13 7 8}{1 9 2}
8. diena: {10 9 14}{12 8 6}{2 4 5}{7 11 1}{13 3 15}
9. diena: {1 14 3}{2 13 12}{15 11 5}{9 8 6}{10 4 7}
10. diena: {9 15 14}{4 12 5}{1 6 13}{2 11 8}{7 3 10}
11. diena: {15 1 6}{8 7 14}{11 13 5}{9 3 12}{2 10 4}
12. diena: {13 8 11}{1 15 2}{7 12 5}{9 4 3}{14 6 10}
13. diena: {13 7 15}{9 3 6}{12 10 1}{4 11 5}{2 14 8}
14. diena: {12 6 11}{15 10 8}{3 2 4}{5 13 7}{14 9 1}
15. diena: {8 5 12}{3 15 11}{6 2 13}{7 9 14}{4 10 1}
16. diena: {12 13 14}{8 1 3}{4 15 9}{11 10 5}{7 6 2}
17. diena: {8 6 2}{4 1 7}{11 14 3}{5 12 13}{10 9 15}
18. diena: {7 13 9}{4 8 6}{12 1 5}{10 11 14}{2 3 15}
19. diena: {12 4 6}{7 2 1}{9 8 11}{10 13 3}{14 5 15}
20. diena: {11 8 10}{2 13 5}{9 3 1}{6 12 14}{4 7 15}
21. diena: {14 6 7}{1 2 11}{5 8 10}{9 4 13}{3 15 12}
22. diena: {3 7 8}{9 5 6}{14 11 2}{15 12 4}{1 13 10}
23. diena: {7 11 4}{10 2 9}{1 12 15}{3 6 13}{14 5 8}
24. diena: {4 15 3}{12 11 9}{8 14 13}{5 6 7}{2 1 10}
25. diena: {11 15 12}{5 10 6}{8 4 1}{9 14 3}{7 2 13}
26. diena: {4 6 13}{10 3 11}{12 7 14}{15 8 1}{5 2 9}
27. diena: {13 14 1}{9 8 7}{15 10 6}{2 5 3}{11 4 12}
28. diena: {2 7 15}{10 4 14}{6 5 3}{1 11 13}{12 8 9}
29. diena: {1 12 9}{5 7 15}{10 11 6}{8 4 3}{13 2 14}
30. diena: {13 10 9}{5 1 8}{2 3 12}{15 11 6}{4 7 14}
31. diena: {11 13 4}{14 10 3}{7 9 12}{5 6 1}{2 8 15}
32. diena: {4 5 9}{3 7 1}{10 12 13}{15 14 2}{8 11 6}
33. diena: {3 4 10}{11 9 6}{13 5 15}{2 14 12}{7 1 8}
34. diena: {5 2 10}{3 13 12}{6 1 4}{11 7 9}{15 14 8}
35. diena: {15 8 4}{11 13 9}{1 5 14}{2 10 12}{3 7 6}

4. pielikums. Kirkmana skolnieču modificētā uzdevuma atrisinājums 15 skolniecēm, kas pastaigājas grupās pa 4

15 skolnieces 105 dienas pēc kārtas pastaigājas 3 grupās pa 4 tā, lai katras divas no tām atrastos kopējā grupā tieši 18 dažādās dienās. Visu dienu laikā nekādas divas grupas nedrīkst būt pilnībā vienādas.

1. diena: {4 15 2 5}{10 12 8 11}{1 6 9 13}
2. diena: {9 15 10 6}{2 5 14 11}{1 13 3 7}
3. diena: {6 1 4 15}{11 13 7 9}{3 14 10 8}
4. diena: {11 7 2 5}{13 14 9 10}{8 6 1 12}
5. diena: {1 11 4 12}{13 3 8 5}{10 14 15 6}
6. diena: {3 14 7 6}{11 13 10 1}{4 2 12 8}
7. diena: {1 10 8 4}{11 9 14 5}{6 15 7 13}
8. diena: {10 14 3 4}{8 12 2 7}{5 1 11 6}
9. diena: {5 15 13 2}{11 6 8 3}{9 7 14 4}
10. diena: {1 14 7 15}{3 10 9 6}{13 2 11 8}
11. diena: {2 1 6 5}{7 10 9 8}{15 4 3 12}
12. diena: {12 2 4 1}{15 9 10 13}{11 8 3 5}
13. diena: {10 12 15 5}{8 7 3 13}{1 4 11 9}
14. diena: {10 4 6 15}{5 13 14 12}{7 9 3 8}
15. diena: {3 4 6 1}{5 8 9 2}{15 11 7 14}
16. diena: {12 14 6 13}{1 8 4 15}{9 11 10 7}
17. diena: {8 13 2 15}{9 7 12 14}{3 4 11 6}
18. diena: {2 10 12 6}{11 1 5 4}{7 13 8 14}
19. diena: {15 4 6 14}{2 1 9 3}{11 10 8 13}
20. diena: {11 8 10 14}{6 12 9 3}{1 13 5 2}
21. diena: {3 6 10 1}{7 11 4 9}{13 2 8 5}
22. diena: {9 6 8 10}{4 3 14 11}{2 12 15 5}
23. diena: {2 11 4 10}{5 14 9 12}{1 3 6 13}
24. diena: {12 15 3 14}{4 5 8 13}{11 9 6 7}
25. diena: {14 15 10 1}{12 6 2 11}{7 5 8 4}
26. diena: {7 13 10 5}{14 12 11 6}{3 15 8 1}
27. diena: {3 8 15 9}{7 2 4 10}{14 1 5 6}
28. diena: {2 4 6 10}{7 5 15 13}{3 1 11 8}
29. diena: {5 7 14 15}{12 9 10 11}{6 8 4 3}
30. diena: {10 6 1 7}{4 13 2 14}{12 15 11 3}
31. diena: {1 2 14 11}{13 3 6 9}{12 5 10 4}
32. diena: {4 5 9 3}{7 1 10 12}{13 15 14 2}
33. diena: {3 1 11 15}{9 5 7 4}{12 2 14 10}
34. diena: {13 3 15 4}{9 1 10 12}{2 11 7 6}
35. diena: {4 12 2 9}{13 8 1 7}{14 10 15 11}
36. diena: {3 2 13 9}{11 15 7 10}{4 6 5 8}
37. diena: {3 9 2 7}{4 5 15 6}{1 14 12 8}
38. diena: {3 8 5 9}{4 14 15 2}{7 10 6 13}
39. diena: {7 14 1 9}{8 10 12 6}{11 4 13 3}
40. diena: {4 11 15 13}{1 7 5 12}{8 9 2 14}
41. diena: {1 7 2 11}{15 6 9 12}{13 5 14 3}
42. diena: {8 13 10 2}{9 12 15 14}{4 3 11 7}
43. diena: {1 7 4 14}{13 15 9 5}{8 3 2 6}
44. diena: {11 8 6 4}{1 12 2 3}{9 15 10 5}
45. diena: {3 14 9 15}{6 13 5 4}{8 12 2 10}
46. diena: {4 13 10 1}{2 12 11 9}{8 14 5 6}
47. diena: {1 9 2 5}{12 4 3 7}{11 10 15 6}
48. diena: {15 9 5 4}{1 8 13 14}{3 11 12 7}
49. diena: {8 5 6 12}{13 10 1 2}{15 3 9 11}
50. diena: {13 8 9 4}{5 15 10 14}{1 3 2 6}
51. diena: {5 11 6 10}{4 2 8 9}{13 7 15 12}
52. diena: {3 2 6 14}{1 11 5 15}{13 4 7 9}
53. diena: {3 11 13 5}{15 1 7 2}{12 9 4 10}
54. diena: {7 12 5 8}{2 10 13 3}{6 11 14 1}
55. diena: {1 7 4 6}{14 12 8 3}{5 2 10 11}
56. diena: {15 12 6 5}{8 3 10 9}{14 11 4 1}
57. diena: {5 7 1 3}{12 8 11 15}{2 6 14 10}
58. diena: {11 7 13 15}{2 8 4 1}{5 3 12 10}
59. diena: {14 3 8 5}{6 11 13 4}{2 15 9 7}
60. diena: {13 8 11 12}{3 7 15 1}{2 5 9 10}
61. diena: {6 15 13 12}{5 7 2 3}{11 14 9 8}
62. diena: {4 11 7 14}{6 12 3 5}{9 15 8 1}
63. diena: {2 11 3 14}{9 8 6 15}{12 5 4 13}
64. diena: {4 15 11 5}{12 8 13 2}{7 14 1 10}
65. diena: {12 6 7 5}{9 4 13 2}{8 10 11 1}
66. diena: {9 6 7 12}{2 15 8 4}{13 11 14 1}
67. diena: {10 6 4 13}{12 2 15 3}{8 9 14 5}
68. diena: {10 13 9 3}{15 1 2 14}{7 12 11 4}

69. diena: {1 10 15 3}{12 4 7 2}{6 9 13 14}
70. diena: {7 3 2 1}{14 5 13 4}{12 6 15 11}
71. diena: {6 5 2 14}{10 4 8 3}{12 1 13 9}
72. diena: {5 1 9 12}{13 4 14 3}{10 2 15 8}
73. diena: {12 1 6 5}{11 3 9 2}{14 10 15 13}
74. diena: {14 2 6 7}{15 5 10 1}{11 12 4 3}
75. diena: {2 3 5 4}{13 9 1 10}{15 7 14 6}
76. diena: {13 6 2 4}{10 12 7 8}{11 3 9 1}
77. diena: {10 4 9 15}{14 8 5 13}{2 7 6 3}
78. diena: {3 8 14 4}{5 11 7 10}{12 6 9 1}
79. diena: {12 11 13 14}{6 3 7 5}{4 8 9 1}
80. diena: {12 14 13 4}{9 10 3 5}{7 15 1 8}
81. diena: {11 13 2 6}{5 1 14 9}{3 15 12 8}
82. diena: {14 3 10 7}{1 9 13 11}{12 8 6 15}
83. diena: {10 3 2 14}{12 4 11 5}{9 1 8 6}
84. diena: {6 13 2 7}{10 5 11 8}{1 14 3 12}
85. diena: {10 11 12 13}{2 5 14 1}{15 7 8 6}
86. diena: {10 5 4 7}{12 2 9 14}{8 1 6 11}
87. diena: {15 9 3 7}{2 1 5 12}{8 11 6 13}
88. diena: {12 10 2 13}{14 8 11 7}{3 15 1 5}
89. diena: {5 8 15 7}{9 6 2 4}{14 12 1 10}
90. diena: {13 5 14 7}{9 15 12 11}{1 3 4 10}
91. diena: {13 1 15 4}{9 2 6 7}{3 5 12 11}
92. diena: {9 7 11 5}{2 6 15 10}{8 4 14 1}
93. diena: {2 8 7 15}{5 13 10 3}{14 4 6 9}
94. diena: {10 2 11 3}{9 8 15 13}{6 4 14 7}
95. diena: {14 8 7 2}{11 9 6 13}{1 12 4 10}
96. diena: {9 2 11 14}{1 7 8 10}{12 15 13 3}
97. diena: {3 10 7 5}{2 1 12 13}{8 11 15 14}
98. diena: {7 12 15 4}{13 1 9 5}{6 3 14 10}
99. diena: {12 8 14 9}{3 6 13 15}{7 10 11 2}
100. diena: {2 11 9 15}{12 1 7 13}{8 10 5 4}
101. diena: {3 4 2 15}{13 10 7 12}{5 1 6 9}
102. diena: {6 8 5 11}{7 13 12 3}{4 10 15 14}
103. diena: {13 1 11 15}{7 4 8 12}{14 10 6 5}
104. diena: {2 8 1 15}{14 3 13 12}{10 9 4 7}
105. diena: {2 15 5 11}{12 14 9 4}{13 7 6 8}

5. pielikums. Kirkmana skolnieču modificētā uzdevuma atrisinājums 15 skolniecēm, kas pastaigājas grupās pa 5

15 skolnieces 63 dienas pēc kārtas pastaigājas 2 grupās pa 5 tā, lai katras divas no tām atrastos kopējā grupā tieši 12 dažādās dienās. Visu dienu laikā nekādas divas grupas nedrīkst būt pilnībā vienādas.

1. diena: {4 15 13 1 10}{12 8 9 14 6}
2. diena: {11 3 10 6 2}{9 14 5 1 13}
3. diena: {6 9 4 7 8}{13 5 10 3 14}
4. diena: {12 15 3 6 13}{14 7 11 8 4}
5. diena: {13 7 12 15 4}{3 8 5 2 14}
6. diena: {3 14 7 13 11}{6 10 1 12 2}
7. diena: {13 1 8 4 3}{2 14 5 7 15}
8. diena: {2 3 9 15 8}{11 12 4 5 1}
9. diena: {2 1 10 11 9}{6 8 3 12 7}
10. diena: {3 14 12 10 5}{11 9 6 13 2}
11. diena: {5 11 6 12 7}{10 9 8 1 4}
12. diena: {8 7 11 1 6}{12 14 9 4 2}
13. diena: {8 15 5 13 1}{7 3 10 2 4}
14. diena: {10 8 11 15 5}{13 14 12 7 9}
15. diena: {7 1 6 12 5}{8 9 2 15 11}
16. diena: {7 14 4 3 1}{8 5 15 12 11}
17. diena: {9 1 15 8 3}{6 7 14 11 5}
18. diena: {5 11 15 9 4}{14 2 6 7 13}
19. diena: {12 8 6 5 2}{1 9 3 11 10}
20. diena: {2 4 10 14 6}{11 12 3 1 13}
21. diena: {15 14 3 1 7}{6 4 9 13 2}
22. diena: {7 15 9 1 5}{3 6 11 2 12}
23. diena: {4 11 8 5 13}{2 1 12 9 3}
24. diena: {4 15 3 14 6}{12 7 13 10 9}
25. diena: {8 15 12 2 10}{6 9 14 3 5}
26. diena: {2 4 10 5 14}{12 11 13 3 15}
27. diena: {3 13 15 9 7}{11 4 10 14 1}
28. diena: {9 4 11 12 3}{8 10 13 2 15}
29. diena: {11 4 2 15 3}{1 10 5 12 8}
30. diena: {6 11 1 5 4}{13 2 14 3 15}
31. diena: {2 1 7 8 13}{3 6 9 12 5}
32. diena: {4 11 2 9 7}{6 10 5 13 15}
33. diena: {14 1 11 15 9}{5 8 4 7 2}
34. diena: {6 9 15 8 10}{1 5 12 2 7}
35. diena: {12 5 11 2 13}{9 1 15 6 10}
36. diena: {14 5 13 1 12}{15 7 10 4 6}
37. diena: {8 3 12 7 4}{10 15 13 1 14}
38. diena: {12 9 13 3 4}{5 15 2 7 11}
39. diena: {7 3 1 9 5}{10 12 14 11 15}
40. diena: {14 6 8 13 3}{7 12 10 4 15}
41. diena: {1 7 2 14 15}{6 10 4 13 5}
42. diena: {8 13 7 2 11}{3 1 14 10 6}
43. diena: {6 11 1 4 13}{7 9 14 8 3}
44. diena: {13 10 2 7 1}{5 6 4 9 15}
45. diena: {7 11 9 10 6}{13 14 4 8 1}
46. diena: {6 4 3 1 2}{11 12 10 8 14}
47. diena: {15 14 6 8 12}{13 3 2 11 10}
48. diena: {2 9 12 10 1}{8 13 14 4 11}
49. diena: {5 8 6 9 13}{11 1 12 15 3}
50. diena: {14 9 12 4 8}{15 10 6 7 3}
51. diena: {7 15 6 4 12}{11 8 9 14 2}
52. diena: {8 2 6 14 1}{7 5 10 13 4}
53. diena: {12 11 10 4 14}{1 7 2 5 9}
54. diena: {15 14 5 12 2}{10 13 8 6 9}
55. diena: {1 7 6 11 14}{3 8 10 5 2}
56. diena: {15 12 1 4 8}{5 10 9 14 11}
57. diena: {8 7 10 5 3}{4 14 15 2 6}
58. diena: {11 9 13 15 7}{8 4 1 5 3}
59. diena: {11 8 1 6 15}{12 13 4 2 9}
60. diena: {7 8 11 10 3}{9 15 13 5 14}
61. diena: {3 11 13 6 5}{9 10 12 7 14}
62. diena: {13 7 12 8 10}{2 3 5 4 15}
63. diena: {4 10 3 5 9}{13 6 12 2 1}

6. pielikums. Piektās pakāpes patvaļīga mainīgo skaita polinoma meklēšanas rijīgā algoritma galvenās funkcijas pirmkods

```
/*
5. pakāpes "variables" mainīgo polinoma meklēšana ar rijīgo algoritmu.
reactive nozīmē, ka viena gājiena ietvaros netiek apskatīti visi iespējamie
soļi, bet gan paņemts pirmais sastaptais, kas uzlabo novērtējumu.
evaluationBoundary un partialEvaluation norāda uz funkcijas daļēju
novērtēšanu laika resursu taupības dēļ, tie sīkāk ir aprakstīti
evaluateCoefficients komentāros.
fromLength nosaka, kuras pakāpes monomus mēģināt nomainīt uz citiem pirmām
kārtām, kā arī incLength nosaka, kuras pakāpes monomus pārbaudīt pēc tam.
saveInFile norāda, ka katrā solī atrastās pozīcijas jāsaglabā failā ar
nosaukumu storeFileName.
Masīvs functionStatistics satur viena mainīgā polinomu viena mainīgā
polinoma veidā.
Mainīgais allowedAmountOfFailures nosaka to, pēc cik neveiksmīgiem soļiem,
kad neizdevās uzlabot pozīcijas novērtējumu, pārlases procedūra ir
jāapstādina. (-1 priekš bezgalības)
Mainīgais polynomNumber satur tā viena mainīgā polinoma kārtas numuru, kas
pašlaik kalpo par pamatu sākotnējā polinoma meklēšanai.
*/
void solutionSearch(bool reactive, int evaluationBoundary, bool
partialEvaluation,
                    int fromLength, int incLength, bool saveInFile,
                    int* functionStatistics, int variables, string
storeFileName,
                    string resultFileName, int allowedAmountOfFailures, int
polynomNumber)
{
    // Ja incLength ir 0, tad katru reizi pārbaudīsim visus *patvaļīgas (3,
4 vai 5) pakāpes* monomus, kurus mēģināsim aizvietot ar citu monomu
novērtējuma uzlabošanai.
    bool randomLength = (incLength == 0);
    // Mainīgie, kuri ir nepieciešami varbūtisku izvēļu izdarīšanai.
    int currentCombinationID[3] = {0, 0, 0};
    int combinationIDInc[3] = {1, 1, 1};
    int queuedCombinations[3];
    // Mainīgie priekš novērtējumu glabāšanas.
    pair<int, int> score, possibleScore, bestScore;
    int victimID, predatorID, replaceCombinationLength;
    bool needReaction;
    int evenScores;
    // Patērētā laika mērīšana
    clock_t start, end;

    // Atrod sākuma pozīcijas novērtējumu..
    score = evaluateCoefficients(make_pair(EPS, EPS), evaluationBoundary,
partialEvaluation, functionStatistics, variables);
    // Saglabā neapskatīto monomu skaitu, kurus iespējams vērts nomainīt uz
citiem monomiem.
    for (int i = 0; i < 3; ++i) queuedCombinations[i] =
cnk[variables][i+3];
    // Bezgalīgi ilgi izpildām kārtējo soli, mēģinot uzlabot pašreizējās
pozīcijas novērtējumu.
    for (int iterations = 1; score.first != 0 || score.second != 0;
++iterations)
    {
        start = clock();
```

```
        // Pārbaudām, ka tekošā polinoma reprezentācija tiešām satur
nepieciešamo monomu skaitu.
        for (int combinationLength = 0; combinationLength < 3;
++combinationLength)
        {
            int counter = 0;
            for (int combinationID = 0; combinationID <
cnk[variables][combinationLength + 3]; ++combinationID)
                if
(combinationIsSufficient[combinationLength][combinationID])
                    ++counter;
            if (counter != sufficientCombinations[combinationLength])
system("pause");
        }
        // Pašlaik nav atrasta nekāda potenciālā jaunā pozīcija
        bestScore = make_pair(EPS, EPS);
        needReaction = false;
        // Ja jāizvēlās patvaļīgas pakāpes monomu apstrādi, darām to
varbūtiski.
        if (randomLength)
        {
            int alwaysSomeChance = rand() % 50;
            if (alwaysSomeChance == 0) fromLength = 3; else
            if (alwaysSomeChance == 1) fromLength = 4; else
            if (alwaysSomeChance == 2) fromLength = 5; else
            {
                // Katras pakāpes līdz šim necaurskatīto monomu skaits
ietekmē šīs pakāpes izvēli.
                int dice = rand() % (3 + queuedCombinations[0] +
queuedCombinations[1] + queuedCombinations[2]);
                if (dice < 1 + queuedCombinations[0]) fromLength = 3; else
                if (dice < 2 + queuedCombinations[0] +
queuedCombinations[1]) fromLength = 4;
                else fromLength = 5;
            }
            // Izvēloties pirmo apstrādes pakāpi, turpmāk secīgi
izvēlēsimies blakus pakāpi.
            if (fromLength == 3) incLength = 1; else
            if (fromLength == 5) incLength = -1; else
            if (queuedCombinations[0] < queuedCombinations[2]) incLength =
1;
            else incLength = -1;
        }
        // Iesākam ar pakāpes fromLength monomu apskati, tad secīgi
palielinām/samazinām šo pakāpi.
        for (int combinationLength = fromLength - 3; !needReaction;
combinationLength += incLength)
        {
            evenScores = 0;
            // Secīgi skatāmies visus konkrētās pakāpes monomus.
            // Ja pakāpe jau ir par mazu vai par lielu - apstājamies un
pielietojam uz šo brīdi labāko pozīciju.
            if (combinationLength < 0 || combinationLength > 2) break;
            // Ja visi šīs pakāpes monomi jau ir izskatīti.
            if (queuedCombinations[combinationLength] == 0)
            {
                // Sākam tos izskatīt pilnīgi no jauna.
                queuedCombinations[combinationLength] =
cnk[variables][combinationLength + 3];
                if (randomLength)
                {
                    // Varbūtiski izvēlamies kārtību, kurā apskatīt visus
dotās pakāpes monomus.
```

```
                    if (rand() % 2 == 0)
                    {
                        currentCombinationID[combinationLength] = 0;
                        combinationIDInc[combinationLength] = 1;
                    }
                    else
                    {
                        currentCombinationID[combinationLength] =
cnk[variables][combinationLength + 3] - 1;
                        combinationIDInc[combinationLength] = - 1;
                    }
                }
                else
                {
                    currentCombinationID[combinationLength] = 0;
                    combinationIDInc[combinationLength] = 1;
                }
            }
            // Kāmer nav nepieciešams apstāties un pielietot atrasto
izdevīgo pozīciju:
            while (!needReaction)
            {
                // Ja visi monomi jau izskatīti - izejam no cikla.
                if (currentCombinationID[combinationLength] == -1 ||
currentCombinationID[combinationLength] == cnk[variables][combinationLength
+ 3]) break;
                // Atzīmējam, ka tūlīt izskatīsim vēl vienu monomu.
                --queuedCombinations[combinationLength];
                printf("combinationLength = %d, combinationID = %d.    \r",
combinationLength + 3, currentCombinationID[combinationLength]);
                // Ja šis monoms vēl nav paņemts tekošajā polinoma
kombinatoriskajā reprezentācijā:
                if
(!combinationIsSufficient[combinationLength][currentCombinationID[combinati
onLength]])
                    // Skrienam pāri visiem izvēlētiem šīs pakāpe monomiem.
                    for (int victimCombinationID = 0; !needReaction &&
victimCombinationID < cnk[variables][combinationLength + 3];
++victimCombinationID)
                        if
(combinationIsSufficient[combinationLength][victimCombinationID])
                        {
                            // Mēģinam kādu no reprezentācijas monomiem
nomainīt ar tekošo monomu-kandidātu.

combinationIsSufficient[combinationLength][victimCombinationID] = false;

combinationIsSufficient[combinationLength][currentCombinationID[combination
Length]] = true;
                            // Novērtējam šīs reprezentācijas novērtējumu.
                            possibleScore = evaluateCoefficients(bestScore,
evaluationBoundary, partialEvaluation, functionStatistics, variables);
                            // Ja novērtējums ir labāks par iepriekšējo -
saglabājam to kā labāko iespējamo.
                            if (bestScore.first == EPS && bestScore.second
== EPS || possibleScore < bestScore)
                            {
                                if (possibleScore == score) ++evenScores;
                                // Ja labākā atrastā pozīcija ir at tādu
pašu novērtējumu kā pašreizējās pozīcijas novērtējums, tad šo pozīciju
saglabājam ar varbūtību 1/(šajā solī atrasto pozīciju skaits ar šādu pašu
novērtējumu).
```

```
                                if (possibleScore != score || rand() %
evenScores == 0)
                                {
                                    bestScore = possibleScore;
                                    replaceCombinationLength =
combinationLength;
                                    victimID = victimCombinationID;
                                    predatorID =
currentCombinationID[combinationLength];
                                    // Ja ir atzīmēts, ka algoritms ir
reaktīvs, atrodot labāku pozīciju par tekošo, uzreiz izejam no cikla un
pārejam uz šo pozīciju.
                                    if (reactive && bestScore < score)
needReaction = true;
                                }
                            }
                            // Atgriežamies pie reprezentācijas monomu
sākuma pozīcijas.

combinationIsSufficient[combinationLength][victimCombinationID] = true;

combinationIsSufficient[combinationLength][currentCombinationID[combination
Length]] = false;
                        }
                // Pamainām tekošās monoma jeb combinationLength vieninieku
kombinācijas no variables izvēlēm kārtas numuru
                currentCombinationID[combinationLength] +=
combinationIDInc[combinationLength];
            }
        }
        // Mainām tekošo pozīciju uz tikko atrasto labāko iespējamo
pozīciju.
        if (score == bestScore)
        {
            if (allowedAmountOfFailures > 0) --allowedAmountOfFailures;
            // Ja sasniegts neveiksmīgo soļu limits, tad procedūras izpilde
beidzas.
            if (allowedAmountOfFailures == 0) break;
        }
        if (bestScore.first == EPS || bestScore.second == EPS) break;
        score = bestScore;
        combinationIsSufficient[replaceCombinationLength][victimID] =
false;
        combinationIsSufficient[replaceCombinationLength][predatorID] =
true;
        printf("Score: {%d;%d}. Position numbers (%d): %d -> %d.
\n", score.first, score.second, replaceCombinationLength + 3, victimID,
predatorID);
        if (saveInFile)
        {
            // Saglabājam tikko iegūto pozīciju failā.
            FILE* fin = fopen(storeFileName.c_str(), "a");
            printPosition(fin, variables);
            fclose(fin);
        }
        end = clock();
        printf("Pateretais laiks vienam solim: %.3f\n", ((double)(end -
start)) / CLOCKS_PER_SEC);
    }
    if (saveInFile)
    {
        // Polinoms ir atrasts, saglabājam to atbildes failā.
        FILE* fin = fopen(resultFileName.c_str(), "a");
```

```
        fprintf(fin, "Polynom number = %d, Score = {%d,%d}.\n",
polynomNumber, score.first, score.second);
        printf("Variables = %d, done with score {%d, %d}.
\n", variables, score.first, score.second);
        if (score.first == 0 && score.second == 0) printPosition(fin,
variables);
        fclose(fin);
    }
}
```

7. pielikums. Piektās pakāpes patvaļīga mainīgo skaita polinoma pozīcijas novērtējuma funkcijas pirmkods

```
/*
Polinoma kombinatorās reprezentācijas novērtējuma noteikšana.
maxProfitableScore norāda, ka, ja novērtējums paliek lielāks par šo
vērtību, tad var vairs neturpināt - šī pozīcija mūs neinteresēs.
evaluationBoundary norāda, ka ir vēlēšanās pārbaudīt tikai visus funkcijas
ievaddatus ar 0 vieninieku skaitu līdz evaluationBoundary vieninieku
skaitam, bet tālāk vairs netuprināt.
partialEvaluation norāda, ka ir vēlēšānās pārbaudīr tikai visus funkcijas
ievaddatus ar 0 vieninieku skaitu līdz 5 vieninieku skaitam un tad
atsevišķi arī evaluationBoundary vieninieku skaitu.
Masīvs functionStatistics satur viena mainīgā polinomu.
Tiek vērtēts "variables" mainīgo 5. pakāpes polinoms.
*/
pair<int, int> evaluateCoefficients(pair<int, int> maxProfitableScore, int
evaluationBoundary, bool partialEvaluation, int* functionStatistics, int
variables)
{
    // Mainīgo inicializācija.
    int functionResults[2], combination[15], subCombination[5],
tmpCombination[5];
    pair<int, int> score = make_pair(0, 0);
    int functionResult, indexIndent, subCombinationID,
subCombinationNumber;

    // Ja nav ierobežojumu uz novērtējumu, kad jāapstājas - uzliekam to
nesasniedzami lielu.
    if (maxProfitableScore.first == EPS && maxProfitableScore.second ==
EPS) maxProfitableScore = make_pair(INF, INF);
    // Katram no vieninieku skaitam ievaddatu virknē.
    for (int combinationLength = 3; score < maxProfitableScore &&
combinationLength <= evaluationBoundary; ++combinationLength)
    {
        // Ja funkcijas parametri uz to norāda, tad pārlecam uz
evaluationBoundary vieninieka skaita novērtējuma izveidi.
        if (combinationLength == 6 && partialEvaluation) combinationLength
= evaluationBoundary;
        // Uz šo brīdi atrastas 0 ievaddatu virknes ar funkcijas vērtību 0
vai 1.
        functionResults[0] = functionResults[1] = 0;
        // Sākuma ievaddatu virkne secīgi saturēs combinationLength pēc
kārtas vieniniekus.
        for (int i = 0; i < combinationLength; ++i) combination[i] = i;
        // Katrai no ievaddatu virknēm pie dotā vieninieku skaita -
atradīsim funkcijas rezultātu un atjaunosim statistiku.
        for (int combinationID = 0; combinationID <
cnk[variables][combinationLength]; ++combinationID)
        {
            // Pie jebkuras ievaddatu virknes, iepriekš zinām 1. un 2.
pakāpes aktīvo monomu koeficientu summu.
            functionResult = cnk[combinationLength][1] -
cnk[combinationLength][2];
            // Lai uzzinātu tekošās ievaddatu virknes rezultātu, jāsaskaita
visu tādu monomu koeficienti, kurus ietver šī ievaddatu kombinācija.. mūs
interesē monomi ar pakāpēm 3, 4 un 5
            for (int subCombinationLength = 3; subCombinationLength <
min(combinationLength, 6); ++subCombinationLength)
```

```
            {
                memcpy(subCombination, zeroCombination, 5*sizeof(int));
                indexIndent = cnkSum[subCombinationLength];
                // Pārlasām tekošās ievaddatu virknes visas iespējamās
subCombinationLength garas kombinācijas no vienīnieka bitu apakškopām
                for (int subCombinationNumber = 0; subCombinationNumber <
cnk[combinationLength][subCombinationLength]; ++subCombinationNumber)
                {
                    // Masīvā apskatāmies numuru tekošai kombinācijai no
vienīnieka bitu apakškopas.
                    // Glabājam to masīvā, jo šis ir programmas kritiskais
brīdis, kurā nepieciešams pēc iespējas labāk optimizēt patērējamos
skaitļošanas resursus.
                    if (subCombinationLength == 3)
                        subCombinationID =
combinationID3[combination[subCombination[0]]][combination[subCombination[1
]]][combination[subCombination[2]]];
                    else if (subCombinationLength == 4)
                        subCombinationID =
combinationID4[combination[subCombination[0]]][combination[subCombination[1
]]][combination[subCombination[2]]][combination[subCombination[3]]];
                    else
                        subCombinationID =
combinationID5[combination[subCombination[0]]][combination[subCombination[1
]]][combination[subCombination[2]]][combination[subCombination[3]]][combina
tion[subCombination[4]]];

                    // Pašreizējiem ievaddatiem pieskaitām atrastā monoma
koeficientu.
                    functionResult += polynomCoefficient[indexIndent +
subCombinationID];

                    // Ar iepriekš aprēķinātu masīvu palīdzību atrodam
nākamo veidu, ka paņemt kombināciju no ievaddatu vienīnieku vitu
apakškopām.
                    memcpy(tmpCombination, subCombination, 5*sizeof(int));
                    if (subCombinationLength == 3)
                    {
                        subCombination[0] =
nextCombination3[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][0];
                        subCombination[1] =
nextCombination3[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][1];
                        subCombination[2] =
nextCombination3[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][2];
                    }
                    else if (subCombinationLength == 4)
                    {
                        subCombination[0] =
nextCombination4[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][tmpCombination[
3]][0];
                        subCombination[1] =
nextCombination4[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][tmpCombination[
3]][1];
                        subCombination[2] =
nextCombination4[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][tmpCombination[
3]][2];
```

```
                        subCombination[3] =
nextCombination4[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][tmpCombination[
3]][3];
                    }
                    else
                    {
                        subCombination[0] =
nextCombination5[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][tmpCombination[
3]][tmpCombination[4]][0];
                        subCombination[1] =
nextCombination5[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][tmpCombination[
3]][tmpCombination[4]][1];
                        subCombination[2] =
nextCombination5[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][tmpCombination[
3]][tmpCombination[4]][2];
                        subCombination[3] =
nextCombination5[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][tmpCombination[
3]][tmpCombination[4]][3];
                        subCombination[4] =
nextCombination5[combinationLength -
3][tmpCombination[0]][tmpCombination[1]][tmpCombination[2]][tmpCombination[
3]][tmpCombination[4]][4];
                    }
                }
            }
            // Ja pašlaik apskatām ievaddatus ar mazāk kā 6 vieninieka
bitiem, tad polinoma kombinatorizkās reprezentācijas masīvā jāatrod, vai
šai ievaddatu virknei ir jāatbilst funkcijas nulles vai vieninieka
vērtībai.
            if (combinationLength < 6)
            {
                // Vai šī virkne ir pieminēta reprezentācijas masīvā.
                bool isSufficient =
combinationIsSufficient[combinationLength - 3][combinationID];
                // Vai nu šim bitu skaitam reprezentācijas masīvs apraksta
vieninieka funkcijas rezultātus un šī virkne ir pieminēta, vai arī
reprezentācijas masīvs apraksta nuļļu funkcijas rezultātus un šī virkne nav
pieminēta -- tad šī ievaddatu virkne atbilst vieninieka funkcijas
rezultātam.
                if (isSufficient && functionStatistics[combinationLength]
== sufficientCombinations[combinationLength - 3]
                || !isSufficient && functionStatistics[variables -
combinationLength] == sufficientCombinations[combinationLength - 3])
                {
                    polynomCoefficient[cnkSum[combinationLength] +
combinationID] = 1 - functionResult;
                    ++functionResults[1];
                }
                // citādāk - nē:
                else
                {
                    polynomCoefficient[cnkSum[combinationLength] +
combinationID] = 0 - functionResult;
                    ++functionResults[0];
                }
            }
            else
            {
```

```
                // Jau zinām, kāda ir funkcijas vērtība šai ievaddatu
virknei, un atjaunojam glabāto statistiku par šīm vērtībām.
                // Iespējams, palielinām novērtējumu - ja funkcijas vērtība
nav 0 vai 1.
                if (functionResult == 0) ++functionResults[0];
                else if (functionResult == 1) ++functionResults[1];
                else score.first += 1;
            }
            // Ģenerējam nākamo ievaddatu virkni ar tādu pašu vieninieku
skaitu.
            nextCombination(combination, combinationLength, variables);
        }
        // Pieskaitām novērtējumam tik punktu, cik stipri iegūtā funkcijas
vērtību statistika atšķirās no vēlamās.
        score.second += abs(functionResults[0] -
functionStatistics[variables - combinationLength]);
        score.second += abs(functionResults[1] -
functionStatistics[combinationLength]);
    }
    // Atgriežam pozīcijas novērtējumu.
    return score;
}
```

DOKUMENTĀRĀ LAPA

Bakalaura darbs

„Zemu polinoma pakāpju Būla funkciju vaicājumu sarežģītība”

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai. Piekrītu sava darba publicēšanai internetā.

Autors: _____
(Autora paraksts)

Ar savu parakstu apliecinu, ka esmu lasījis augšminēto bakalaura darbu un atzīstu to par **piemērotu/nepiemērotu** (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu bakalaura studiju programmas gala pārbaudījuma komisijas sēdē.

Darba vadītājs(-ja): _____
(Vadītāja paraksts)

Darbs iesniegts Datorikas fakultātē _____.
(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.
Metodiķe: _____.
(Metodiķes paraksts)

Recenzents: _____

Darbs aizstāvēts bakalaura darbu gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____, vērtējums _____
(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____
(Sekretāra paraksts)