ՀՀ ԳԱԱ ԻՆՖՈՐՄԱՏԻԿԱՅԻ ԵՎ ԱՎՏՈՄԱՏԱՑՄԱՆ ՊՐՈԲԼԵՄՆԵՐԻ ԻՆՍՏԻՏՈՒՏ

Նահապետյան Հայկ Էդվարդի

ԻՆՔՆԱԿԱԶՄԱԿԵՐՊՎՈՂ ՀԱՄԱԿԱՐԳԵՐՈՒՄ ԴԻՆԱՄԻԿ ՊՐՈՑԵՍՆԵՐԻ ՀԵՏԱԶՈՏՄԱՆ ԳՈՐԾԻՔԱՅԻՆ ԾՐԱԳՐԱՅԻՆ ՄԻՋՈՑՆԵՐԻ ՄՇԱԿՈՒՄ

Ատենախոսություն

Ե.13.04 – «Հաշվողական մեքենաների, համալիրների, համակարգերի և ցանցերի մաթեմատիկական և ծրագրային ապահովում» մասնագիտությամբ տեխնիկական գիտությունների թեկնածուի գիտական աստիճանի համար

Գիտական ղեկավար՝ տեխ.գիտ.դոկտոր
Յու.Հ. Շուքուրյան

Երևան – 2018

Բովանդակություն

ԱՌԱՋԱԲԱՆ 4

ԳԼՈՒԽ 1. ԻՆՔՆԱԿԱԶՄԱԿԵՐՊՎՈՂ ՀԱՄԱԿԱՐԳԵՐԻ և ԴՐԱՆՑ ՀԵՏԱԶՈՏՄԱՆ ՆՊԱՏԱԿՈՎ ՆԱԽԱԳԾՎԱԾ ԾՐԱԳՐԱՅԻՆ ԳՈՐԾԻՔԱՄԻՋՈՑՆԵՐԻ ՎԵՐԼՈՒԾՈՒԹՅՈՒՆ ...11

1.1 Ինքնակազմակերպվող համակարգեր, ավազակույտի և rotor-router մոդելներ 12

1.2. Ինքնակազմակերպվող համակարգերի հետազոտման արդի ծրագրային գործիքամիջոցների վերլուծությունը 18

1.3 «Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդիրը և գոյություն ունեցող լուծումները 26

1.4 Ավազակույտի մոդելի վրա կառուցված կլաստերային գոյություն ունեցող համակարգերի վերլուծությունը 27

Եզրակացություն առաջին գլխի վերաբերյալ 28

ԳԼՈՒԽ 2. ԱՎԱԶԱԿՈՒՅՏԻ ՄՈԴԵԼԻ ԱՇԽԱՏԱՆՔԻ ԶՈՒԳԱՀԵՌԱՑՄԱՆ ՄԱԹԵՄԱՏԻԿԱԿԱՆ ՄՈԴԵԼԻ ԱՌԱՋԱԴՐՈՒՄԸ 30

2.1. Խորանարդային ցանցերում աստղային ծածկույթները նկարագրող բանաձևը 30

2.2. Ավազահատիկների քանակի ստորին գնահատականը փակ եզրերով եռաչափ ցանցում անվերջ անկայուն վիճակի հասնելու համար: 35

2.3. Rotor-router համակարգի առանձնահատկությունները 37

Եզրակացություն երկրորդ գլխի վերաբերյալ 38

ԳԼՈՒԽ 3. ՄՇԱԿՎԱԾ ԾՐԱԳՐԱՅԻՆ ՓԱԹԵԹՆԵՐԻ ՆԿԱՐԱԳՐՈՒԹՅՈՒՆԸ 39

3.1. Բազմաօգտատեր «CA Simulator» համակարգի նկարագրությունը 40

3.2. Բազմաօգտատեր SandGame «լուրջ խաղի» օրինակը 47

3.3. Արհեստական նեյրոնային ցանցերի ներգրավմամբ «Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդրի լուծման դիտարկումը 54

3.4. «SandScheduler» ծրագրային փաթեթը 64

Եզրակացություն երրորդ գլխի վերաբերյալ 73

ԳԼՈՒԽ 4. ՍՏԱՑՎԱԾ ՏԵՍԱԿԱՆ և ՓՈՐՁԱՐԱՐԱԿԱՆ ԱՐԴՅՈՒՆՔՆԵՐԻ ՎԵՐԼՈՒԾՈՒԹՅՈՒՆԸ և ԳՆԱՀԱՏԱԿԱՆԸ 74

4.1. «CA Simulator» ծրագրային փաթեթի գործարկման արդյունքների և գոյություն ունեցող լուծումների համամետական վերլուծությունը: 74

4.2. Ավազակույտի մոդելի սիմուլատորի աշխատանքի զուգահեռ և ոչ զուգահեռ տարբերակների ժամանակների համեմատական վերլուծություն: 76

4.3. «Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդրի շրջանակներում նեյրոնային ցանցերի և ճշգրիտ արդյունքների միջև համեմատական վերլուծությունը: 89
Եզրակացություն չորրորդ գլխի վերաբերյալ 97

ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ 99
ՀՂՈՒՄՆԵՐ..... 101
ՀԵՂԻՆԱԿԻ ԿՈՂՄԻՑ ՀՐԱՏԱՐԱԿՎԱԾ ԱՇԽԱՏՈՒԹՅՈՒՆՆԵՐ.....106

ԱՌԱՋԱԲԱՆ

Թեմայի արդիականությունը

Ժամանակակից գիտական հետազոտություններում առաջնային ուղղություններից մեկը բարդ դինամիկական համակարգերի վարքի ուսումնասիրումն է: Դինամիկական համակարգերը հետազոտության համար գրավիչ են այնքանով, որ նրանք կարող են հանդիսանալ մի շարք ֆիզիկական և կենսաբանական երևույթների, երկրաշարժերի, տնտեսագիտության, հեռահաղորդակցության և շատ այլ ոլորտների նկարագրման հարմարավետ մոդել:

Բարդ դինամիկական համակարգերի կառուցվածքը բնորոշվում է միլիոնավոր և ավել գագաթանի գրաֆի/ցանցի հանգույցներում/բջիջներում տեղադրված պրոցեսորներով, որոնք միմիանց հետ կապված են ինֆորմացիոն կապերով: Համակարգի վիճակները ներկայացվում են հանգույցներում տեղադրված պրոցեսոներների վիճակների վեկտորով, իսկ բջջի վիճակների անցումները ձևավորվում են լոկալ անցումային ֆունկցիաներով՝ կախված հարևան պրոցեսորների վիճակներից: Արտաքին մուտքային ազդանշանների հաջորդականություններն ընդհանուր դեպքում ունեն ստոխաստիկ բնույթ և արտահայտում են արտաքին միջավայրի ազդեցությունը համակարգի վրա:

Սույն աշխատանքը վերաբերում է բարդ դինամիկական համակարգերի ենթադաս հանդիսացող **ինքնակազմակերպվող համակարգերին** [1], որոնցում, որպես կանոն, կարող են առաջանալ վիճակների առաձնահատուկ կարգավորվածություններ: Ժամանակի ընթացքում «դանդաղ» փոփոխելով վիճակը, թռիչքային անցում է կատարվում դեպի այսպես կոչված «ռեկուրենտ» վիճակների բազմություն, որը համակարգը չի լքում և, արդյունքում, ռեկուրենտ վիճակների բազմության մեջ շարժման սահմանափակումը ծնում է ներքին կարգավորվածություն: Կարգավորվածության առաջացման և կրիտիկական հատկությունների նման մեխանիզմն անվանվում է **ինքնակազմակերպվող կրիտիկականություն** [2-3]: Նշված համակարգերի դասի համար որպես

մոդելավորման հարմարավետ միջոց կարող են հանդիսանալ բջջային ավտոմատները [4-5], պայմանով, որ դրանց միջոցով հնարավոր լինի նկարագրել համակարգերի ինքնակազմակերպվող կրիտիկականությունը: Կատարված հետազոտություններում որպես բջջային ավտոմատներ ընտրվել են **ավազակույտի** [6] և **rotor-router** [7-8] մոդելները:

Նշենք, որ ավազակույտի մոդելը ամենապարզ տեսական մոդելն է՝ ինքնակազմակերպվող կրիտիկականության լավագույնս նկարագրման համար [9-10]:

Ավազակույտի մոդելով նկարագրվող ֆիզիկական երևույթների ուսումնասիրությունն ավելի արդյունավետ դարձնելու նպատակով մշակվել և գոյություն ունեն գործիքաշարեր, որոնք իրականացնում են ավազակույտի մոդելի սիմուլացիա և տեսաբերում (վիզուալիզացիա) [11-13] : Ավազակույտի մոդելի աշխատանքի ալգորիթմական կիրառություն է կատարվել նաև ոչ բնական համակարգերում, ինչպիսիք են՝ օրինակ, կլաստերային և ամպային համակարգերը [14-15]:

Այնուամենայնիվ, ավազակույտի մոդելի հետազոտության գոյություն ունեցող գործիքաշարերում առկա են մի շարք թերացումներ, որոնցից են, մասնավորապես՝ ցանցի տեղայնացվածությունը, ավազակույտի մոդելն իրագործող ցանցի ցածր չափայնությունը, համակարգի միաժամանակյա օգտատերերի քանակի խիստ սահմանափակումը, համատեղ աշխատանքների կատարման ընթացքում օգտատերերի ֆունկցիոնալ սահմանափակումները, սիմուլատորի աշխատանքի հաջորդականային բնույթը և համակարգի ըստ պահանջի ընդլայնումների հնարավորության բացակայությունը:

Հաշվի առնելով ինքնակազմակերպվող համակարգերի ուսումնասիրման կարևորությունը նաև «լուրջ խաղերի» տեսության մեջ [16], ավազակույտի մոդելի վրա կառուցված «լուրջ խաղի» օրինակը կարող է գրավիչ հարթակ հանդիսանալ ինքնակազմակերպ կրիտիկականության տեսողական հետազոտության և ուսուցողական նպատակների համար:

Մյուս կողմից, նկատի ունենալով ավազակույտի և rotor-router մոդելների պիտանելիությունը կլաստերային և ամպային համակարգերի կազմակերպման գործընթացներում, ինչպիսիք են՝ մասնավորապես, էներգապահպանման և հավասարաչափ բաշխվածության ապահովումը, լիարժեք սիմուլյացնող գործիքաշարի ստեղծումը արդիական խնդիր է:

Ատենախոսության նպատակն է հետազոտել և մշակել ծրագրային փաթեթներ ինքնակազմակերպ համակարգերում դինամիկ պրոցեսների ուսումնասիրության համար:

Նշված նպատակին հասնելու համար առաջադրվել և լուծվել են հետևյալ խնդիրները.

- մշակել ավազակույտի մոդելով նկարագրվող ինքնակազմակերպվող դինամիկ պրոցեսների հետազոտման բազմաօգտատեր ծրագրային փաթեթ երկչափ և եռաչափ տեսաբերման հնարավորությամբ:
- դուրս բերել d չափանի խորանարդային ցանցի համար գագաթային ծածկույթների հայտնաբերման օպտիմալ ալգորիթմ և իրականացնել ավազակույտի մոդելի աշխատանքի զուգահեռացում տվյալ տոպոլոգիական տարածությունում:
- հավելել «լուրջ խաղեր»-ի տեսության գրադարանը ևս մեկ «լուրջ խաղ»-ի օրինակով՝ հիմնելով ավազակույտի մոդելի աշխատանքի վրա:
- հետազոտել նեյրոնային ցանցերի հնարավոր օգտագործումը ավազակույտի մոդելի որոշակի խնդիրների լուծման համար
- նախագծել կլաստերային համակարգի բաշխիչի սիմուլատոր՝ հիմնվելով ավազակույտի և rotor-router մոդելների աշխատանքի ալգորիթմի վրա:

Հետազոտման մեթոդները

Կատարված հետազոտությունները հիմնված են դիսկրետ մաթեմատիկայի, գրաֆների տեսության, ավտոմատների տեսության դրույթների, ինչպես նաև

զուգահեռ ծրագրավորման մեթոդների և նեյրոնային ցանցերի մեթոդաբանության օգտագործման վրա:

Գիտական նորությունը

- Մշակվել է d չափանի խորանարդային ցանցի համար գագաթային ծածկույթների հայտնաբերման օպտիմալ ալգորիթմ, որը հիմք է հանդիսացել ցանցերում ավազակույտի մոդելի աշխատանքի զուգահեռացման օպտիմալ ապահովմանը:
- Մշակվել և իրականացվել է լուրջ խաղի ալգորիթմ՝ հիմված ավազակույտի մոդելի վրա:
- Ստեղծվել է կլաստերային համակարգերում խնդիրների տարաբաշխման նոր ալգորիթմ՝ հիմնված ավազակույտի և rotor-router մոդելների վրա:

Արդյունքների կիրառական նշանակությունը

Մշակված բազմաօգտատեր ծրագրային համակարգը՝ հիմնված ավազակույտի մոդելով նկարագրվող ինքնակազմակերպվող դինամիկ պրոցեսների հետազոտման համար, թույլ է տալիս հետազոտել և հաշվարկել տարբեր ֆիզիկական, ինֆորմացիոն բնութագրիչներ երկչափ և եռաչափ տարածություններում, ինչպես նաև կատարել համատեղ հետազոտություններ՝ աշխատանքների միաժամանակյա դիտման, փոփոխությունների կատարման, մոդելներում վիճակների պահպանման, ստացման և ուղարկման հնարավորություններով:

Իրականացված ծրագրային փաթեթների միջոցով d չափանի խորանարդային ցանցում ավազակույտի մոդելի աշխատանքի զուգահեռացումը՝ հիմնված նույն տոպոլոգիական տարածությունում աստղային լրիվ ծածկույթները նկարագրող բանաձևի վրա, հնարավորություն են տալիս օգտագործողներին կատարել բնութագրիչների հաշվարկ միչև 3 անգամ ավելի արագ նույն CPU-ի

սահմաններում, և մինչև 50 անգամ ավելի արագ համարժեք վիդեոկարտայի դեպքում:

Ստեղծված բազմաօգտատեր «լուրջ խաղ»-ի օրինակը՝ հիմնվելով ավազակույտի մոդելի վերաբերող բազմաթիվ թեորեմների վրա և չունենալով հաղթանակին տանող ակընհայտ ստրատեգիա, խթան է հանդիսանում ինքնակազմակերպվող համակարգերի հանդեպ եղած հետաքրքրության մեծացմանը, հետևաբար նաև համատեղ ուսուցմանը և հետազոտմանը:

«Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդրի հետազոտմանը միտված նեյրոնային ցանցերի մշակումը հնարավորություն է ստեղծում փոխարինել ավազակույտի աշխատանքի սիմուլյացիան համապատասխան նեյրոնային ցանցով՝ սույնով օպտիմիզացնելով պահաջվող սպասվելիք արդյունքների ստացման ժամանակը:

Նախագծված կլաստերային համակարգի սիմուլատորը՝ հիմված ավազակույտի և rotor-router մոդելների աշխատանքի ալգորիթմի վրա, դյուրացնում է ինքնակազմակերպվող համակարգերի աշխատանքային ալգորիթմի դրսևորման հետազոտումը կլաստերային համակարգերում: Ինչպես նաև նկարագրված և առաջ բերված հիպոթեզը rotor-router մոդելների տրամաբանությամբ խնդիրների տարաբաշխման համակարգի վերաբերյալ կարող է դինամիկ խնդիրների պլանավորման նոր հիմք հանդիսանալ:

Պաշտպանությանը ներկայացվող դրույթները

- d չափանի խորանարդային ցանցի համար գագաթային ծածկույթների հայտնաբերման օպտիմալ մեթոդի մշակումը,
- լուրջ խաղի ալգորիթմի մշակումը՝ հիմված ավազակույտի մոդելի վրա,
- կլաստերային համակարգերում խնդիրների տարաբաշխման նոր ալգորիթմի մշակումը՝ հիմնված ավազակույտի և rotor-router մոդելների վրա,
- վերոհիշյալ դրույթների ալգորիթմական ներկայացումը բազմաօգտատեր միջավայրում:

Աշխատանքի արդյունքների ներդրումը

Մշակված համակարգերը ներդրվել և փորձարկվել են Հայկական ազգային հաշվողական գրիդ ենթակառուցվածքում (Երևան, Հայաստան): Մշակված համակարգը օգտագործվել է որպես գործիքային ծրագրային միջոց վիճակագրական ֆիզիկայի խնդիրների լուծման համար Դուբնայի ՄՀՄԻ-ի և ԻԱՊԻ-ի համատեղ հետազոտողական խմբի կողմից:

Աշխատանքի ապրոբացիան

Ատենախոսության հիմնական արդյունքները զեկուցվել են.

- Միջազգային գիտաժողով՝ "Computer Science and Information Technology'' (CSIT-2013), Երևան, Հայաստան, 2013 թ.
- Միջազգային գիտաժողով՝ "Computer Science and Information Technology'' (CSIT-2017), Երևան, Հայաստան, 2017 թ.
- ՀՀ ԳԱԱ ԻԱՊԻ ընդհանուր սեմինարներում
- Թուլուզի IRIT (Կոմպյուտերագիտության ինստիտուտ) լաբորատորիայի սեմինարին

Հրատարակումներ

Աշխատանքի հիմնական արդյունքները ներկայացված են 6 գիտական աշխատություններում, որոնց ցանկը բերված է ատենախոսության վերջում:

Ատենախոսության կառուցվածքը

Ատենախոսությունը բաղկացած է առաջաբանից, 4 գլուխներից, 63 անուն պարունակող գրականության ցանկից: Աշխատանքի ընդհանուր ծավալը կազմում է 106 էջ:

Աշխատանքի հիմնական բովանդակությունը

Ներածության մեջ հիմնավորվել է ատենախոսության թեմայի արդիականությունը, ներկայացվել են հետազոտության նպատակն ու խնդիրները, գիտական նորույթը, պաշտպանությանը ներկայացվող հիմնական դրույթները և հետազոտության տեսական ու գործնական նշանակությունը:

Ատենախոսության առաջին գլխում նկարագրվել են ինքնակազմակերպվող համակարգերը, դրանց աշխատանքի սկզբունքը և հիմնախնդիրները: Ներկայացվել են նաև ինքնակազմակերպվող համակարգերը մոդելավորող և սիմուլյացնող արդի ծրագրային փաթեթները, ինչպես նաև նկարագրվել են «լուրջ խաղերի» հայեցակետերը և կլաստերային համակարգերի գոյություն ունեցող իրականացումները հիմված ինքնակազմակերպվող համակագրերի մոդելների վրա:

Ատենախոսության երկրորդ գլուխը նվիրված է աշխատանքում ստացված տեսական արդյունքների և մշակված ալգորիթմների նկարագրությանը, որոնք հետագայում ծրագրային փաթեթների աշխատանքի հիմք են հանդիսանում:

Ատենախոսության երրորդ գլուխը նվիրված է մշակված ծրագրային փաթեթների, իրականացման համար ընտրված միջավայրի, ծրագրավորման լեզվի, ալգորիթմական աշխատանքների և օգտագործման սկզբունքների նկարագրությանը:

Ատենախոսության չորրորդ գլխում զետեղված են ինքնակազմակերպվող համակարգերի ուսումնասիրմանը նվիրված ծրագրային փաթեթների, ստացված տեսական արդյունքերի հիման վրա փորձարկված ինչպես նաև մշակված նեյրոնային ցանցերի կողմից ստացված արդյունքերի համեմատական:

ԳԼՈՒԽ 1. ԻՆՔՆԱԿԱԶՄԱԿԵՐՊՎՈՂ ՀԱՄԱԿԱՐԳԵՐԻ և ԴՐԱՆՑ ՀԵՏԱԶՈՏՄԱՆ ՆՊԱՏԱԿՈՎ ՆԱԽԱԳԾՎԱԾ ԾՐԱԳՐԱՅԻՆ ԳՈՐԾԻՔԱՄԻՋՈՑՆԵՐԻ ՎԵՐԼՈՒԾՈՒԹՅՈՒՆ

Ժամանակակից գիտական հետազոտություններում առաջնային ուղղություններից մեկը բարդ դինամիկական համակարգերի վարքի ուսումնասիրումն է: Դինամիկական համակարգերը հետազոտության համար գրավիչ են այնքանով, որ նրանք կարող են հանդիսանալ մի շարք ֆիզիկական և կենսաբանական երևույթների, երկրաշարժերի, տնտեսագիտության, հեռահաղորդակցության և շատ այլ ոլորտների նկարագրման հարմարավետ մոդել:

Բարդ դինամիկական համակարգերի առանձնահատկությունն այն է, որ նրանց հատուկ են որոշակի տիպի կարգավորվածություններ: Բարդ համակարգերը ժամանակի ընթացքում անցում են կատարում դեպի այսպես կոչված «ռեկուրենտ» վիճակների բազմություն, որից համակարգը դուրս չի գալիս և, արդյունքում, ռեկուրենտ վիճակների բազմության մեջ շարժման սահմանափակումը ծնում է ներքին կարգավորվածություն: Կարգավորվածության առաջացման և կրիտիկական հատկությունների նման մեխանիզմն անվանվում է **ինքնակազմակերպ կրիտիկականություն:**

Մյուս կողմից, բջջային ավտոմատները կարող են բարդ դինամիկական համակարգերի մոդելավորման հարմարավետ միջոց հանդիսանալ այնքանով, որքանով դրանց միջոցով հնարավոր է նկարագրել համակարգերի ինքնակազմակերպ կրիտիկականությունը: Կատարված հետազոտություններում որպես բջջային ավտոմատ ընտրվել է ավազակույտի և rotor-router մոդելները: Ավազակույտի մոդելով նկարագրվող ֆիզիկական երևույթների ուսումնասիրությունն ավելի արդյունավետ դարձնելու նպատակով մշակվել և գոյություն ունեն գործիքաշարեր, որոնք իրականացնում են ավազակույտի մոդելի սիմուլյացիա և տեսաբերում: Ավազակույտի մոդելի աշխատանքի ալգորիթմական կիրառություն է կատարվել նաև ոչ բնական համակարգերում, ինչպիսիք են՝ օրինակ, կլաստերային համակարգերը:

Այնուամենայնիվ, ավազակույտի մոդելի հետազոտության գոյություն ունեցող գործիքաշարերում առկա են մի շարք թերացումներ, որոնցից են, մասնավորապես՝ ցանցի տեղայնացվածությունը, ավազակույտի մոդելն իրագործող ցանցի ցածր չափայնությունը, համակարգի միաժամանակյա օգտատերերի քանակի խիստ սահմանափակումը, համատեղ աշխատանքների կատարման ընթացքում օգտատերերի ֆունկցիոնալ սահմանափակումները, սիմուլյատորի աշխատանքի հաջորդականային բնույթը և համակարգի ըստ պահանջի ընդլայնումների հնարավորության բացակայությունը: Հաշվի առնելով ինքնակազմակերպ համակարգերի ուսումնասիրման կարևորությունը նաև «լուրջ խաղերի» տեսության մեջ, ավազակույտի մոդելի վրա կառուցված «լուրջ խաղի» օրինակը կարող է գրավիչ հարթակ հանդիսանալ ինքնակազմակերպ կրիտիկականության տեսողական հետազոտության համար: Մյուս կողմից, նկատի ունենալով ավազակույտի և rotor-router մոդելների պիտանելիությունը կլաստերային համակարգերի կազմակերպման գործընթացներում, ինչպիսիք են՝ մասնավորապես, էներգապահպանման և հավասարաչափ բաշխվածության ապահովումը, լիարժեք սիմուլյացնող գործիքաշարի ստեղծումը արդիական խնդիր է:

1.1 Ինքնակազմակերպվող համակարգեր, ավազակույտի և rotor-router մոդելներ

Ինքնակազմակերպվող կրիտիկականության հայեցակետն առաջին անգամ դիտարկվել է Բակի, Թանգի և Ուիզենֆիլդի [17] կողմից 1987 թվականին: Ընդսմին՝ կատարված հետազոտություններում որպես բջջային ավտոմատներ ընտրվել են **ավազակույտի** և **rotor-router** մոդելները, որոնք մինչ այժմ հայտնի են որպես ինքնակազմակերպ կրիտիկականությունը նկարագրող մեծ տարածում և հետաքրքրություն առաջացնող համակարգեր: Ավազակույտի մոդելը, լինելով բջջային ավտոմատների դաս, հանդիսանում է ամենապարզ տեսական մոդելը, որտեղ տեղի ունի ինքնակազմակերպ կրիտիկականությունը [18]: Գոյություն ունի

ավազակույտի մոդելների հատուկ ենթադաս, որը կոչվում է աբելյան ավազակույտի մոդել:

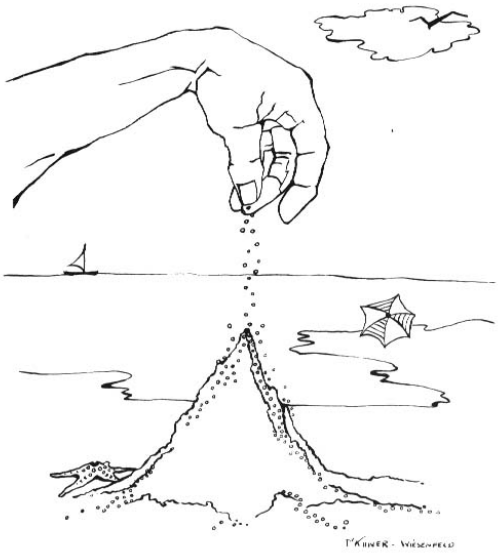
Կասենք, որ ավազակույտն ունի աբելյան հատկություն այն և միայն այն դեպքում, երբ ավազակույտի վերջնական կայուն վիճակը կախված չէ հանգույցների վիճակների փոփոխության հերթականությունից: Այս հատկությունը կարևոր դերակատարում ունի աբելյան ավազակույտի մոդելի ինչպես թվային, այնպես էլ վերլուծական ուսումնասիրությունների կատարման ընթացքում [19, 20, 21, 22, 23, 24, 25, 26, 27]: Հաշվի առնելով ինքնակազմակերպվող համակարգերի կարևորությունը բնական երևույթների ուսումնասիրման և մոդելավորման գործընթացներում, մինչ այժմ նախագծվել և հաջողությամբ զարգացվում են համապատասխան ծրագրային գործիքամիջոցներ՝ վերոհիշյալ խնդիրների համակարգչային սիմուլացման և տեսաբերման համար:

Ավազակույտի ներըմբռնողական մոդել է, օրինակ՝ սեղանի մակերևույթին դրա գոյացումը, որն ինքնին թույլ ինտերակտիվ համակարգի տեսակ է (Նկար 1): Վերևից անընդհատ լցնելով ավազահատիկներ սեղանի պատահականորեն ընտրված կոնկրետ տեղամասի վրա, նկատում ենք, որ լցված ավազահատիկները մինչև ինչ-որ ժամանակ ոչ մի ներգործություն չեն ունենում ավազակույտի այլ տեղամասերում: Այնուամենայնիվ, ընթացակարգը շարունակելու դեպքում ավազակույտը հասնում է այսպես կոչված «կրիտիկական վիճակի», որտեղ կույտն այլևս չի կարող աճել առանց փլուզման [28-29]:

Կախված ավազակույտի փլուզման պահին արձանագրված կոնֆիգուրացիայից, փլուզումները կարող են լինել տարբեր ծավալների: Բակը այս կրիտիկական վիճակները բնութագրում է որպես ինքնակազմակերպ կրիտիկականության վիճակներ (SOC), այսինքն, վիճակներ, որտեղ համակարգն ինքնակազմակերպվում է կայուն վիճակի:

Ավազի ևս մեկ հատիկի ավելացումը մի այնպիսի տեղամասում, որը գտնվում է SOC վիճակում, կարող է հանգեցնել «լավայի» կամ «ավազի սողանքի», այսինքն,

ավազի կասկադային փլուզմանը, իսկ ավազակույտի հանգույցներում ավազահատիկները հնարավոր է նաև ցած թափվեն սեղանի եզրից:



Նկար 1.Ավազի կույտ:[1]

Ավազակույտի մոդելն արտահայտվում է $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ գրաֆով, որտեղ V-ն գագաթների, իսկ E-ն կողերի բազմություններն են:

Սահմանում 1: $\eta : \mathrm{V} \to \mathrm{N}$ (N-ը բնական թվերի բազմությունն է) արտապատկերումը կոչվում է **կոնֆիգուրացիա**, որտեղ $\eta(\mathrm{v})$-ն $\mathrm{v} \in \mathrm{V}$ գագաթում պարունակվող ավազահատիկների քանակն է, կամ որ նույնն է՝ **v գագաթի հզորությունը**:

Սահմանում 2: G գրաֆի $S \subset V$, $S \neq \emptyset$ գագաթները կոչվում են **փոսեր**:

Սահմանում 3: G գրաֆում $v \in V \backslash S$ գագաթը կանվանենք **կայուն**, եթե $\eta(v) < d_v$, $d_v = \sum_{w \in V} a_{v,w}$, որտեղ $a_{v,w}$-ն $v, w \in V$ գագաթները կապող կողերի քանակն է: Նշված պայմանին չբավարարող գագաթները կոչվում են **անկայուն**:

Սահմանում 4: **Փլուզում** կանվանենք v անկայուն գագաթից իտերացիայի ամեն քայլում մեկական ավազահատիկի փոխանցումը հարևան գագաթներին

[1] https://math.hmc.edu/seniorthesis/archives/2009/ndurgin/ndurgin-2009-thesis.pdf

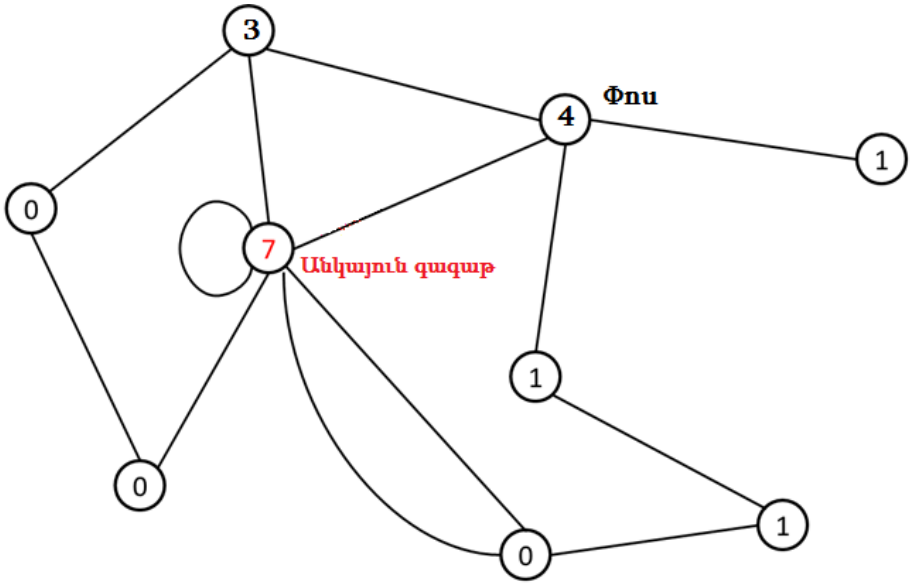
(հնարավոր է նաև ինքն իրեն), որի հետևանքով ծնվում է նոր η' կոնֆիգուրացիա, որտեղ $\eta'(v) = \eta(v) - d_v$ և $\eta'(w) = \eta(w) + a_{v,w}$, $\forall\, w \in V$:

Նկատենք, որ փոսերն, ըստ սահմանման կայուն են, ուստի և փլուզման ենթակա չեն:

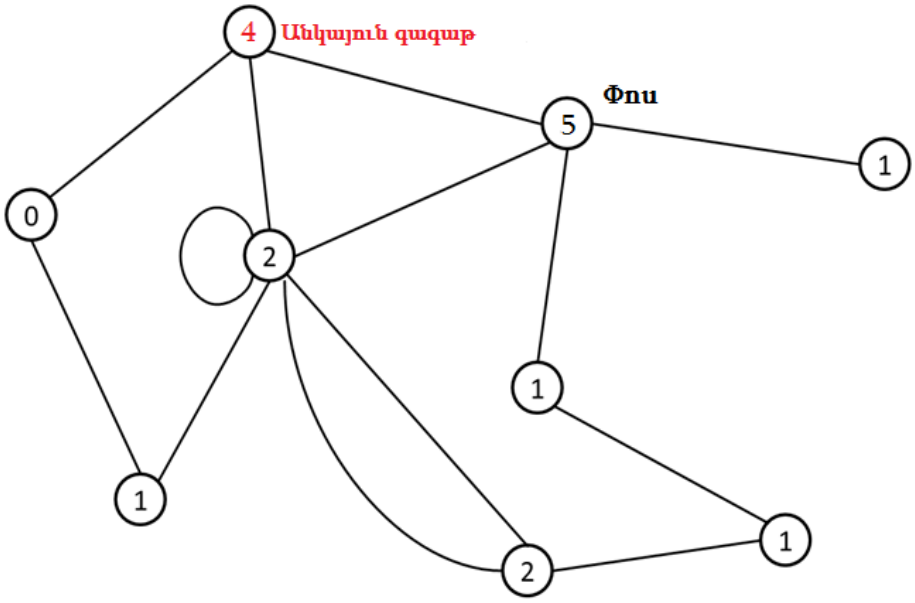
Սահմանում 5: η **կոնֆիգուրացիան կանվանենք կայուն**, եթե այդ կոնֆիգուրացիայում բոլոր գագաթները կայուն են:

Ինչպես նշվեց վերևում, ավազակույտի մոդելն օժտված է աբելյան հատկությամբ, որը նշանակում է, որ անկախ փլուզումների հերթականությունից, մոդելի վերջնական կայուն վիճակը միշտ նույնն է:

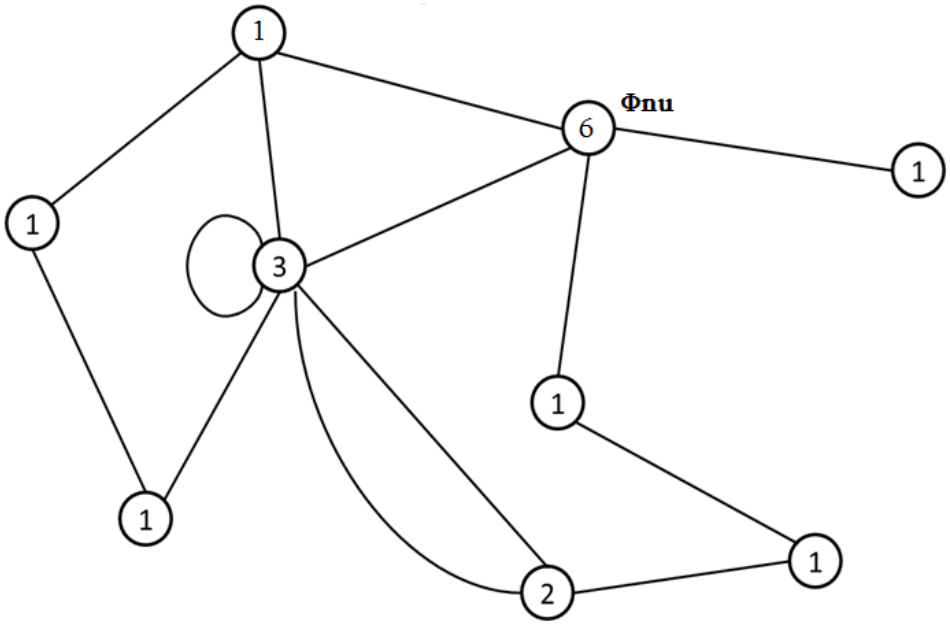
Ավազակույտի մոդելի մեկ այլ օրինակ բերված է նկարներ 2,3,4-ում, որտեղ կարմիրով նշված են անկայուն գագաթները: Օրինակում երևում է, որ մոդելը ի սկզբանե գտնվում է անկայուն վիճակում, այնուհետև փլուզման հետևանքով հայնտվում է նոր անկայուն գագաթ, որը արդեն փլուզելով ստանում ենք մոդելի կայուն վիճակ: Նկատենք, որ վերջնական վիճակում «փոսային» գագաթում ավազահատիկների քանակը ավել է այդ գագաթի կողերի քանակից, սակայն այն ըստ սահմանման չի փլուզվում և չի ազդում մոդելի տվյալ վիճակի կայուն լինելուն:



Նկար 2. Ավազակույտի մոդել(անկայուն վիճակ):



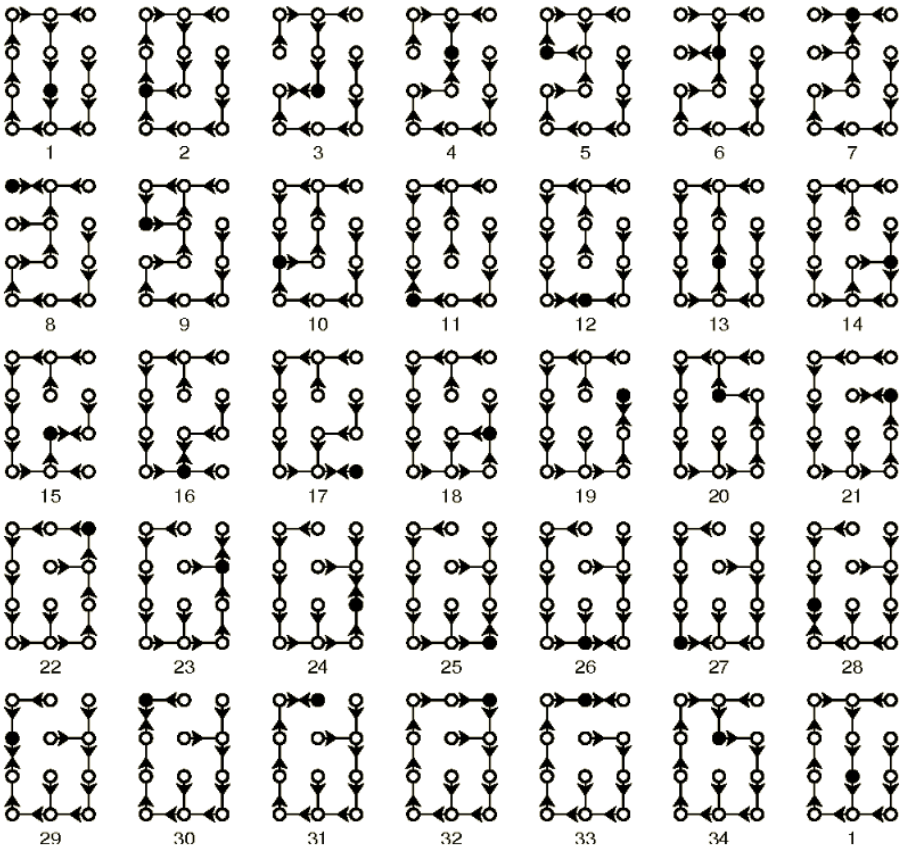
Նկար 3. Ավազակույտի մոդել(անկայուն վիճակ):



Նկար 4. Ավազակույտի մոդել(կայուն վիճակ):

Ինչ վերաբերում է **rotor-router** մոդելին, ապա նշենք, որ այն ևս հիմնված է ավազակույտի մոդելի վրա, սակայն ունի մի քանի տարբերիչ առանձնահատկություններ: Մասնավորապես՝ ավազահատիկների փլուզումն այս դեպքում տեղի է ունենում ամեն գագաթում ուղղորդիչի միջոցով:

Դիտարկենք Նկար 5-ում պատկերված ուղղորդված քառակուսային ցանցի գրաֆի մոդելը, որտեղ յուրաքանչյուր գագաթի ելից կողի վրա սահմանված է ցիկլիկ կարգավորվածություն: Յուրաքանչյուր v գագաթի համապատասխանության մեջ դնենք ρ_v երթուղիչ: Ավազահատիկի տեղաշարժը կատարվում է հետևյալ սկզբունքով. նախ՝ պտտում ենք երթուղիչի ուղղությունն ըստ ցիկլիկ կարգավորվածության, այնուհետև ավազահատիկն ուղարկում այդ ուղղությամբ: Նկար 5-ում բերված է rotor-router մոդելի օրինակը փոքրաչափ ցանցային գրաֆի կիրառմամբ, որտեղ երևում է, որ վերջավոր քայլերից հետո մոդելը վերադառնում է սկզբնական վիճակի, ընդսմին վերականգնելով նաև երթուղիչների նախնական կարգավորվածությունը:



Նկար 5.Rotor-router մոդելի աշխատանքը 3*4 ցանցում:[2]

[2] http://pi.math.cornell.edu/~levine/sand.pdf

1.2. Ինքնակազմակերպվող համակարգերի հետազոտման արդի ծրագրային գործիքամիջոցների վերլուծությունը

Բջջային ավտոմատների և ավազակույտի աբելյան մոդելների վերաբերյալ բազմաթիվ հետազոտությունների կատարման նպատակով նախագծվել են համապատասխան գործիքաշարեր, որոնք ապահովում են նաև դրանց մոդելավորումն ու համակարգչային սիմուլացիան: Նշված հետազոտությունների արդյունավետ անցկացման հնարավորություններից մեկը այդ գործիքաշարերի տեսաբերման և ինտերակտիվ գործունեության ապահովումն ու ավազակույտի մոդելի հիմնական բնութագրիչների, օրինակ՝ միջին խտություն, կրիտիկական խտություն և այլն, հաշվարկի իրականացումն է: Բացի այդ, կարևոր է նաև համապատասխան ծրագրային գործիքամիջոցների ներդնումը ուսուցողական պրոցեսներում:

Վերոհիշյալ նպատակների իրականացումը ենթադրում է համապատասխան գործառույթով ծրագրային լուծումներ, որոնք կներառեն նաև միաժամանակյա փոփոխություններ կատարելու, մոդելը դիտարկելու, ինչպես նաև մոդելի աշխատանքին միջամտելու հնարավորությունների տրամադրումը: Ցանկալի է նաև, որ ներկայացված լուծումներն ապահովեն մոդելի դիտարկման ընթացքում կատարված փոփոխությունների պահպանումը, բնութագրիչների հաշվարկի կատարումը իրական ժամանակում, ինչպես նաև մոդելի տվյալ վիճակի պահպանումը՝ հետագա ուսումնասիրությունների համար: Մյուս կողմից, վիրտուալ լաբորատորիաների հետազոտողների համար արդիական խնդիր է այնպիսի ծրագրային գործիքամիջոցների առկայությունը, որը հնարավորություն կընձեռի թիմային ձևով և միաժամանակ դիտարկել ու հետազոտել միևնույն մոդելը՝ անկախ օգտագործողների աշխարհագրական դիրքից:

Ասպարեզում գոյություն ունեն մի շարք ծրագրային լուծումներ, որոնցից յուրաքանչյուրն ապահովում է վերոնշյալ պահանջների ինչ-որ ենթադաս[12-13, 30-

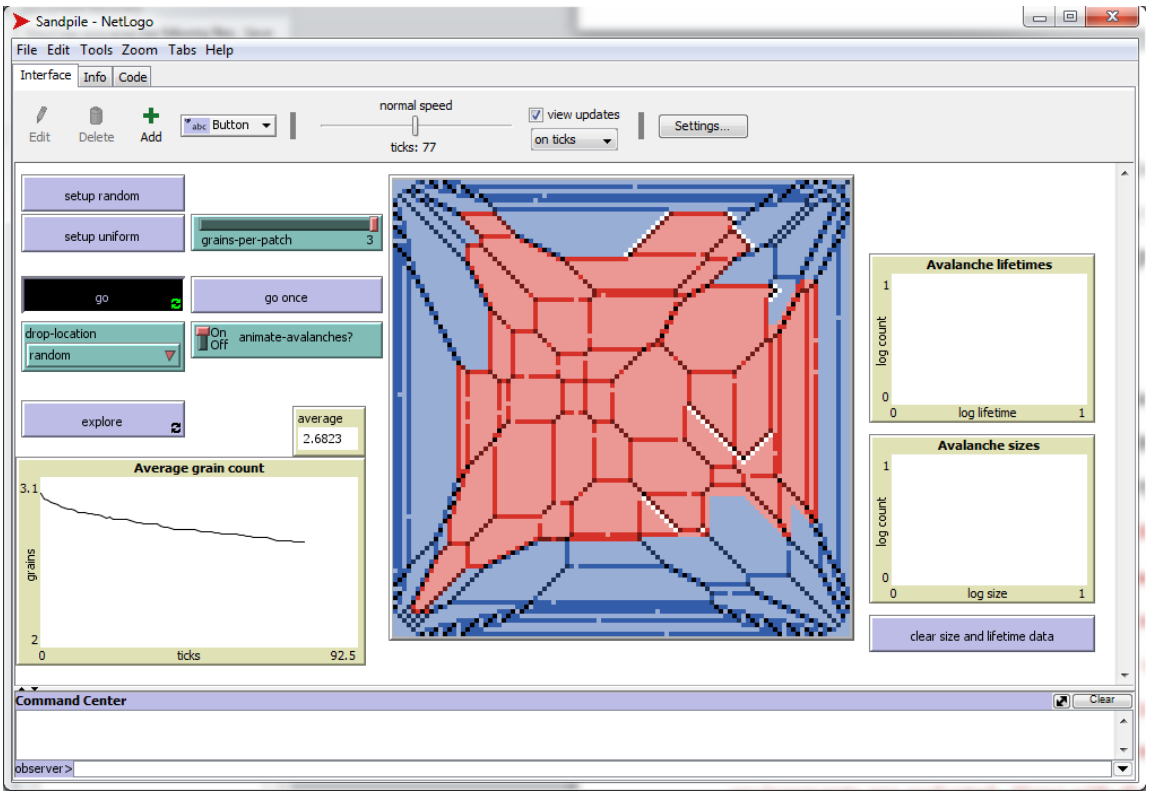
33]: Ստորև բերված է այդ գործիքամիջոցներից մի քանիսի նկարագրությունը, որոնցից ամենահայտնի և ֆունկցիոնալ առումով ամենահարուստը NetLogo-ն է:

NetLogo-ն բազմաագենտ համակարգերի ծրագրավորման միջավայր է՝ հարուստ գրադարանով, որտեղ զետեղված են 50 անուն դիսկրետ մոդելներ: Նույնանուն թիմը ստեղծել է NetLogo միջավայրում ծրագրավորելու համար առանձին լեզու՝ հիմնված ֆունկցիոնալ ծրագրավորման մեթոդաբանության վրա, որը բավականաչափ պարզեցված է, միևնույն ժամանակ նաև սահմանափակ է, որը լուրջ թերություն է բարդ դինամիկական համակարգերի տեսաբերման և մեծ թվով օգտատերերի ինտերակտիվ աշխատանքի ապահովման համար:

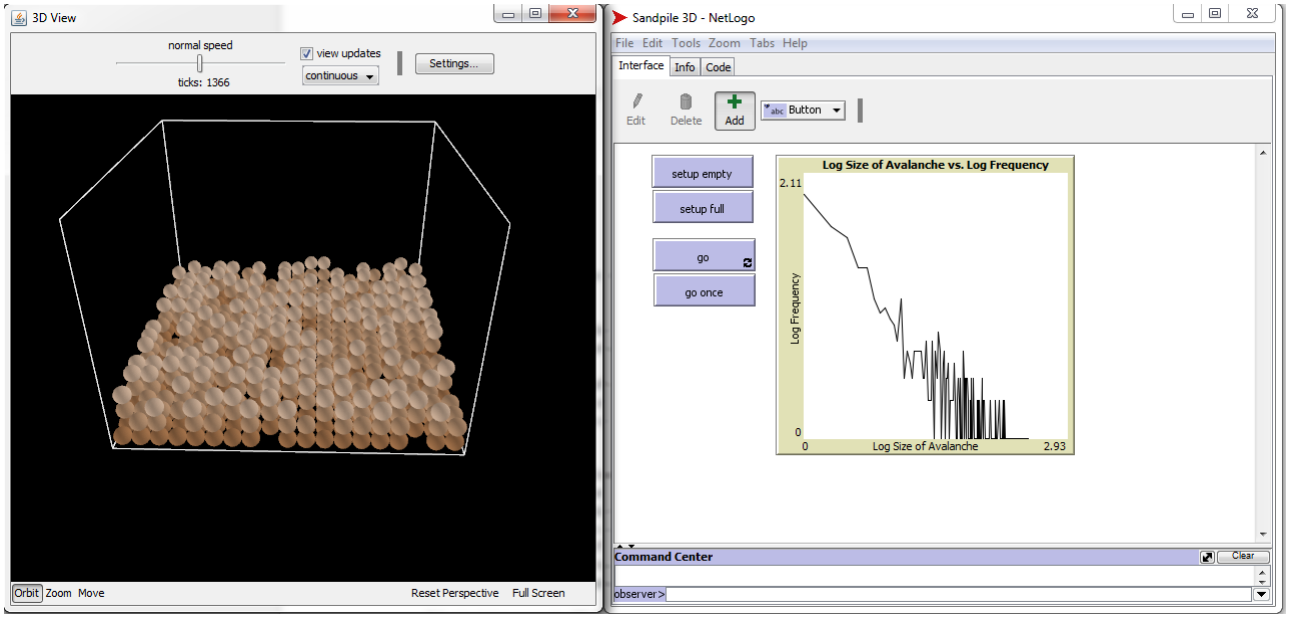
NetLogo-ի գրադարանը, որտեղ տեղ են գտել բազմաթիվ բազմաագենտ համակարգերի մոդելներ, ներառում է նաև ավազակույտի մոդելը նկարագրող երկու տարբեր ծրագրային փաթեթ:

Նկար 6-ում ներկայացված է այդ փաթեթներից առաջինը՝ երկչափ ցանցում ավազակույտի մոդելի երկչափ արտապատկերմամբ, որը հնարավորություն է ընձեռում կատարել ինչպես պատահականության, այնպես էլ միօրինակության սկզբունքով մոդելի սկզբնարժեքավորում, փլուզման ալիքների տեսաբերում և մոդելի ցանկալի կամ պատահական սկզբունքով ընտրված գագաթին ավազահատիկի ավելացում: Նշենք նաև, որ կատարվում է մոդելի հիմնական բնութագրիչների հաշվարկ իրական ժամանակում, ինչպիսիք են, օրինակ՝ գագաթների միջին հզորությունը, փլուզման ալիքների ժամանակահատվածը և չափերը:

Ինչ վերաբերում է գրադարանի երկրորդ ծրագրային փաթեթին, ապա այն օժտված է եռաչափ տեսաբերման հնարավորությամբ և իրականացնում է երկչափ ցանցում ավազակույտի մոդելի եռաչափ տեսաբերում՝ պատահականության սկզբունքով սկզբնարժեքավորման և հզորության ավելացման հնարավորությամբ:



Նկար 6.NetLogo ծրագրային համակարգում երկչափ ցանցում ավազակույտի մոդելի տեսաբերումը:[3]

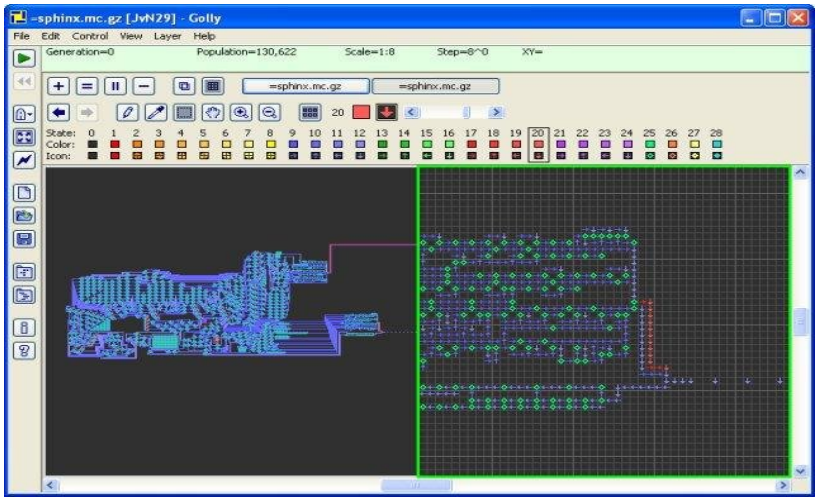


Նկար 7. NetLogo ծրագրային համակարգում եռաչափ ցանցում ավազակույտի մոդելի տեսաբերումը:[3]

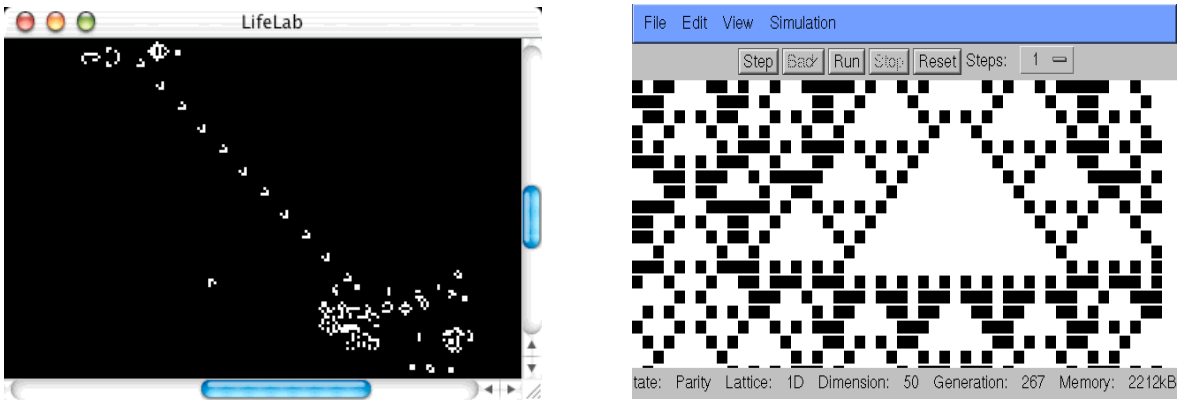
[3] https://ccl.northwestern.edu/netlogo/

Գոյություն ունեն նաև բջջային ավտոմատների այլ սիմուլատորներ, մասնավորապես Golly-ն, lifeLab-ը, JCASim-ը, CellDemo-ն և այլն [30-33](Նկար 8 և 9):

Նշենք, որ բոլոր նշված համակարգերի հիմնական թերությունն այն է, որ դրանք չեն ապահովում զուգահեռ աշխատանք միևնույն մոդելի վրա, ինչպես նաև օժտված չեն մոդելի տվյալ վիճակի պահպանման և հետագա օգտագործման հնարավորությամբ, որը սահմանափակում է ավազակույտի մոդելը միայն երկչափ ցանցում:



Նկար 8. Golly ծրագրային համակարգի տեսքը:[4]



Նկար 9. LifeLab[5] և CellDemo[6] ծրագրային համակարգերի տեսքը:

[4] https://sourceforge.net/projects/golly/
[5] http://www.trevorrow.com/lifelab/
[6] https://github.com/devinacker/celldemo

Նշենք, որ եռաչափ տեսաբերում և ավազակույտի եռաչափ ցանցի վրա դիտարկում հնարավոր է ապահովել նաև Wolfram Mathematica [34] կամ MatLab[35] միջավայրերում, որոնք, սակայն համատեղ ուսումնասիրությունների հնարավորություն չեն ընձեռում: Կատարվել են ուսումնասիրություններ տեղեկատվության փոխանակման խնդրի վերաբերյալ, ինչպես օրինակ [36]-ում, որտեղ ներկայացված է event-driven մոդուլ՝ վեբ տեխնոլոգիայի վրա հիմված պլատֆորմներում եռաչափ մոդելների համատեղ հետազոտությունների համար և [37]-ում, որտեղ դիտարկվել են վիրտուալ միջավայրում համատեղ աշխատանքները, արդի տեխնոլոգիաները և հնարավոր բարելավումները:

Ստորև բերված Նկար 10-ում ներկայացված է ավազակույտի մոդելի և բջջային ավտոմատների սիմուլացիա և տեսաբերում ապահովող գործիքամիջոցների համեմատական վերլուծությունը:

	Golly	JCASim	CellDemo	NetLogo
Մոդելավորում	☑	☑	☑	☑
Սիմուլյացիա	☑	☑	☑	☑
Երկչափ ցանցում վիզուալիզացիա	☑	☑	☑	☑
Եռաչափ ցանցում վիզուալիզացիա	☐	☐	☐	☐
Մոդելի փոփոխություններ	☑	☐	☑	☑
Բնութագրիչների հաշվարկ	☐	☐	☐	☐
Մոդելի տվյալ պահին վիճակի պահպանում և բեռնում (save/load)	☑	☑	☐	☐

Նկար 10.Ավազակույտի մոդելի և բջջային ավտոմատների սիմուլացիա և տեսաբերում ապահովող գործիքամիջոցների համեմատություն:

Մի շարք աշխատանքներում, ավազակույտի ֆիզիկական հատկությունների վերաբերյալ անալիտիկ տվյալների բացակայության պատճառով, կիրառվել են վիճակագրական վերլուծության տարբեր մեթոդներ, մասնավորապես՝

ավազակույտի մոդելների հիպոթետիկ և վերլուծական արտահայտությունները հաշվարկելու կամ հաստատելու համար: Առկա են նաև այլ գործիքաշարեր, որոնք նպատակաուղղված են մոդելի աշխատանքի սիմուլացիան արագացնելու համար: Դրանցից է օրինակ [38]-ում նկարագրված, բազմամիջուկանոց համակարգիչների վրա ավազակույտի աբելյան մոդելի զուգահեռ սիմուլացիա իրականացնող ծրագրային փաթեթը, որը արագագործության առումով արդարացնում է իրեն այն դեպքում, երբ ժամանակի յուրաքանչյուր պահին մոդելում առկա կլինեն քիչ քանակությամբ անկայուն գագաթներ: Ընդհանուր առմամբ, երբ սկզբնական անկայուն կոնֆիգուրացիան պարունակում է բազմաթիվ անկայուն գագաթներ, այս մոտեցումը լավ արդյունքներ չի տալիս խոշոր ցանցերում:

Այս ալգորիթմի հիմնական գործողությունը կայանում է հետևյալում: Դիցուք ունենք ցանց, որի վրա լցնում ենք ինչ-որ քանակությամբ ավազահատիկներ այնքան, մինչև որ մոդելը բերվում է անկայուն վիճակի: Ծրագրային փաթեթն իրականացնում է հնարավոր բոլոր փլուզումները և վերադարձնում արդեն կայուն վիճակի բերված ցանցը:

Առաջին (հաջորդականային) ծրագրային համակարգի գործողության ալգորիթմը հետևյալն է.

Քայլ 1. Անցնում է կատարվում ցանցի բոլոր հանգույցների վրայով, և բոլոր կրիտակական հանգույցների համարները գրանցվում են ստեկում:

Քայլ 2. Դիտարկվում են ստեկում գրանցված համարներով գագաթները: Համաձայն ստեկի գործողության սկզբունքի, ստեկից վերցվում է հերթական հանգույցի համարը, ըստ որի տվյալ հանգույցը փլուզում ենք, այնուհետև դիտարկվում են վերջինիս հարևան հանգույցները՝ կրիտիկականության հայտնաբերման նպատակով: Կրիտիկականության հայտնաբերման դեպքում տվյալ հանգույցի համարը նույնպես գրանցվում է ստեկում:

Քայլ 3. Ստուգում ենք ստեկի պարունակությունը: Դատարկ լինելու դեպքում ծրագիրն ավարտում է աշխատանքը: Հակառակ դեպքում՝ ալգորիթմը վերադառնում է քայլ 2-ին:

Երկրորդ (զուգահեռ) ալգորիթմի էությունը կայանում է հետևյալում: Դիցուք ունենք անկայուն վիճակում գտնվող ցանց:

Քայլ 1. Անցնում է կատարվում ցանցի բոլոր հանգույցների վրայով, և բոլոր կրիտակական հանգույցների համարները գրանցվում են ստեկում:

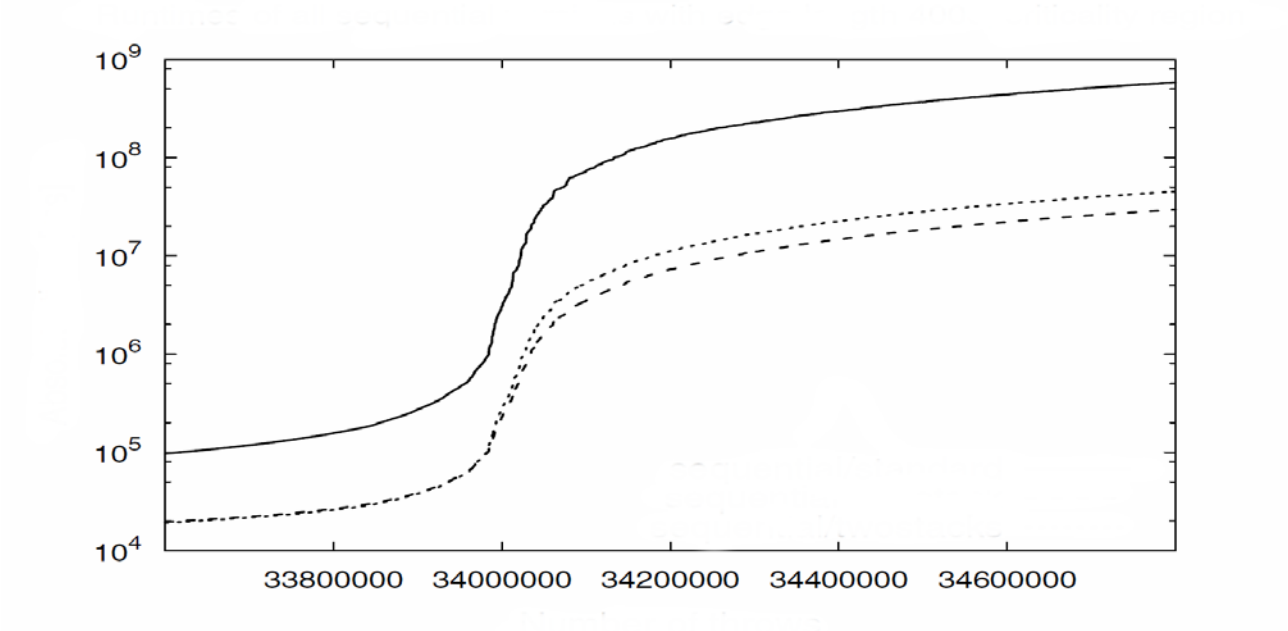
Քայլ 2. Դիտարկվում են ստեկում գրանցված համարներով գագաթները: Համաձայն ստեկի գործողության սկզբունքի, ստեկից վերցվում է հերթական հանգույցի համարը, ըստ որի նրա համար ստեղծվում է նոր հոսք/thread և այդպես ստեկի բոլոր գագաթների համար քանի դեռ նոր հոսք ստեղծելու հնարավորություն կա: Նոր ստեղծված հոսքի ֆունկցիան հետևյալն է.

1. ընտրված հանգույցը փլուզվում է՝ սույնով դառնալով կայուն
2. ստուգվում են հարևան հանգույցների կայուն կամ անկայուն լինելը
3. անկայուն հարևան հայտնաբերելու դեպքում ստեղծվում է նոր ստեկ, որում գրանցվում են անկայուն հարևան հանգույցների համարները, որից հետո անցում է կատարվում քայլ 2-ին:

Քայլ 3. Բոլոր հոսքերի աշխատանքի ավարտին ավարտվում է նաև ծրագրային ամբողջ փաթեթի աշխատանքը:

Տվյալ աշխատանքում [38] նշված է, որ խնդրի երկրորդ լուծումն իրականացվում է զուգահեռ ծրագրավորման մեթոդների և միջոցների կիրառմամբ: Կիրառված ալգորիթմների վերլուծության արդյունքում, այնուամենայնիվ, պարզվեց որ զուգահեռացումն օպտիմալ չէ: Ծրագրի գործողության արդյունքերը, մասնավորապես 16.000.000 հանգույց ունեցող քառակուսային ցանցի համար տեղադրված են Նկար 11-ում, որտեղ աղյուսակի մի առանցքը փլուզվող ավազահատիկների քանակն է, մյուսը՝ ցանցը կայուն վիճակի բերելու անհրաժեշտ

ժամանակը ներկայացված միլիվարկյաններով: Փորձը կատարվել է AMD Opteron 16 միջուկանոց պրոցեսորի վրա:



Նկար 11.Ավազակույտի կայունացման անհրաժեշտ ժամանակը փլուզվող ավազահատիկների քանակից կախված:

Հաշվի առնելով նաև «լուրջ խաղերի» կարևորությունը, որոնց միջոցով ընձեռնվում է հնարավորություն օգտվողներին փորձարկել ժամանակի, անվտանգության, գումարային արժեքի և այլ պատճառներով իրական կյանքում անհավանական իրավիճակներ, դրական նպաստումը խաղացողների մի շարք տարբեր հմտությունների զարգացմանը և հնարավոր դրական դերակատարումը հասարակության տարբեր շերտերի վրա ինքնակազմակերպվող համակարգերի ուսումնասիրմանը նպատակաուղված ավազակույտի մոդելի վրա կառուցված «լուրջ խաղի» ծրագրային օրինակը պահանջվող գործիք է հանդիսանում, որը կարող է գրավիչ հարթակ հանդիսանալ ինքնակազմակերպ կրիտիկականության տեսողական հետազոտության և ուսուցողական նպատակների համար:

1.3 «Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդիրը և գոյություն ունեցող լուծումները

«Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդիրը ձևակերպվել է ավազակույտի մոդելի կառուցման և հետազոտման նպատակով և մեծ հետաքրքրություն է ներկայացնում ֆիզիկայի և մի շարք բանագավառների խնդիրների մոդելավորման գործընթացում:

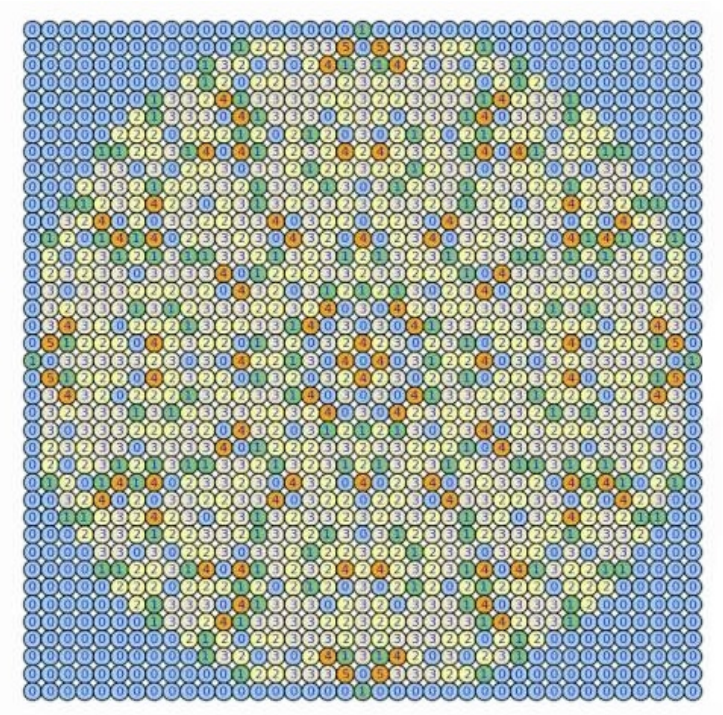
Սահմանում 6: $r_{i,j}$-ով նշանակենք v_i և v_j գագաթների միջև հեռավորությունը, կամ այլ կերպ ասած՝ նվազագույն կողերի քանակը, որն անհրաժեշտ է v_i –ից v_j հասնելու համար:

Սահմանում 7: $c_{i,j}$-ով նշանակենք ավազահատիկների այն նվազագույն քանակը, որոնք լցվելով v_i գագաթի վրա և արդյունքում կայունացնելով մոդելը, ապահովում է v_j գագաթում առնվազն 1 ավազահատիկի գոյությունը:

Դիտարկենք անվերջ երկչափ ցանց, որտեղ $\eta(v) = 0 \quad \forall v \in V$: Ընտրենք կամայական v_i հանգույց: Այդ դեպքում «Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդիրն իրենից ներկայացնում է $c_{i,j}$-ի կախվածության դուրս բերումը $r_{i,j}$ –ից (Նկար 12):

Գոյություն ունեն մի շարք աշխատություններ, որտեղ դիտարկվել և փորձ է կատարվել ճշգրիտ լուծում տալ վերոհիշյալ խնդրին, սակայն մինչև այժմ այս խնդիրը ճշգրիտ լուծում չունի: Խնդրին անդրադարձել են նաև Լեվինը [39] և Ֆեյը [40]: Ըստ Լեվինի աշխատության, դասական ավազակույտի մոդելում v_i-ում $n = \pi r^2$ ավազահատիկների ավելացման դեպքում $r_{i,j}$ –ի հնարավոր արժեքների բազմությունը կլինի հետևյալը.

$$\frac{r}{\sqrt{3}} \leq r_{i,j} \leq (r + o(r))/\sqrt{2}$$



Նկար 12.Ավազակույտի մոդելը կենտրոնից ավազահատիկներ լցնելու դեպքում:[7]

1.4 Ավազակույտի մոդելի վրա կառուցված կլաստերային գոյություն ունեցող համակարգերի վերլուծությունը

Գոյություն ունեն կլաստերային համակարգերի բազմաթիվ սահմանումներ, սակայն, ըստ դասական սահմանման [41], կլաստերը զուգահեռ և տարաբաշխված համակարգի տարատեսակ է, որը բաղկացած է մի քանի համախմբված համակարգիչներից և օգտագործվում է ինչպես մեկ միասնականացված համակարգչային ռեսուրս: Սահմանումից հետևում է, որ կլաստերը որոշ թվով համակարգիչների համախմբություն է, որոնք ղեկավարվում և օգտագործվում են որպես մեկ ամբողջություն: Կլաստերի յուրաքանչյուր հանգույց իրենից ներկայացնում է առանձին համակարգիչ, որտեղ տեղադրված է գործառնական համակարգի (համակարգչային ծրագրերի համախումբ, որը ղեկավարում է համակարգչի ապարատային և ծրագրային ռեսուրսները) անհատական նմուշը: Կլաստերային համակարգերը տարբերվում են միմյանցից մի քանի

[7] https://youtube.com/watch?time_continue=22&v=DGw_2dfcTjQ

ատրիբուտներով, բայց առավել գրավիչ են կլաստերում խնդիրների տարաբաշխման և էներգախնայողության խնիդրներին վերաբերող լուծումները:

Կախված հաշվողական կլաստերներում խնդիրների կազմակերպման եղանակից և ծանրաբեռնվածությունից, տարբեր առաջադրանքներ նպատակահարմար է տեղադրել տարբեր հանգույցներում կամ հանգույցների տարբեր հերթերում: Կլաստերներում արդիական է հանդիսանում հերթերի անհավասարաչափ բաշխումը, կամ որ նույնն է՝ տարբեր հանգույցներում խնդիրների անհավասարաչափ ծանրաբեռնվածությանը վերաբերող լուծումները:

Ավազակույտի մոդելի վրա հիմված կլաստերային համակարգերում խնդիրների և ընհանրապես ավազահատիկների տեղաբաշխմանը և հավասարաչափ ծանրաբեռնվածությանը վերաբերող աշխատություններից են աշխատանքներ[14-15, 42]: Հարկ է նշել սակայն, որ դրանցում, այնուամենայնիվ օգտագործված չեն rotor-router մոդելը և դրա հնարավորությունները, որոնց շնորհիվ կարող են լուծվել այնպիսի խնդիրներ, ինչպիսիք են, օրինակ՝ ցանցի հանգույցների ծանրաբեռնվածության դինամիկ բալանսավորումը, էներգախնայողությունը և այլն, քանի որ տվյալ մոդելը կարողանում է ապահովել մեկ գագաթի շուրջ ավազահատկների կենտրոնացումը:

Եզրակացություն առաջին գլխի վերաբերյալ

Ուսումնասիրելով և վերլուծելով ավազակույտի գոյություն ունեցող մոդելները, կիրառված ալգորիթմները և համապատասխան ծրագրային լուծումները, հստակորեն նշմարվում է նոր մոդելների, ալգորիթմների և գործիքամիջոցների պահանջ, որոնց միջոցով հնարավորություն կընձեռնվի կատարել դիտարկվող մոդելների սիմուլացում, մոդելավորում, երկչափ և եռաչափ տեսաբերում, ինչպես նաև օգտատերերի համատեղ միաժամանակյա գործարկում անկախ աշխարհագրական դիրքից: Բացի այդ պահանջ կա նաև իրականացնել դիտարկվող մոդելների աշխատանքի արդյունավետ զուգահեռացում ժ-չափանի

խորանարդային ցանցերի վրա անկախ մոդելի նախնական վիճակից, ավազահատիկների քանակից և դասավորությունից:

Կլաստերային համակարգերի օրեցօր զարգացումը ստիպում է դրանց հետ առընչվող խնդիրների և դրանց լուծումների շարունակական վերաձևակերպում և լավորակում: Միևնույն ժամանակ, հաշվի առնելով ավազակույտի և rotor-router մոդելների առանձնահատկությունները, մեկ այլ հետաքրքրություն է առաջանում դրանք կիրառել կլաստերային համակարգերում արդի որոշակի խնդիրների լուծման նպատակով:

ԳԼՈՒԽ 2. ԱՎԱԶԱԿՈՒՅՏԻ ՄՈԴԵԼԻ ԱՇԽԱՏԱՆՔԻ ԶՈՒԳԱՀԵՌԱՑՄԱՆ ՄԱԹԵՄԱՏԻԿԱԿԱՆ ՄՈԴԵԼԻ ԱՌԱՋԱԴՐՈՒՄԸ

Ավազակույտի առաջադրված մոդելի մշակման նպատակով անհրաժեշտություն է առաջացել առաջադրել d չափանի խորանարդային ցանցում աստղային ծածկույթները նկարագրող բանաձև և ապահովել այդ բանաձևի անալիտիկ ստացումը [43, 44], ինչպես նաև ավազակույտի մոդելն անվերջ անկայուն վիճակում գցելու համար անհրաժեշտ ավազահատիկների մինիմալ քանակի արժեքի դուրս բերումը երկչափ և եռաչափ խորանարդ ցանցերի դեպքում [45]: Բացի դրանից, առաջարկվել է ներգրավել rotor-router մոդելը կլաստերային համակարգերում առաջադրանքների բաշխիչի նախագծման նպատակով [46]:

2.1. Խորանարդային ցանցերում աստղային ծածկույթները նկարագրող բանաձևը

Դիտարկենք d չափանի ցանց L_d՝ n գծային երկարությամբ, պարբերական սահմանային պայմաններով, որն իրենից ներկայացնում է գրաֆ՝ գագաթների հետևյալ բազմությամբ

$$V = V(L_d) = \{v = (x_1, x_2, \dots, x_d) : x_i = 0, 1, 2, \dots, n-1;\ i = 1, 2, \dots, d\} \quad \textbf{(1)}$$

և ոչ ուղղորդված $E = E(L_d)$ կողերով, որոնք սահմանվում են հետևյալ կանոններով: Յուրաքանչյուր $v = (x_1, x_2, \dots, x_d) \in V$ գագաթ հարևան է $2d$ քանակությամբ գագաթների՝ տրված հարևանության ցուցակին համապատասխան.

$$Adj(v) = \{(x_1, x_2, \dots, (x_k \pm 1)\ mod\ n, \dots, x_d) : k = 1, 2, \dots, d\} \quad \textbf{(2)}$$

Գագաթների և կողերի քանակը համապատասխանաբար հավասար է $|V| = n^d$, $|E| = d * n^d$. Կասենք, որ երկու v_1 և v_2 գագաթներ **անկախ** են, եթե նրանք հարևան չեն և չունեն ընդհանուր հարևան՝ հետևյալ կերպ.

$$v1 \notin Adj(v_2),\ v_2 \notin Adj(v_1)\ and\ Adj(v_1) \cap Adj(v_2) = \emptyset \quad \textbf{(3)}$$

cԱյդ դեպքում, աստղային ծածկույթի խնդիրն իրենից ներկայացնում է L_d ցանցի ծածկումը չհատվող աստղային գրաֆներով:

Դիտարկենք $\boldsymbol{V'}$ աստղային ծածկույթը, որի աստղային գրաֆների կենտրոնական գագաթների բազմությունը բավարարում է հետևյալ 2 պայմաններին.

1. ամբողջական ծածկույթի պայման՝

$$\left(\bigcup_{\boldsymbol{v} \in \boldsymbol{V'}} \boldsymbol{Adj}(\boldsymbol{v})\right) \bigcup \boldsymbol{V'} = \boldsymbol{V}$$

2. չհատման պայման. կամայական երկու $v_1, v_2, \in V'$ անկախ են:

Ակնհայտ է, որ օպտիմալ աստղային ծածկույթ գոյություն ունի այն և միայն այն դեպքում, երբ V-ն անմնացորդ բաժանելի է $2d + 1$-ի վրա:

Դիտարկենք $v = (x_1, x_2, \ldots, x_d) \in V$ գագաթը և $\vec{r} = (p_1, p_2, \ldots, p_d) \in Z$ վեկտորը: Սահմանենք $v + \vec{r}$ գումարման գործողությունը, որի արդյունքը կլինի գագաթ V-ում հետևյալ կոորդինատներով.

$$v + \vec{r} \equiv ((x_1 + p_1) \bmod n, (x_2 + p_2) \bmod n, \ldots, (x_d + p_d) \bmod n) \in V \quad \textbf{(4)}$$

Համապատասխանաբար, $\vec{r}$ վեկտորի և բոլոր $v \in V'$ գագաթների գումարի բազմությունը կստացվի հետևեյալ ենթաբազմությունը V-ից:

$$V' + \vec{r} \equiv \{v + \vec{r} : v \in V'\} \subseteq V \qquad \textbf{(5)}$$

Նկատենք, որ V'-ի ենթաբազմությամբ նկարագրվող աստղային ծածկույթի դեպքում $V' + \vec{r}$-ը նույնպես նկարագրում է աստղային ծածկույթ, որտեղ $\vec{r} \in V'$:

Թեորեմ: Դիտարկենք d չափանի ցանց L_d՝ n գծային երկարությամբ և պարբերական սահմանային պայմանով: Ենթադրենք, որ n-ը անմնացորդ բաժանելի է $2d + 1$-ի վրա: Ապա գոյություն ունի L_d-ի աստղային ծածկույթ, որի

աստղային գրաֆների կենտրոնական գագաթները նկարագրվում են հետևյալ բանաձևով.

$$V' = \{v = (x_1, x_2, \ldots, x_d) : (x_1 + 2x_2 + 3x_3 + \ldots + d\,x_d)\ mod\ (2d + 1) = 0, v \in V\} \quad \textbf{(6)}$$

Ապացույց:

Տրված է $\vec{r} = (p_1, p_2, \ldots, p_d) \in Z$ վեկտորը, սահմանենք $Q(\vec{r})$ ֆունկցիան, որտեղ.

$$Q(\vec{r}) = p_1 + 2p_2 + 3p_3 + \cdots + dp_d:$$

Սահմանենք d չափանի հենքային վեկտորները հետևյալ կերպ.

$$\vec{e_k} = (\underbrace{0, 0, \ldots, 0}_{\text{k-1}}, 1, \underbrace{0, 0, \ldots, 0}_{\text{d-k}}), \quad k = 1, 2, 3, \ldots, d,$$

Այդ դեպքում դատարկ վեկտորը՝ $\overrightarrow{e_0}$-ն, կունենա $\overrightarrow{e_0} = (0,0, \ldots,0)$ տեսքը: Նախ ապացուցենք ամբողջական ծածկույթի պայմանը:

Տրված է կամայական $v \in V \setminus V'$ գագաթ, անհրաժեշտ է ապացուցել, որ գոյություն ունի $u \in Adj(v)$ գագաթ այնպիսին, որը հանդիսանում է աստղային ծածկույթի կենտրոն: Այլ կերպ ասած՝

$$\forall \vec{r} = (p_1, p_2, \ldots, p_d) \in Z \qquad 0 \le p_i \le n - 1$$

վեկտորի համար բավարարվում է հետևյալ 3 պայմաններից մեկը.

1. $Q(\vec{r}) mod (2d + 1) = 0$
2. $\exists k = 1,2, \ldots, d; Q(\vec{r} - \overrightarrow{e_k}) mod (2d + 1) = 0$
3. $\exists k = 1,2, \ldots, d; Q(\vec{r} + \overrightarrow{e_k}) mod (2d + 1) = 0$

Այժմ ենթադրենք, որ $Q(\vec{r})$-ը ունի հետևյալ տեսքը.

$$Q(\vec{r}) = (2d + 1)s + t, 0 \le t < 2d + 1, s \ge 0$$

Ընդունենք, որ k –ի արժեքը հետևյալն է,

$$k = \begin{cases} 0 & \text{եթե } t = 0; \Rightarrow Q(\vec{r}) mod(2d+1) = 0 \\ t & \text{եթե } t < d; \Rightarrow Q(\vec{r} - \overrightarrow{e_k}) mod(2d+1) = 0 \\ 2d+1-t & \text{եթե } t > d; \Rightarrow Q(\vec{r} + \overrightarrow{e_k}) mod(2d+1) = 0 \end{cases},$$

որն էլ հենց ապացուցում է ամբողջական ծածկույթի լինելիության պայմանը:

Այժմ ապացուցենք ծածկույթի կենտրոնական գագաթների չհատման պայմանը, որը պնդում է, որ

$\forall \vec{r} = (p_1, p_2, \dots, p_d) \in Z\ 0 \le p_i \le n-1, i = 1,2, \dots, d$, վեկտորի համար, որը բավարարում է հետևյալ 2 պայմաններին.

1. $Q(\vec{r}) mod(2d+1) = 0$
2. $\forall \Delta\vec{r} = (\Delta p_1, \Delta p_2, \dots, \Delta p_d)$, որի համար ճիշտ է հետևյալ արտահայտությունը՝ $0 < |\Delta p_1| + |\Delta p_2| + \cdots + |\Delta p_d| \le 2$:

Հետևաբար կունենանք հետևյալ անհավասարությունը.

$$Q(\vec{r} + \Delta\vec{r})\ mod\ (2d+1) \neq 0:$$

Քանի որ Q-ն գծային է, հարկավոր է ապացուցել, որ

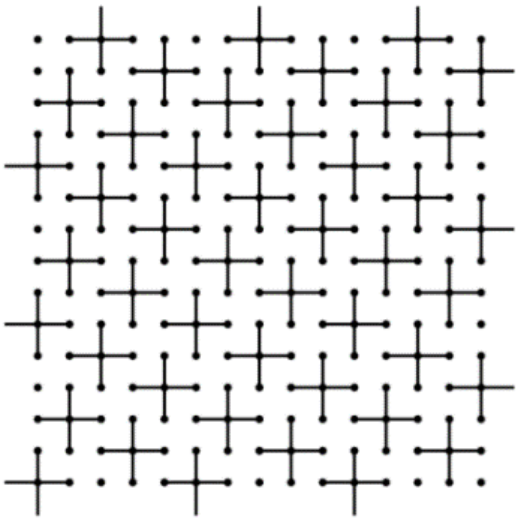
$$Q(\Delta\vec{r})\ mod\ (2d+1) \neq 0:$$

Վերոհիշյալ արտահայտությունը հավասարազոր է $0 < |Q(\Delta\vec{r})| < 2d+1$ անհավասարության ապացուցմանը: Այդ նպատակով առանձնացվում են 3 հնարավոր դեպք, որոնք բավարարում են 2րդ պայմանին.

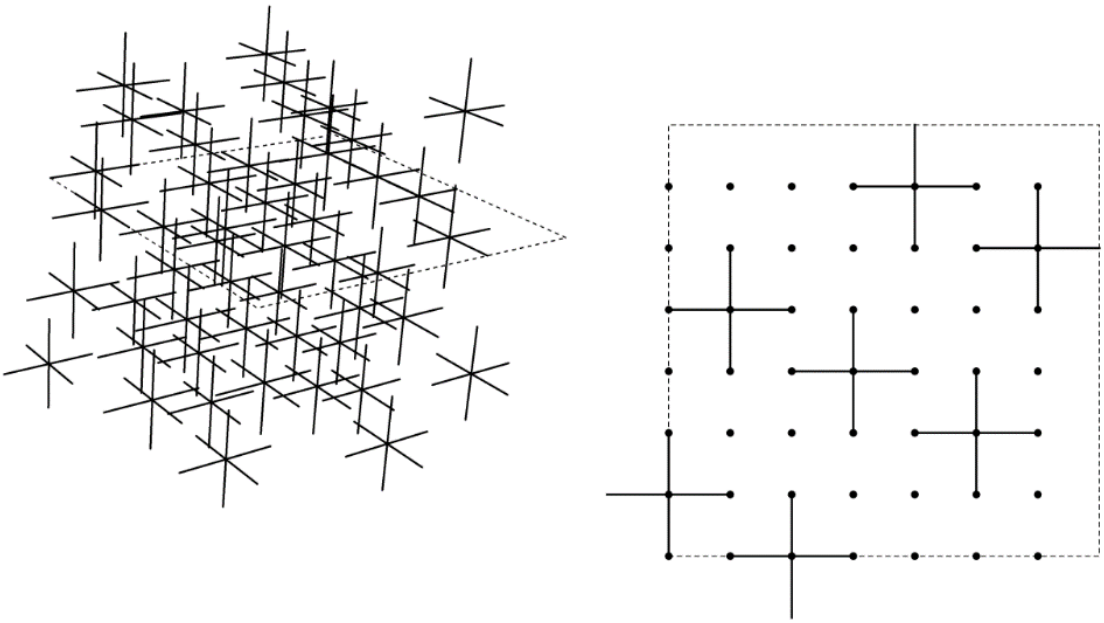
1. $\exists k_0 = 1,2, \dots, d \quad \Delta p_{k_0} = \pm 2 \quad \Delta p_s = 0 \quad \forall s \neq k_0$, հետևաբար.
$$0 < |Q(\Delta\vec{r})| = 2k_0 < 2d+1$$
2. $\exists k_0 = 1,2, \dots, d \quad \Delta p_{k_0} = \pm 1 \quad \Delta p_s = 0 \quad \forall s \neq k_0$
$$0 < |Q(\Delta\vec{r})| = k_0 < 2d+1$$
3. $\exists k_1, k_2 = 1,2, \dots, d;\ k_1 \neq k_2 \quad \Delta p_{k_1} = \pm 1 \quad \Delta p_{k_2} = \pm 1 \quad \Delta p_s = 0 \quad \forall s \neq k_1, k_2$

$$0 < |Q(\Delta\vec{r})| = \left|k_1 \Delta p_{k_1} + k_2 \Delta p_{k_2}\right| \leq k_1 + k_2 < 2d + 1$$

Մասնավորապես, երկչափ, եռաչափ և քառաչափ ցանցերի դեպքում ունենք Նկար 13-ում և 14-ում բերված պատկերները:



Նկար 13.Երկչափ ցանցում ծածկույթների օրինակ:



Նկար 14.Եռաչափ և քառաչափ ցանցերում ծածկույթների օրինակ, z=4 առանցքը պատկերված է առանձին:

Այսպիսով, ունենալով L_d՝ n գծային երկարությամբ d չափանի ցանցի ծածկույթները, կարող ենք տվյալ ցանցում առանձնացնել անկախ գագաթների

բազմություններ, և ամեն բազմության մեջ կարող ենք փլուզումները կատարել զուգահեռ, համոզված լինելով, որ դրանք չեն առաջացնի միևնույն հիշողության տիրույթին միաժամանակ դիմման խնդիրներ:

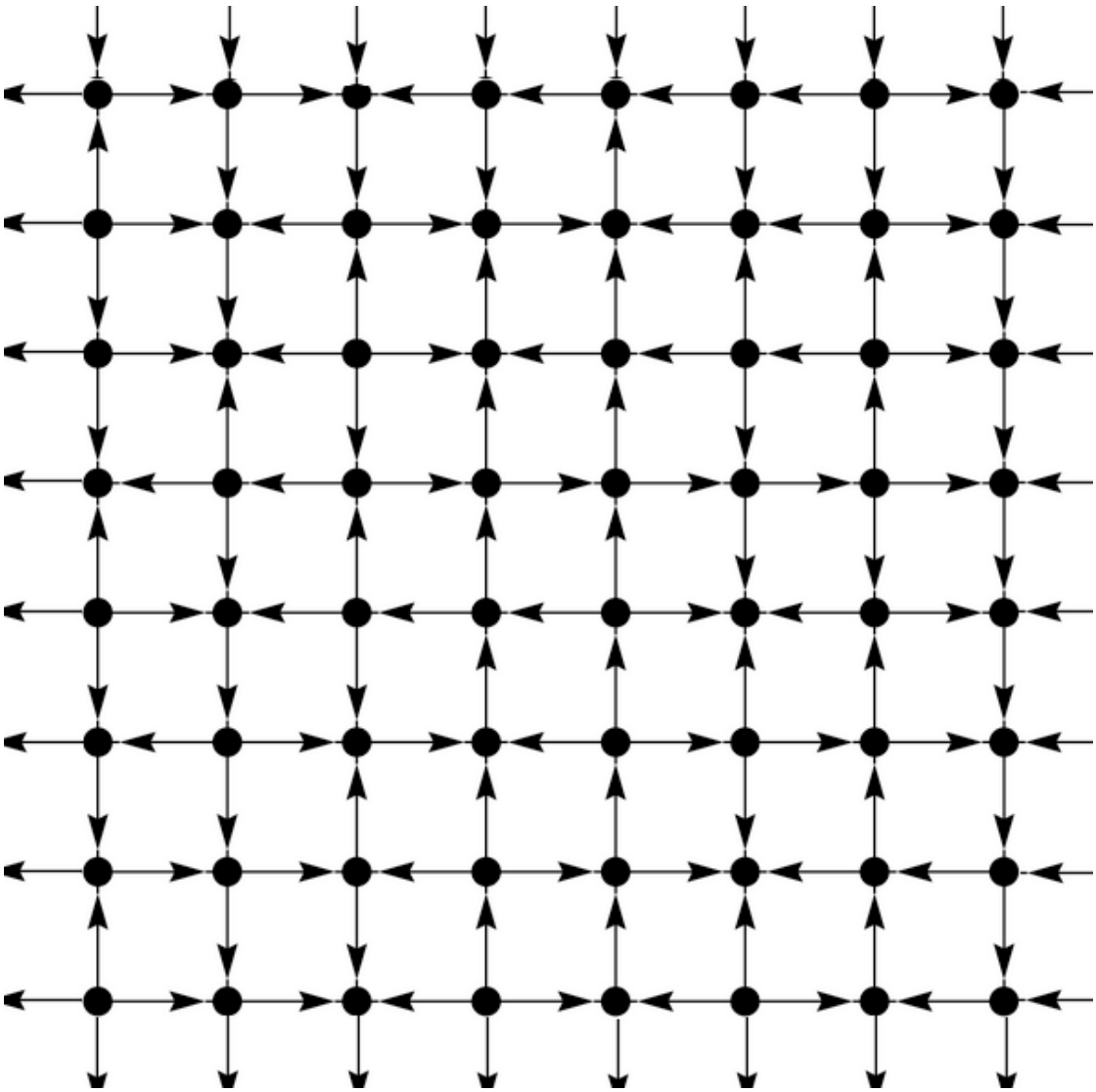
2.2. Ավազահատիկների քանակի ստորին գնահատականը փակ եզրերով եռաչափ ցանցում անվերջ անկայուն վիճակի հասնելու համար:

Դիտարկենք ավազակույտի մոդելը երկչափ ուղղորդված n-գծային երկարությամբ ցանցային գրաֆի վրա, որտեղ կողերի ուղղվածությունը դասավորված է պատահականության սկզբունքով և մոդելում գոյություն չունեն փոսեր (Նկար 15):

Սահմանում 8: D_i-ով նշանակենք կողերի քանակը ուղղված դեպի V_i գագաթ:

Սահմանում 9: Կասենք, որ մոդելն **անվերջ անկայուն վիճակում** է, եթե այն երբեք չի կայունանում: Նման իրավիճակ հնարավոր է միայն մոդելի փակ լինելու պարագայում, երբ մոդելում փոսեր չկան: Իսկ ստուգելու համար, արդյոք մոդելը հայտնվել է արդեն անվերջ անկայուն վիճակում թե ոչ, կարելի է միայն համոզվելով, որ փլուզման ալիքի ընթացքում ամեն գագաթ փլուզվել է գոնե մեկ անգամ:

Ավազահատիկների քանակի ստորին գնահատականի ստացման համար դիտարկենք մոդել և փորձենք հնարավորինս քիչ ավազահատիկներով բերել նրան անվերջ անկայուն վիճակի: Դրա համար դիտարկենք կամայական մոդելի օրինակ, որը դիտարկված կլինի ուղղորդված գրաֆի վրա ինչպես պատկերված է Նկար 15-ում:



Նկար 15. Ուղղորդված քառակուսային ցանց:

Վերոնշյալ ուղղորդված գրաֆում բացառենք ցիկլերի գոյությունը: Ավազակույտի մոդելը դիրտարկված տվյալ գրաֆի վրա կընկնի անվերջ անկայուն վիճակ, եթե ամեն V_i գագաթ սկզբանարժեքավորվի $4 - D_i$ ավազահատիկներով: Ընդհանուր մինիմում ավազահատիկների $C^2{}_{unlimited}$ քանակը մոդելը անվերջ անկայուն վիճակում գցելու համար կստացվի $C^2{}_{unlimited} = 4 * n^2 - \sum_{i=0}^{n^2-1} D_i$, որն էլ իր հերթին հավասար է

$$C^2{}_{\text{unlimited}} = 2 * n^2:$$

Եռաչափ փակ ցանցի դեպքում նույն դատողություններով մինիմում ավազահատիկների քանակը մոդելը անվերջ անկայուն վիճակում գցելու համար կստանանք $C^3_{\text{unlimited}} = 3 * n^3$: Այս գնահատականն օգտագործվել է բաժին 3.2-ում նկարագրված «լուրջ խաղի» օրինակում հաղթողի ճանաչմանը անհրաժեշտ քայլերի քանակի նվազեցման նպատակով:

2.3. Rotor-router համակարգի առանձնահատկությունները

Դիտարկվող rotor-router մոդելը օժտված է մի շարք առանձնահատկություններով, որոնցից ամենաէականը ավազահատիկների տեղաբաշխմանը վերաբերող հատկությունն է: Հիմնվելով Պրյեժեվի [47, 48] և Դհարի [49] գիտական աշխատանքների վրա, համոզվում ենք, որ rotor-router մոդելում ավազահատիկների տեղաբաշխումը տեղի է ունենում հավասարաչափ տարբեր տոպոլոգիական տարածություններում:

Մեր կողմից այս առանձնահատկությունն օգտագործվել է կլաստերային համակարգերի, ավազակույտի և rotor-router մոդելների աշխատանքային սկզբունքներով օժտված, սիմուլատորի բաշխիչի ստեղծման համար, որտեղ ավազահատիկները փոխարինվել են խնդիրներով [46]: Բացի այդ, առաջ է քաշվել հիպոթեզ, որ rotor-router մոդելի հիման վրա ստեղծված կլաստերային համակարգում խնդիրների հավասարաչափ բաշխվածություն կապահովվի ոչ միայն ստատիկ, այլև դինամիկ խնդիրների դեպքում, որը տեսանելի է մեր կողմից ստեղծած վերոնշյալ բաշխիչի միջոցով: Ընդսմին, դինամիկ խնդիրներ ասելով հասկանում ենք, որ յուրաքանչյուր խնդիր ունի կատարման անհրաժեշտ ժամանակ, և ավարտվելուն (կատարվելուն) պես ազատում է զբաղեցրած հանգույցը:

Որպես օրինակ դիտարկենք երկչափ քառակուսային ցանցի տեսք ունեցող կլաստերային համակարգը: Վերցնելով որևէ կենտրոնական գագաթ, և խնդիրները անընդհատ ավելացնելով տվյալ գագաթից, նկատում ենք, որ խնդիրները

դասավորվում են տվյալ կետին մոտ գտնվող հանգույցներում՝ սույնով ապահովելով տվյալ հանգույցներում հավասարաչափ ծանրաբեռնվածություն, իսկ ավելի հեռու գտնվող հանգույցներն հնարավորության դեպքում ազատ են մնում:

Եզրակացություն երկրորդ գլխի վերաբերյալ

Ունենալով d չափանի խորանարդային ցանցերում աստղային ծածկույթները նկարագրող բանաձևը, հնարավորություն ունենք նույն տոպոլոգիական տարածությունում առանձնացնել անկախ գագաթներ, որոնք հարևան չլինելով և ընհանուր հարևան չունենալով հնարավորություն կտան կատարել տվյալ գագաթների զուգահեռ փլուզում տարբեր հոսքերում: Սույնով բացառվում է համակարգի աշխատանքի խափանումը կամ հոսքերի ռեսուրսաքաղցը, միևնույն ժամանակ բացառելով տարբեր հոսքերի կողմից հիշողության միևնույն տիրույթին դիմելը:

Ավազակույտին անվերջ անկայուն վիճակի հասցնելու համար անհրաժեշտ ավազահատիկների ստորին գնահատականը կարելի է օգտագործել «Լուրջ խաղերի» տեսության մեջ նոր, ավազակույտի մոդելի վրա հիմված, «լուրջ խաղի» օրինակի ստեղծման մեջ:

Rotor-router համակարգի ներկայացված առանձնահատկությունները բավարար հատկանիշներ ունեն այն կլաստերային համակարգում բաշխիչի աշխատանքում ընգրկելու, ինչպես նաև կլաստերային համակարգում արդի մի շարք խնդիրների լուծման համար, ինչպիսիք են, օրինակ՝ առաջադրանքների համասեռ բաշխվածությունը և էներգախնայողությունը:

ԳԼՈՒԽ 3. ՄՇԱԿՎԱԾ ԾՐԱԳՐԱՅԻՆ ՓԱԹԵԹՆԵՐԻ ՆԿԱՐԱԳՐՈՒԹՅՈՒՆԸ

Այս գլուխը նվիրված է մշակված ծրագրային փաթեթների, իրականացման համար ընտրված միջավայրի, ծրագրավորման լեզվի, ալգորիթմական աշխատանքների և օգտագործման սկզբունքների նկարագրությանը:

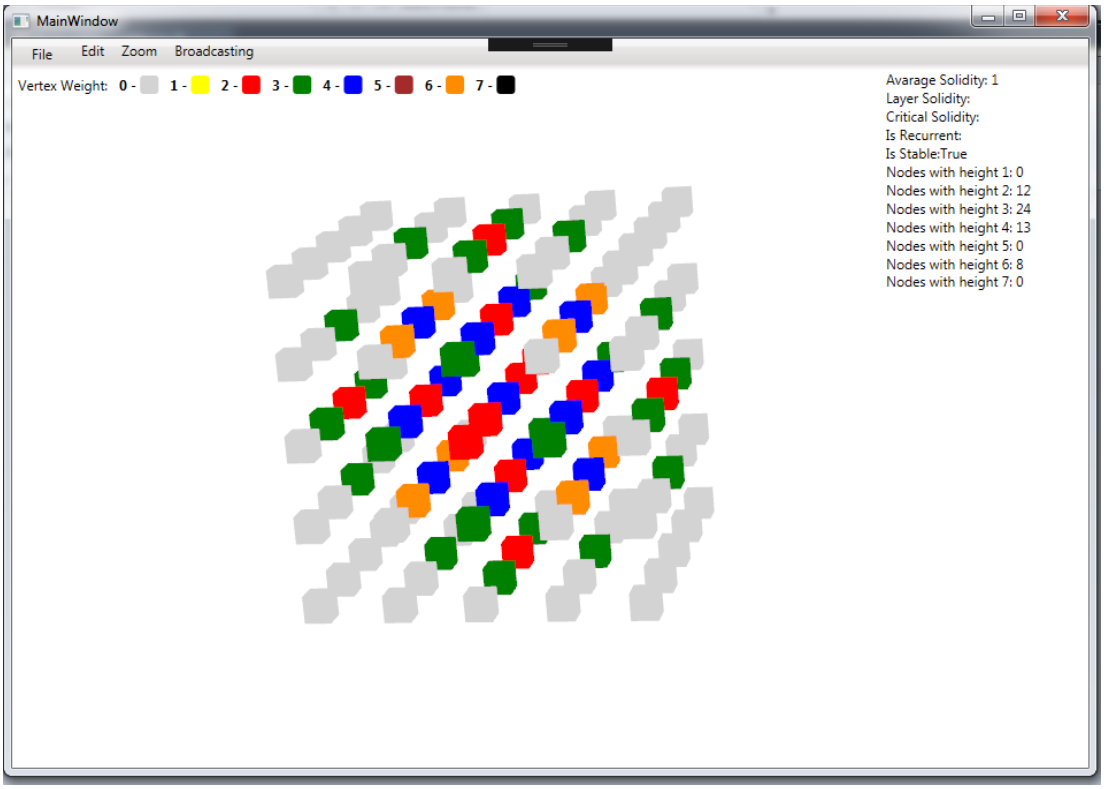
Մասնավորապես, մշակված բազմաօգտատեր «CA Simulator» համակարգը, նախատեսված լինելով ինքնակազմակերպվող համակարգերի հետազոտման նպատակով և միավորելով արդի ծրագրային փաթեթների առավելությունները մեկ տեղում, բավարարում է ինչպես լոկալ այնպես էլ վիրտուալ լաբորատորիաների հետազոտողների հիմնական պահանջներին [50]: Իսկ SandGame [45] «լուրջ խաղի» օրինակը արդեն իր հարուստ ֆունկցիոնալ ստեկով և ժամանակակից լուծումներով գրավիչ հարթակ է հանդիսանում խաղացողներին ինքնակազմակերպվող կրիտիկականության կոնցեպտը դյուրին ընկալելու համար:

NeurophStudio [51] միջավայրի միջոցով ստեղծված նեյրոնային ցանցերը հնարավորություն կտան որոշ դեպքերում ավազակույտի մոդելի սիմուլացիան փոխարինել համապատասխան նեյրոնային ցանցով ժամանակային շահույթի համար:

SandScheduler [46] համակարգը լինելով կլաստերային համակարգի սիմուլյատոր, հիմված ավազակույտի և rotor-router մոդելների վրա, իր տեսաբերման, ատրբիուտների հաշվարկման և սիմուլացիոն ճկուն հնարավորությունների հետ միասին նպատակահարմար միջոց է հանդիսանում վերոնշյալ մոդելների աշխատանքային ալգորիթմի վրա հիմված բաշխիչի աշխատանքի, թերությունների և առավելությունների հետազոման համար:

3.1. Բազմաօգտատեր «CA Simulator» համակարգի նկարագրությունը

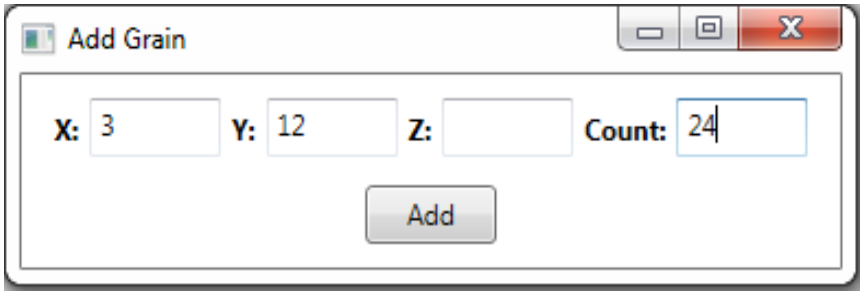
Նախագծված է «CA Simulator» համակարգը [50], որը նախատեսված է ինքնակազմակերպվող համակարգերի հետազոտման համար (Նկար 16): Նախագիծը հիմնված է ավազակույտի աբելյան մոդելի վրա, որը ներկայացնում է ինքնակազմակերպվող համակարգի օրինակ: «CA Simulator» համակարգը, նպատակաուղղված լինելով բազմաօգտագործողների համար, հնարավորություն է ընձեռում կատարել հետազոտություններ երկչափ և եռաչափ տարածություններում՝ համատեղ կատարվող հետազոտական աշխատանքների միաժամանակյա դիտման և փոփոխությունների կատարման հնարավորություններով:



Նկար 16.CA Simulator միջավայրը ավազակույտի աբելյան մոդելի օրինակով:

Ավազահատիկների ավելացումը հնարավոր է կատարել ինչպես կոնկրետ կետի, այնպես էլ շերտի կամ ամբողջ մոդելի վրա ընտրելով մեկ կետին բաժին ընկնող ցանկալի քանակությամբ ավազահատիկների քանակը: Նկար 17-ում պատկերված է օգտագործողի պատուհան, որտեղ օգտագործողը կարող է նշել

ավազահատիկների ավելացման ցանկալի կոորդինատը և ավազահատիկների քանակը: Նշեմ որ կոորդինատներից որևէ մեկի բացակայության դեպքում նշված ավազահատիկները ավելացնում են բացակայող կոորդինատի երկայնքով:

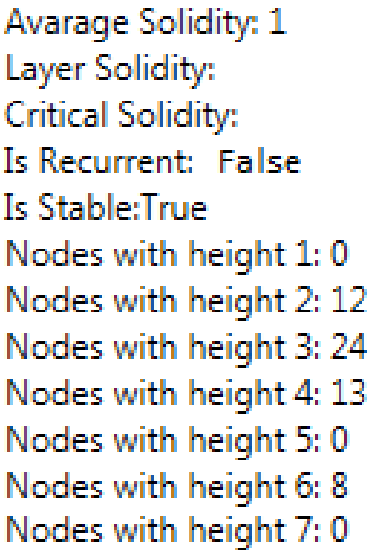


Նկար 17.Ավազահատիկների ավելացման պատուհան ցանկալի կոորդինատներով և քանակով:

«CA Simulator» համակարգն օժտված է նաև տեսաբերվող մոդելի դիտման անկյունների փոփոխության, ինչպես նաև մոտարկման և հեռարկման (zooms in/out) հնարավորություններով: Նշենք, որ զուգահեռ աշխատանքի ժամանակ օգտագործողներից մեկի դիտման անկյունի փոփոխությունը կամ մոտարկումը ազդեցություն չի ունենում մյուս օգտագործողների դիտման անկյունների և դիտարկվող մոտարկման հեռավորության վրա: Օգտագործողները հնարավորություն ունեն նաև դիտարկել մոդելի ենթամասեր առանձին, ինչպիսիք են օրինակ եռաչափ ցանցի շերտերը: «CA Simulator»-ը ընձեռնված լինելով մոդելների վիճակների պահպանման և վերբեռնման հնարավորությամբ՝ իրականանալի է դարձնում կոնկրետ օգտագործողի զուգահեռ հետազոտությունների կատարումը տարբեր մոդելների վրա:

Համակարգում նախատեսված է նաև մոդելի տարբեր ֆիզիկական և ինֆորմացիոն բնութագրիչների տեսաբերման հնարավորություն իրական ժամանակում, որն ավելի նպատակահարմար է դարձնում «CA Simulator» համակարգի օգտագործումը: Հաշվարկվող բնութագրիչներն են օրինակ մոդելի

միջին խտությունը, շերտի խտությունը, կրիտիկական խտությունը, մոդելի ռեկուրենտ վիճակում լինել/չլինելը, կայուն լինել/չլինելը, կոնկրետ հզորությամբ հանգույցների քանակները և այլն (Նկար 18):



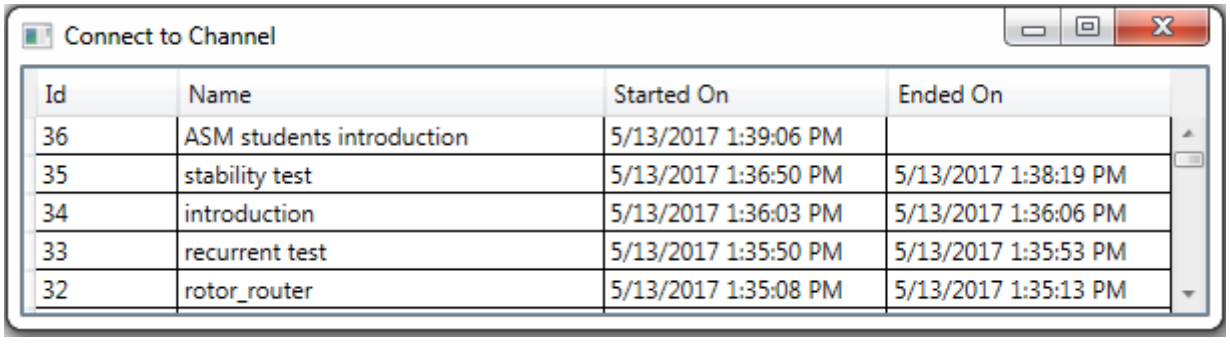
Նկար 18.CA Simulator-ում հաշվարկվող ատրիբուտների ցուցակ:

Նոր մոդել ստեղծելու համար օգտագործողը պետք է ընտրի "File" օպցիան վերին վահանակից, այնուհետև ընտրում է "New Sandpile Model" տարբերակը, ապա մուտքագրում է ցանցային գրաֆի գծային երկարությունը, որի վրա դիտարկվելու է ավազակույտի մոդելը: Փոփոխությունների համար, ինչպիսիքն են օրինակ ավազահատիկ ավելացնելը, օգտագործողը պետք է ընտրի "Edit" օպցիան վերին վահանակից, այնուհետև "Add Grain" տարբերակը, որից հետո կհայտնվի Նկար 19-ում պատկերված պատուհանը: Նշեմ նաև, որ ավելացվող ավազահատիկների քանակության որևէ սահմանափակում չկա:

Ինչպես արդեն նշվել է, «CA Simulator» համակարգը ապահովում է բազմաօգտատեր շահագործում, որը հնարավոր է եղել ստանալ Microsoft .Net [52] միջավայրի միջոցով, որը ինքնին հնարավորություն է տալիս ստեղծել ծրագրային փաթեթներ, որոնց միջոցով կարելի է կապել ինչպես օգտագործողներին, այնպես էլ տարբեր համակարգեր և սարքեր, դրանով իսկ դյուրացնելով ինֆորմացիայի

կիսումը/sharing: Օգտագործելով .Net միջավայրը, «CA Simulator» համակարգում մշակվել է հաճախորդ-սերվեր-հաճախորդ փոխկապակցվածությունը՝ նպատակ ունենալով հնարավորություն տալ օգտագործողներին կատարել համատեղ հետազոտություններ միևնույն մոդելների վրա: Սերվերի տեղակայման համար ընտրվել են Microsoft Azure [53] ամպային տեխնոլոգիաների հարթակը:

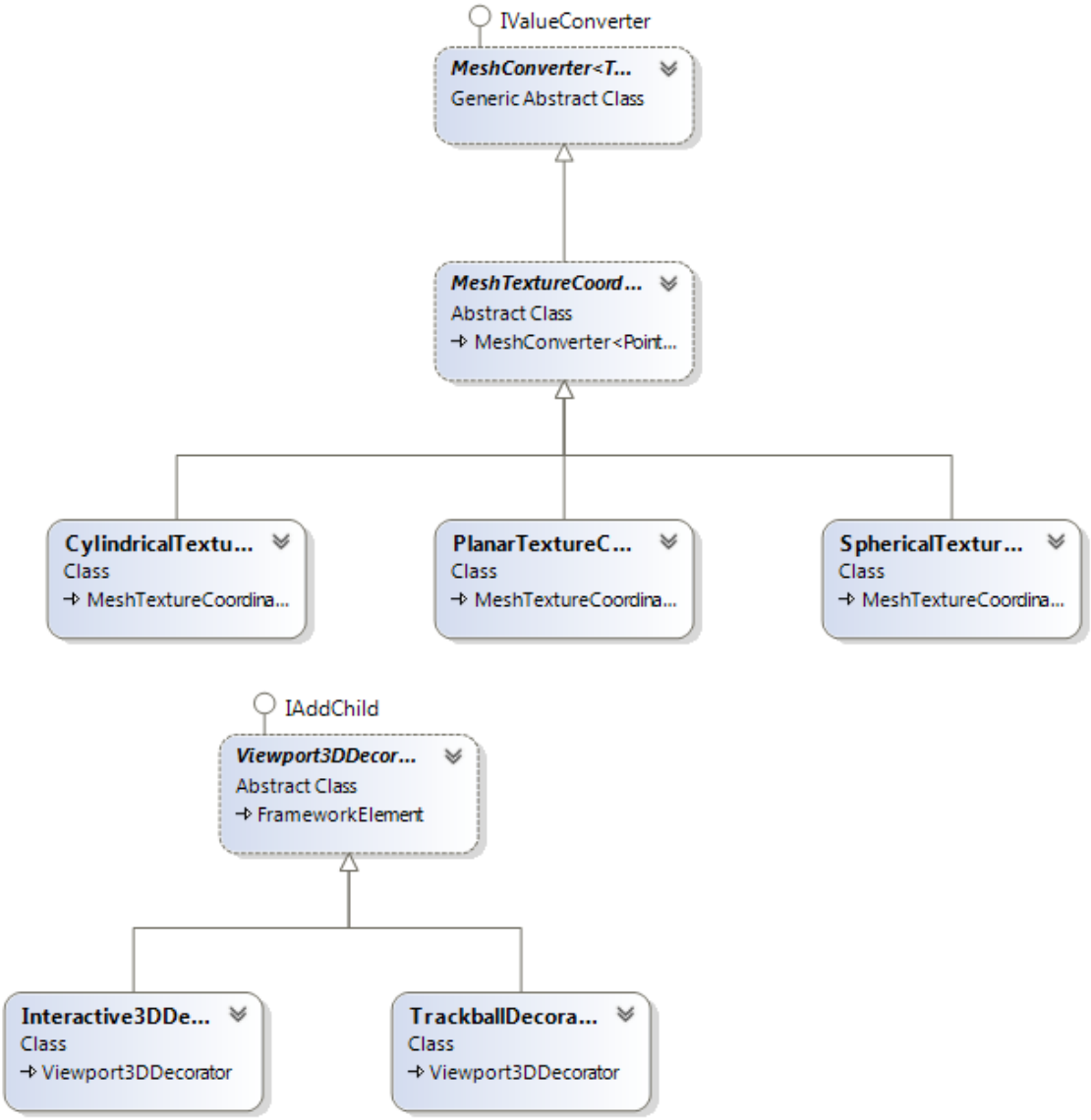
Համատեղ հետազոտություն իրականացնելու համար օգտագործողը պետք է ստեղծի մոդել, որի վրա որ կատարվելու է հետագա համատեղ հետազոտությունը, այնուհետև պետք է կիսվի ստեղծած մոդելով, որի համար ընտրում է «Broadcasting» օպցիան վերին վահանակից, այնուհետև ընտրում է «Start Broadcasting» տարբերակը՝ մուտքագրելով ալիքի անվանումը: Մինչդեռ մյուս օգտագործողները բացելով ալիքների ցուցակը «Broadcasting» օպցիայից «Connect to Chanel» տարբերակը ընտրելով, տեսնում են ալիքների պատմությունը ցուցադրող պատուհանը (նկար 19), որտեղ կարող են ընտրել ցանկալի ալիքը տվյալ ցանկից և միանալ դրան: «CA Simulator» համակարգի շատ կարևոր առավելություններից է նաև այն, որ այն պահպանում է մոդելի վրա կատարված հետազոտության ընթացքում կիրառված բոլոր փոփոխությունները, և հնարավորություն տալիս ալիքին ուշ միացած օգտագործողներին տեսնել դրանք քայլ առ քայլ, ամեն քայլի հետ թարմացնելով բնութագրիչների հաշվարկը:



Id	Name	Started On	Ended On
36	ASM students introduction	5/13/2017 1:39:06 PM	
35	stability test	5/13/2017 1:36:50 PM	5/13/2017 1:38:19 PM
34	introduction	5/13/2017 1:36:03 PM	5/13/2017 1:36:06 PM
33	recurrent test	5/13/2017 1:35:50 PM	5/13/2017 1:35:53 PM
32	rotor_router	5/13/2017 1:35:08 PM	5/13/2017 1:35:13 PM

Նկար 19.Ալիքների պատմությունը ցուցադրող պատուհան:

Ինչպես արդեն նշվեց, «CA Simulator» համակարգը մշակվել է օգտագործելով .Net միջավայրը և C# լեզուն: Իսկ գլոբալ ցանցում աշխատանքի ապահովման համար ընտրվել է Microsoft Azure-ը: Ծրագրավորողների տեսանկյունից «CA Simulator» համակարգը կարող ենք բաժանել 3 մոդուլի. տեսաբերում, լոկալ սիմուլացիա և կլիենտ-սերվեր ճարտարապետություն: Տեսաբերման, պտույտների և դիտարկման հեռավորության ապահովման համար օգտագործվել են հենց .Net միջավայրի գրադարանները: Տեսաբերող դասը կունենա գծապատկեր 1-ի տեսքը:



Գծապատկեր 1.Տեսաբերող դասերի կառուցվածքային նկարագրությունը:

Լոկալ սիմուլացիայի մոդուլում նախագծվել է **GuiHelper** դասը, որը ապահովում է մոդելի ստեղծումը, տեսաբերումը, տեսաբերման ֆունկցիոնալ բոլոր հնարավորությունները, պահպանումը, վերբեռնումը, ավազահատիկների ավելացումը, գագաթների փլուզումը, ինչպես նաև ատրիբուտների հաշվարկը: Տվյալ դասի ֆունկցիոնալ ստեկը ունի Նկար 20-ի տեսքը:

```
public static class GuiHelper
{
event EventHandler GrainAdded;
Viewport3D _mainViewPort;
int _size;
List<InteractiveSphere> _points;
List<InteractiveSphere> Points

//Initialize 3D view
public static void Init(Viewport3D vp);

//Creates model with given sizes
public static void CreateModel(Size size);

//Draws Sandpile model
public static void DrawSandpileModel();

//Add grain on Sandpile model
public static void AddGrain(Position pos);

//Adds grain from visual aspects
private static void addGrainOnVertex
        (InteractiveSphere point);

//Returns color regarded to grains count
private static Brush GetColorByWeight
        (int weight);

#region File/String IO

//Save model in file
public static void WriteToFile() {}

//Load model from file
public static void LoadFromFile(){}

#endregion

}
```

Նկար 20.GuiHelper դասի նկարագրություն:

Սերվեր-կլիենտ/ Service-client մոդուլի ճարտարապետության շրջանակներում ունենք BroadcastingHelper դասը (Նկար 21), որը պարունակում է անհրաժեշտ ֆունկցիաները հեռարձակող-բաժանորդ կապի իրականացման համար:

Նախագծված ճարտարապետության հիմքում ընկած է այն գաղափարը, որ հեռարձակողը ինքնին հանդիսանում է նաև բաժանորդ նույն ալիքի համար, որը հնարավորություն է ընձեռում այլ օգտվողների փոփոխությունների ընդունումը նաև հեռարձակողի կողմից: Ինչպես նշվեց տվյալ ալիքում պահպանվում է բոլոր բաժանորդների կողմից կատարված բոլոր փոփոխությունները, որի համար ստեղծվել է տվյալների հենք, որի իրականացման համար ընտրվել է SQLite- ը:

```
public static class BroadcastingHelper
{
public static long SelfChannelId;
public static long SubscribedChannelId;
private static long LastActionId;
private static Timer timer;
private static ActionModel locker;
public static EventHandler<> ChannelClosed;

//Starts to listen to the given channel
public static void ListenChannel
        (ChannelModel channel);

//Disconnects from channel if it's closed
static void timer_Elapsed
        (object sender, ElapsedEventArgs e)

//Ends broadcasting
public static void EndBroadcasting();

//Disconnects from channel
public static void DisconnectFromChannel();


public interface ISeService
{
[OperationContract]
long StartBroadcasting(string name);

[OperationContract]
void EndBroadcasting(long id);

[OperationContract]
void AddAction(long channelId,
        ActionType type, string data);

[OperationContract]
ActionModel GetNextAction(long channelId,
        long lastActionId);

[OperationContract]
List<ChannelModel> GetActiveChannels();
}
```

Նկար 21.BroadcastingHelper դասի նկարագրություն:

Ինչպես արդեն նշվեց, CA simulator համակարգը ստեղծվել է ավազակույտի մոդելի օրինակով: Առկա են երկու հիմնական գործառույթներ կապված ավազակույտի հետ.

- DrawSandpileModel (), որն ապահովում է ավազակույտի տեսաբերումը, մոդելի փոփոխությունների արտացոլումը և
- addgrain(Position pos) ֆունկցիան, որը օգտագործողին հնարավորություն է տալիս կատարել փոփոխությունները ավազակույտի մոդելի վրա:

Նկատենք, որ հավելյալ բջջային մոդելի մշակումը նույնպես դժվարություն չի ներկայացնում. պարզապես կատարվում է տեսաբերման եւ մոդելի փոփոխման գործառույթների փոփոխություն: Իսկ արդեն տեղեկությունների փոխանակման համար նոր բջջային ավտոմատի դեպքում անհրաժեշտ է ISeService ինտերֆեյսի մի քանի գործառույթների ադապտացում տվյալ մոդելին SeService դասում: CA simulator ծրագրային համակարգի աղբյուրները (source code) կարելի է գտնել Bitbucket-ում url {https: //nhayk@bitbucket.org/nhayk/ca_simulator.git} հղումով:

3.2. Բազմաօգտատեր SandGame «լուրջ խաղի» օրինակը

Նշենք, որ «Լուրջ խաղերը» հստակ սահմանում չունեն, և դրանց յուրաքանչյուր մշակող յուրովի է սահմանում տվյալ դասի խաղերը [54-55]: Այնուամենայնիվ, տարածված կարծիքի համաձայն «լուրջ խաղերը» (թվային) խաղերն են, որոնք օգտագործվում են ոչ միայն որպես զվարճանք, այլև լուրջ գիտական հետազոտություններ և ուսուցողական նպատակներ են հետապնդում: Բացի նրանից, որ լուրջ խաղերը հնարավորություն են ընձեռում օգտվողներին փորձարկել ժամանակի, անվտանգության, գումարային արժեքի և այլ պատճառներով իրական կյանքում անհավանական իրավիճակներ, նրանք կարող են նպաստել խաղացողների մի շարք տարբեր հմտությունների զարգացմանը: Սակայն սա չի նշանակում, որ բոլոր խաղերը նպատակահարմար են բոլոր վերոհիշյալ

գործընթացներում: Լուրջ խաղերի օգտագործման վերաբերյալ գիտական ուսումնասիրություններ կատարվել են [56]-ում:

Գոյություն ունեն «Լուրջ խաղերի» հայտնի մի քանի օրինակներ որոնք, համարվում են, որ փոխել են աշխարհը: Դրանցից են՝ «MICROSOFT FLIGHT SIMULATOR (1982)»-ը [57], TILTFACTOR LABORATORY-ի կողմից ստեղծած խաղերը [58], FORCE MORE POWERFUL (2006) [59] և այլն (Նկար 22):

Թռիչքի սիմուլատորները հանդիսանում են լուրջ խաղերի նախապապերը, իսկ MICROSOFT FLIGHT SIMULATOR-ը նրանցից ֆինանսապես ամենահաջողվածն է: 1982 թվականից ի վեր, Microsoft Flight Simulator- ը մշակվել է որպես քաղաքացիական ավիացիայի համապարփակ մոդելավորում, և մեկն է գոյություն ունեցող ստվար քանակով ոչ մարտական սիմուլատորների մեջ:

Նկար 22. Հայտնի «լուրջ խաղերի» օրինակներ:

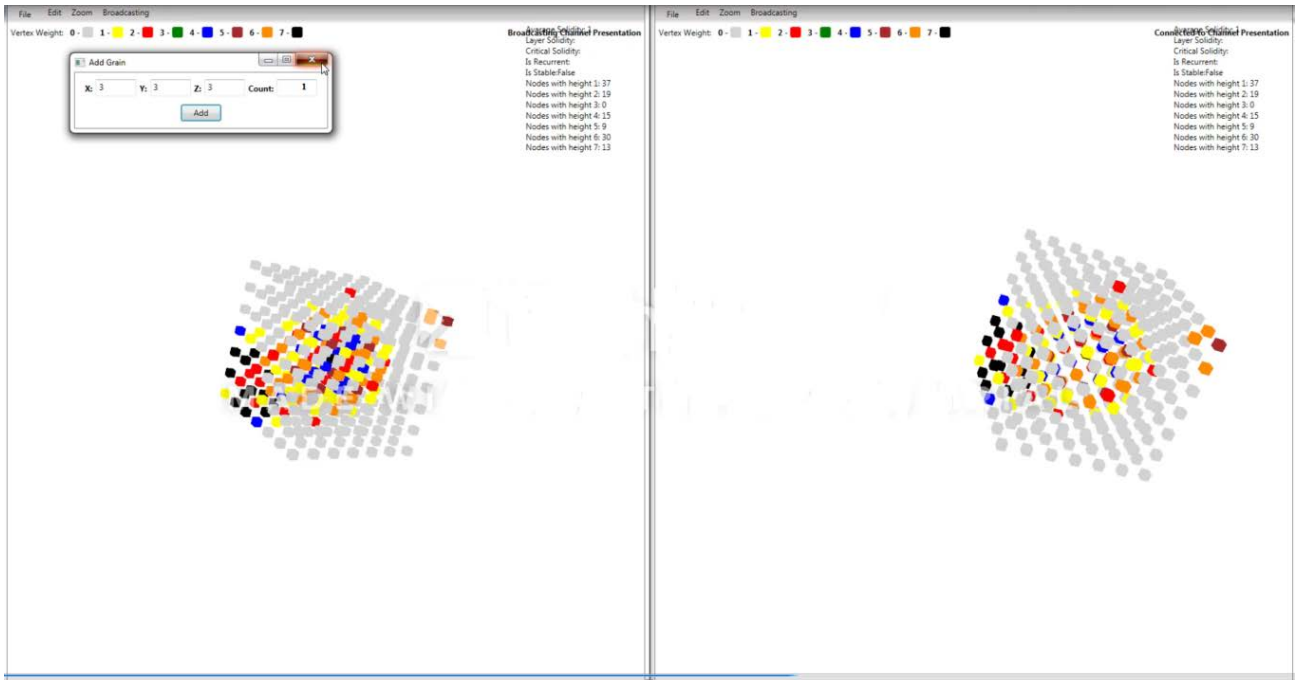
2003թ.-ին հիմնադրված լուրջ խաղերի հետազոտական կենտրոնը՝ Tiltfactor լաբորատորիան, վերջին տարիներին հաջողություններ գրանցեց իրենց նորարարական քարտային խաղերով: Ընկերության կարգախոսն է «Խաղերի դիզայնը սոցիալական փոփոխության համար», իսկ Pox և Awkward Moment լուրջ խաղերի միջոցով, նրանք ուսուցանում են խաղացողներին այնպիսի լուրջ թեմաներ, ինչպիսիք են հակավակցինային շարժումները և սոցիալական կարծրատիպերից խուսափումը:

1999թ.-ին PBS- ը թողարկեց «Force More Powerful» ֆիլմը ոչ բռնի դիմադրության մասին: Այնուհետև Breakaway Games- ը ստեղծեց վիդեո խաղ՝ հիմնված Սերբիայի Otpor շարժման վրա: Խաղը նախատեսված էր սովորեցնել վարել հակամարտություն օգտագործելով ոչ բռնի մեթոդներ:

Առաջարկված «CA simulator» ծրագրային համակարգում ավելացվել է նոր մոդուլ, որը հանդիսանում է «լուրջ խաղի» օրինակ SandGame անվանմամբ: «SandGame»-ը [45] իրականավել է հիմնվելով ավազակույտի աբելյան մոդելի վրա՝ նպատակ ունենալով խթանել ինքնակազմակերպվող համակարգերի հետազոտումը և դյուրըմբռումը, և ունի լուրջ մտածված կրթական նպատակ, հետևաբար չի նախատեսվում օգտագործել միայն զվարճանքի համար, մինչդեռ այն բավականին գրավիչ է զվարճանքի տեսանկյունից: Իրականացված խաղի մոդելի գաղափարը հիմնված է ավազակույտի մոդելին առնչվող տարբեր թեորեմների և տեսությունների վրա, որոնք իրենց հերթին բարելավվում են ուսուցման խորությունը և բարձրացնում են գրավչությունը: Անշուշտ, բուհերի և դպրոցների համար անհրաժեշտ է գիտական խաղերի հագեցած ուսուցման մեխանիզմների գոյությունը: SandGame խաղում ավազակույտի մոդելը դիտարկվել է եռաչափ և երկչափ կապակցված ցանցի վրա (Նկար 23): Ցանցի կապակցվածությունը հնարավորություն է տալիս մոդելում ունենալ հավասարազոր գագաթներ ըստ հարևանների քանակի, ինչպես նաև թույլ չի տալիս մոդելին կորցնել ավազահատիկ:

«SandGame» խաղը միաժամանակ ապահովում է երկչափ և եռաչափ տեսաբերում, ինչպես նաև դիտման անկյունների փոփոխություն և շերտերի առանձին դիտման հնարավորություն: Վերոնշյալից զատ, SandGame-ում կա նաև մոդելի տվյալ պահին վերաբերող համապատասխան բնութագրիչների հաշվարկ և տեսաբերում իրական ժամանակում, որոնք, ավազակույտի մոդելին վերաբերող բազմաթիվ թեորեմների հետ մեկտեղ նպաստում են օգտատերերի ավելի ճշգրիտ

որոշումների կայացմանը: Նշենք նաև, որ SandGame-ը չունի ինչպես հոսթինգների, այնպես էլ միևնույն հոսթինգի օգտատերերի քանակի սահմանափակումներ:



Նկար 23.SandGame խաղը երկու խաղացողների դեպքում:

Խաղում օգտագործողների նպատակն է հասնել անվերջ անկայուն վիճակի հերթով գցելով ավազահատիկներ՝ օգտագործողների կողմից ընտրված կոորդինատներին: Խաղը՝ դիտարկված լինելով փակ եզրակետերով ցանցի վրա, հնարավորություն է ընձեռում մոդելին չկորցնել ավազահատիկ: Հետևաբար, կարելի է վստահ լինել վերջավոր ժամանակում հաղթող ճանաչվելու փաստում: Բաժին 2,2-ում նկարագրված է ավազահատիկների քանակի $C^3{}_{\text{unlimited}} = 3 * n^3$ ստորին գնահատականը, որը անհրաժեշտ է մոդելը անվերջ անկայուն վիճակի մեջ գցելու համար: Ուստի մոդելը նախօրոք սկզբնարժեքավորվում է $C^3{}_{\text{unlimited}} - 1$ քանակությամբ ավազահատիկներով, որոնք ավելացվում են պատահականության սկզբունքով ընտրված գագաթների վրա՝ ամեն անգամ հնարավորություն տալով նոր խաղային ռազմավարության: Բացի այդ, նշված քանակությամբ ավազահատիկներով սկզբնարժեքավորումը զգալիորեն կրճատում է հաղթողի

ճանաչմանն անհրաժեշտ քայլերի քանակը: Օրինակ ԼԻՆՔ[33] աշխատությունում կատարվել է ստատիստիկ հաշվարկներ, ըստ որի մոդելը անվերջ անկայուն վիճակում գցելու համար անհրաժեշտ ավազահատիկների միջին քանակը երկչափ ցանցի դեպքում ստացվում է մոտարկմամբ $C^2{}_{\text{avarage}} = 2.12588\ldots * n^2$: Ուստի երկչափ ցանցում փլուզելով $C^2{}_{\text{unlimited}} - 1$ ավազահատիկ, խաղացողներին միջինում անհրաժեշտ է լինում փլուզել ևս

$$C^2{}_{\text{avarage}} - C^2{}_{\text{unlimited}} + 1 = 0.12588\ldots * n^2$$

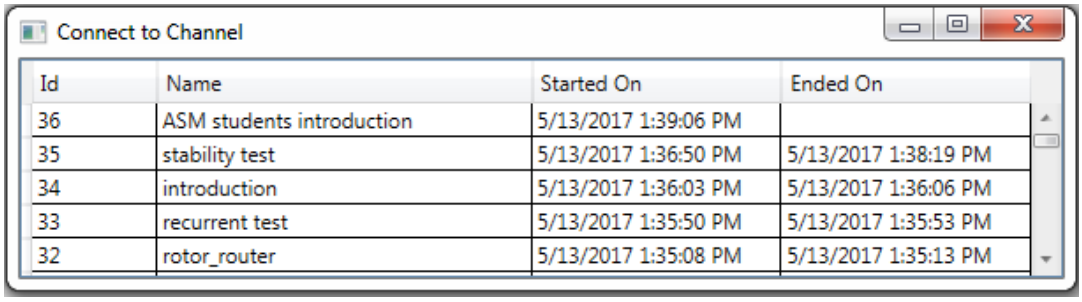
քանակությամբ ավազահատիկ միջինում: Հետևելով տվյալ աշխատության նույն տրամաբանությանը եռաչափ ցանցի դեպքում ավազահատիկների միջին քանակը անհրաժեշտ հաղթողի ճանաչման համար կստացվի $0.12588\ldots * n^3$:

Խաղի իմաստը կայանում է հետևյալում: Կա պատահականության սկզբունքով սկզբնարժեքավորված գագաթներով ավազակույտի մոդել, և կան խաղացողներ, որոնք պետք է խաղան միմյանց դեմ: Յուրաքանչյուր խաղացող կարող է ավազահատիկ ավելացնել տվյալ մոդելի ցանկալի հանգույցին, և այս քայլը կատարվում է բոլոր խաղացողների կողմից հերթականությամբ այնքան, մինչև մոդելը ընկնի անվերջ անկայուն վիճակում: Հաղթողը կլինի այն խաղացողը, ում գցած ավազահատիկը հետևանք կդառնա մոդելի ավերջ անկայուն վիճակում ընկնելու:

Ներկայացնենք թե ինչպես են խաղում SandGame-ը: Խաղի քայլերի հերթանականությունը բաժանենք 3 բաժինների: նախապատրաստություն, գործողություններ, հաղթողի ճանաչում:

- Նախապատրաստություն – նոր խաղ սկսելու համար խաղացողը ընտրում է "File" վերին պանելից, այնուհետև ընտրում "New game" տարբերակը, որից հետո հայտնվում է պատուհան ցանցի չափի մուտքագրման համար: Մուտքագրելուց հետո ստեղծվում է նոր մոդել, որն արդեն սկզբնարժեքավորված կլինի $0.12588\ldots * n^3$ քանակությամբ ավազահատիկներով: Մոդելով կիսվելու և խաղի ալիքը մյուս

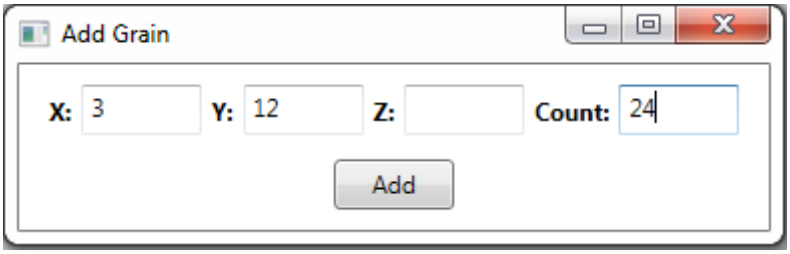
խաղացողների համար հասանելի դարձնելու համար խաղացողը պետք է ընտրի "Broadcasting" տարբերակը վերևի պանելից, այնուհետև ընտրի "Start Broadcasting" կոճակը, որից հետո կհայտնվի պատուհան, որտեղ խաղացողը պետք է մուտքագրի խաղի ալիքի անվանումը(օրինակ ASM students introduction), որով մյուս խաղացողները կգտնեն տվյալ ալիքը: Դրանից հետո մյուս խաղացողները կարող են բացել ալիքների ցուցակը ընտրելով "Broadcasting" օպցիան վերևի պանելից և սեղմելով «Connect to Chanel» կոճակը: Կբացվի ալիքների պատմությունը ցույց տվող պատուհան, որտեղ հասանելի կլինեն ինչպես ակտիվ, այնպես էլ պասիվ/ավարտված ալիքները (Նկար 24): Ալիքների ցանկից ընտրելով ցանկալի ալիքը (ASM students introduction մեր դեպքում) նրանք կմիանան միևնույն ալիքին դիտարկելով միևնույն մոդելը:



Նկար 24.Ալիքների ցուցակ:

- Գործողություններ – Նույն ալիքին/մոդելին միանալուց հետո խաղացողները կարող են սկսել ավազահատիկների ավելացման պրոցեսը, որը կատարվում է ըստ ալիքին միացման հերթականությամբ: Նշենք որ ամեն խաղացողին թույլատրվում է ավելացնել միայն մեկ ավազահատիկ ամեն քայլում (Նկար 25): Դրա համար խաղացողը ընտրում է "Edit" օպցիան վերևի պանելից, այնուհետև սեղմելով "Add Grain" կոճակը հայտնվում է նոր պատուհան,

որտեղ խաղացողը մուտքագրում է իրեն ցանկալի կոորդինատները, և վերջացնելուն պես սեղմում պատուհանի «Add» կոճակը:



Նկար 25.Ավազահատիկ ավելացնելու պատուհան:

- Հաղթողի ճանաչում – Այն խաղացողը, որի ավելացրած ավազահատիկը դրդում է մոդելին ընկնել անվերջ անկայուն վիճակի մեջ ճանաչվում է հաղթող: Այստեղից կարելի է եզրակացնել, որ խաղացաղները կարող են գցել ավազահատիկ միայն այն ժամանակ, երբ ըստ հերթականության նրանցից նախորդ խաղացողը արդեն ավելացրել է ավազահատիկ և դրանից հետո մոդելում կատարվել են բոլոր անհրաժեշտ փլուզումները, և մոդելը կայունացել է: Իսկ մոդելի անվերջ անկայուն վիճակի հասնելը ստուգումը կատարվում է փլուզման ալիքի ընթացքում բոլոր գագաթների առկայությունով: Այլ կերպ ասված փակ համակարգում եթե փլուզման ալիքի ընթացքում ամեն գագաթ փլուզվել է գոնե մեկ անգամ կարելի է համոզված լինել որ մոդելը ընկել է անվերջ անկայուն վիճակում:

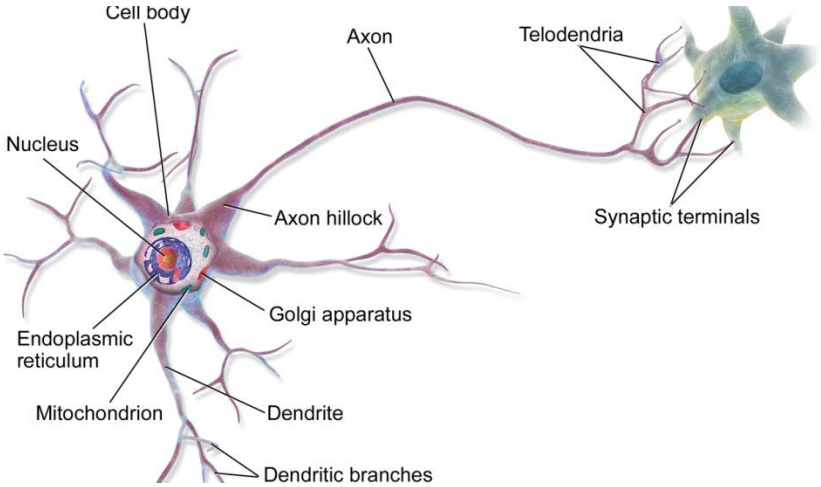
SandGame-ի հիմքում եղած լուրջ տեսությունը դեպի հաղթանակ տանող ճանապարհը ավելի դժվար է դարձնում, երբ հակառակորդները պրոֆեսիոնալ խաղացողներն են լինում: SandGame-ը հնարավորություն է տալիս յուրաքանչյուր խաղացողի ստեղծել իր սեփական հաղթանակի տանող ռազմավարությունը, սակայն հաղթողը կլինի այն մեկը, որի ռազմավարությունը ավելի լավ կլինի:

Օրինակ, ամենապարզ ռազմավարություններից մեկը կարող է լինել հետևյալ. ավազահատիկներ ավելացնում ենք այն գագաթներին, որոնք ավելի մոտ են անկայուն վիճակի ընկնելու համար, կամ ավելի հետաքրքիր կլինի մոդելը փորձել բերել կրիտիկական վիճակներից մեկին, որի դեպքում անվերջ անկայունության հասնելու համար անհրաժեշտ ավազահատիկների քանակը մոտ է նվազագույնին:

3.3. Արհեստական նեյրոնային ցանցերի ներգրավմամբ «Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդրի լուծման դիտարկումը

Արհեստական նեյրոնային ցանցերը մաթեմատիկական մոդելներն են, ինչպես նաև նրանց ծրագրային կամ սարքային իրականացումներ, որոնք կառուցված են բիոլոգիական նեյրոնային ցանցերի (կենդանի օրգանիզմի նեյրոնային բջիջների) հիման վրա: Այս հասկացությունը առաջացել է գլխուղեղում առաջացող պրոցեսների ուսումնասիրման և այդ պրոցեսների մոդելավորման փորձերի արդյունքում: Այդպիսի առաջին փորձը Մակկալոկի և Պիթսի նեյրոնային ցանցերն էին [60]: Հետագայում, ուսուցողական ալգորիթմների մշակումից հետո, ստացված մոդելները սկսեցին կիրառել պրակտիկ նպատակներով՝ կանխատեսման խնդիրներում, կերպարների ճանաչման համար, կառավարման խնդիրներում և այլն: Մասնավորապես Նկար 26-ում պատկերված է բնական ուղեղի նեյրոնի կառուցվածքը, որտեղ նշված ամեն մաս ունի ուրույն աշխատանքը.

- Dendrite – ստանում են տվյալներ
- Axon – ուղարկում են տվյալներ
- Synaptic terminal – Երկու նեյրոնի միացման կետերն են
- Neurotransmitter – փոխանցում է տվյալներ երկու նեյրոնների միջև

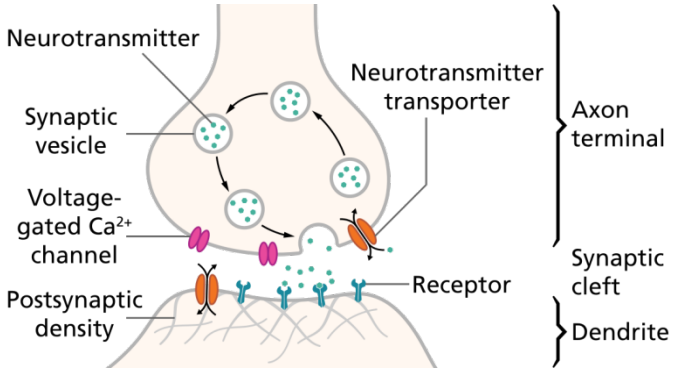


Նկար 26.Բիոլոգիական նեյրոնային ցանց:[8]

Եթե ավելի խորը դիտենք տվյալների փոխանցման հատվածը կունենանք Նկար 27-ի պատկերը, որտեղ.

- Receptor – տվյալների ընդունիչ
- Neurotransmitter transporter – Neurotransmitter-ի փոխադրիչ

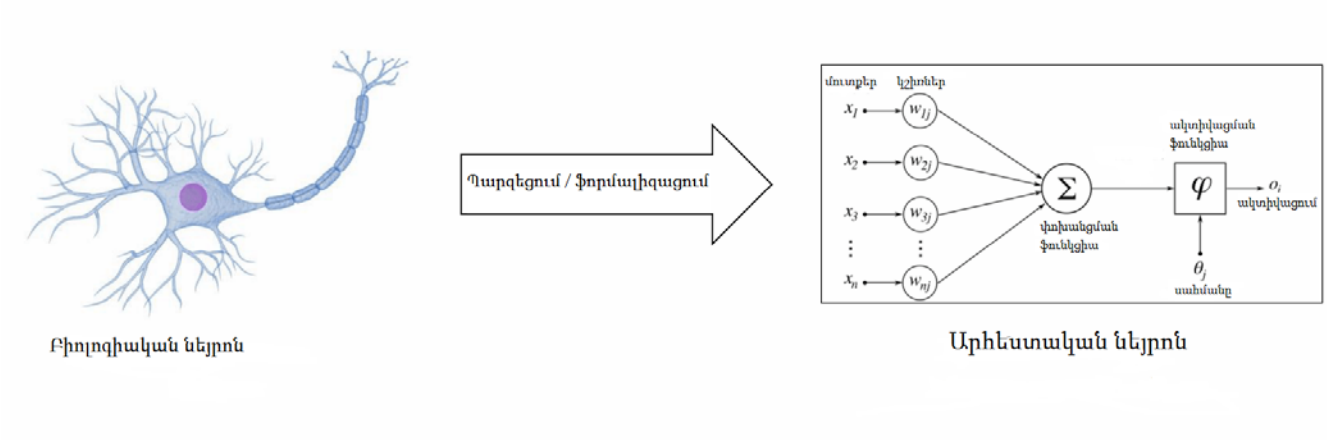
Նշեմ նաև որ ինչքան կարևոր է նեյրոններում փոխանցվող ինֆորմացիան այնքան ավելի շատ Neurotransmitter-ներ են արտադրվում և փոխանցվում:



Նկար 27.Բիոլոգիական նեյրոնային ցանցում ինֆորմացիայի փոխանակում:[9]

[8] https://en.wikipedia.org/wiki/Neural_circuit
[9] https://da.wikipedia.org/wiki/Neurotransmission

Արհեստական նեյրոնային ցանցերի դեպքում կունենանք Նկար 28-ի պատկերը:



Նկար 28.Բիոլոգիական նեյրոնի պարզեցում:

Ընհանրապես գոյություն ունեցող խնդիրները կարող ենք բաժանել 3 դասի.

1. Խնդիրներ, որոնց լուծումը հայտնի է (կան բանաձևեր, ալգորիթմներ և այլն),
2. Խնդիրներ, որոնց լուծումը հայտնի է միայն մասամբ, այլ կերպ ասված լուծող ալգորիթմը աշխատում է միայն հիմնական կամ հաճախ հանդիպող դեպքերի համար
3. Խնդիրներ, որոնք չունեն լուծում.

Առաջին դասին պատկանող խնդիրներից են, օրինակ՝ տվյալների տեսակավորումը կամ տվյալների մեջ տրված արժեքով տվյալի որոնումը: Երկրորդ դասի խնդիրներ կարող են հանդիսանալ պատկերում մեքենայի հայտնաբերումը կամ զենք կրող մարդու միաժամանակ ոստիկանի հագուստ կրելու փաստի հաստատումը: Երրորդ դասի խնդիրներից են եղանակի տեսության կամ տարբեր դրամական արժույթների կանխատեսումը: Նշենք, որ նեյրոնային ցանցերի օգտագործումը ամենանպատակահարմարն է հենց երրորդ դասի խնդիրների դեպքում: Հաշվի առնելով որ «Մինիմում ավազահատիկներ և մաքսիմալ

հեռավորություն» խնդիրը պատկանում է հենց 3րդ դասին, ապա կարելի է փորձել կիրառել նեյրոնային ցանցերը տվյալ խնդրի լուծման նպատակով:

Արհեստական նեյրոնային ցանցերն իրենցից ներկայացնում են փոխկապված և փոխհամագործակցող (արհեստական նեյրոնների) պարզ պրոցեսորների համակարգ: Այդպիսի պրոցեսորները սովորաբար բավականին պարզ են, հատկապես, համեմատած անհատական համակարգիչներում կիրառվող պրոցեսորների հետ: Նմանատիպ ցանցի յուրաքանչյուր պրոցեսոր գործ ունի միայն ազդանշանների հետ, որոնք պարբերականորեն ստանում է, և ազդանշանների, որոնք պարբերաբար ուղարկում է այլ պրոցեսորների: Այնուամենայնիվ, այդպիսի լոկալ պարզ պրոցեսորները միասին ընդունակ են կատարելու բավականին բարդ խնդիրներ: Մեքենայական ուսուցման տեսանկյունից նեյրոնային ցանցը իրենից ներկայացնում է կերպարների ճանաչման, դիսկրիմինանտ վերլուծության, կլաստերիզացիայի մեթոդի և նմանատիպ այլ մեթոդների լուծման յուրօրինակ տարբերակ: Մաթեմատիկական տեսանկյունից, նեյրոնային ցանցերի ուսուցումը ոչ գծային օպտիմիզացման բազմապարամետրական խնդիր է: Կիբեռնետիկայի տեսանկյունից նեյրոնային ցանցը կիրառվում է հարմարվողական/adaptive կառավարման խնդիրներում և որպես ալգորիթմ կիրառվում է ռոբոտատեխնիկայում: Հաշվողական տեխնիկայի և ծրագրավորման զարգացման տեսակյունից նեյրոնային ցանցը արդյունավետ զուգահեռացման խնդիրների լուծման միջոց է:

Նեյրոնային ցանցերը չեն ծրագրավորվում բառից բուն իմաստով, դրանք ուսուցանվում են: Ուսուցանելու հնարավորությունը նեյրոնային ցանցերի գլխավոր առանձնահատկություններից է: Տեխնիկապես ուսուցումը կայանում է նեյրոնների միջև կապերի գործակիցների առկայությամբ: Ուսուցման պրոցեսում նեյրոնային ցանցը ունակ է հայտնաբերել մուտքային և ելքային տվյալների միջև բարդ կախվածություններ, ինչպես նաև կատարել ընդհանրացում, որի հետևանքով «ամրապնդվում» է նեյրոնների միջև կապերի կշիռները: Դա նշանակում է, որ հաջող ուսուցման դեպքում ցանցը կարող է վերադարձնել ճիշտ արդյունք այն

տվյալների հարցումների դեպքում, որոնք բացակայում էին ուսուցողական ընտրանքում, ինչպես նաև հնարավոր են լինել ոչ լիարժեք և/կամ «աղմկոտ», մասամբ աղավաղված արդյունքներ:

Նեյրոնային ցանցերի ուսուցման մի քանի եղանակ գոյություն ունի, բայց հիմնականում օգտագործվողներից են «Ուսուցչով/ supervised» և «Առանց ուսուցչի/unsupervised» տարբերակները: Այս երկու տարբերակների նկարագրությունները հետևյալն են.

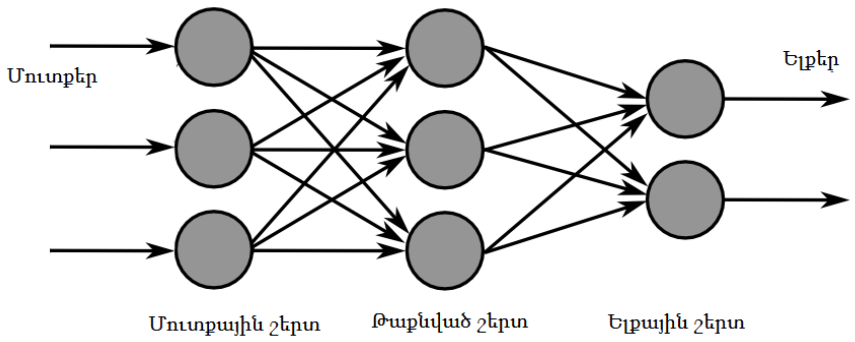
Ուսուցչով ուսուցման դեպքում նեյրոնային ցանցը

- փորձում է կանխատեսել սպեցիֆիկ քանակ
- պարունակում է նախնական պիտակավորված ուսուցողական նյութ
- կարողանում է անմիջականորեն չափել ճշգրտությունը

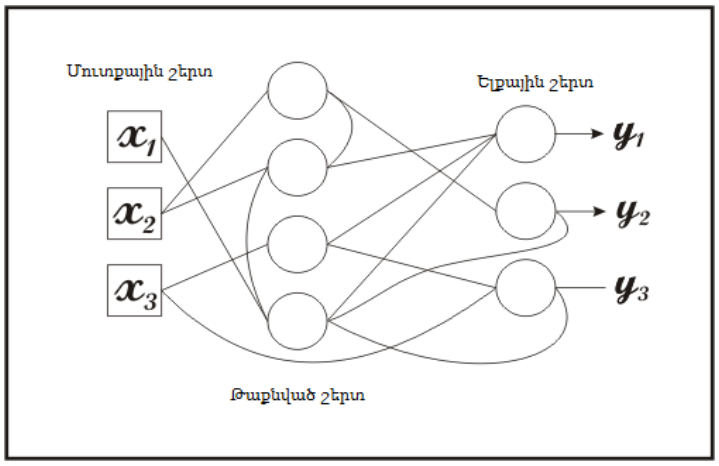
Առանց ուսուցչի տեսակի դեպքում նեյրոնային ցանցը

- փորձում է «հասկանալ» տվյալները
- փնտրում է կառուցվածքային կամ շաբլոնային համաչափություն
- չի պահանջում պիտակավորված ուսուցողական նյութ
- գնահատում է հիմնականում անուղղակի/indirect կամ որակով

Արհեստական նեյորնային ցանցերի հիմնական երկու տիպերն են Feedforward ցանցեր և Ռեկուրենտ ցանցեր (Նկար 29 և 30):



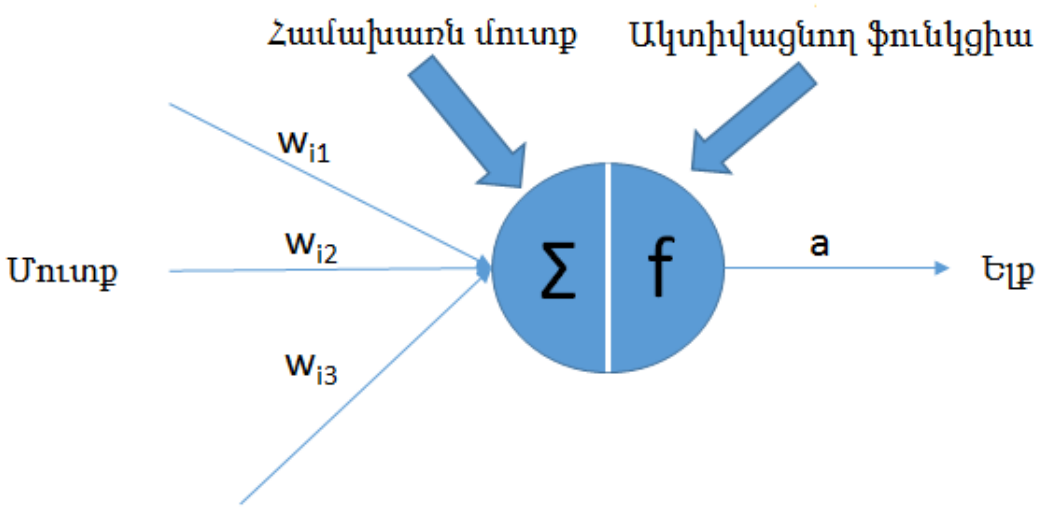
Նկար 29.Feedforward արհեստական նեյրոնային ցանց:



Նկար 30.Ռեկուրենտ արհեստական նեյրոնային ցանց:

Feedforward նեյրոնային ցանցը առաջին և ամենապարզ նեյրոնային ցանցի տիպն է: Այս տեսակի ցանցերում տեղեկատվությունը շարժվում է միայն մեկ ուղղությամբ, մուտքի հանգույցներից, թաքնված հանգույցների միջոցով (եթե առկա է) դեպի ելքային հանգույցներ: Տվյալ տիպի ցանցերում ցիկլեր կամ հանգույցներ չկան ի տարբերություն ռեկուրենտ տիպի ցանցերի: Ռեկուրենտ նեյրոնային ցանցը արհեստական նեյրոնային ցանցերի դաս է, որտեղ հանգույցների միջև կապերը հաջորդականությամբ ուղղորդված գրաֆիկ են կազմում: Ի տարբերություն feedforward նեյրոնային ցանցերի, ռեկուրենտ նեյրոնային ցանցը կարող է օգտագործել իր ներքին վիճակը (հիշողություն) մուտքերի հաջորդականության մշակման համար, որը թույլ է տալիս նմանատիպ ցանցերի կիրառությունը այնպիսի խնդիրների դեպքում, ինչպիսիք են ոչ սեգմենտավորված և կապված ձեռագիր գրությունների ճանաչումը կամ խոսքի ճանաչում:

Ինչպես նշվեց, արհեստական նեյրոնային ցանցերը բաղկացած են պրոցեսինգային միավորներից (նշանակենք՝ PU) , որոնք հանդիսանում են արհեստական նեյրոնային ցանցերի հիմնասյուները: PU- ն սովորաբար ունի Նկար 31-ում ներկայացված ձևը և էմուլացնում է (ենթադրաբար) ուղեղի նեյրոնի ֆունկցիան:



Նկար 31.Նեյրոն-պրոցեսինգային միավոր:

Հիմնականում PU-ները տրամաբանություն մշակող մոդուլներ են, որոնք օժտված են ֆունդամենտալ ֆունկցիայով, որը իրենից ներկայացնում է մուտքային «կշռված» տվյալների գումարի և որոշակի շեմային արժեքի վրա հիմնված գործառույթով: Մաթեմատիկական առումով սա արտահայտվում է որպես

$$y_i = f_i\left(\sum_{j=1}^{n} w_{ij}x_j - s_i\right) = f_i(a_i)$$

որտեղ y_i, w_{ij}, x_j, և s_i, ատրիբուտները համապատասխանաբար PU-ի ելքային տվյալն է, j-ից i կապուղղու կշիռն է, PU-ի մուտքային տվյալն է, PU-ի շեմային արժեքն է: PU_i-ի մուտքային արժեք կարող է լինել կամ PU_s-ի ելքային արժեքը, կամ ուղղակիորեն նեյրոնային ցանցի արտաքին աշխարհից տրված արժեք: PU_i-ի ելքային արժեքը կարող է օգտագործվել որպես PU_s մուտքային արժեք հետագայում, կամ նեյրոնային ցանցից որպես ելքային արժեք: w_{ij} արժեքը սահմանում է, թե որքանով է PU_j-ի ելքը ազդում PU_i-ի վրա: Կշիռը կարող է փոխվել ժամանակի ընթացքում: Ուսուցման ընթացքում, հիմնականում, այս մեխանիզմն է, որ հնարավորություն է տալիս PU-ին ադապտացվել նոր արդյունքներ ստանալու և ուսուցումը իրականցնելու համար: Ինչպես պարզ կդառնա ավելի ուշ, ամբողջ

կշիռների W մատրիցան արտացոլում է նեյրոնային ցանցի գիտելիքներն ու հմտությունները, որոնք ուսուցանվել են նախորդ վերապատրաստման միջոցով և, հետևաբար, այն կոչվում են նեյրոնային ցանցի երկարատև հիշողություն:

Նկատենք, որ s_i շեմը գործում է որպես մուտքային ազդանշանների ֆիլտր: Հավասարության a_i ատրիբուտը հայտնի է որպես PU_i ակտիվացման ֆունկցիա, որ ժամանակավոր և տեղական տեղեկատվություն է տրամադրում տվյալ PU-ի մասին և հետեւաբար այն անվանվում է, այսպես կոչված, PU-ի կարճաժամկետ հիշողություն:

Մուտքերի արդյունքում ստացված a_i արժեքը փոխակերպվում է PU-ի f_i ելքային ֆունկցիայի միջոցով, որը որոշում է ընթացիկ ազդանշանի մեծությունը: Նեյրոնային ցանցեր կառուցելու համար օգտագործվել են մի շարք ակտիվացման ֆունկցիաներ, որոնցից քայլ/step ֆունկցիան ամենապարզն է: Քայլ/step ակտիվացիոն ֆունկցիայի միջոցով PU-ի ելքային տվյալները կարող են ընդունել կամ 0 կամ 1, կախված ակտիվացիոն ֆունկցիայի արժեքից ցածր կամ բարձր լինելուց (Նկար 32): Այլ կերպ ասած, ունենք հետևյալը.

$$y_i = \begin{cases} 1, & \text{եթե } a_i > 0 \\ 0, & \text{հակառակ դեպքում} \end{cases}$$

որտեղ

$$a_i = \left(\sum_{j=1}^{n} w_{ij} x_j - s_i \right)$$

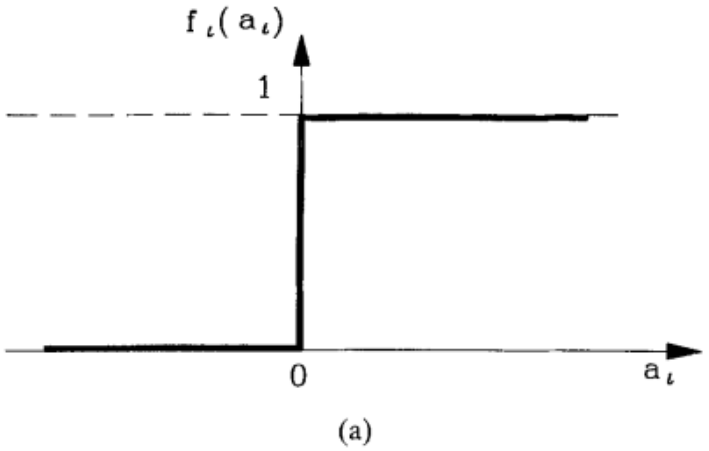
Այնուամենայնիվ, աղմուկը զտելու համար, հետևաբար բարելավվելու և իրական կայուն վիճակի հասնելու նկատառումներով որոշ նեյրոնային ցանցերի համար սովորաբար օգտագործվում է սիգմոիդ ակտիվացումը, արտահայտված Նկար 33-ի ձևով.

$$y_i = f_i(a_i) = \frac{1}{1 + e^{-ca_i}}$$

որտեղ

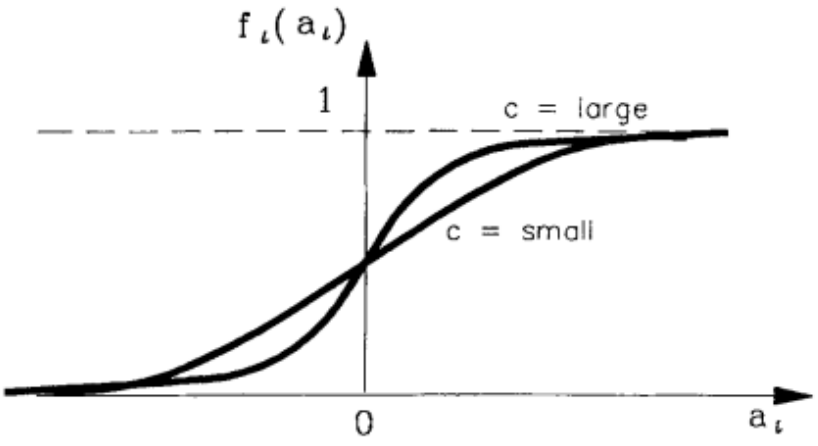
$$a_i = \left(\sum_{j=1}^{n} w_{ij} x_j - s_i \right)$$

Wu



(a)

Նկար 32.Քայլ/Step ակտիվացիոն ֆունկցիա:



(b)

Նկար 33.Սիգմոիդ ակտիվացիոն ֆունկցիա:

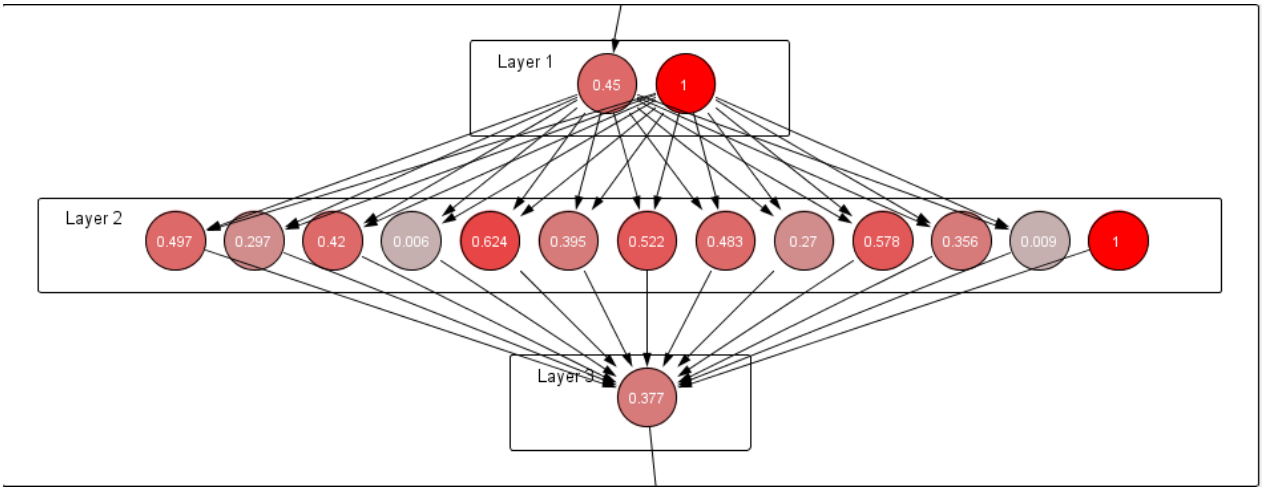
Այստեղ c-ն հաստատուն է, որը սահմանում է PU_i-ի աշխատանքում ներկայացված շեմի մակարդակը: Այս տեսակի գործառույթի առաջարկած որոշ առավելություններ հետագայում ավելի պարզ կդառնա ($1/c$ հայտնի է նաև որպես

«ջերմաստիճան»): Որոշ դեպքերում դրա արժեքը կարող է սահմանվել արհեստականորեն բարձր մակարդակի վրա, որը կստիպի նեյրոնային ցանցին ավելի լավ կատարել աշխատանքը, այսինքն կբարձրանա հավանականությունը ավելի կայուն վիճակին հասնելու:

«Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդիր գրավիչ է այնքանով, որ հանդիսանում են իրական կյանքում որոշ բնական երևույթների նկարագիր ավազակույտի մոդելի վրա: Մինչև այժմ չունենալով ճշգրիտ լուծում տվյալ խնդրի համար, նեյրոնային ցանցերի օգտագործումը արդիական նպատակ դարձավ: Նեյրոնային ցանցերի մշակման հարմարավեր միջոց է Neuroph Studio-ն [51], որը հանդիսանում է նեյրոնային ցանցերի ֆրեյմվորկ ստեղծված Java ծրագրավորման լեզվի միջոցով և նպատակաուղված է ընհանուր նեյրոնային ցանցերի նախագծման և մշակման համար: Neuroph Studio-ն պարունակում է նաև լավ նախագծված բաց կոդով Java գրադարան քիչ քանակությամբ նեյրոնային ցանցերի հիմնական աշխատանքը սիմուլացնող հիմնական դասերի ամբողջությամբ: Այն ունի նաև շատ հարմարատավետ գրաֆիկական միջավայր, որը հնարավորություն է տալիս առանց որևէ ծրագրավորման լեզվի տիրապետելու և օգտագործելու մշակել նեյրոնային ցանցեր: Մեր կողմից նույնպես ընտրվել է Neuroph Studio-ն «մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդրի ուսումնասիրության շրջանակներում նեյրոնյանին ցանցերի ճարտարապետության մշակման, տեստավորման և արդյունքների ստացման համար:

Մասնավորապես, մեր կողմից առաջարկվող տարբերակում [61] ընտրվել են բազմաշերտ բաղադրիչ(multilayer perceptron) տիպի նեյրոնային ցանցերի տեսակը մեկ մուտքային և մեկ ելքային նեյրոններով: Նեյրոնային ցանցերում օգտագործվել են Bias նեյրոնները, որպես փոխանցման ֆունկցիա ընտրվել է սիգմոիդ տիպը, մինչդեռ ուսուցողական կանոնը մշակվում է back-propagation մեթոդաբանության հիման վրա:

Դիտարկված նեյրոնային ցանցերում կա միայն մեկ մակադակի թաքնված շերտ (Նկար 34), որի նեյրոնների քանակությունը տարբերակվում է նշված համեմատական անալիզներում:



Նկար 34.Մշակված նեյրոնային ցանցի օրինակ 12 թաքնված նեյրոնով:

3.4. «SandScheduler» ծրագրային փաթեթը

Ժամանակակից կլաստերային համակարգերում մուտքագրվող առաջադրանքների ճշգրիտ արդյունքներ ապահովելուց բացի առկա են նաև այլ խնդիրներ, ինչպիսիք են՝ մուտքագրվող առաջադրանքների հավասարաչափ բաշխվածության ապահովումը և էներգիայի խնայողությունը:

Ինչպես նշվեց բաժին 1.4-ում, արդեն գոյություն ունեն որոշակի ալգորիթմներ կլաստերային համակարգերում առաջադրանքների տեղաբաշխումը ավազակույտի մոդելի հիման վրա կառուցված: Մեր կողմից ստեղծված SandScheduler [46] ծրագրային փաթեթն իրենից ներկայացնում է կլաստերային համակարգերի աշխատանքը սիմուլացնող և տեսաբերող ծրագրային փաթեթ, որտեղ խնդիրների բաշխիչի աշխատանքը հիմված է ոչ միայն ավազակույտի, այլև rotor-router համակարգի հիման վրա:

Նշենք, որ առաջարկվող մոդելը հանդիսանում է ապակենտրոնացված համակարգ, այսինքն՝ չկա կենտրոնացված կառավարող մոդուլ, և ամեն մի հանգույց յուրովի որոշում է կայացնում, որը կարող է ազդել հարևան հանգույցների վրա:

Հաշվի առնելով rotor-router համակարգի առանձնահատկությունները նկարագրված բաժին 2.3-ում, նկատվել է դրա նպատակահարմար կիրառումը նաև կլաստերային համակարգերում: Արդյունքում ստացվել է կանխատեսումների տեսանելի արդյունք SandScheduler-ի միջոցով, որոնցից է, օրինակ՝ 2.3 բաժնում բերված հիփոթեզը:

Կլաստերային համակարգի սիմուլացիան կատարվել է երկչափանի քառակուսային ցանցի վրա, որտեղ ամեն հանգույց իրենցից ներկայացնում են առանձին հաշվողական բլոկ, այլ կերպ ասած համակարգիչ իր առանձին հիշողության տիրույթով: Կախված տվյալ հանգույցում առաջադարանքների ծանրաբերնվածության, տվյալ հանգույցն է որոշում ուղարկել առաջադրանքը հարևան հանգույցներից մեկին թե ոչ: Դիտարկված մոդելում գոյություն ունեն հետևյալ նախապայմանները.

- Մոդելում ունենք նախորոք մուտքագրված քանակությամբ քառակուսային ցանցի տեսք ունեցող համասեռ հանգույցները, այսինքն հանգույցներում տեղադրված հիշողության տիրույթները, պրոցեսսները, օպերատիվ հիշողությունները և մնացած բոլոր տեխնիկական պարամետրները նույն են:
- Ամեն հանգույց ունի «հասանելի» տիրույթ առաջադրանքների համար, որտեղ կարող են տեղավորվել վերջավոր քանակությամբ խնդիրներ(տվյալ դեպքում ամենաշատը 3 առաջադրանք), և առաջադրանքի տեղավորվելուն պես այն դիտարկվում է որպես արդեն կատարվող: Եթե հանգույցի «հասանելի» տիրույթում կան 3 արաջադրանքներ, ապա համարվում է, որ տվյալ հանգույցը

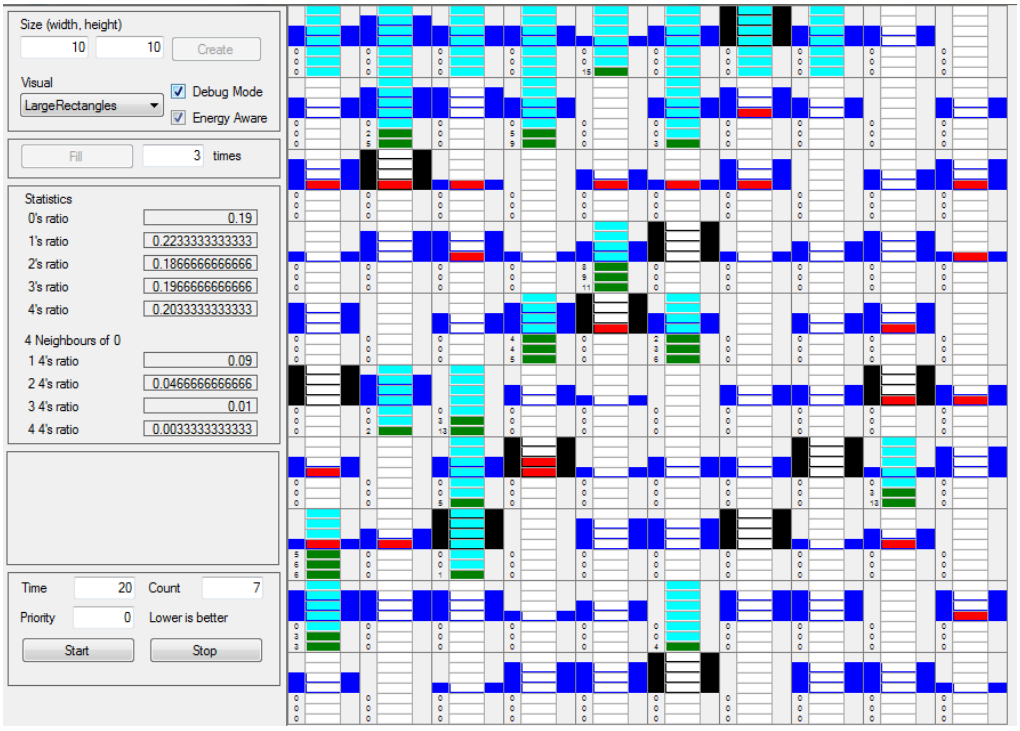
աշխատում է միաժամանակ այդ 3 առաջադրանքների իրականացման շուրջ:

- Հանգույցները միացված են իրար կապերով և ամեն հանգույց կարող է ունենալ ամենաշատը 4 հարևան (համակարգում դիտարկվում է նաև հանգույցների անսարքության դեպքեր տեխնիկական պատճառներով)
- Մուտքագրվող առաջադրանքերը միանման են և տարբերվում են կատարման համար անհրաժեշտ ժամանակով միայն: Ինչպես նաև կա հնարավորություն մուտքագրվող առաջադրանքերին տալ առաջնահերթություն:

Ինչպես արդեն նշվել է, առաջադրանքների տեղաբաշխումը կատարվում է երկու եղանակով.

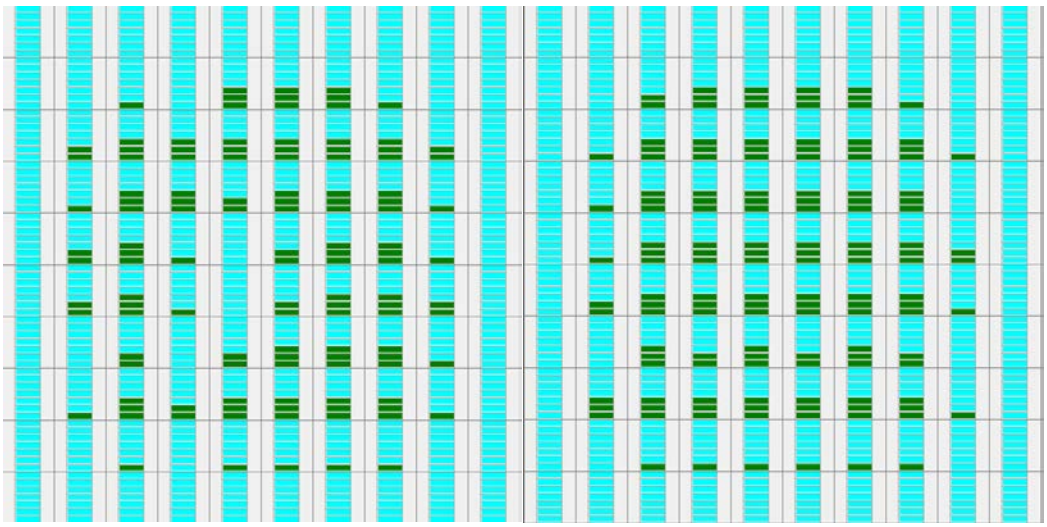
1. ավազակույտի մոդելի աշխատանքային սկբունքի, որի դեպքում համակարգում առաջադրանքների տեղաբաշխումը կատարվում է ավազակույտում ավազահատիկների փլուզման սկզբունքով: Ավազակույտի մոդելում ինչպես արդեն նշվել է ամեն հանգույցին համապատասխանեցված է արժեք, որը համապատասխանում է այդ հանգույցում ավազահատիկների քանակին: Բակը, Թանգը եւ Ուիզենֆիլդը դիտարկել են ցանցի վրա ավազահատիկների պատահական տեղադրման գործընթացը ցանցում: Ավազահատիկի տեղադրումը որոշակի հանգույցում կարող է որեւէ ազդեցություն չունենալ կամ կարող է առաջացնել փլուզման ալիքներ, որոնք կարող են կասկադային ազդեցություն ունենալ բազմաթիվ հանգույցների վրա: Հիմնական հետաքրքրությունը այս մոդելի շրջանակներում այն է, որ ցանցերի սիմուլյացիայի ժամանակ, երբ ցանցը բերվում է կրիտիկական վիճակի, այդ ժամանակ համակարգի կորելացիայի երկարությունը և տևողությունը ձգտում են անվերջության առանց համակագրի որևէ ատրիբուտի փոփոխության: Ավազակույտի մոդելում մեկ ավազահատիկի ավելացումը որևէ հանգույցին, կարող է չհանգեցնել որևէ փլուզման, կամ կարող է հանգեցնել հանգույցների զանգվածային փլուզմումների: Մենք

օգտագործում ենք վերոնշյալ առանձնահատկությունը SandScheduler համակարգում դինամիկ խնդիրների տեղաբաշխման համար, որը տեսենալի է տվյալ համակարգում «Debug enabled» վիճակում, Նկար 35: Այսպիսով SandScheduler համակարգում առաջադրանքների ավազակույտի մոդելի աշխատանքի սկզբունքով տեղաբաշխման դեպքում համակարգում ունենք դատարկ առաջադրանքներ, որոնք չեն կարող զբաղեցնել հանգույցների «հասանելի» տիրույթները: Դատարկ առաջադրանքների իմաստը կայանում է նրանում, որ դրանք ապահովվելով բավարար քանակ՝ մոդելը պահում են կրիտիկական վիճակում՝ իրականացնելով ավազակույտի մոդելին բնորոշ փլուզումները, դրանով իսկ ապահովվելով իրական առաջադրանքների տեղաշարժը ցանցում: Իրական խնդիրը, հայտնվելով հանգույցում, որի «հասանելի» տիրույթում առկա է ազատ տեղ, զբաղեցնում է համապատասխան տեղը, դուրս գալով ավազահատիկի մոդելի փլուզումների շարքից, որտեղ դրա դուրս գալուն պես ի հայտ է գալիս նոր դատարկ առաջադրանք որպես փոխարինող:



Նկար 35.SandScheduler ծրագրային փաթեթ:

2. rotor-router մոդելի աշխատանքի սկզբունքը թույլ է տալիս զերծ մնալ դատարկ/ֆիկտիվ առաջադրանքներից իրական առաջադրանքների բաշխման նկատառումներով: Տվյալ դեպքում ցանցի ամեն հանգույց ունենում է ուղղորդիչ բնորոշ rotor-router մոդելին և իրական առագադրանքը մուտք լինելով որևէ հանգույցի, զբաղեցնում է տվյալ հանգույցի հասանելի տիրույթը վերջինիս ազատ լինելու դեպքում, կամ տեղափոխվում է հարևան հանգույց ըստ տվյալ հանգույցի ուղղորդիչի ուղղության: Այս գործողությունը կրկնվում է այնքան մինչև առաջադրանքը կգտնի ազատ հանգույց և կզբաղեցնի համապատասխան տիրույթը: Հիշեցնեմ որ փակ rotor-router համակարգում ավազահատիկը պտտվելով վերջավոր քանակությամբ՝ անցում է կատարում բոլոր հանգույցներով և վերադառնում է հին դիրքին, ինչպես նաև վերադառնալուն պես մոդելը արդեն ընդունած է լինում սկզբնական վիճակը: Ինչպես նաև անդրադառնալով rotor-router համակարգի բաժին 2,3-ում նկարագրված առանձնահատկությանը, կարող ենք համոզված լինել, որ ստատիկ(կատարման ժամանակը անվերջ է) առաջադրանքների մեկ կետից ավելացման դեպքում կլաստերային համակարգում կունենանք համասեռ բաշխվածություն տվյալ հանգույցի շուրջ, կամ այլ կերպ ասված, մեկ հանգույցում առաջադրանքներ ավելացնելու դեպքու կարող ենք համոզված լինել, որ տվյալ հանգույցից r հեռավորություն ունեցող հանգույցում լցված առաջադրանքների քանակը չի կարող ավել լինել $r-1$ հեռավորության վրա գտվնող հանգույցի առաջադրանքների քանակից: Ասեմ ավելին մեր կողմից առաջ է քաշվել հիպոթեզ, որ դինամիկ(կատարման ժամանակը վերջավոր է) խնդիրների դեպքում ևս կունենանք նույն պատկերը: Տրված հիպոթեզի իսկությունը երևում է SandScheduler ծրագրային համակարգի միջոցով նկար 36:



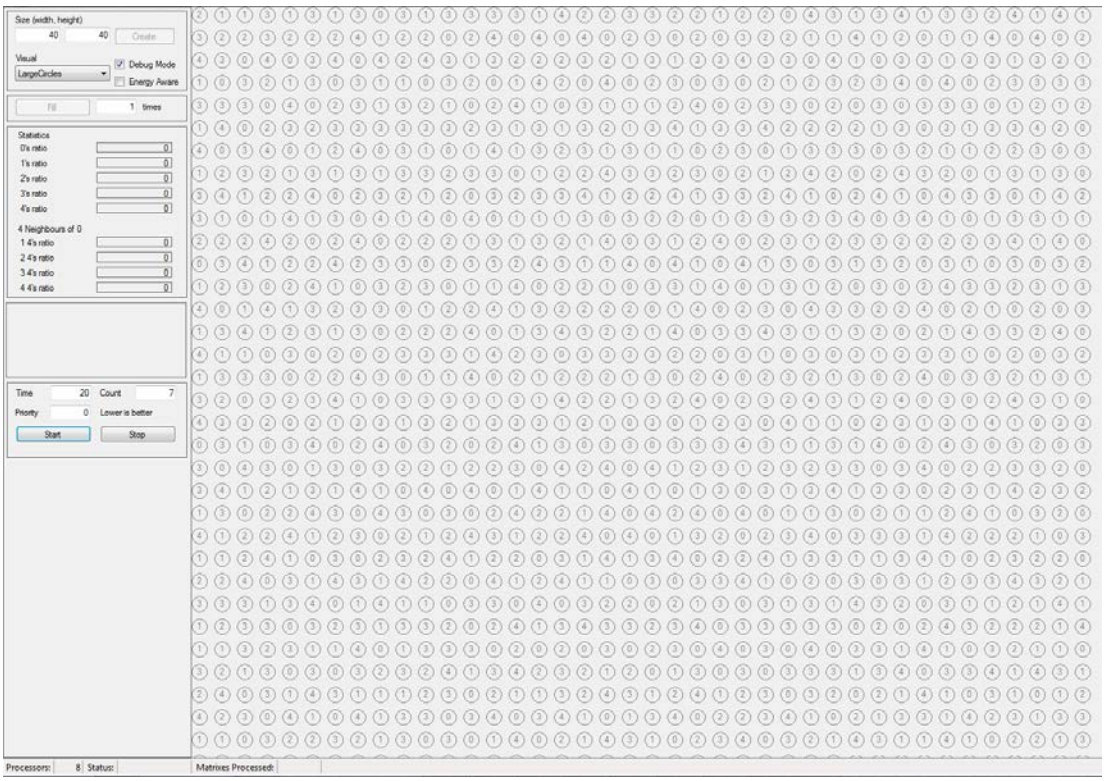
Նկար 36. SandScheduler-ը rotor-router վիճակում:

SandScheduler համակարգի տեսքը պատկերված է Նկար 38-ում, որտեղ

1. Մուտք են արվում մատրիցի/կլաստերի չափսերը և 'Create' կոճակով ստեղծվում է համապատասխան չափի մատրից/կլաստեր:

2. Ոչ գրաֆիկական ռեժիմում կատարում է ավազահատիկների փլման, համապատասխանաբար նաև խնդիրների տեղաբաշխման պրոցեսը ստեղծելով նշված քանակությամբ մատրիցներ, և վերջում տպում է ստացված միջին արդյունքները «Statistics» բաժնում

3. Տարբեր խնդիրների համար պահանջվում են գրաֆի տարբեր ներկայացումներ, կա երկու տեսակ ներկայացում.
    a. Մեծ շրջաններ, որը հարմար է ավազահատիկների փլուզումը ցույց տալու համար նկար 37
    b. Մեծ ուղղանկյուններ, որը հարմար է ցույց տալու խնդիրների բաշխման անիմացիան, որտեղ յուրաքանչյուր վանդակ պետք է ցույց տա իր մեջ գտնվող, կատարվող և հերթի մեջ գտնվող խնդիրները նկար 35:



Նկար 37.SandScheduler համակարգում շրջաններով տեսաբերում:

4. Energy Aware վանդակի չնշված դեպքում մոդելում առաջադրանքների տեղաբաշխումը կատարում է ավազակույտի մոդելի աշխատանքային սկզբունով, իսկ նշված դեպքում կիրառվում է rotor-router մոդելի սկզբունքները:

5. Ոչ գրաֆիկական ռեժիմում կատարում է ավազահատիկների փլման պրոցեսը ստեղծելով նշված քանակությամբ մատրիցներ, և վերջում տպում է ստացված միջին արդյունքները 'Statistics' բաժնում

6. 0,1,2,3 և 4 ավազահատիկներ պարունակող վանդակների քանակների գումարների հարաբերությունը ընդհանուր վանդակների թվին

7. Ավազահատիկ չպարունակող վանդակների հարևան վանդակների 1,2,3,4 ավազահատիկներ պարունակող վանդակների գումարների հարաբերությունը ընդհանուր վանդակների թվին

8. Խնդիրների բաշխման խնդրի մեկ խնդրի կատարման համար պահանջվող տակտերի քանակը և մկնիկի քլիքի միջոցով ավելացվող խնդիրների քանակը լռությամբ

9. Ավելացվող խնդիրների առաջնահերթության աստիճանը, այլ կերպ ասված, որքան կարևոր են մուտքագրվող խնդիրները

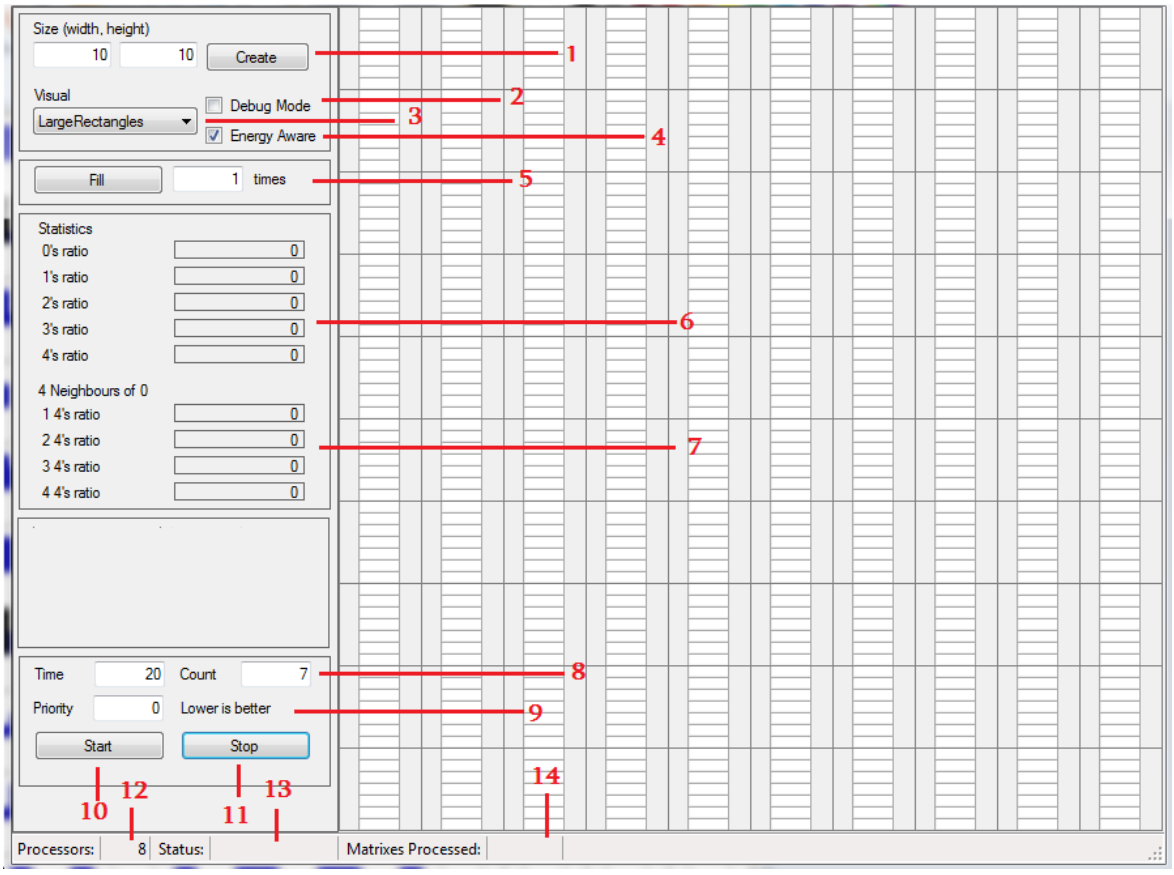
10. Սկսում է խնդիրների բաշխման անիմացիան

11. Դադարեցնում է խնդիրների բաշխման անիմացիան

12. Համակարգում առկա պրոցեսների քանակը, որը հասանելի է ծրագրին

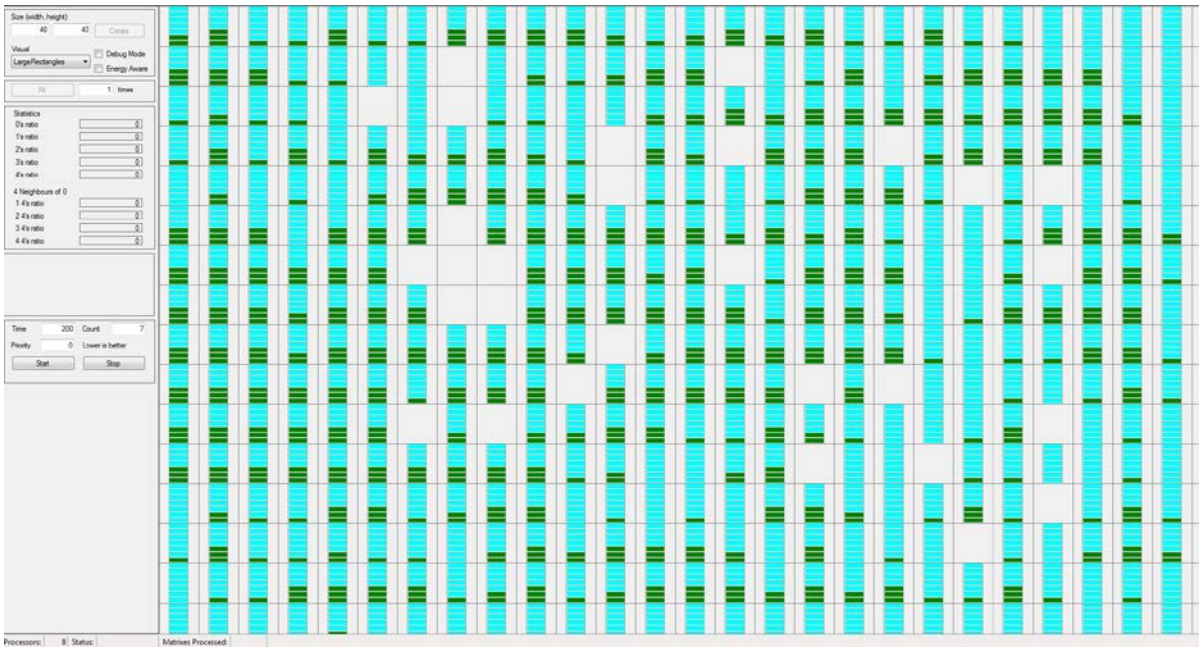
13. Առաջադրանքի կատարման համար ծախսված առավելագույն ժամանակը

14. Ցույց է տալիս կատարված և վերջացած մատրիցների քանակը



Նկար 38.SandScheduler համակարգի տեսքը:

SandScheduler համակարգով նախագծված կլաստերային համակարգի բաշխիչը ունի նաև հնարավորություն դիմակայել որոշ տեխնիկական խնդիրների, ինչպիսիք են օրինակ որոշ հանգույցի/հանգույցների անսպասելի անջատումները: Տվյալ դեպքում տվյալ հանգույցի հարևանները ավտոմատ կերպով ինֆորմացվում են դրա անսարքության մասին և, հիմնվելով ավազակույտի մոդելի փլուզումների տրամաբանությանը, հասկանում ենք որ առաջադրանքների տեղաշարժ դեպի վթարված հանգույց չի լինում: Միաժամանակ ընդհանուր կլաստերային համակարգը շարունակում է աշխատել սովորականի պես առանց որևէ խնդրի, բացառությամբ իհարկե վթարված հանգույցներում իրականացվող առաջադրանքների կորստի (Նկար 39):



Նկար 39.SandScheduler համակարգը որոշ անջատված հանգույցներով:

Եզրակացություն երրորդ գլխի վերաբերյալ

Այս գլխում նկարագրվեցին CA Simulator, SandGame, SandScheduler ծրագրային փաթեթները, ինչպես նաև նկարագրվեցին «մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդրի լուծմանը նպատակաուղղված մշակված նեյրոնային ցանցերը:

CA Simulator համակարգը իր լուրջ ֆունկցիոնալ հնարավորություններով ավելի դյուրին կդարձնի ավազակույտի մոդելի ուսումնասիրությունները, ինչպես նաև հնարավորություն կնձեռնի տարբեր աշխատահգրական դիրքերում գտնվող օգտագործողներին կատարել հետազոտություններ միասին միևնույն մոդելի վրա:

SandGame «լուրջ խաղի» օրինակը իր գրավչությամբ և պիտանելիությամբ կարող է հիմք հանդիսանալ շատերի համար ինքնակազմակերպվող կրիտիկականության կոնցեպտի դյուրընկալման:

SandScheduler ծրագրային փաթեթի միջոցով հնարավոր է ուսումնասիրել ավազակույտի և rotor-router մոդելների պիտանելությունը և կարևորությունը կլաստերային համակարգերում, և դրանց առավելությունների մեջ համոզվելու դեպքում կարելի է արդեն SandScheduler-ում բաշխիչի նկարագրված աշխատանքը իրականացնել իրական կլաստերային համակարգերում:

Մշակված նեյրոնային ցանցերը հնարավորություն են տալիս ժամանակի շահման ավազակույտի մոդելի վրա նկարագրվող «մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդրի շրջանակներում՝ փոխարինելով մոդելի սիմուլացիան նկարագրված և մշակված նեյրոնային ցանցով:

ԳԼՈՒԽ 4. ՍՏԱՑՎԱԾ ՏԵՍԱԿԱՆ և ՓՈՐՁԱՐԱՐԱԿԱՆ ԱՐԴՅՈՒՆՔՆԵՐԻ ՎԵՐԼՈՒԾՈՒԹՅՈՒՆԸ և ԳՆԱՀԱՏԱԿԱՆԸ

Ատենախոսության **չորրորդ գլխում** զետեղված են ինքնակազմակերպվող համակարգերի ուսումնասիրմանը նվիրված ծրագրային փաթեթների և արդեն գոյություն ունեցող լուծումների միջև համամետական վերլուծությունը: Մասնավորապես դիտարկվել է «CA Simulator» համակարգը, և առաջարկված զուգահեռացման ալգորիթմը համապատխան փորձարկումներով: Սույն գլխում բերված է նաև ավազակույտի մոդելի սիմուլատորի կողմից ստացված ճշգրիտ արդյունքների և մշակված նեյրոնային ցանցերի կողմից ստացված արդյունքերի համեմատական վերլուծություն:

4.1. «CA Simulator» ծրագրային փաթեթի գործարկման արդյունքների և գոյություն ունեցող լուծումների համամետական վերլուծությունը:

Դիտարկենք ներկայացված «CA Simulator» ծրագրային փաթեթի գործարկման արդյունքների և գոյություն ունեցող լուծումների միջև տարբերությունը: Մասնավորապես, դիտարկվել է NetLogo միջավայրը, որը, լինելով բազմաագենտ համակարգերի ծրագրավորման միջավայր և ունենալով հարուստ գրադարան, ամենահայտնի և ֆունկցիոնալ առումով ամենահարուստն է մինչև այժմ գոյություն ունեցող լուծումների միջև: Այնուամենայնիվ «CA Simulator»-ը, օժտված լինելով բազում ֆունկցիոնալ հատկություններով, ոչ միայն լրացնում է ամենահայտնի լուծումներում հանդիպող բացթողումները, այլ և ներառում է, բայց չի սահմանափակվում արդի լուծումների ֆունկցիոնալ հնարավորություններով, ինչպիսիք են, օրինակ՝ բազմաօգտատեր միջավայրի ապահովումը գլոբալ ցանցում, եռաչափ ցանցի կառուցումը և տեսաբերումը, ինչպես նաև տեսաբերվող ցանցի դիտման անկյունների փոփոխությունները, դիտարկվող ցանցի շերտերի և ենթամասերի առանձակի տեսաբերումը, մոդելի ինչպես կետային, այնպես էլ

մասնակի փոփոխությունները պատահական և կոնկրետ սկզբունքով, ֆիզիկական և ինֆորմացիոն բնութագրիչների հաշվարկը և այլն:

Նշենք նաև, որ «CA Simulator» փաթեթի միջոցով կարելի է կատարել բազմաթիվ տարանջատված համատեղ հետազոտություններ, և պահպանել հետազոտությունների ընթացքում կամայական պահին մոդելի վիճակը՝ հետագա ինչպես համատեղ, այնպես էլ միանձնյա հետազոտությունների համար: «CA Simulator»-ը ստեղծվել է օգտագործելով .Net միջավայրը և C# լեզուն, իսկ սերվերի տեղակայման համար ընտրվել է Microsoft Azure-ը: Իրականացման ընթացքում պահպանվել են ՕԿԾ-ի (OOP) բոլոր ստանդարտները և SOLID-ի սկզբունքները, որոնք խիստ անհրաժեշտ են այնքանով, որ հնարավորություն են ընձեռում ավելի դյուրին կերպով կատարել համակարգի ֆունկցիոնալ ընդլայնումը և հավելումը այլ բջջային ավտոմատների մոդելներով:

	Golly	JCASim	CellDemo	NetLogo	CA Simulator
Մոդելավորում	☑	☑	☑	☑	☑
Սիմուլյացիա	☑	☑	☑	☑	☑
Երկչափ ցանցում վիզուալիզացիա	☑	☑	☑	☑	☑
Եռաչափ ցանցում վիզուալիզացիա	☐	☐	☐	☐	☑
Մոդելի փոփոխություններ	☑	☐	☑	☑	☑
Բնութագրիչների հաշվարկ	☐	☐	☐	☐	☑
Մոդելի տվյալ պահին վիճակի պահպանում և բեռնում (save/load)	☑	☑	☐	☐	☑
Մոդելի կիսում (share)	☐	☐	☐	☐	☑
Միևնույն մոդելի վրա զուգահեռ աշխատանք անկախ աշխարհագրական դիրքից	☐	☐	☐	☐	☑

Աղյուսակ 1.Հասանելի համակարգերի և նախագծված CA Simulator-ի համեմատական վերլուծություն:

Այսպիսով, եթե կատարենք համեմատական վերլուծություն արդի համակարգերի և «CA Simulator»-ի միջև ապա կունենանք Աղյուսակ 1-ի պատկերը:

4.2. Ավազակույտի մոդելի սիմուլատորի աշխատանքի զուգահեռ և ոչ զուգահեռ տարբերակների ժամանակների համեմատական վերլուծություն:

Հիմնվելով բաժին 2,1-ում նկարագրված d չափանի խորանարդային ցանցերում աստղային ծածկույթները նկարագրող բանաձևի վրա, կատարվել է d չափանի խորանարդային ցանցերում ավազակույտի աբելյան մոդելի աշխատանքի զուգահեռացումը: Ինչպես նշվել է բաժին 1,2-ում, արդեն գոյություն ունեն ավազակույտի մոդելի սիմուլացիայի արագագործությանը նպատակաուղված ծրագրային փաթեթներ:

Առաջարկվող լուծման հիմքում ընկած է զուգահեռ ծրագրավորման և խնդրի զուգահեռացման մեթոդաբանությունը: Տարաբնույթ հաշվարկների զուգահեռ իրականացումը թույլ է տալիս առավել մեծ ու բարդ խնդիրները տրոհել մասերի և այն հաշվարկել իրարից անկախ գործողություններով, դրանք իրականացնելով միաժամանակ: Խնդիրների լուծման նման մոտեցումը մարդկությանը հայտնի էր վաղուց, սակայն նրա հանդեպ հետաքրքրվածությունը մեծացավ միայն վերջին տասնամյակում, երբ սկսվեցին արտադրվել մեկից ավելի պրոցեսորներ ունեցող համակարգիչներ և մշակվեցին մեթոդներ ցանցային ծրագրավորման զարգացման համար՝ այսինքն իրական հնարավորություններ ստեղծվեցին որևէ խնդրի իրարից անկախ մասերը զուգահեռ լուծել՝ օգտագործելով մի քանի պրոցեսորներ: Իսկ քանի որ արդեն աշխարհում բոլոր համակարգիչները բազմապրոցեսոր են, դա նշանակում է, որ մինչ այս միայն մեկ հաշվարկող պրոցեսորի համար նախագծված բոլոր ալգորիթմները կորցնում են իրենց արդիականությունը, եթե դրանք կարելի է լուծել զուգահեռացված տարբերակով:

Ավազակույտի աշխատանքի զուգահեռացման արդյունքում որոշ հանգույցներ պետք է փլուզվեն միաժամանակ: Առաջարկվող լուծումներում

դիտարկվում է 2 տարբերակ. առաջինը, որ միաժամանակ փլուզվող գագաթները կարող են ունենալ ընդհանուր հարևաններ, և երկրորդ՝ չունեն ընդհանուր հարևան: Քննարկվող երկրորդ տարբերակի առավելությունը ի տարբերություն առաջինին այն է, որ 2 կամ ավել գագաթների միաժամանակյա փլուզումը չի հանգեցնի միևնույն հանգույցին միաժամանակյա դիմումի խնդրի, որի հետևանքով մոդելի կայունացման ժամանակը կերկարեր:

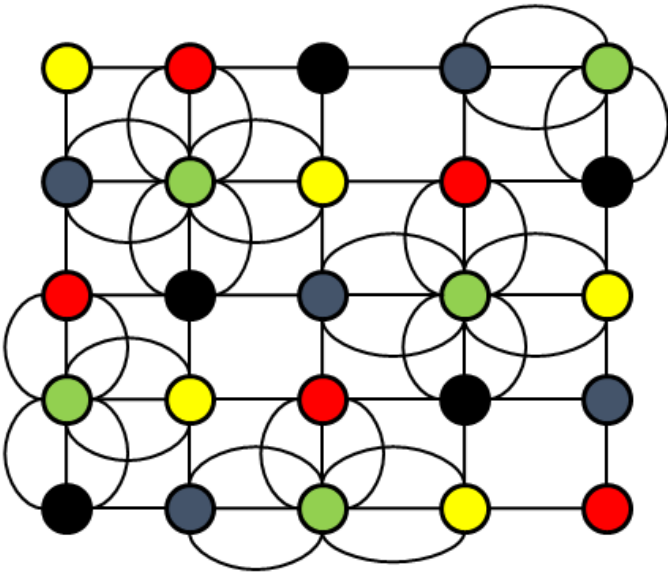
Դիտարկենք զուգահեռացման ալգորիթմներից առաջինը: Ավազակույտի մոդելների օրինակներից մեկը պատկերված է Նկար 40-ում: Այս օրինակի ուսումնասիրությունը հնարավորություն է տվել կապ տեսնել և ստեղծել ինտերվալային ներկումների և ավազակույտի մոդելի միջև, ինչպես նաև դիտարկվել են ինֆորմացիոն ինտերպոլացիա տալու համար, որի միջոցով հայտնաբերվել են ցանցում ինֆորմացիա տարածելու արդեն հայտնի կարճագույն ճանապարհներ:

4	0	4	0	4	0
0	4	0	4	0	4
4	0	4	0	4	0
0	4	0	4	0	4
4	0	4	0	4	0

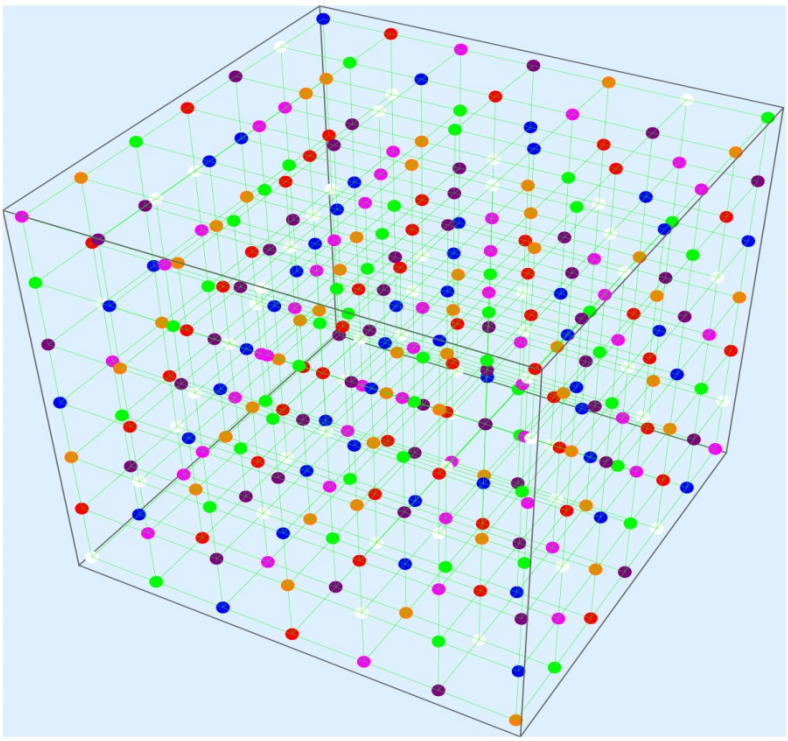
Նկար 40.Ավազակույտի մոդելի օրինակ:

Այժմ դիտարկենք զուգահեռացման ալգորիթմներից երկրորդը, որը հիմված է հենց աստղային ծածկույթները նկարագրող բանաձևի հիման վրա: Այսպիսով, ցանցի հանգույցները բաժանենք խմբերի այնպես, որ գոյություն չունենան այնպիսի

երկու հանգույց պատկանող նույն խմբին, որ ունենան ընդհանուր հարևան(այդպիսի հանգույցները անվանվում են անկախ հանգույցներ): Մասնավորապես, երկչափ և եռաչափ ցանցերի դեպքում կունենանք Նկար 41 և 42-ում բերված պատկերները, որտեղ միևնույն գույնով նշված գագաթներն անկախ են:



Նկար 41.Ավազակույտի ծածկույթները երկչափ ցանցում:



Նկար 42.Ավազակույտի ծածկույթները եռաչափ ցանցում:

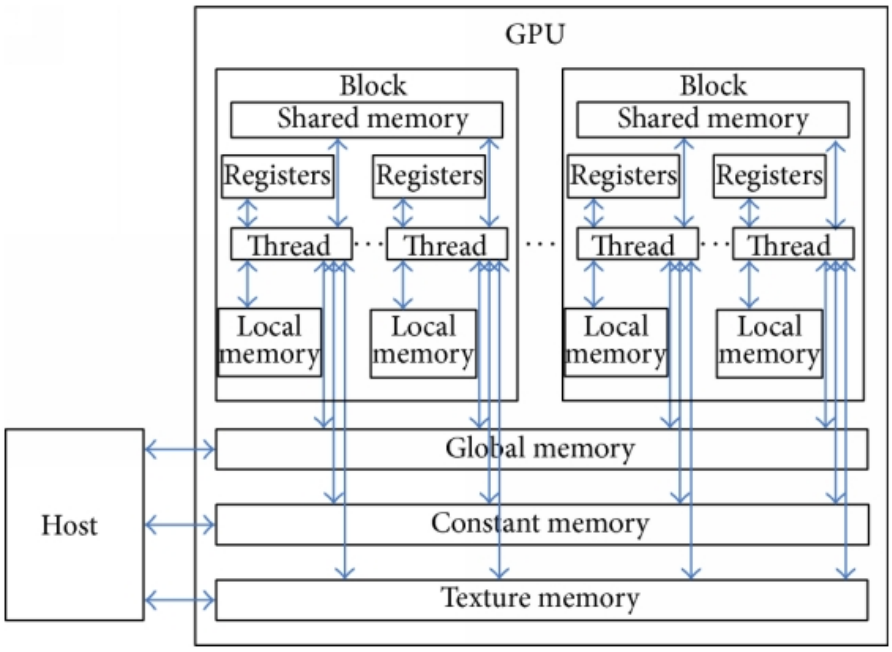
Անկախ հանգույցները կարող են միաժամանակ փլուզվել՝ բերվել կայուն վիճակի, առանց «խանգարելու» միմյանց:

Այսպիսով, երկչափ ցանցի դեպքում ունենք 5 խումբ՝ յուրաքանչյուրում $n/5$ հատ անկախ հանգույցներ, որտեղ n-ը ցանցի հանգույցների քանակն է, և պարզ է, որ անկախ հանգույցները միաժամանակ կայունացնելը կապահովի ժամանակային օպտիմիզացիա: Փորձերը ցույց են տվել, որ պրոցեսների ավելացման հետ զուգահեռ գծայնորեն նվազում է ցանցի կայունացման անհրաժեշտ ժամանակը, սակայն որոշակի քանակություն հետո պրոցեսների ավելացումը ժամանակի վրա դեր չի խաղում: Զուգահեռացման ծրագրային իրականացման համար ընտրվել են openMP [62] և CUDA [63] տեխնոլոգիաները:

Նշենք, որ openMP-ն ստանդարտ է C, C++ լեզուներով՝ ստեղծված ծրագրերի զուգահեռացման համար: Այն նկարագրում է շրջակա միջավայրի փոփոխականները, գրադարանի ֆունկցիաները և պրոցեդուրաները, որոնք օգտագործվում են բազմահոսքային ծրագրեր ծրագրավորելու համար ընհանուր հիշողությամբ բազմապրոցեսային համակարգերում: Այն իրականացնում է զուգահեռ հաշվարկ՝ օգտագործելով բազմահոսքությունը (multi-threading), որտեղ գլխավոր հոսքը ստեղծում է "հպատակ" հոսքեր, և աշխատանքը բաժանվում է նրանց միջև: Ենթադրվում է, որ հոսքերը կատաարվում են զուգահեռ, բազմապրոցեսորային մեքենաների վրա, որտեղ պրոցեսների քանակը պարտադիր չէ մեծ կամ հավասար լինի հոսքերի քանակին:

CUDA-ն զուգահեռ հաշվարկման պլատֆորմա է և ծրագրային մոդել՝ ստեղծված NVidia-ի կողմից և իրականացված GPU-ների վրա (Graphic proccesing unit), որոնք ստեղծվում են հենց իրենց կողմից: CUDA-ն ծրագրավորողներին հնարավորություն է տալիս մուտք, հասանելիություն CUDA GPU-ների վիրտուալ ինստրուկցիաների բազմություն և զուգահեռ հաշվարկման էլեմենտների հիշողություն: Այս ծրագրաապարատային զուգահեռ հաշվարկման

ճարտարապետությունը թույլ է տալիս բարձրացնել հաշվողական արտադրողականությունը: Այսինքն CUDA-ի միջոցով վերը նշված ալգորիթմը աշխատեցնում ենք վիդեոքարտի(GPU) վրա: Հիշեցնեմ որ GPU-ն ունի բազմաթիվ միկրոպրոցեսներ, որոնց քանակությունը զգալի շատ է CPU-ի պրոցեսորների քանակից: Օգտագործողը կարող է կանչել 1-512 հոսքերից կազմված բլոկը: Իսկ բլոկերն էլ իրենց հերթին կազմում են միկրոպրոցեսների ամբողջ ցանցը: Ամեն հոսք բլոկի մեջ ունի յուրահատուկ ID: Բլոկները նույնպես ընդհանուր ցանցի մեջ ունեն յուրահատուկ ID: Ինչպես նաև կան տարբեր հիշողություններ, որոնց հետ հոսքերը տարբեր արագությամբ են աշխատում: Նկար 43-ից GPU-ի աշխատանքը ավելի պարզ է դառնում: CUDA տեխնոլոգիա օգտագործող ծրագրային համակարգի արագությունը կախված է ինչպես հիշողության՝ այնպես էլ բլոկերի և հոսքերի հարաբերակցության ճիշտ ընտրությունից: Օրինակ միևնույն ծրագրի աշխատանքը 2 բլոկ՝ յուրաքանչյուրում 20-ական հոսք և 1 բլոկ 40-ական հոսքերով կարող է ժամանակային 4 անգամ տարբերություն տալ:



Նկար 43.GPU-ի կառուցվածք:[10]

[10] https://www.researchgate.net/figure/The-structure-of-GPU_fig6_272421718

Այսպիսով, ծրագրային ալգորիթմանակն լուծումը երկչափ քառակուսային ցանցի դեպքում կայանում է հետևյալում: Լուծման մեջ օգտագործվում է արդեն հիշատակված անկախ հանգույցների մեթոդը:

Քայլ 1. Հանգույցների բազմությունը բաժանում ենք 5 մասի, ամեն մասում պահպանելով հանգույցների անկախությունը: Վերցնում ենք 5 մասիվ և զբաղեցնում արդեն առանձնացված անկախ հանգույցներով:

Քայլ 2. Վերցնում ենք առաջին մասիվը և անցնում նրա պարունակության մեջ գտնվող հանգույցների վրայով: Անկայուն գագաթ գտնելու դեպքում բերում ենք կայուն վիճակի:

Քայլ 3. 2-րդ քայլում նշված գործողությունները կատարում ենք 2-րդ, 3-րդ, 4-րդ և 5-րդ մասիվների հետ:

Քայլ 4. Աշխատանքի ավարտի պահը ֆիքսելու երկու եղանակ կա.

1. Անցնում ենք ցանցի վրայով և ստուգում արդյոք անկայուն գագաթ մնացել է թե ոչ:
2. պահում ենք գլոբալ բուլյան փոփոխական isBoardReady, որի true լինելու դեպքում ալգորիթմը դադարեցնում է աշխատանքը: Առաջին քայլից սկսած թափում կատարելու դեպքում isBoardReady փոփոխականին վերագրում ենք false, և ամեն քայլից հետո ստուգում փոփոխականի արժեքը , true լինելու դեպքում ավարտում աշխատանքը: Ամեն 5 քայլի ավարտից հետո փոփոխականին վերադարձնում ենք true արժեքը:

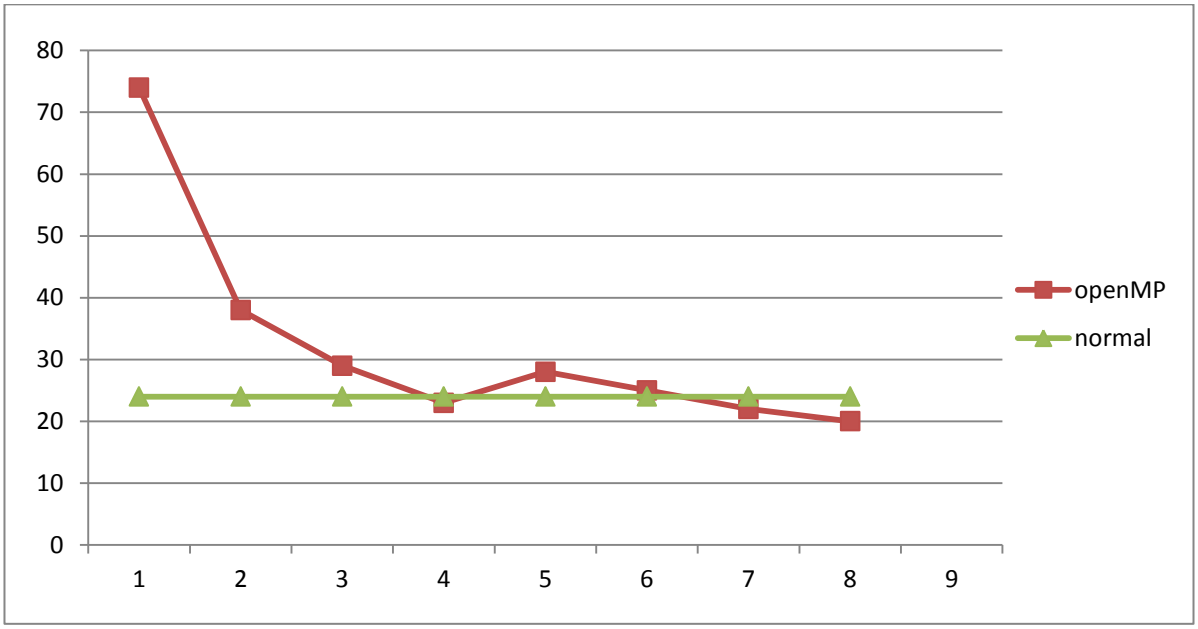
Աշխատանքի ավարտի պահը ֆիքսելու երկրորդի թերությունը հետևյալումն է: Եթե մենք կատարում ենք զուգահեռ աշխատանք , այսինքն ամեն հոսք կատարում է իրեն հատկացված աշխատանքը, ապա ամեն հոսք պարտավոր է աշխատել isBoardReady փոփոխականի հետ թափում կատարելու դեպքում, իսկ

քանի որ բոլորը միաժամանակ միևնույն հիշողության հետ աշխատել չեն կարող, ստիպված պետք է շատերը սպասման վիճակի անցնեն, իսկ դա երկարացնում է ալգորիթմի աշխատանքի ժամանակահատվածը:

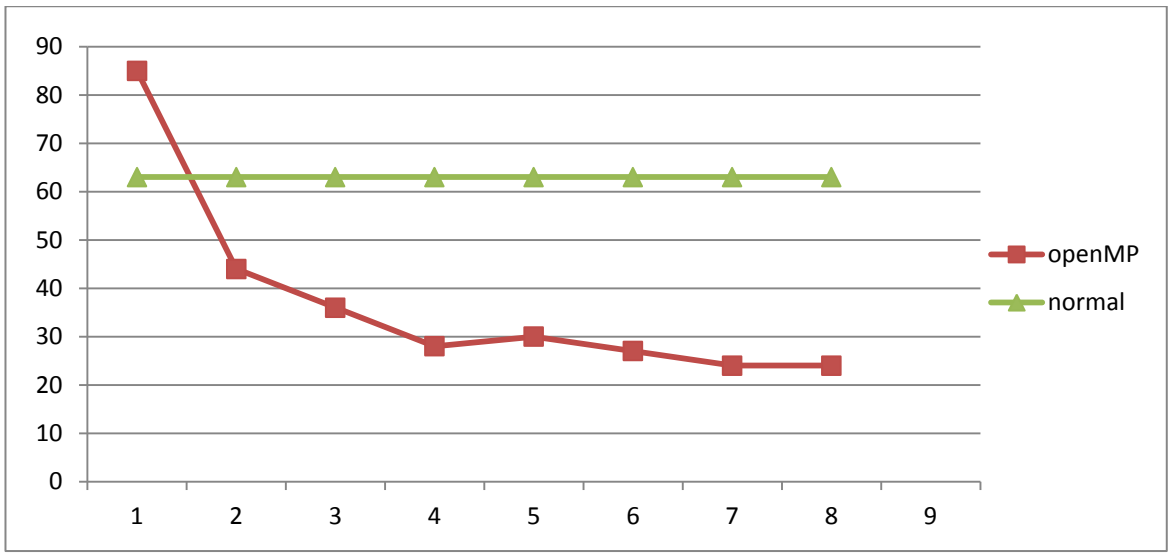
Առաջին և երկրորդ դեպքերը պետք է ընտրվեն կախված խնդրից և ցանցի չափերից: Նույն տրամաբանությամբ կարելի է իրականացնել d չափանի խորանարդային ցանցերի վրա դիտարկվող ավազակույտի մոդելի աշխատանքի սիմուլացիայի զուգահեռացում:

Այժմ ներկայացնենք տարբեր հզորությամբ CPU-ների և GPU-ների վրա փորձարկված արդյունքների վիճակագրական վերլուծությունը:

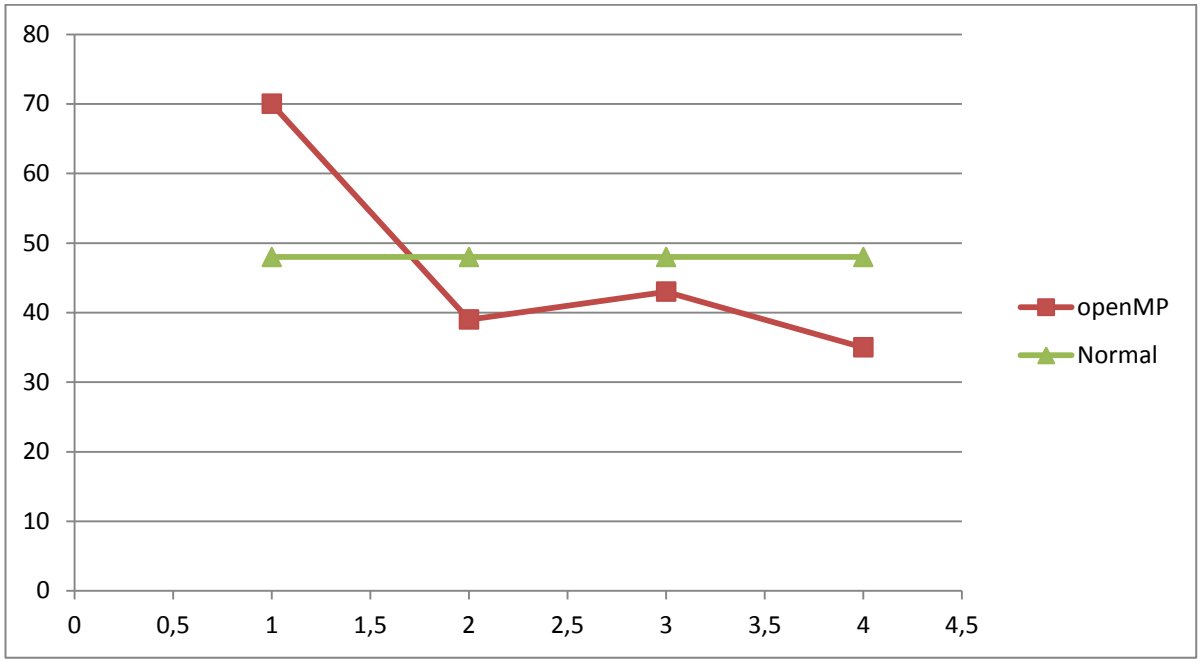
Երկչափ ցանցերի դեպքում ունենք համեմատական վերլուծություն՝ պատկերված Գծապատկեր 2, 3, 4, 5-ում:



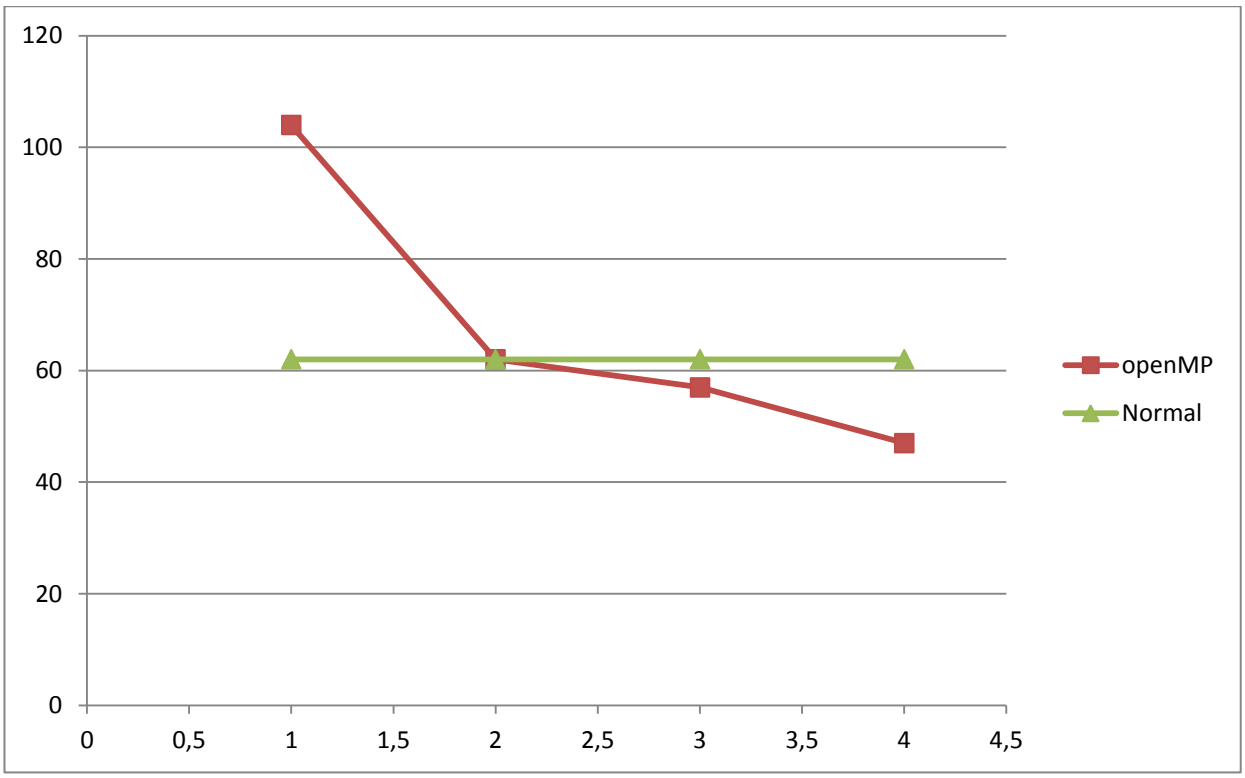
Գծապատկեր 2. 1. CPU-ի մոդելը. Intel i7 2670QM. Ցանցի չափերը. 90.000 հանգույց .Ավազահատիկների քանակը. 9.000.000 ավազահատիկ:



Գծապատկեր 3. CPU-ի մոդելը. Intel i7 2670QM . Ցանցի չափսերը. 250.000 հանգույց .Ավազահատիկների քանակը. 1.000.000 ավազահատիկ:



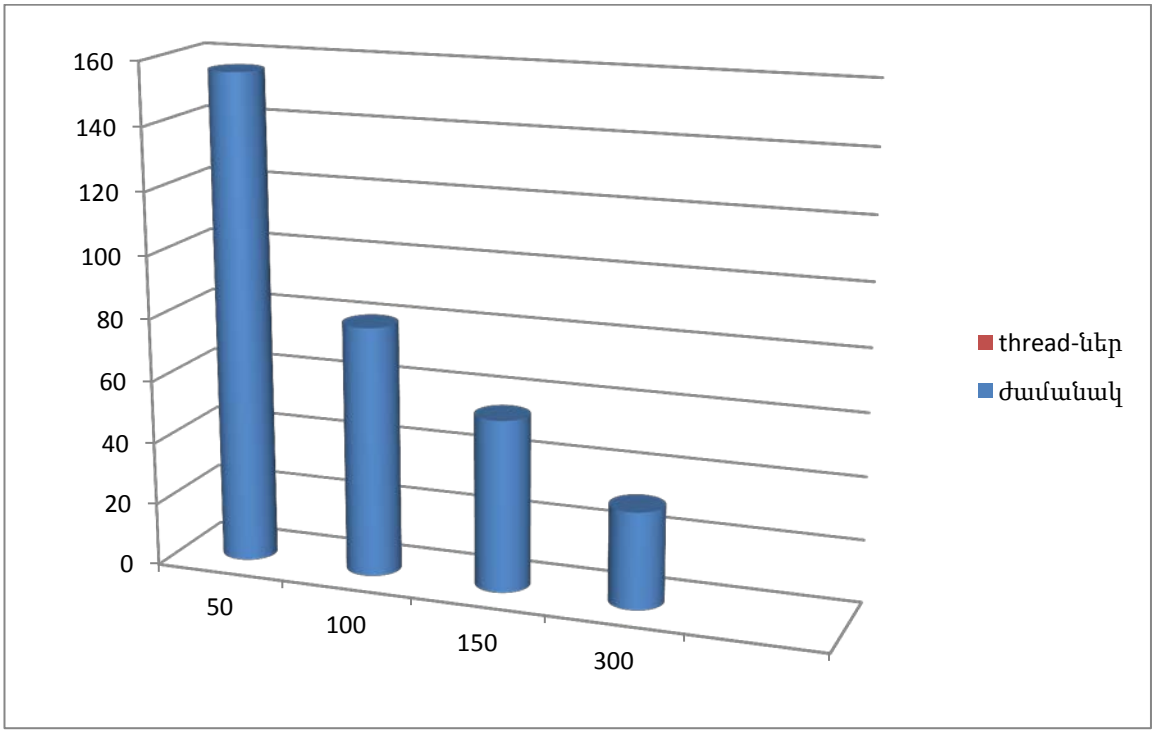
Գծապատկեր 4. CPU-ի մոդելը. Intel i3 2100, Ցանցի չափսերը. 90.000 հանգույց. Ավազահատիկների քանակը. 9.000.000 ավազահատիկ:



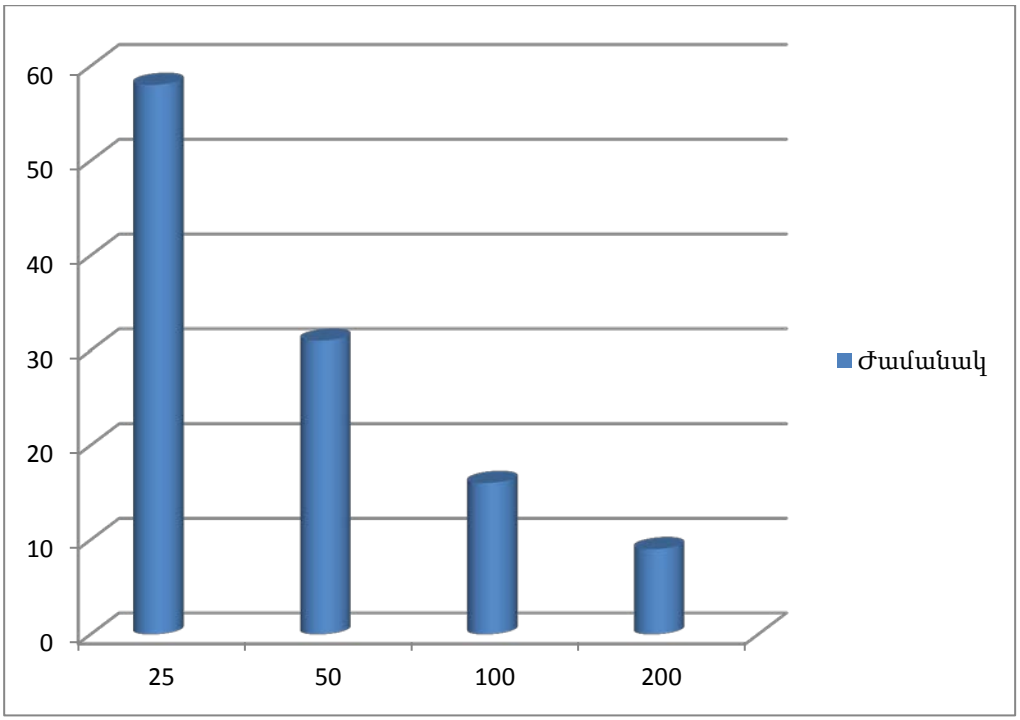
Գծապատկեր 5. CPU-ի մոդելը. Intel i3 2100. Ցանցի չափսերը. 250.000 հանգույց .Ավազահատիկների քանակը. 1.000.000 ավազահատիկ:

Գծապատկերներից պարզ է դառնում, որ ցանցի մեծացման հետ ժամանակային տարբերությունը ավելի է մեծանում: Ուստի մեծ ցանցերի դեպքում զուգահեռացումը օգտագործելը շատ ավելի օգտավետ կլինի: Հիշեցնեմ նաև, որ ժամանակը ավելի փոքր կարող է լինել ճիշտ ծրագրային մինիմիզացիաների դեպքում:

Գծապատկերներ 6-ում և 7-ում տեղադրված են արդյունքները CUDA տեխնոլոգիայի միջոցով տեստավորված Dell T5500 Workstation with NVIDIA Tesla C1060 մոդելի GPU-ի վրա: Բոլոր տեստերում ցանցի ամեն հանգույցի ավազահատիկների քանակը հավասար է 5-ի: Օրինակ՝ 2500 հանգույց ունեցող ցանցի դեպքում ընդամենը 1 պրոցեսորի աշխատանքը տևում է 3 վայրկյան: Հիշենք, որ նշված պրոցեսները նույն GPU-ի ֆիզիկական պրոցեսները չեն: Օրինակ, տվյալ մոդելում կա 240 ֆիզիկական պրոցեսոր, բայց վիրտուալ պրոցեսների քանակը կարող է հասնել մինչև 65.000-ի:



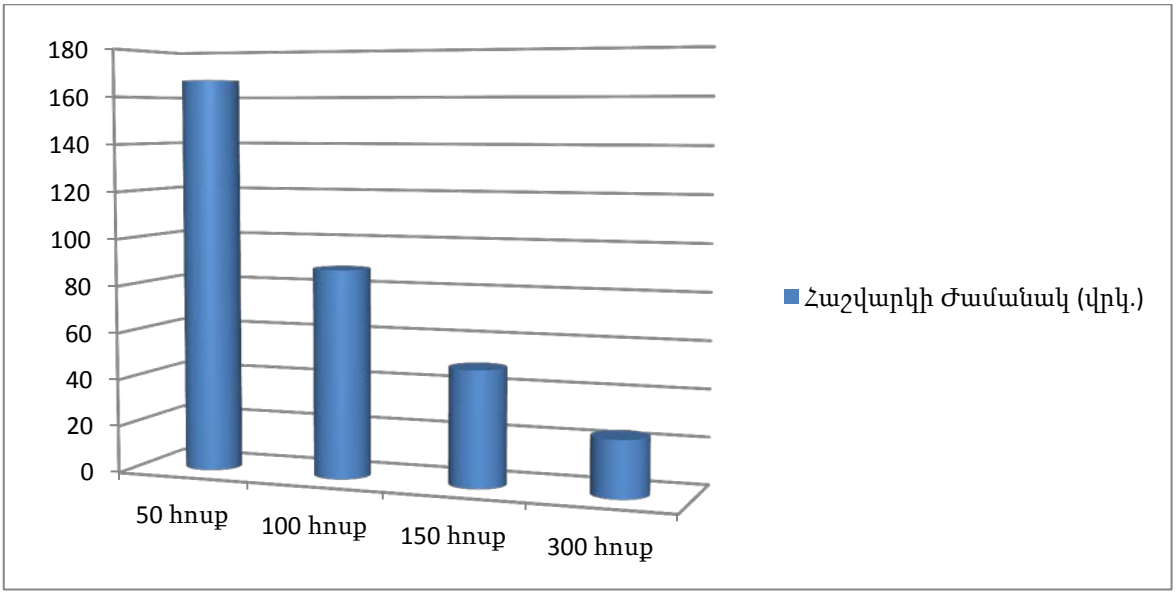
Գծապատկեր 6. 1. Ցանցի չափերը. 40.000 հանգույց, հոսքերի քանակը հաջորդաբար 25, 50, 100, 200:



Գծապատկեր 7. Ցանցի չափերը. 90.000 հանգույց, հոսքերի քանակը հաջորդաբար 50, 100, 150, 300:

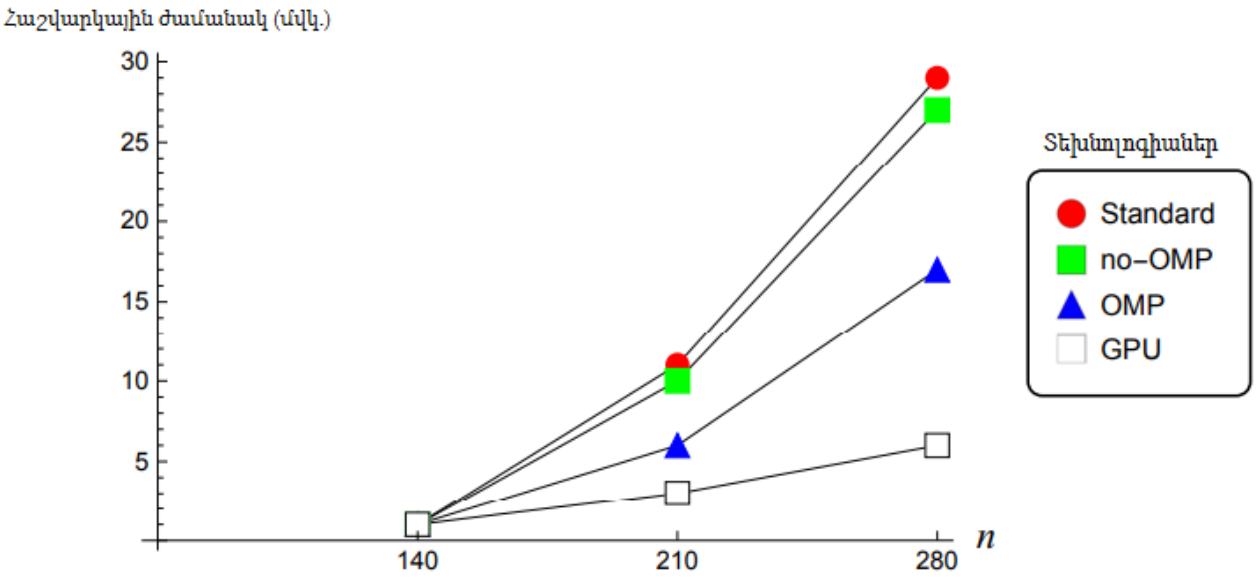
Ինչպես տեսնում ենք փորձարկումների արդյունքներից, պրոցեսների աճելուն զուգընթաց կրճատվում է նաև ավազակույտի մոդելի կայունացման անհրաժեշտ ժամանակը:

Եռաչափ ցանցերի դեպքում պատկերը մի փոքր այլ է: Մասնավորապես, CUDA տեխնոլոգիայի դեպքում ունենք 8րդ գծապատկերը:

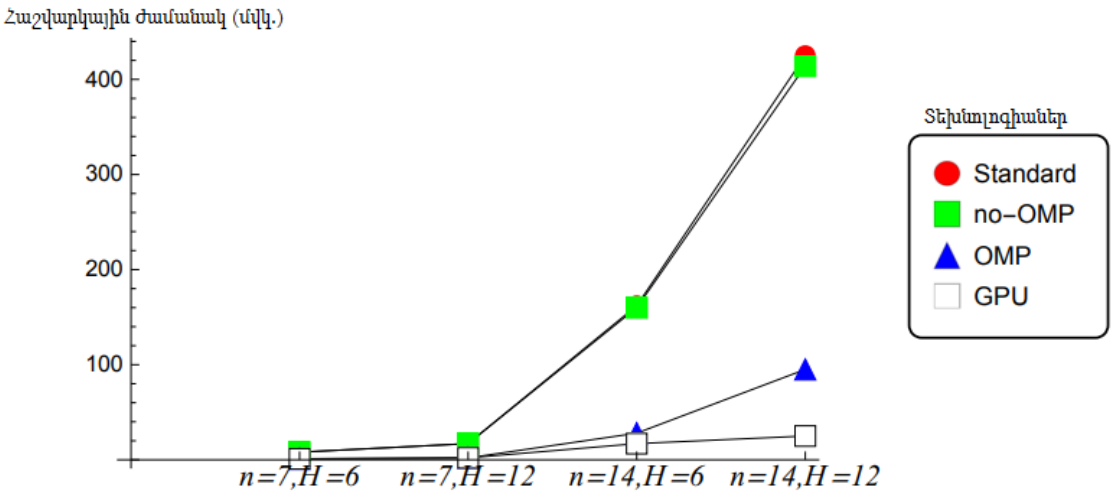


Գծապատկեր 8. GPU - Nvidia GT-540m, 250.000 հանգույց երկչափ ցանցում, 1,000,000 ավազահատիկ:

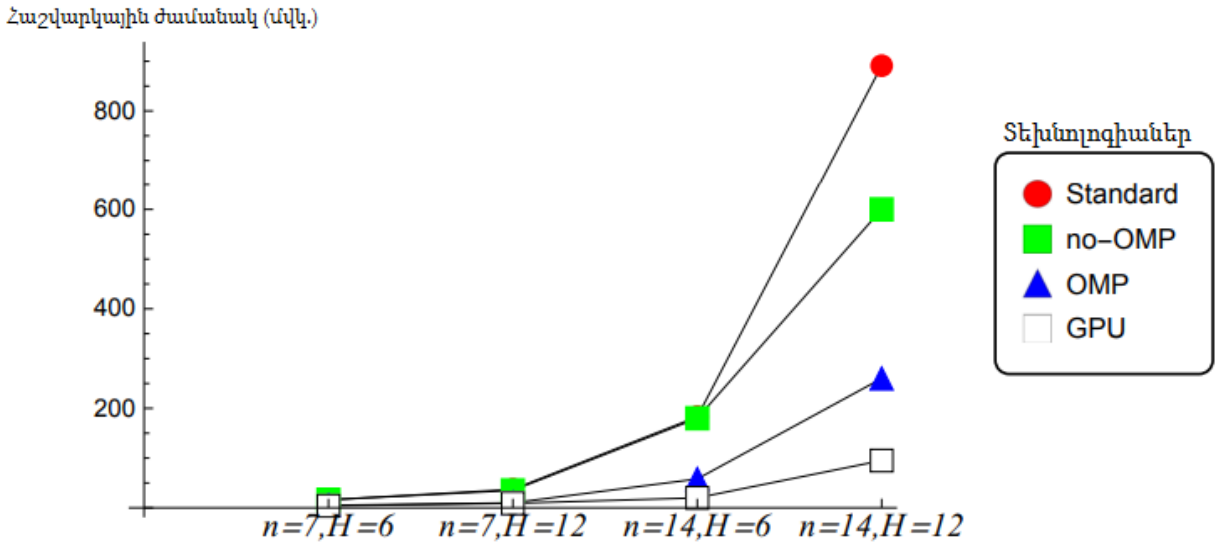
Կատարվել է համեմատական վերլուծություն նաև բոլոր տեխնոլոգիաների միջև, որտեղ դիտարկվել է, որ փլուզումից զատ կա նաև կատարվող որևէ այլ գործողություն, որը տևել է կոնկրետ T_{wait} ֆիքսված ժամանակ (Գծապատկեր 9-12):



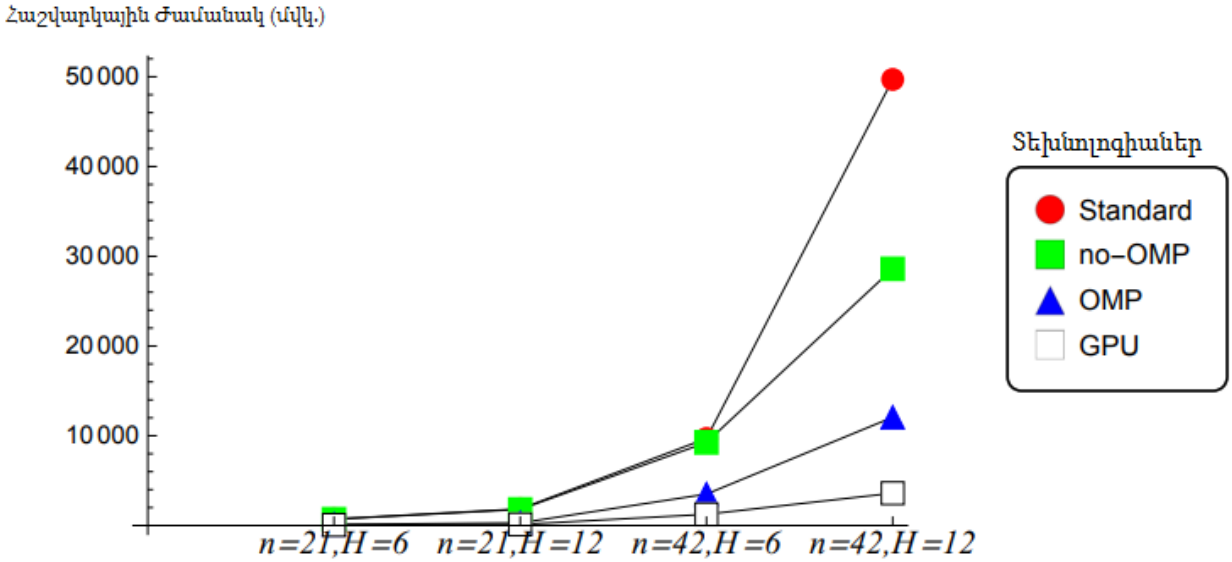
Գծապատկեր 9. Եռաչափ ցանցում համեմատական վերլուծություն, ավազահատիկները $10 * n^3$, $T_{wait} = 0$:



Գծապատկեր 10. Եռաչափ ցանցում համեմատական վերլուծություն, ավազահատիկները ունեն H-բարձրությունը, $T_{wait} = 5$ ms:



Գծապատկեր 11. Եռաչափ ցանցում համեմատական վերլուծություն, ավազահատիկները ունեն H-բարձրությունը, $T_{wait} = 10$ ms:



Գծապատկեր 12. Եռաչափ ցանցում համեմատական վերլուծություն, ավազահատիկները ունեն H-բարձրությունը, $T_{wait} = 3$ ms:

Զուգահեռացման շնորհիվ նույն ռեսուրսների միջոցով հնարավոր է դարձել ստանալ մինչև 3 անգամ ավել արագագործություն openMP տեխնոլոգիայի միջոցով: Իսկ պատկերված գծապատկերներով կարելի է տեսնել CUDA-ի միջոցով ստացված ժամանակային լուրջ արդյունքները: Սովորական փլուզումների դեպքում օգտագործելով համարժեք VideoCard հնարավոր է ստանալ մինչև 50 անգամ

արագագործություն, իսկ փլուզման ժամանակ փլուզման գործողությունից զատ՝ մոդելում այլ ժամանակատար փոփոխության դեպքում տարբերությունն էլ ավելի է աճում, որը պարզ տեսանելի է գծապատկերներում:

4.3. «Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդրի շրջանակներում նեյրոնային ցանցերի և ճշգրիտ արդյունքների միջև համեմատական վերլուծությունը:

Այս ենթաբաժնում բերված է ավազակույտի մոդելի սիմուլացիայի արդյունքում ստացված տվյալների և մշակված նեյրոնային ցանցերի կողմից ստացված արդյունքների համեմատական վերլուծությունը «Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդրի ուսումնասիրության շրջանակներում: Բերված համեմատական վերլուծություններում նեյրոնային ցանցերի տարբերությունը միայն տաքնված նեյրոնների քանակի մեջ է: Ընդհանրապես ըստ նեյրոնների քանակական ճարտարապետության նեյրոնային ցանցերը կարելի է բաժանել երեք տիպերի.

1. Անգիր անող – երբ նեյրոնների քանակը բավականին շատ է՝ մշակված նեյրոնային ցանցը այսպես ասված անգիր է անում մուտքային տվյալները և արդեն հարցման ժամանակ տրված պատասխանները տալով առանց «մտածելու» մեկը մեկին կհամապատասխանեն ուսուցման համար տրված տվյալներին, իսկ այնտեղից բացակայելու դեպքում արդյունքը գոհացուցիչ չի լինի,
2. Անխելք - երբ նեյրոնների քանակը բավականին քիչ է՝ մշակված նեյրոնային ցանցը ամեն հարցման վրա սկսում է շատ երկար մտածել, և հարցմանը տրված պատասխանների սխալանքը բավականին մեծ է լինում անկախ հարցվող տվյալի առկայությանը ուսուցողական տվյալների հենքում,
3. Ոսկե միջին - երբ նեյրոնների քանակը բավականին է նեյրոնային ցանցի լավ աշխատանքի համար՝ ոչ այնքան շատ, որ «անգիր» արվի, ոչ էլ հակառակը:

Բարեբախատաբար, «Ոսկե միջինին» պատկանող նեյրոնային ցանցերում նեյրոնների քանակի միջակայքը բավականաչափ լայն է, ուստի այդքան էլ բարդ չէ մշակել ցանց, որը կպատկանի երրորդ տիպին:

Որպես նախագծված նեյրոնային ցանցերի ուսումնառության մեթոդ ընտրվել է «Ոսուցչով տարբերակը», որը նախատեսում է սկզբնական տվյալների հենք նեյրոնային ցանցերի ուսուցման համար: Այդ նպատակով մշակվել է ծրագրային փաթեթ, որը սիմուլացնելով ավազակույտի մոդելը անվերջ երկչափ ցանցի վրա տվել է հեռավորության 0-ից 40 արժեքներին համապատասխան անհրաժեշտ ավազահատիկների քանակը: Այնուհետև ստացված տվյլաները բերվել են նեյրոնային ցանցի մուտքին համապատասխան վիճակի և տրվել որպես վերապատրաստման հավաքածու/training set (Աղյուսակ 2):

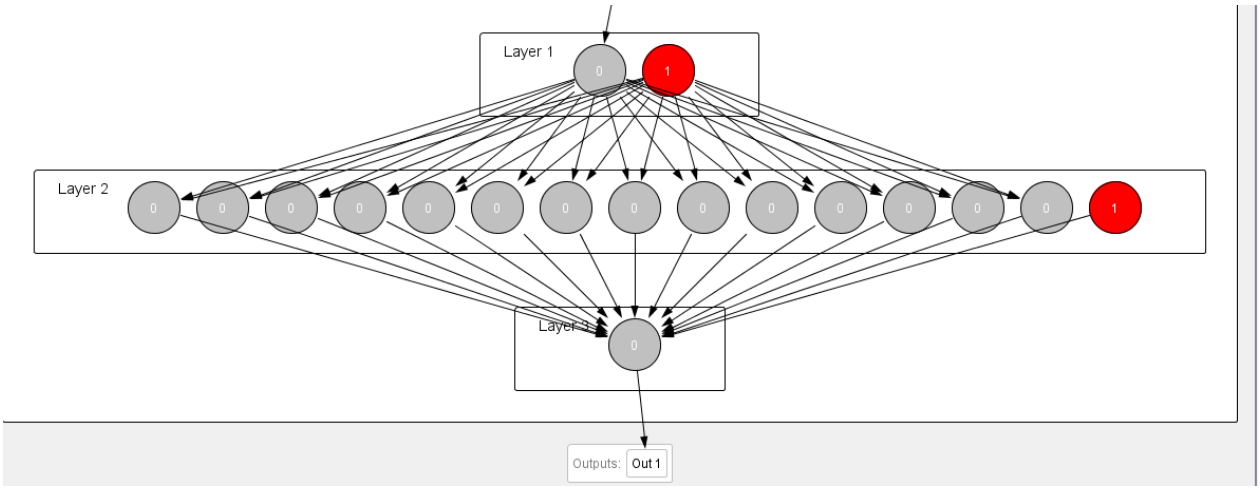
Հեռավորություն	Ավազահատիկների քանակ
1	1
2	4
3	16
4	44
5	88
6	144
7	208
8	320
9	408
10	512
11	672
12	788

13	948
14	1096
15	1288
16	1552
17	1768
18	1960
19	2208
20	2456
21	2708
22	3028
23	3384
24	3700
25	3996
26	4394
27	4800
28	5098
29	5494
30	5900
31	6388
32	6786
33	7204
34	7678
35	8302
36	8694

37	9186
38	9804

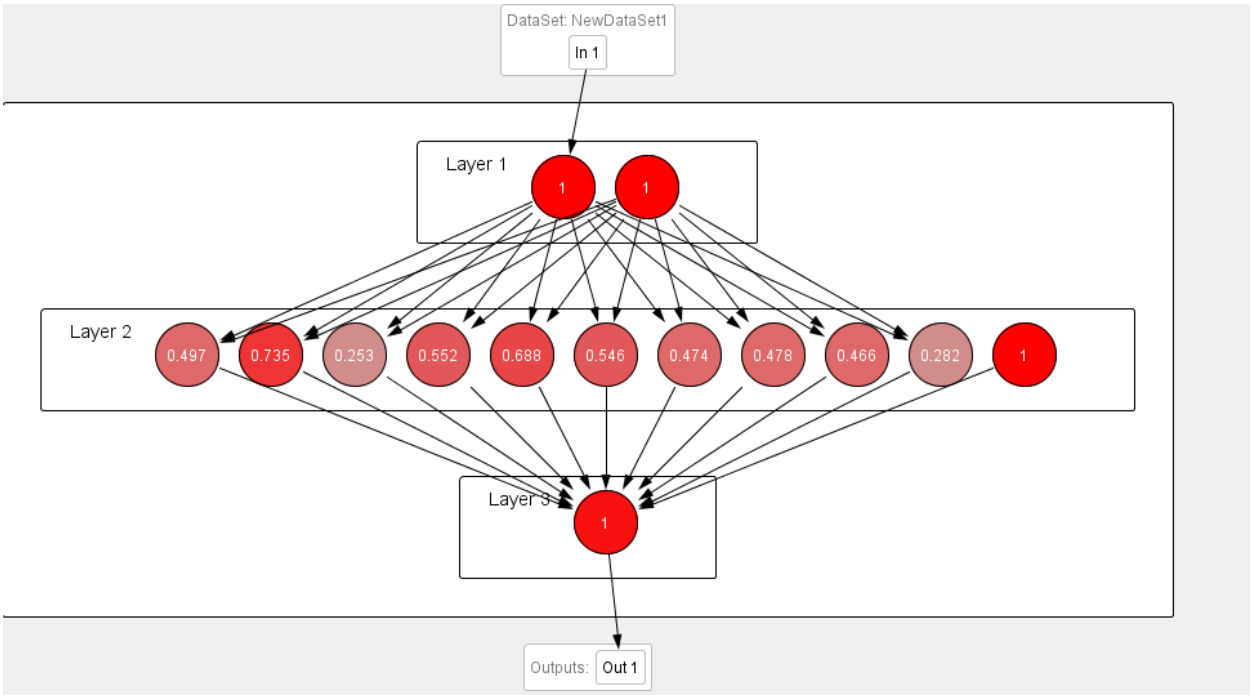
Աղյուսակ 2. Ավազակույտի մոդելի միջոցով ստացված արդյունքներ:

Նկար 44-ում ներկայացված է մշակված նեյրոնային ցանցերի սկզբնական վիճակը մինչև ուսուցումը, որտեղ երևում է, որ նեյրոնների կշիռը հավասար է 0: Իսկ արդեն ուսուցումից հետո ունենում ենք այլ պատկեր, որտեղ նեյորնային ցանցը ուսուցման արդյունքում արժեքավորած է լինում նեյրոնների կշիռները ակտիվացիոն ֆունկցիայի շեմը ճիշտ հաղթահարելու և ցանկալի արդյունքներ տալու համար(Նկար 45):

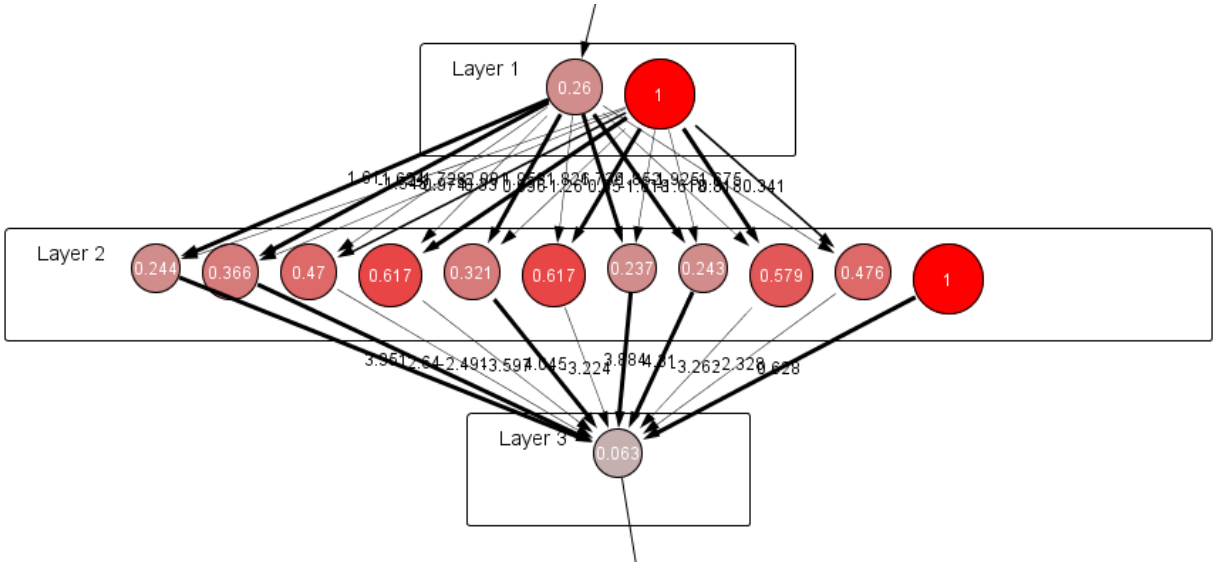


Նկար 44.Նեյրոնային ցանցը մինչ ուսուցումը:

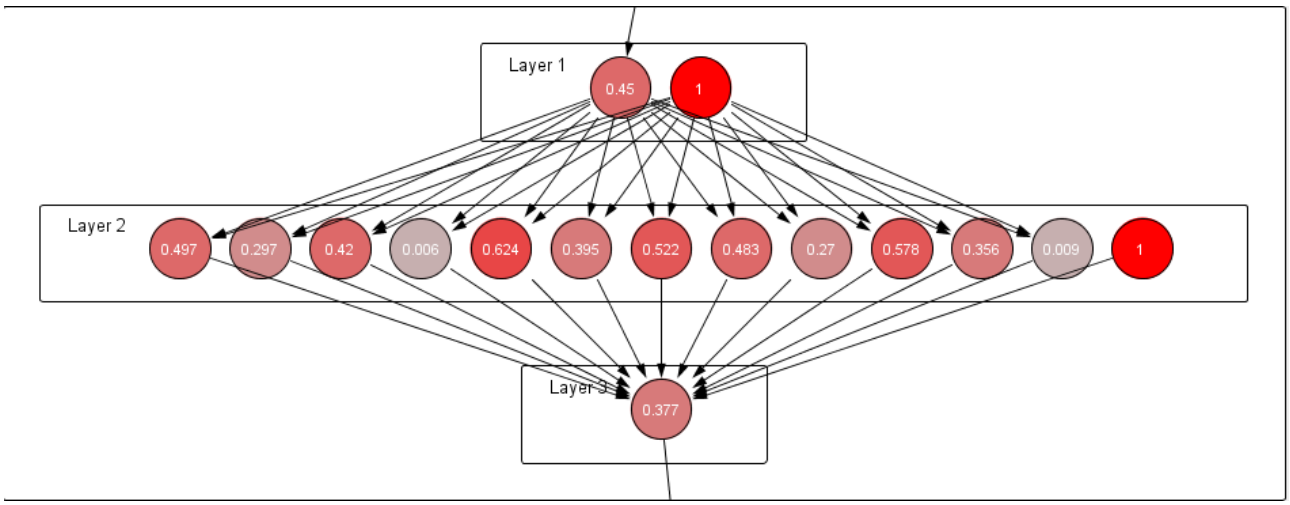
Նկար 46-ում պատկերված են նաև նեյրոնների միացումների «ամրությունը»: Կենսաբանական ուղեղի հետ համեմատ, արհեստական նեյրոնային ցանցում նեյրոնների կշիռները/կապերը ունեն նույն իմաստը թե որքան կարևոր է տվյալ նեյրոնից եկած ինֆորմացիան:



Նկար 45.Նեյրոնային ցանցը ուսուցուման ավարտից հետո:



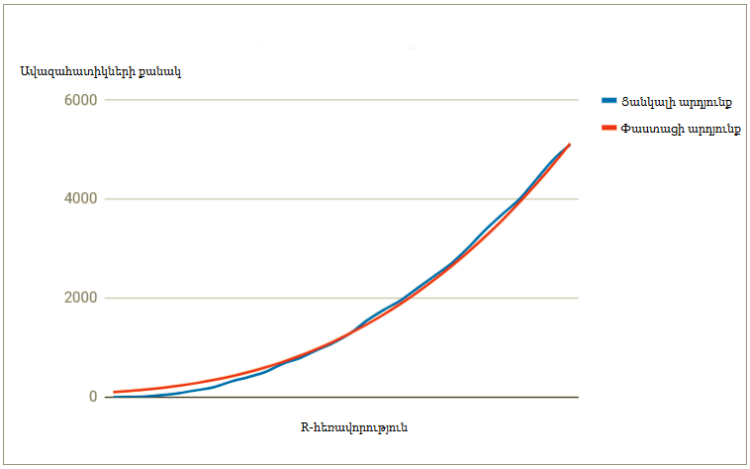
Նկար 46.Նեյրոնների կապերի «ամրությունը»:



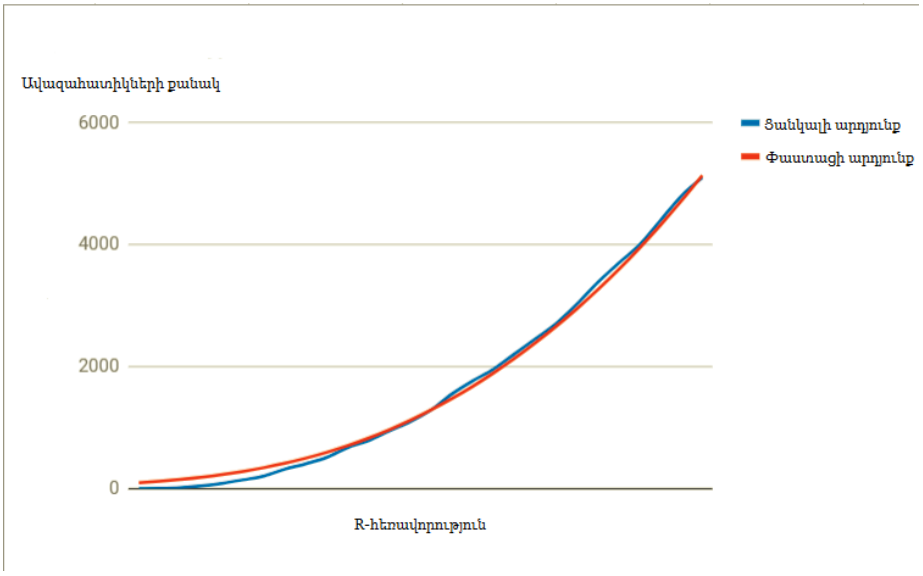
Նկար 47.Նեյրոնների կշիռները 12 թաքնված նեյրոնի դեպքում:

Նկար 46 և 47-ից երևում է տարբեր քանակությամբ նեյրոններ պարունակող նեյրոնային ցանցերի միջև ստացված կշիռների տարբերությունը, որոնք ստացվել են նույն ուսուցողական տվյալների հենքի հիման վրա:

Մշակվել և տեստավրովել են թաքնված նեյրոնների տարբեր քանակությամբ նեյրոնային ցանցեր, որոնցից լավագույն արդյունք են տվել այն ցանցերը, որոնք բաղկացած են եղել 10 և 14 թաքնված նեյրոններից: Նշեմ նաև, որ այդ նեյրոնային ցանցերի կողմից տրված արդյունքները բավականաչափ մոտ են եղել միմիանց (Նկար 48 և 49):



Նկար 48.14 թաքնված նեյրոններից ստացված արդյունքի և ճշգրիտ արդյունքների համեմատությունը:



Նկար 49.10 թաքնված նեյրոններից ստացված արդյունքի և ճշգրիտ արդյունքների համեմատությունը:

Մասնավորապես, 10 թաքնված նեյրոն ունեցող ցանցի դեպքում աղյուսյական պատկերը բերված է աղյուսակ 3-ում:

Հեռավորություն	Ցանկալի արդյունք	Ստացված արդյունք	Սխալանք	Տոկոսային սխալանք
1	1	97	-96	9600%
2	4	121	-117	2925%
3	16	151	-135	843.75%
4	44	187	-143	325%
5	88	231	-143	162.5%
6	144	283	-139	96.5277777777778%

7	208	344	-136	65.3846153846154%
8	320	416	-96	30%
9	408	499	-91	22.3039215686275%
10	512	595	-83	16.2109375%
11	672	704	-32	4.76190476190476%
12	788	828	-40	5.0761421319797%
13	948	966	-18	1.89873417721519%
14	1096	1120	-24	2.18978102189781%
15	1288	1290	-2	0.15527950310559%
16	1552	1476	76	4.89690721649485%
17	1768	1678	90	5.09049773755656%
18	1960	1898	62	3.16326530612245%
19	2208	2134	74	3.35144927536232%
20	2456	2389	67	2.72801302931596%
21	2708	2661	47	1.73559822747415%
22	3028	2953	75	2.47688243064729%
23	3384	3264	120	3.54609929078014%
24	3700	3595	105	2.83783783783784%
25	3996	3948	48	1.2012012012012%
26	4394	4323	71	1.61583978152025%
27	4800	4720	80	1.66666666666667%
28	5098	5138	-40	0.78462142016471%
29	5494	5576	-82	1.49253731343284%
30	5900	6029	-129	2.1864406779661%
31	6388	6493	-105	1.64370695053225%
32	6786	6959	-173	2.54936634246979%

33	7204	7418	-214	2.9705719044975%
34	7678	7858	-180	2.34436051054962%
35	8302	8268	34	0.409539869910865%
36	8694	8636	58	0.667126754083276%
37	9186	8956	230	2.50381014587416%
38	9804	9223	581	5.92615259077927%

Աղյուսակ 3. 10 թաքնված նեյրոն ունեցող ցանցի դեպքում ստացված արժեքներ:

Աղյուսակ 3-ից և 48,49 նկարներից երևում է, որ նեյրոնային ցանցերի կողմից ճշգրիտ արդյունքներին ավելի մոտ արդյունք ստացվում է R հեռավորության մեծացմանը զուգընթաց: Այսպիսով «Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդրի հետազոտմանը միտված նեյրոնային ցանցերի մշակումը հնարավորություն է ստեղծում փոխարինել ավազակույտի աշխատանքի սիմուլյացիան համապատասխան նեյրոնային ցանցով՝ սույնով օպտիմիզացնելով պահաջվող սպասվելիք արդյունքների ստացման ժամանակը:

Եզրակացություն չորրորդ գլխի վերաբերյալ

Բջջային ավտոմատների և ինչպես նաև ինքնակազմակերպ համակարգերի ուսումնասիրման նկատառումներով և հաշվի առնելով CA Simulator համակարգի ֆունկցիոնալ և տեխնիկական առավելությունները համեմատած արդի լուծումների հետ, CA Simulator-ը կարող է հանդիսանալ նպատակահարմար ծրագրամիջոց ավազակույտի մոդելի ուսումնասիրման համար: Հաշվի առնելով նաև փաթեթի ընդլայնելիությունը, նոր մոդելների ներմուծման դյուրին լինելը, սիմուլատորը կարող է դառնալ հիմնական հետազոտական գործիք բազմաթիվ լոկալ և վիրտուալ լաբորատորիաների համար:

d չափանի խորանարդային ցանցերում աստղային ծածկույթները նկարագրող բանաձևի հիման վրա նույն տոպոլոգիական տարածությունում ավազակույտի աբելյան մոդելի աշխատանքի զուգահեռացումը հնարավոր է տալիս կատարել տվյալ մոդելի ցանկալի բնութագրիչների 50 և ավել անգամ ավելի արագ քան մինչ այժմ գոյություն ունեցող գործիքներն են: Հաշվի առնելով նաև բազմաթիվ բնութագրիչների գոյությունը, որոնց արժեքները նկարագրող բանաձևեր գոյություն չունեն, ավազակույտի մոդելի աշխատանքի սիմուլացիան հանդիսանում էր արդի լուրջ խնդիր:

«Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդիրը լուծող նեյրոնային ցանցի գոյությունը հնարավորություն է ընձեռում ավազակույտի մոդելի սիմուլացիան փոխարինել տվյալ նեյրոնային ցանցով՝ ժամանակի խնայման նկատառումներով:

ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ

Մշակված բազմաօգտատեր ծրագրային համակարգը՝ հիմնված ավազակույտի մոդելով նկարագրվող ինքնակազմակերպվող դինամիկ պրոցեսների հետազոտման համար, թույլ է տալիս հետազոտել և հաշվարկել տարբեր ֆիզիկական, ինֆորմացիոն բնութագրիչներ երկչափ և եռաչափ տարածություններում, ինչպես նաև կատարել համատեղ հետազոտություններ՝ աշխատանքների միաժամանակյա դիտման, փոփոխությունների կատարման, մոդելներում վիճակների պահպանման, ստացման և ուղարկման հնարավորություններով:

Իրականացված ծրագրային փաթեթների միջոցով d չափանի խորանարդային ցանցում ավազակույտի մոդելի աշխատանքի զուգահեռացումը՝ հիմնված նույն տոպոլոգիական տարածությունում աստղային լրիվ ծածկույթները նկարագրող բանաձևի վրա, հնարավորություն են տալիս օգտագործողներին կատարել բնութագրիչների հաշվարկ միչև 3 անգամ ավելի արագ նույն CPU-ի սահմաններում, և մինչև 50 անգամ ավելի արագ համարժեք վիդեոկարտայի դեպքում:

Ստեղծված բազմաօգտատեր «լուրջ խաղ»-ի օրինակը՝ հիմնվելով ավազակույտի մոդելի վերաբերող բազմաթիվ թեորեմների վրա և չունենալով հաղթանակին տանող ակընհայտ ստրատեգիա, խթան է հանդիսանում ինքնակազմակերպվող համակարգերի հանդեպ եղած հետաքրքրության մեծացմանը, հետևաբար նաև համատեղ ուսուցմանը և հետազոտմանը:

«Մինիմում ավազահատիկներ և մաքսիմալ հեռավորություն» խնդրի հետազոտմանը միտված նեյրոնային ցանցերի մշակումը հնարավորություն է ստեղծում փոխարինել ավազակույտի աշխատանքի սիմուլյացիան համապատասխան նեյրոնային ցանցով՝ սույնով օպտիմիզացնելով պահաջվող սպասվելիք արդյունքների ստացման ժամանակը:

Նախագծված կլաստերային համակարգի սիմուլատորը՝ հիմված ավազակույտի և rotor-router մոդելների աշխատանքի ալգորիթմի վրա, դյուրացնում է ինքնակազմակերպվող համակարգերի աշխատանքային ալգորիթմի դրսևորման հետազոտումը կլաստերային համակարգերում: Ինչպես նաև նկարագրված և առաջ բերված հիպոթեզը rotor-router մոդելների տրամաբանությամբ խնդիրների տարաբաշխման համակարգի վերաբերյալ կարող է դինամիկ խնդիրների պլանավորման նոր հիմք հանդիսանալ:

ՀՂՈՒՄՆԵՐ

[1] W. Banzhaf, Self-organizing systems. In: Meyers, R.A. (ed.) Encyclopedia of Complexity and Systems Science, Springer, Heidelberg, pp. 8040–8050, 2009.

[2] Introduction to self organized crititicality, [online]. Available: https://en.wikipedia.org/wiki/Self-organized_criticality

[3] D. L. Turcotte, “Self-organized criticality”, Rep. Prog. Phys., vol *62,* 1377, 1999.

[4] Jarkko Kari , “Theory of cellular automata: A survey”, Theoretical computer science, vol. 334, Issues 1-3, pp. 3-33, 15 apr.2005

[5] S. Wolfram, “Cellular automata as models of complexity”, Nature 311, pp. 319-424, 04 oct. 1984

[6] (2002)The Wikipedia website, [online]. Available: https://en.wikipedia.org/wiki/Abelian_sandpile_model

[7] L. Levine, “The Rotor-Router Model”, [Online]. Available arXiv:math/0409407

[8] B. Priezzhev, D. Dhar, A. Dhar, and S. Krishnamurthy, “Eulerian walkers as a model of self-organised criticality”, Phys. Rev. Lett. 77, 50795082, 1996.

[9] Per Bak, *How Nature Works: the science of self-organized criticality*, New York, NY: Copernicus Press, 1996.

[10] A. Fey, L. Levine and D. B. Wilson, “Approach to criticality in sandpiles”, Phys. Rev. E 82, 031121, 15 sept. 2010

[11] The Abelian Sandpile Model visualising tool, [Online], Available: http://www.natureincode.com/code/various/sandpile.html

[12] Seth Tisue and Uri Wilensky, “NetLogo: A Simple Environment for Modeling Complexity”, International Conference on Complex Systems, Boston, May 1621, 2004.

[13] Visual Sandpile, [Online], Available: https://github.com/flrs/visual_sandpile

[14] J. L. J. Laredo, P. Bouvry, F. Guinand, B. Dorronsoro and C. Fernandes "The sandpile scheduler", Cluster Computing, vol. 17, pp 191–204, June 2014.

[15] J. Gąsior and F. Seredyński, "A Sandpile cellular automata-based scheduler and load balancer", Journal of Computational Science, vol. 21, pp. 460-468, July 2017.

[16] Seriouse game description, [Online], Available: https://en.wikipedia.org/wiki/Serious_game

[17] P. Bak, C. Tang and K. Wiesenfeld, “Self-organized criticality: An explanation of the 1/f noise”, Phys. Rev. Lett., vol. 59, no. 4, pp. 381–384, 1987.

[18] D. Dhar, “Self-organized critical state of sandpile automaton models”, Phys. Rev. Lett., vol. 64, no. 14, pp. 1613–1616, 1990.

[19] J. A. Bonachela and M. A. Mu˜noz, “Confirming and extending the hypothesis of universality in sandpiles”, Phys. Rev. E, 78(4):041102, 2008. arXiv:0806.4079.

[20] P. Grassberger and S. S. Manna, “Some more sandpiles”, J. Phys. France, vol. 51, pp. 1077–1098, 1990.

[21] V. S. Poghosyan, S. Y. Grigorev, V. B. Priezzhev and P. Ruelle, “Pair correlations in the sandpile model: A check of logarithmic conformal field theory”, Phys. Lett. B, vol. 659, pp. 768–772, 2008.

[22] Su. S. Poghosyan, V. S. Poghosyan, V. B. Priezzhev and P. Ruelle, “Numerical study of correspondence between the dissipative and fixed-energy Abelian sandpile models”, Phys.Rev. E, 84, 066119, 2011.

[23] A. Fey, L. Levine and D.B. Wilson, “Driving Sandpiles to Criticality and Beyond”, Phys. Rev. Lett., 104, 145703, 2010.

[24] A. Fey, L. Levine and D. B. Wilson, “Approach to criticality in sandpiles”, Phys. Rev. E 82, 031121, 2010.

[25] V. S. Poghosyan, S. Y. Grigorev, V. B. Priezzhev and P. Ruelle, “Logarithmic two-point correlators in the Abelian sandpile model”, J. Stat. Mech., vol. 2010, no. 07, P07025, 2010.

[26] V. S. Poghosyan and V. B. Priezzhev, “The problem of predecessors on spanning trees”, Acta Polytechnica, vol. 51, no. 1, pp. 59–62, 2011.

[27] S. N. Majumdar and D. Dhar, “Height correlations in the Abelian sandpile model”, J. Phys. A: Math. Gen., vol. 24, no. 7, L357–L362, 1991.

[28] N. Winslow, "Introduction to Self-Organized Criticality & Earthquakes", Internet document, [online] Available: http://www.geo.lsa.umich.edu/~ruff/DGeo105.W97/SOC/SOCeq.html.

[29] P. Bak and K. Chen, “Self-Organized Criticality”, Scientific American, vol. 264, no. 1, pp. 46–53, 1991.

[30] Cellular automata simulator Golly, [Online] Available: https://sourceforge.net/projects/golly/

[31] Cellular atomata simulator LifeLab, [Online] Available: http://www.trevorrow.com/lifelab/

[32] Cellular automata simulator Jcasim, [Online]. Available: http://www.jcasim.de/

[33] Cellular automata simulator celldemo, [Online]. Available: https://github.com/devinacker/celldemo

[34] Wolfram Mathematica wegpage [Online]. Available: https://www.wolfram.com/mathematica/

[35] MathLab webpage [Online]. Available: https://www.mathworks.com/products/matlab.html

[36] C. Desprat, J.P. Jessel and H. Luga, “3DEvent: a framework using event-sourcing approach for 3D web-based collaborative design in P2P”. International Conference on Web3D Technology (Web3D 2016), Anaheim, CA, 22/07/16-24/07/16, ACM, p. 73-76, July 2016.

[37] Thi Thuong Huyen Nguyen and Thierry Duval, "A Survey of Communication and Awareness in Collaborative Virtual Environments", 2014 International Workshop on Collaborative Virtual Environments (3DCVE), Mar 2014, Minneapolis, United States. IEEE, DOI: 10.1109/3DCVE.2014.7160928, 2014.

[38] S. Frehmel, “The Sandpile Model: Parallelization of Efficient Algorithms for Systems with Shared Memory”. In: Bandini S., Manzoni S., Umeo H., Vizzari G. (eds) Cellular Automata. ACRI 2010. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, vol. 6350, pp. 35-45, 2010.

[39] L. Levine and Y. Peres, “Asymptotics for Rotor-Router Aggregation and the Divisible Sandpile”, Potential Anal 30, 2009: [Online] Available: https://doi.org/10.1007/s11118-008-9104-6

[40] A. Fey and F. Redig, “Limiting shapes for deterministic centrally seeded growth models”, J. Stat. Phys. 130, no. 3, 579597, 2008.

[41] Cumputer cluster definition [online]. Available: https://en.wikipedia.org/wiki/Computer_cluster

[42] Y. Rabani, A. Sinclair and R. Wanka, “Local divergence of Markov chains and the analysis of iterative load-balancing schemes”. In IEEE Symp. on Foundations of Computer Science, DOI: 10.1109/SFCS.1998.743520, 1998.

[43] V. S. Poghosyan, S. S. Poghosyan and H. E. Nahapetyan, “The Investigation of Models of Self-Organized Systems by Parallel Programming Methods Based on the Example of an Abelian Sandpile Model”, Proc. CSIT Conference 2013, Yerevan Armenia, Sept. 23-27, pp. 260-262, 2013.

[44] H. E. Nahapetyan, S. S. Poghosyan, V. S. Poghosyan and Yu. H. Shoukourian, “The Parallel Simulation Method for d-dimensional Abelian Sandpile Automata”, Mathematical Problems of Computer Science, vol. 46, 117–125, 2016.

[45] H. E. Nahapetyan, “An Example of a Multiplayer Serious Game on 3D Sandpile Model”, Mathematical Problems of Computer Science, vol. 48, pp. 5–13, 2017.

[46] H. E. Nahapetyan and S. S. Poghosyan, “Dynamic Task Scheduling Based on Abelian Sandpile and Rotor-Router Models”, Mathematical Problems of Computer Science, vol. 49, pp. 41-48, 2018.

[47] A. M. Povolotsky, V. B. Priezzhev and R. R. Shcherbakov, ”Dynamics of Eulerian walkers”, Physical Review e, vol. 58, no. 5, 10.1103/PhysRevE.58.5449, 1998

[48] V. B. Priezzhev, ”Self-organized criticality in self-directing walks”, [Online] Available: arXiv:cond-mat/9605094v1

[49] D. Dhar, ”Theoretical studies of self-organized criticality”, Physica A: Statisitcal Mechanics and its Applications, vol 369, no. 1, pp. 29-70, 1 sept. 2006. [Online] Available: https://doi.org/10.1016/j.physa.2006.04.004

[50] H. E. Nahapetyan , J.-P. Jessel , S. S. Poghosyan, Yu. H. Shoukourian, “A multi-user and multi-purpose CA simulator ”, IEEE conference proceedings, CSIT, pp. 23 – 26, 10.1109/CSITechnol.2017.8312133, 2017.

[51] Neuroph Studio [Online]. Available: http://neuroph.sourceforge.net/

[52] Micorosft .Net technology [Online]. Available: https://www.microsoft.com/net/

[53] Microsoft Azure [Online]. Available: https://azure.microsoft.com/en-us/?v=18.20

[54] Djaouti Damien; Alvarez Julian; Jessel Jean-Pierre. "Classifying Serious Games: the G/P/S model" [Online]. Available: DOI: 10.4018/978-1-60960-495-0.ch006

[55] Graafland M, Schraagen JM, Schijven MP. "Systematic review of serious games for medical education and surgical skills training” The British Journal of surgery 2012 Oct;99(10):1322-30. doi: 10.1002/bjs.8819

[56] Cheng Meng-Tzu, Chen Jhih-Hao, Chu Sheng-Ju and Chen Shin-Yen, "The use of serious games in science education: a review of selected empirical research from 2002 to 2013", Journal of Computers in Education, pp. 353-375, 01 sep. 2015.

[57] Microsoft flith simulator [Online]. Available: https://en.wikipedia.org/wiki/Microsoft_Flight_Simulator

[58] TiltFactor company webpage [Online]. Available: http://www.tiltfactor.org/games/

[59] A force more powerful game description [online]. Available: http://www.mobygames.com/game/a-force-more-powerful-the-game-of-nonviolent-strategy

[60] McCulloch, Warren and Walter Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. vol. 5, no. 4, pp. 115–133, doi:10.1007/BF02478259, Dec. 1943.

[61] H. E. Nahapetyan, “Usage of Neural Networks for ASM Research”, Mathematical Problems of Computer Science, vol. 49, pp. 58-65, 2018.

[61] OpenMP technology webpage [Online]. Available: https://en.wikipedia.org/wiki/OpenMP

[63] Cuda technology webpage [Online]. Available: https://developer.nvidia.com/cuda-zone

ՀԵՂԻՆԱԿԻ ԿՈՂՄԻՑ ՀՐԱՏԱՐԱԿՎԱԾ ԱՇԽԱՏՈՒԹՅՈՒՆՆԵՐ

1. V. S. Poghosyan, S. S. Poghosyan and H. E. Nahapetyan, “The Investigation of Models of Self-Organized Systems by Parallel Programming Methods Based on the Example of an Abelian Sandpile Model”, Proc. CSIT Conference 2013, Yerevan Armenia, Sept. 23-27, pp. 260-262, 2013.
2. H. E. Nahapetyan, S. S. Poghosyan, V. S. Poghosyan and Yu.H. Shoukourian, “The Parallel Simulation Method for d-dimensional Abelian Sandpile Automata”, Mathematical Problems of Computer Science, vol. 46, pp. 117–125, 2016.
3. H. E. Nahapetyan, J.-P. Jessel, S. S. Poghosyan, Yu. H. Shoukourian, “A multi-user and multi-purpose CA simulator”, IEEE conference proceedings, pp.23 – 26, 2017, 10.1109/CSITechnol.2017.8312133
4. H. E. Nahapetyan, “An Example of a Multiplayer Serious Game on 3D Sandpile Model”, Mathematical Problems of Computer Science, vol. 48, pp. 5–13, 2017.
5. H. E. Nahapetyan, S. S. Poghosyan, “Dynamic Task Scheduling Based on Abelian Sandpile and Rotor-Router Models”, Mathematical Problems of Computer Science, vol. 49, pp. 41-48, 2018.
6. H. E. Nahapetyan, “Usage of Neural Networks for ASM Research”, Mathematical Problems of Computer Science, vol. 49, pp. 58-65, 2018.