O’ZBEKISTON RESPUBLIKASI OLIY VA O’RTA MAXSUS TA’LIM VAZIRLIGI MIRZO ULUG’BEK NOMIDAGI O’ZBEKISTON MILLIY UNIVERSITETI MATEMATIKA FAKULTETI ALGORITMLAR VA DASTURLASH TEXNOLOGIYALAR KAFEDRASI

Qo’lyozma huquqida

UDK 004.8

Kasimov Saidaxmad Nadimulla o’g’lining

“NP SINFIDAGI MASALANI GENETIK ALGORITMLAR VA AVTOMATLASHGAN DASTURLASH ORQALI “AQLLI CHUMOLI HAQIDA” MISOLIDA YECHISH”

5A330101 – Axborot tizimlarining matematik va dasturiy ta’minoti Magistr akademik darajasini olish uchun yozilgan

DISSERTATSIYA

Ilmiy rahbar: f-m.f-n.dots.Gaynazarov S.N.

Toshkent – 2018 yil

MUNDARIJA

KIRISH

Magistrlik dissertatsiyasi mavzusining asoslanishi va uning dolzarbligi. Oxirgi paytlarda boshqaruvchi tizim dasturiy ilovasini ishlab chiqishda avtomatlashgan dasturlashni qo'llash keng ishlatilmoqda – bu dasturlashning paradigmasi bo'lib, bunda boshqaruv obyektlarini avtomatlashtirilgan ko'rinishda qurish tavsiya qilinadi, bunda har bir obyekt o'zida boshqaruv tizimini o'z ichiga oladi (bir yoki bir necha o'zaro ta'sirlashadigan boshqaruvchi avtomatlar) va boshqaruv obyekti.

Boshqaruv obyekti ko'plab **hisoblash holatlari** bilan tavsiflanadi, shu bilan birga ikkita funksiya bilan: ko'plab predikatlar (bular avtomatga kiruvchi o'zgaruvchilar yoki avtomatga kiruvchi ta'sir qiluvchi hisoblanadi), hisoblash holatini mantiqiy qiymatini ko'rsatadi (rost yoki yolg'on), ko'plab ta'sir qiluvchilar hisoblash holatini o'zgartirib yubora oladigan avtomatning chiqish ta'sir qiluvchisi hisoblanadi. Bundan tashqari, tashqi muhit hodisani generatsiya qilishi mumkin. Bu esa boshqaruv avtomatning kirish qismiga tushishi mumkin bo'ladi.

So'nggi o'n yilliklarda *dasturiy ilovaning qidirish injeneriyasi* (*Search-Based Software Engineering, SBSE*) izlanish sohasi jadal rivojlanib kelmoqda. Bu soha doirasida *dasturiy injeneriya* masalasini yechish uchun (talab tahlili, ishlab chiqish jarayonini bashorat qilish, loyihalash, testdan o'tkazish va refaktoring shular jumlasidan) *qidirish optimizatsiyasi algoritmi* taklif qilinadi. Aniqlangan qidirish injeneriya dasturiy ilovalarini ishlatish metodlari ichiga evolutsion algoritmlar (*genetik algoritmlar, genetik dasturlash va evolutsion strategiyalar*) va *chumoli algoritmi, qismlarni kavlash metodi, kuyishni imitatsiya qilish metodi, qo'yib yuborish metod*lari kiradi. Lekin shunda ham hozirda eng ko'p tarqalgan algoritm evolutsion algoritmlardir.

Tadqiqot obyekti va predmeti. Dissertatsiya ishining obyekti "Aqlli chumoli" masalasi va bu masalani yechishda foydalaniladigan algoritm va

usullardir. Predmeti esa, berilgan masalani yechimi sifatida dasturiy ta'minot yaratilishidir.

Tadqiqot maqsadi va vazifalari. Ancha yillardan beri muhokamaga sabab bo'lib kelayotgan, hali aniq yechimga ega bo'lmagan masalalarga yechim topish va uni dasturiy ta'minot sifatida tadbiq etish. Tadqiqot genetik algoritm masalasi bo'lgan aqlli chumoli masalasiga yechim topishga yo'naltirilgan.

Ilmiy yangiligi. Dissertatsiya ishida hozirgi kunda actual bo'lib turgan genetik algoritm, avtomatlashtirilgan dasturlash va fitness funksiyalar ko'rib chiqilgan bo'lib, amaliyotda qo'llash doirasi keng hisoblanadi. Yangilik sifatida shuni aytish mumkinki, dissertatsiya ishida genetik algoritm, fitness funksiya va avtomatlashgan dasturlashning vazifasi etib aqlli chumoli masalasi qo'yilgan, masalaning absolyut qiymatiga yaqin yechimi keltirib chiqarilgan. Chunki bunday masalalarda taqribiy yechim hisobga olinadi.

Tadqiqotning asosiy masalalari va farazlari. Qo'yilgan masalaga genetik algoritm, fitness funksiyalardan foydalangan holda, aqlli chumoli masalasi yechimi misol tariqasida olingan, aqlli chumoli berilgan hudud ichida 200 ta qadam ichida iloji boricha eng ko'p olmani yeyishiga erishish ko'zda tutiladi.

Tadqiqot mavzusi bo'yicha adabiyotlar sharhi (tahlili). Amaliy masalalarni yechishda qo'llaniladigan intellektual tahlil, avtomatlashgan dasturlash masalalari, fitness funksiya masalalari, genetik algoritmlar bo'yicha ko'plab ilmiy izlanuvchilar izlanish olib bormoqdalar. Bulardan: D.Xopkroft, P.Motvani, D.Ulman, L.Gladkov, V.Kureychik ilmiy izlanishlari bo'yicha o'ziga xos usullar yaratganlar.

Tadqiqotda qo'llaniladigan metodikaning tavsifi. Tadqiqot jarayonida fitness funksiya, genetik algoritmlar orqali "Aqlli chumoli" masalasi yechilgan, avtomatlashgan dasturlash usulida dasturiy ta'minoti tarkibi yaratilgan va shular orqali natija tahlil qilingan.

Tadqiqot natijalarining nazariy va amaliy ahamiyati. Dasturiy ta’minotni yaratish va uni real masalada amaliyotdan o’tkazish ko’zda tutilgan. Olingan natijalar nazariy va amaliy ahamiyatga ega bo’lib, genetik algoritm masalalarini hayotga tadbiq etishda qo’llanilishi mumkin.

Ish tuzilmasining tavsifi. Yaratilgan magistrlik dissertatsiya ishi kirish qismi, uch bob, xulosa, foydalanilgan adabiyotlar va ilovadan iborat.

Izlanish (tadqiqot) metodlari. Dissertatsiya ishida diskret matemetika va evolutsion algoritmlar avtomatlar nazariyasi metodi qo’llaniladi.

1 - bobda NP sinf masalalari tushunchalari, genetik algoritm, avtomatlashgan dasturlash, “Aqlli chumoli haqida” gi ma’lumotlar berilgan.

Dissertatsiya ishida qo’yilayotgan masalaning berilishi tavsiflangan va o’rganib, yechim uchun misollar keltirilgan.

2 - bobda dissertatsiyada ishida berilgan masala yechimi, genetik algoritmlarni qo’llash, qo’llanilgan algoritmlarga avtomat dasturlash orqali dastur tuzish sxemasi, masala yechimi tadbiqi sifatida populyatsiya misoli olinishi va uni yechish usuli, genetik operatorlar tushunchasi, foydalanilgan algoritmlarning to’g’irlanadigan parametrlari va kelib chiqqan natija tavsiflari keltirib o’tilgan.

3 - bobda, algoritm ishlashi amaliy tadbiqi keltirib o’tilgan bo’lib, 3.1 bo’limda bu ishni amalga oshirishda foydalanilgan global optimizatsiya freymvorki to’g’risida ma’lumot berib o’tilgan, uni tadbiq etish dastur kodi esa ilovada keltirilgan. 3.2. bo’limda esa dasturdan foydalanish ketma-ketligi, ya’ni foydalanuvchiga qo’llanma berilgan.

Ushbu dissertatsiya ishida biz genetik algoritm deb ataladigan bir qator murakkab algoritmlarni sinab ko'rdik, dasturiy ta'minotni qurish uchun avtonom yondashuvni ko'rib chiqdik, umumjahon optimallashtirishning odatiy muammolarini o'rganib chiqdik va ushbu muammoga dasturiy ta'minot shaklida yechim topish uchun algoritm ishlab chiqdik.

I BOB. MASALA QO'YILISHI HAQIDA UMUMIY MA'LUMOT

1.1.NP SINF MASALALARI

NP sinf masalalari (ing. *Non-deterministic polynomial*) nazariyasida ko'plab muammoni yechimi deb ataladi. Bu masala yechimlarini Tyuring mashinasida vaqtga tekshirish mumkin, bunda kiruvchi ma'lumotlar polinomdan oshib ketmasligi kerak bo'ladi, qo'shimcha tarzda ma'lumotlar ham bo'lishi kerak (*yechim sertifikati* deb ataladi).

NP sinfida ekvivalent masalani o'z ichiga oluvchisini aniqlash mumkin, bu o'z qatorda polinomial vaqtda nodeterminallashgan Tyuring mashinasida yechimni aniqlashga olib keladi.

Polinomial vaqtga ega bo'lgan algoritmlar yechimini kompyuterda tezroq yechish mumkin, bunda eksponensial vaqtda to'g'ridan-to'g'ri tanlov amalga oshadi. Bu P va NP sinflari tengligi muammosining amaliy qiymatini shartlashishiga olib keladi.

NP sinfiga tegishli qo'shimcha tillar sinflari NP-osti sinflari deb ataladi. Hali isbotlanmagan bo'lsa ham, bu sinf NP sinfdan farq qiladi. NP va NP-osti sinflarini kesishishi P sinfni o'z ichiga oladi. NP sinfi o'z ichiga P sinfni oladi. Lekin bu holatning qat'iligi haqida hech narsa ma'lum emas.

P va NP sinflarning tengligi masalasi algoritmlar nazariyasining markaziy muammolaridan biri hisoblanadi. Agar ular teng bo'lsa, NP sinfdagi istalgan masalani tez yechish mumkin (polinomial vaqt hisobiga). Lekin ilmiy jamoa bu savolga javobni inkor qilish tarafdoridirlar.

NP sinfi boshqa kengroq sinflarga qo'shiladi, masalan PH sinfiga. Shu bilan birga boshqa sinflarga NP sinfini qo'shilishi qat'iligi haqida ochiq savollar mavjud.

Hozirgi kungacha P sinfi NP sinfiga tegishliligi to'g'risidagi ko'plab masalalarni keltirish mumkin, bulardan:

- Bul formulasini bajarilishi haqidagi masala: quyidagi bul formula orqali unga kiruvchi o’zgaruvchilarni 1 ga aylantirishni aniqlash. Sertifikat – shunday tanlov asosida.
- Tugmani bosish masalasi: quyidagi berilgan o’lchamdagi grafa orqali unda tugmalar bor yoki yo’qligi aniqlansin (to’liq grafostilari). Sertifikat- tugmani hosil qiluvchi cho’qqilar raqami.
- Grafada gamilton siklini aniqlash. Sertifikat – gamilton siklini tashkil qiluvchi cho’qqilar ketma-ketligi.
- Kommivoyajer masalasining optimizatsiya qilinmagan variant (k qiymatidan uzun bo’lmagan marshrut mavjudmi) – bundan oldingi masalani keng va real holatga yaqinlik darajasi yuqori yechimi. Sertifikat – shunday marshrut.
- Berilgan tizim chiziqli tengsizlikning butun sonli yechimining mavjudligi. Sertifikat- yechim.

NP sinfining masalalari ichidagi “eng qiyin” masalalari – NP to’liq masalalaridir. Agar ularning istalgan birini polinomial vaqt ichida yechishni iloji bo’lsa, u holda NP sinf masalalarini polinomial vaqt ichida yechish mumkin bo’ladi.

Kommivoyajer masalasining yopiq versiyasida, grafaning barcha cho’qqilarini ko'rib chiqish kerak, so'ngra original cho’qqiga qaytish kerak bo’ladi. Yopiq versiya ochiqdan farq qiladi, chunki u boshlang'ich cho’qqiga qaytmasligi kerak.

Masalaning ochiq versiyasi cho’qqi uchidan boshlab tegishli yoylari og'irliklarini almashtirish orqali yopilgan versiyaga o’tiladi. Optimal kommivoyajer masalasi yechimi uchun yo'l - bu chizma yopilgan optimal yo'nalishidir, asl chizma bo'lmagani esa yopiqqa mos keladi. Keyin siz chizma uchun yangi tepalik nuqtasiga (Biz original chizma uchlari shu boshlang'ich uchidan boshlaymiz, ya’ni 0 dan va qolgan chizma uchlari ham raqamlanadi, deb taxmin qilinadi) qiymat kiritishingiz lozim bo'ladi. Quyidagi cho’qqida paydo bo'lgan va kirgan yoylarning qiymati quyidagicha tavsiflanadi:

Grafika bo'yicha kommivoyajer masalasi optimal yechimi yopiq bo'lmagan yo'nalishi asl grafigida kommivoyajerning optimal yopiq marshrutiga mos keladi va qo'shimcha qiymatga ega bo’ladi.

Yechim usullari

Soddalari:

- To'liq algoritmlar
- Tasodifiy qidiruv
- Ochko'z algoritmlar
- Eng yaqin qo'shni usul
- Eng yaqin shaharni kiritish usuli
- Eng arzonroq usulni qo'llash
- Eng kam spanning daraxt usuli
- Tavlanishga simulyatsiya usuli

Kommivoyajer masalasini hal qilish uchun barcha samarali (to'liq qidirishni kamaytirish) usuli evristik usulidir. Ko'pgina tarjimai holi usullarida eng samarali yo'l yo'q, ammo bu taxminiy yechim hisoblanadi. Ko'pincha har qanday vaqtli algoritm deb ham ataladi, ya'ni [aniqlashtirish uchun], hozirgi taxminiy echimni asta-sekin yaxshilaydi.

Shuningdek, kommivoyajer masalasi NP-to’liq masalalar sinfiga tegishli komplektiga aylanishi mumkin. Odatda, bunday bayonotlar quyidagilarga o'xshaydi: G ga nisbatan uning narxi x dan oshmaydigan qilishni shunday bir yo'li bormi?. Ko'p holatlarda to'liq tergovning evristik qisqartirishiga yangi yondashuvlar qo'yiladi.

Amalda, samarali usullarning turli modifikatsiyalari qo'llaniladi: filial va chegara usuli, genetik algoritmlar usuli va chumoli koloniyaning algoritmi singarilardir[9,10,11].

1.2.Genetik algoritm

Genetik algoritm – bu evolyutsion dinamikaning tamoyillariga mos tarzda populyatsiya rivojlanishining genetik protseduralarini imitatsiya qilishga

asoslangan algoritm. U ko'pincha ko'p mezonli optimizatsiya, izlash va boshqarish masalalarini yechish uchun ishlatiladi.

Berilgan masala NP-to’liq masalalari sinfiga tegishli hisoblanadi, algoritm ishlash vaqti kommivoyajer masalasini yechishda qo’llaniladigan kiruvchi ma’lumotlarga bog’liq bo’ladi, ya’ni shaharlar soniga.

Amaliyotda har xil modifikatsiya metodlari qo’llaniladi: tarmoq va chegara metodi va albatta genetik algoritmlar, shuningdek chumoli kaloniyasi algoritmi ham.

Masalani genetik algoritm orqali yechishni aniqlash uchun, aynan nima masalaning yechimi ekanligini aniqlash kerak, masalan xromosomalarni kodlashtirish orqali xromosomalarning moslashuvchanlik qobiliyati funksiyasi tuzib chiqiladi. Shundan so’ng masalani yechish mumkin bo’ladi.

Kodlashning bu kamchiligini to’g’irlash uchun bir necha usullar mavjud, lekin ular resurslarning ortiqcha sarfiga olib keladi (hisoblash), xromosomalarni qo’shimcha tekshirish zarur bo’ladi. Yana bir usul moslashuvchanlik qobilyati ichida qaytariluvchi qiymatlarni tekshirib chiqish. Bunda qaytariluvchi qiymatni aniqlab xromosomada uchramaydigan kodga almashtiriladi. Ikkinchi usul – hech narsa tekshirilmaydi, moslashuvchanlik funksiyasining eng kichik qiymatlari xromosomalarga yuklanadi, lekin bu holatda genetik algoritm effektiv ishlamay qo’yadi.

Umuman olganda, genetik algoritmlar uchun genlar ketma-ketligini yechimini kodlash muhim omil bo’lib hisoblanadi. Algoritm ishlash sifatiga qarab yechim aniqlanuvchanligi yuqoriligi kelib chiqadi. Eng asosiysi va muhim jihati shundan iboratki, xromosani kodlashga qo’yilgan talab masala yechimini tavsiflashi kerak, chunki bir xromosomani har xil usulda traklash imkoniyati mavjud bo’lishi kerak emas. Xromosomalar iloji boricha kam bit egallashi kerak. Yana shuni ta’kidlash kerakki kodlash sharti ham sodda bo’lishi kerak. Bunda ishlash tezligi bog’liqligini unutmaslik kerak.

Kodlashdan so’ng istalgan parametrlar bilan genetik algoritm ishga tushiriladi.

Genetik algoritmlar asosiy prinsiplari Golland (Holland 1975-yil) tomonidan formallashtirilgan. Internet saytlarning ko’pida genetik algoritm ustida amalga oshirilgan ishlar haqida bir qator ma’lumotlar berilgan. Hozirgi kunda biologik evolutsiya to’g’risida ko’plab nazariyalar mavjud (J.B.Lamark, P.Teyar de Sharden, K.E.Ber, L.S.Berg, A.A.Lyubishev, S.V.Meyen va boshqalar). Lekin ularning hech qaysi biri umumiy tan olingan nazariya hisoblanmaydi.

Mashhur nazariya Charlz Darvin nazariyasi hisoblanadi. U nazariyasini “Turlar kelib chiqishi” nomli ishida 1859 yilda ko’rsatib o’tgan. Bu nazariya ham boshqa nazariyalar kabi o’zida ko’plab yechilmagan masalalarni jamlaydi. Bu ishdan tashqari bo’lgan muammolar kelib chiqishiga olib keladi.

O’zining ko’plab kamchiliklariga qaramay an’anaviy tarzda Darvin nazariyasi Genetik algoritimda modelizatsiya qilinadi. Albatta, bu boshqa nazariyalarni modelizatsiya qilinmaydi degani emas. Aynan shunday kompyuter modellashtirish va uni Yerdagi real hayot evolutsiyasi bilan solishtirish keyinchalik biologik evolutsiya nazariyasini qoliplashgan yechimiga olib kelishi mumkin.

Darvin nazariyasi alohida turlarga emas, aynan statik yoki dinamik tashqi muhitda yashovchi tirik avlod qoldira oladigan turga tegishli turning populyatsiyasiga tegishli. Darvin evolutsiyasi modelini tuzish asosida tirik organizmga ta’sir qiluvchi tasodifiy o’zgarishlar elementlari yotadi. Va bu o’zgarishlar asta-sekin avloddan-avlodga o’tib boradi. Shuning uchun tabiiy tanlov moslashuvchan organizmlar o’rtasida yuzaga keladi.

Tabiiy tanlov populatsiya ishtirokchilarining raqobati sharti asosida kelib chiqadi. Bu esa moslashuvchan yoki adaptatsiyalashgan tur geni keng tarqaladi va avloddan-avlodga o’tib boradi. Moslasha olmaydiganlar turi esa yo’qolib

boraveradi. Shunday qilib genetik algoritm qaysidir aniqlangan biologik evolutsiya yoki uning variantiga qarab olingan imitatsiya qilingani modeli hisoblanadi. Shu bilan bir qatorda shuni aytib o’tish kerakki, biologik evolutsiya izlanuvchilari hali o’zlari aniq bir to’xtamga kelganlari yo’q.

Genetik algoritm genetika masalalarini yechishda keng foydalaniladi. Bu algoritmning xususiyati – ularning NP-murakkab muammolarni (polinomial o'suvchi algoritmik murakkablikdagi algoritm tuzish mumkin bo'lmagan muammolar) echishda muvaffaqiyat bilan ishlatilishidir. Genetik algoritmlarni boshqa usullar bilan echish mumkin bo'lmagan masalalarni echishda qo'llash mumkin bo'lsa ham ular optimal echimni topish uchun (hech bo'lmaganda, mumkin bo'lgan vaqtda, bu erda polinomial baholar ko'pincha yaroqsiz) kafolat bermaydilar[12,13].

Ko'plab sotsial-iqtisodiy tizimlarni, yagona-evolyutsion nazariya metodlari va vositalari bilan yaxlit pozitsiyalardan bayon etish mumkin.

Evolyutsion modellashtirishtirishda murakkab sotsial-iqtisodiy tizimni modellashtirishtirish jarayoni uning evolyutsiyasi modelini yaratishga yoki tizimning yo'l qo'yiladigan holatlarini izlashga, ko'plab mumkin bo'lgan holatlar (traektoriyalar)ni kuzatish protsedurasiga (algoritmiga) keltiriladi. Sotsial-iqtisodiy tizimlarni evolyutsion modellashtirishtirishda klassik va klassik bo'lmagan matematik modellarni, jumladan, tizimning fazoviy tasnifini (masalan, kaktakli avtomatlar va fraktallar), ichki tizimlarning tasnifi va ierarxiyasini, tajriba va sezgisini hisobga olib ishlatish foydali.

Evolyutsion modellashtirishtirish protsedurasini amalga oshirishning adekvat vositasi genetik algoritmlardir.

Tizim evolyutsiyasini o'rganishda, ta'minot maqsadida, uning ichki tizimlarga dekompozitsiyasi zarur:

• muhit bilan samarali o'zaro harakat;

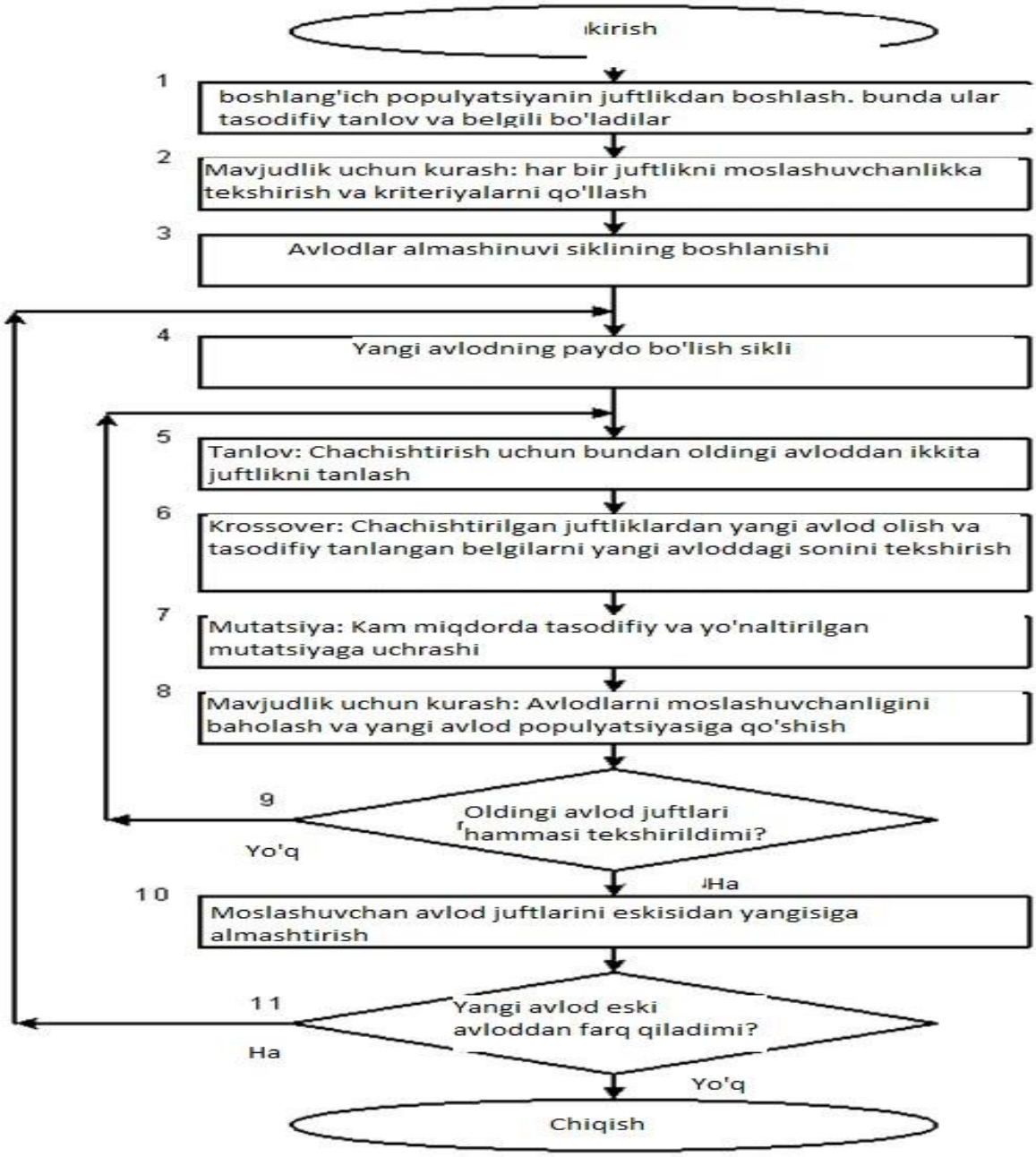
• ichki tizimlar bilan moddiy, energetik, informatsion, tashkiliy resurslarni aniqlovchilar bilan optimal ayirboshlash;

• dinamik almashtirish va maqsadlarning qayta tartiblanish, tasnifli faollik va tizim murakkabligi sharoitlarida Tizimning evolyutsionirlanishi;

• tizimning boshqaruvchanligi, boshqaruvchi ichki tizim va tizim ichki tizimlari bilan samarali bog'liqlik, teskari aloqaning identifikatsiyasi.

Genetik algoritmga misol keltiramiz:

Oddiy genetik algoritm



Genetik algoritm bu yerda iteratsion jarayonni tasvirlamoqda, bu jarayon avlodlar bir-biridan farq qilmaguncha davom etadi. Yoki berilgan vaqt yoki berilgan avlod sonini o'tamaguncha davom etadi. Har bir avlod uchun tanlov, krossover (chachistirish) va mutatsiya amalga oshiriladi.

1-qadam: N ta juftlikdan tasodifiy belgilar tanlovi orqali boshlang'ich populyatsiyani generatsiya qilamiz.

2-qadam: (tiriklik, mavjudlik uchun kurash) har bir juftlikni absalyut moslashuvchanligi *f(i) sharoitdan* hisoblanadi va bu hamma populyatsiyaga tegishli bo'ladi. Keyin proporsional tanlov orqali populyatsiyaga ta'sir qiladigan nisbiy summalar $P_s(i)$ orqali hisoblanadi, absalyut moslashuvchanlik butun populyatsiyaga tegishli bo'lishi tekshiriladi (1.2.1):

$$P_s(i) = \frac{f(i)}{\sum_{i=1}^{N} f(i)} \tag{1.2.1}$$

(1) orqali absalyut moslashuvchanlikni butun populyatsiyaga emas balki aynan i-nchi juftlikka tegishli bo'lishi kelib chiqadi, shu orqali esa o'rtacha moslashuvchanlik qiymati aniqlanadi (1.2.2):

$$\bar{f} = \frac{1}{N}\sum_{i=1}^{N} f(i) \tag{1.2.2}$$

Bundan keyin quyidagi kelib chiqadi (1.2.3):

$$P(i) = \frac{f(i)}{\bar{f}} = \frac{f(i)}{\frac{1}{N}\sum_{i=1}^{N} f(i)} \tag{1.2.3}$$

Agar (3) dan 2 asosli logarifmni oladigan bo'lsak, unda bizda juftlikning belgilari uning tirik qolib avlod qoldirishi haqida ma'lumot beradi (4):

$$I(i) = Log_2 \frac{f(i)}{\bar{f}} \tag{1.2.4}$$

Shuni aytib o’tish kerakki, bu formula Xarkevichning semantik ma’lumotlar soni uchun formula bilan ustma-ust tushadi, agar maqsadni individual tirik qolish va avlodni davom ettirish masalasiga qaratsak. Bu degani, juftlikning ***moslashuvchanligini formallashtiradigan bo’lsak, bu o’zidan ma’lumot sonini keltirib chiqarishi ma’lum bo’ladi. Bu uni fenotipida keyingi avlod davom etishida genotipda saqlanadi.***

Chunki avlod davom ettirish soni uning moslashuvchanligiga proporsional bo’ladi, agarda buni ma’lumot soni deb hisoblaydigan bo’lsak albatta:

- Musbat, berilgan juftlik tirik qoladi va avlodni davom ettiradi, agar ularning soni ma’lumot soniga proporsional bo’lsa;
- Nolga teng, juftlik yetilish yoshigacha yetib boradi, lekin avlod qoldirmaydi(uning soni nolga teng);
- Noldan kamroq, bu juftlik yetilish yoshigacha nobud bo’ladi

Shunday qilib dunyoqarashdan fundamental xulosa chiqarish mumkin, tabiiy tanlov o’zidan generatsiya va tirik qolish, alodni davom ettirish, bir necha avlodlar populyatsiyasi jarayonida ma’lumotni to’planishini tasvirlaydi, keltirib chiqaradi. Bu tizimga aylanadi[10,13].

Bu ma’lumot to’planishi populyatsiyaning har xil ieararxik darajalarida tizimdek amalga oshadi:

- Tizim elementlari: alohida juftliklar;
- elementlarning o'zaro munosabati: avlodlar o'rtasidagi munosabatlar, kelgusi avlodlarga ularning omon qolishlari haqidagi ma'lumotlarning maksimal hajmini yetkazib berish va avlodning davom etishi (eng qulay shaxslardan o'tish va ratsional moslashib olishni meros qilib olsih);

- Tizimning maqsadi: aholining saqlanishi va rivojlanishi jismoniy shaxslarning maqsadlari orqali amalga oshiriladi: individual hayot va avlodning davomiyligi.

Fenotip genotipga mos keladi va uning o'ziga xos xususiyatlaridan biri tashqi ko'rinishidir. Inson atrof-muhit va boshqa jonzotlar bilan fenotipga

muvofiq o'zaro ta'sir qiladi. Agar bu muvaffaqiyatli bo'ladigan bo'lsa, unda genetik ma'lumot fenotipni keyingi avlodlarga yetkazib beradi.

3-qadam: avlodlar davrining boshlanishi.

4-qadam: yangi avlodni shakllantirish davrining boshlanishi.

5-qadam (tanlov): Turning davom etishida ishtirok etishi mumkin bo'lgan shaxslarning nisbiy tanlovi. Yashash va davom etish uchun genotip va fenotipda ijobiy miqdorda ma'lumotga ega bo'lganlar faqatgina tanlanganlar va tanlov ehtimoli bu ma'lumotga mutanosibdir.

6-bosqich (krossover): oldingi bosqichda genlarni davom ettirish uchun tanlangan shaxslar ehtimollikdagilar bilan kesishadi yoki o'zaro faoliyat (rekonstruktsiya) qilinadi.

Agar o'zaro bog'liqlik sodir bo'lsa, avlodlar ota-onalarning har biridan tasodifiy belgilangan xususiyatlarning yarmini oladi. O'g'illarning soni ota-onalarning sog'lig'i bilan mutanosib. Ba'zi GA variantlarida ularning farzandlari paydo bo'lishidan keyin ota-ona genlarini almashtirib, mutatsiyaga kirishadi.

Krossover paydo bo'lmasa, original shaxslar – muvaffaqiyatsizlikka uchraydi va keying avlod ota-onalar mutatsion bosqichga o'tadilar.

7-qadam (mutatsiya): mutatsion operatorlari amalga oshiriladi. Bu holda, ehtimol Pm bilan avlodlarning xususiyatlari tasodifiy boshqalarga o'zgaradi. Tasodifiy mutatsiyalar mexanizmi genetik algoritmlarni foydalanish Monte-Karlo usuli sifatida mashhur, buni unutmang.

8-qadam (mavjudlik uchun kurash): avlodlarning farovonligi baholanadi (2 bosqichda bo'lgani kabi bir xil algoritm bilan).

9-qadam: tanlangan kishilar avlod berganmi-yo'qmi tekshiriladi.

Keyingi bosqichga borish - Agar yo'q bo'lsa 10, aks holda, 5 qadam harakati va yangi avlod shakllanishi.

10-qadam: avlodlar o'zgarishi:

- avlodlar yangi avlodga joylashtiriladi;

- katta avlod Fittestga yangitdan yo'naltiriladi va ularning har biri uchun bir belgilangan aniq bir soni mavjud, bundan ortiq bo'lishi mumkin emas;

- olingan yangi aholi eskisi o'rnini egallaydi.

11-qadam: genetik algoritmni to'xtatish holatining bajarilishi tekshiriladi. Genetik algoritmdan chiqmasdan turib ham yangi avlod avvalgidan sezilarli darajada farq bo'lsa, ular aytganidek, bu, deb, «algoritm yakunlangan", yoki o'tgan avlodlar va biron-bir raqami bor edi deb algoritm belgilangan vaqt (hech qanday "Ketma-ketlik" va dinamik belgilangan vaqt ichida eritmani topib bo'lmaydigan holatda o'chiriladi).

Agar GA yaqinlashgan bo'lsa, bu yechim topilgan degan ma'noni anglatadi, ya'ni, bu turg'un sharoitlariga ideal tarzda mos keladigan avlodga erishildi.

Aks holda - 4 bosqichga o'tish - yangi avlodni shakllantirishning boshlanishi.

Haqiqiy biologik evolyutsiyada bu cheklangan emas, chunki muayyan ekologik mavqei rivojlanishidan tashqari har qanday populyatsiya ham o'z chegaralari doirasidan tashqariga va odatda "qo'shni" bo'lgan boshqa narsalarni o'rganishga harakat qiladi. Bu jarayonlar tufayli dengiz hayotini tark etib, havo maydoniga va tuproqning sirt qatlamiga hayot kirib borgan va tashqi makonni egallagan.

Albatta, ilmiy tadqiqotlar olib boriladigan haqiqiy genetik algoritmlar odatda berilgan misolga juda o'xshash emas. Tadqiqotchilar genetik algoritmlarning turli parametrlari bilan tajriba o'tkazadilar, masalan: o'tish uchun shaxslarni tanlash usullari; fitnes mezonlari va atrof-muhit omillarining qat'iyligi; ota-onadan avlodga uzatiladigan xususiyatlarni tanlash usullari (retsessiv va resesssiz genlar va boshqalar); intensivlik, tasodifiy taqsimot turi va mutatsiyalarning orientatsiyasi; ko'paytirish va tanlashda turli yondashuvlar.

Shuning uchun "genetik algoritmlar" deganda aslida faqat bitta model emas, balki ba'zan bir-biriga juda o'xshamaydigan keng algoritm sinfini tushunish kerak.

Ko'rib chiqish operatorlari, krossover va mutatsiyada turli tanlov berilmoqda: turnir tanlash (Brindle, 1981; Goldberg va Deb, 1991), tanlash uchun n shaxslar, har bir musobaqada k elementlar bir namunasi asosida qurilgan n musobaqalarda amalga oshiradi va tanlash ular orasida eng yaxshi shaxslar (k bilan eng keng tarqalgan turniri tanlash = 2); elita tanlovi (De Jong, 1975) aholining eng yaxshi yoki eng yaxshi a'zolari tanlashda majburiy ravishda yashashini ta'minlashi kerak (eng keng tarqalgan protsedura tanlov jarayoni, krossover va mutatsiyadan o'tmagan bo'lsa, faqat bitta eng yaxshi namunani majburiy himoya qilish); Ikki-nuqta krossoveri (Cavicchio, 1970; Goldberg,

1989c) va yagona qarama-qarshi tomondan (Syswerda, 1989) meros avlodlari alomatlar ota-onalar turli yo'llari bor.

GA amalga oshirilayotgan biologik evolyutsiya modeli, odatda juda, tabiiy asl nisbatan soddalashtirilgan shunday kam bo'lmagan GA muvaffaqiyatli qiyin, shu jumladan, ilovalar keng ko'lamli hal qilish uchun qo'llash mumkin kuchli vositasi ekanligiga qaramay, ba'zan hatto imkonsiz, boshqa yo'llar bilan hal qilish[10,11].

1.3 Avtomatlashgan dasturlash

Avtomatlashtirilgan dasturlash - bu dastur yoki uning qismini rasmiy avtomatning modeli sifatida sharhlagan dasturiy paradigm hisoblanadi. Avtomatlashtirilgan boshqaruv ob'ektlari shaklida kompleks hatti-harakatlarga ega bo'lgan shaxslarning vakolatxonasidan tashkil etilgan va ularning har biri nazorat obyekti va avtomat bo'lib, "avtomatik dasturlash paradigmasi ham mavjud" deb qaraladi. Shu bilan birga, dastur, avtomatik boshqaruvda bo'lgani kabi, avtomatlashtirilgan nazorat ob'ektlari tizimi sifatida ham o'ylangan bo’ladi.

Muayyan vazifaga qarab, avtomatlashtirilgan dasturlash ham murakkab tuzilishga ega bo'lgan ikkita sonli avtomat va avtomatizadan foydalanishi mumkin.

Avtomatik dasturlash uchun quyidagi funktsiyalar juda muhim hisoblanadi:

1. Dasturni bajarish muddati mashinasozlik bosqichlariga bo'linadi, ularning har biri bitta kirish punktiga ega bo'lgan aniq kodni (har bir qadam uchun) bajarish; bunday bo'lim alohida masalani o'z ichiga olishi mumkin va ayrim davlatlar yoki davlatlar toifalariga mos keladigan kichik bo'limlarga bo'linishi mumkin

2. Mashinaning bosqichlari orasidagi ma'lumotlarni uzatish faqat mashina holati deb ataladigan aniq belgilangan o'zgaruvchan majmui orqali amalga oshiriladi; Mashina dasturining (yoki uning qismlari avtonom uslubda ishlab chiqilgan) bosqichlari o'rtasida davlatning yopiq elementlari, masalan, stokdagi mahalliy o'zgaruvchilar qiymatlari, funktsiyalarni qaytarish manzili, mavjud buyruqlar taymerining qiymati va boshqalar bo'lishi mumkin emas; Boshqacha qilib aytganda, dasturning holati avtomatik ravishda har qadamda ikki daqiqaga to'g'ri keladi, faqat avtomat holatini tashkil etuvchi o'zgaruvchan qiymatlardan farq qilishi mumkin (va bunday o'zgaruvchilar aniq belgilanishi kerak)[13].

Avtomat stilidagi kodni to'liq bajarish mashinaning qadamlarining aylanishi (ehtimol yopiq) hisoblanadi.

Avtomatik dasturlash - bu ibora dasturlashning (ijro jarayonining idrok) fikr yo'li deyarli aniq (va hokazo Turing mashinasi, Markov otomat, kabi) rasmiy mashinalari tayyorlashda fikr uslubi qayta ishlab oqlanadi.

Misol uchun, chiziqlardan iborat standart kiritilgan matnni o'qish, C tilida bir dastur yozish kerak va har bir chiziq va chiziq tanaffusi birinchi so'zni chop etdi deylik. Bu holda har bir satrni o'qish davomida birinchi liniyasiga boshida o'tishi, bo'sh joyga o'tish kerak bo'ladi; Keyin so'z hosil qilgan xat o'qilib so'z (ya'ni, yoki chiziq nihoyasiga qilmoqchi emas, yoki bo'shliqni kutib bo'lmaydi) larni chop qilish zarur; birinchi so'z muvaffaqiyatli o'qish va chop qilingan paytda, hech narsa chop qilinmaydigan holat esa xatolik holati bo'ladi. Chiziqli xabarlar belgisi bilan (har qanday bosqichda) duch kelganingizda, qatorni chop etishni boshlashingiz va boshidanoq davom etishingiz kerak. "Faylning oxiri" holati yana bir bor sodir bo'lsa (har qanday bosqichda) ishlamasligi kerak.

Imperativ dastur

Ushbu muammoni an'anaviy uslubda yechadigan dastur (C tili):

```c
#include <stdio.h>

int main() {
  int c;

  do {
    c = getchar();
    while (c == ' ') c = getchar();
    while (c != ' ' && c != '\n' && c != EOF) putchar(c), c = getchar();
    putchar('\n');
    while (c != '\n' && c != EOF) c = getchar();
  } while (c != EOF);

  return 0;
}
```

Dastur avtomatik uslubda

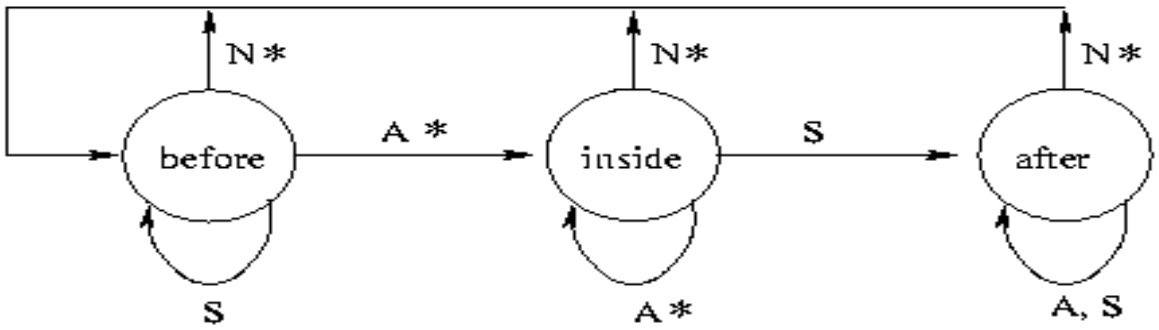
Xuddi shu muammoni sonli avtomatlar nuqtai nazaridan qo'llash orqali hal qilish mumkin. Shuni esda tutingki, chiziqni ajratish uch bosqichga bo'linadi: etakchi bo'shliqlarni atlayarak, so'zni kiritish va qolgan qator belgilarni aytaylik. Dastur endi, masalan, quyidagi kabi ko'rinishi mumkin:

```c
#include <stdio.h>
```

```
int main() {
  enum states { before, inside, after } state;
  int c;
```

```
  state = before;
  while ((c = getchar()) != EOF) {
    switch (state) {
          case before:
        if (c == '\n') putchar('\n');
        else if (c != ' ') putchar(c), state = inside;
        break;
          case inside:
        switch (c) {
          case ' ':
            state = after; break;
          case '\n':
            putchar('\n'), state = before;
            break;
          default: putchar(c);
        }       break;
          case after:
        if (c == '\n') putchar('\n'), state = before;
         }  }
  return 0;
}
```

Kodning ochiq-oshkor bo'lib qolganiga qaramay, u shubhasiz bir afzalliklarga ega: o'qish (ya'ni getchar () funktsiyasini chaqirish) hozir bir joyda amalga oshiriladi. Bundan tashqari, avvalgi versiyada ishlatilgan to'rtta tsikl o'rniga, aylanish jarayoni endi faqat bitta ishlatiladi. Tsiklning tanasi (nomda bajarilgan xatti-harakatlar bundan mustasno) mashinaning qadamidir, tsiklning o'zi mashinaning aylanishini o'rnatadi.



Berilgan vositalar uchun avtomatning ishlashi yoki harakati 0 va boshlang'ich x(0) ning ichki holati ham to'liq aniqlanadi va o'tish va chiqishlar vazifalari bilan aniqlanadi.

O'tish funksiyasi, hozirgi vaqtda ichki holati va ichki holatining holati ustidan avtomatning ichki holatiga bog'liq bo'ladi.

Chiqish funksiyasi mashinaning chiqish holatiga, kirish holatiga va mashina ichki holatiga bog'liqligini aniqlaydi.

Turli avtomatlar uchun ushbu bog'liqliklarning xilma-xilligi sinxron sonli deterministik avtomatlar sinfida avtomatlarning alohida turlarini ajratishga imkon beradi[10,11,12].

Asosiy ikkita model: Mili va Mur.

Mili avtomatik mashinasi quyidagi formulalar bilan tavsiflanadi:

1. Avtomatning ichki holati keyingi davrda avtomatning ichki holatiga va hozirgi vaqtda kirish signaliga bog'liq.

$$x(t+1) = \phi(x(t), \rho(t)) \tag{1.3.1}$$

2. Mashinaning ma'lum bir vaqtdagi chiqish signallari hozirgi paytda kirish signaliga va mashinaning ichki holatiga bog'liq.

$$\lambda(t) = \psi(x(t), \rho(t)) \tag{1.3.2}$$

T davrida avtomat holatining kontseptsiyasi avtomatning ichki holati va shu vaqtning o'zida avtomatning kiritilishi holati bilan belgilanadi.

$$M(t) \equiv (x(t), \rho(t)) \tag{1.3.3}$$

Barcha juftliklarda o'tish vazifalari va chiqish funksiyalari aniqlangan avtomatlarga mutlaqo aniq yoki to'liq avtomat deb ataladi. Shunga mos ravishda, barcha juftliklarda o'tish vazifalari yoki chiqish funksiyalari aniqlanmagan avtomat aniqlanmagan avtomat deb ataladi. Muvofiq juftda belgilangan avtomatning M (t) foydalanilmayotgan nuqtasi deb ataladi. Chiqish funksiyasi avtomatning har qanday holatiga aniqlanmagan bo'lsa, unda u bevosita chiqim holatiga mos keladi.

Mur avtomati

Mur avtomati uchun o'tish va chiqish funksiyalari quyidagicha:

$$x(t+1) = \phi(x(t), \rho(t)) \tag{1.3.4}$$

$$\lambda(t+1) = \psi(x(t+1)) = \psi(\phi(x(t), \rho(t)) = \psi'(x(t), \rho(t)) \quad (1.3.5)$$

Mur avtomatining chiqishi funksiyasi avtomatning ichki holati bilan belgilanadi.

Vaqt mos kelmaydigan mashina uchun.

Bu quyidagi tenglamalar bilan belgilanadi:

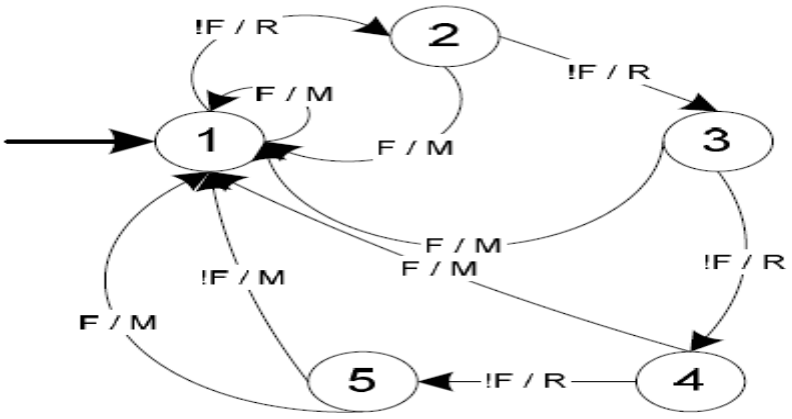
$$x(t+1) = \phi(x(t), \rho(t+1)) \quad (1.3.6)$$

$$\lambda(t+1) = \psi(x(t+1), \rho(t+1)) \quad (1.3.7)$$

Vaqt mos kelmaydigan mashinada, kirish holatini o'zgartirish keyingi ichki holatga o'tishga olib keladi, ya'ni, mashina ichki holati bir vaqtning o'zida kirishning holatiga bog'liq bo'lib, mashinaning chiqishi holati uning kiritilish holatiga bog'liq bo'ladi[9,10].

1.4."Aqlli chumoli haqida"gi masala

"Aqlli chumoli" masalasida, yechim topish usullardan biri sifatida chumolining hatti-harakatini cheklangan avtomat ko’rinishida ta'riflanishi mumkin:



1.4.1-rasm.

1.4.1-rasmda ishlatilganlarni tushuntirib o’tamiz. 1 yozuv. O'tkazmalardagi belgilar holat / harakat formatida. Shartlar quyidagicha ifodalanadi:

• F - chumolon oldida ovqat bor;

•! F - chumoli oldida ovqat yo'q.

Harakatlar quyidagicha ko'rsatiladi:

• M - "oldinga qadam qo'ying";

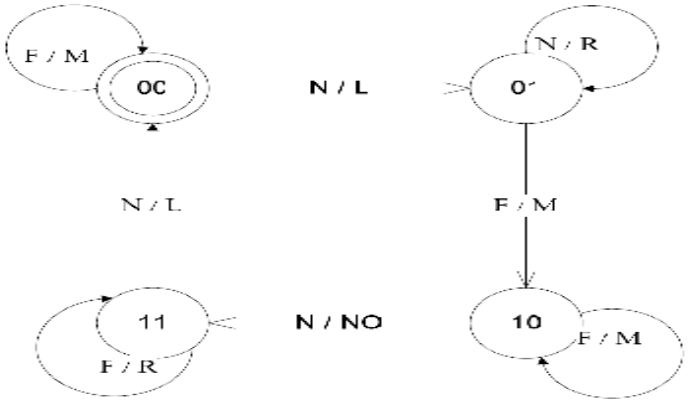
• L - "chapga buriling";

• R - "o'ng tomonga buriling";

• N - "Hech narsa qilmang".

Ushbu ishda genetik algoritmda (bit lentasi) chumolining harakatlarini belgilaydigan avtomat kodlash quyidagi tarzda amalga oshirildi: F ning kirish harakati bir birlik, N esa nolga tenglashdi. To'rtta chumolilarning har biri ikki tomonlama sonlar bilan kodlangan: 00 - NO, 01 - L, 10 - R, 11 - M.

Keling, avtomat kodlash qanday amalga oshirilayotganini ko'raylik. 2-rasm, ma'lum bir chumolining hatti-harakatini tavsiflovchi avtomatning o'tish grafigining namunasini ko'rsatadi[10].



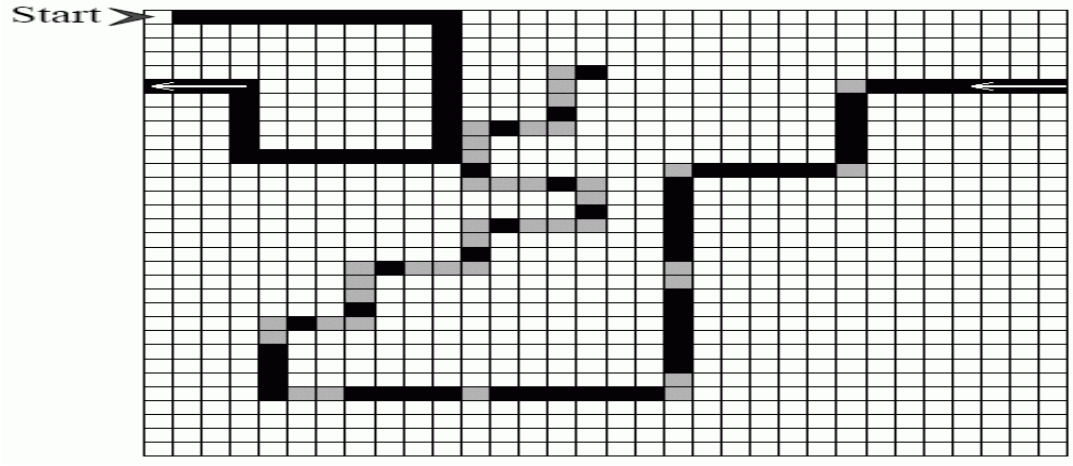
Ushbu o'tish grafikasini kodlash jadvalda keltirilgan.

Vaziyat	Kirish	Yangi holat	Harakat
00	0	01	01
00	1	00	11
01	0	01	10
01	1	10	11
10	0	11	00
10	1	10	11
11	0	00	01
11	1	11	10

Ushbu jadvalni bir bitga aylantiring. Buning uchun avval mashinaning holatini eslab qolishingiz kerak (bu holda to'rtta). Chiziq boshida mashinaning boshlang'ich holatining ikkilik soni berilgan (ushbu misolda 00). Keyin juftliklar (Yangi holat, Amal) saqlanadi. Maydonlarning tarkibi (Status, Input) qayd etilmaydi, chunki ular har bir avtomat uchun bir xil holatlar bilan takrorlanadi. Ko'rib chiqilayotgan misol uchun, qidiruv satri quyidagicha ko'rinadi:

00 0101 0011 0110 1011 1100 1011 0001 1110

32 ga 32 kvadrat o'lchovdagi ikki o'lchovli torus ishlatiladi. Tumanning ayrim hujayralarida olma mavjud - quyidagi rasmda qora hujayralar mavjud. Olma ba'zi singan chiziq bo'ylab joylashgan, ammo uning barcha hujayralarida emas. Olma olmaydigan singan chiziqdagi hujayralar kulrang. Oq rangli hujayralar - buzilgan liniyaga tegishli emas va olma yo'q. Bu erda 89 ta olma mavjud.



"Boshlash" deb nomlangan katakda chumoli turibdu. U maydonning hujayrasini egallaydi va to'rt yo'nalishdan biriga (shimolga, g'arbga, janubga, sharqqa) qaraydi. O'yin boshida chumoli sharqqa qaraydi. U to'g'ridan-to'g'ri oldida olma ekanligini aniqlashni biladi. Biror chumolining davomida to'rtta harakatdan birini bajaradi:

• Olma oldida bo'lsa, bir kvadrat oldinga boradi;

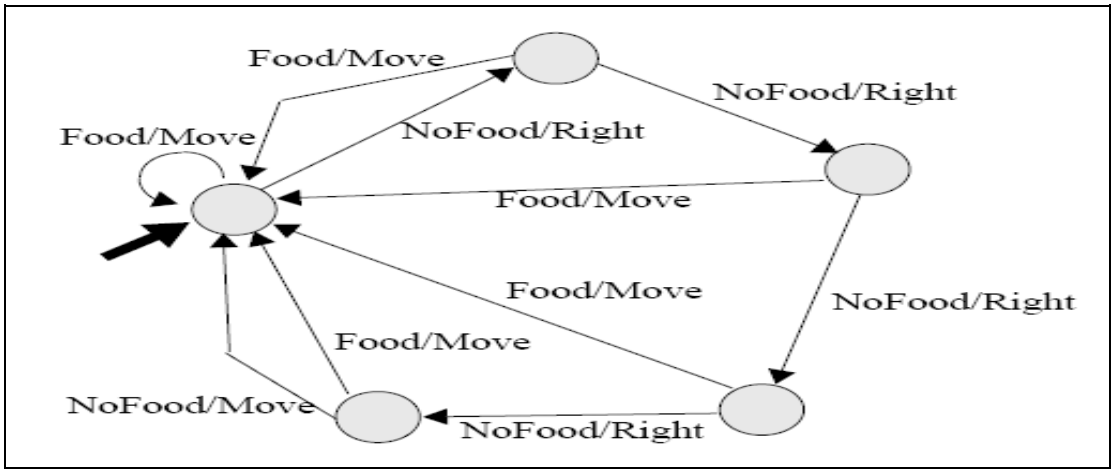
• o'ngga o'giradi;

• chapga buriladi;

• hali ham turibdi.

Bir tomonidan yegan olmasi qaytib to'ldirilmaydi. Chumoli o'yin orqali tirik - oziq-ovqat yeydi bu uning mavjudligi uchun ajralmas manba hisoblanadi. Bu yerda chumolidan tashqari boshqa hech qanday belgi yo'q. Tuzilgan liniya

qat'iyan berilgan. Chumolilar maydonning har bir kvadratida yurishi mumkin. O'yin 200 ta harakatni davom ettiradi, ularning har birida chumoli yuqorida ko'rsatilgan to'rtta harakatdan birini bajaradi. O'yin oxirida chumoli yegan olma soni hisobga olinadi. Ushbu qiymat o'yinning natijasidir. O'yinning maqsadi - 200 ta harakatda iloji boricha ko'p olma iste'mol qiladigan chumolini yaratishdir. Bir xil miqdordagi olmalarni iste'mol qiladigan chumolilar, har bir kishi yeb-ichish jarayoniga sarflanadigan harakatlar sonidan qat'iy nazar o'yinni xuddi shunday natijaga aylantiradi. Shu bilan birga, bu vazifa turli xil o'zgarishlarga ega bo'lishi mumkin, masalan, bir xil miqdorda olmalar yeyilgan bo'lsa, eng yaxshisi kamroq olma yeb qolgan chumoli deb hisoblanadi. Quyida shu ko'rsatilganki, chumolining harakati cheklangan avtomat bilan belgilanishi mumkin. Bunday holatda, muammo barcha olma yeyish uchun bir qator antistatik vaziyatlar yoki bir qator holatlar uchun maksimal olma miqdorini iste'mol qiladigan chumoli mashinasi bilan bir qatorda avtomat qurilishi bilan bog'liq bo'lishi mumkin. Quyidagi rasmda ko'rsatilgandagi sonli holat mashinasi faqat beshta holatni o'z ichiga oladi.



Bu mashina muammolarni hal qilmaydigan chumolining hatti-harakatini tasvirlaydi - 200 ta harakat faqat 81 ta olmoqda, va 314 ta harakat uchun - 89 ta olma. Chumolilar "Men olma ko'rmoqdaman - men oldinga boraman" tamoyili asosida faoliyat yuritadi. Men ko'rmayapman - men o'ng tomonga burilib ketaman. Men aylana yasadim, lekin olma yo’q ekan – men oldinga boraman "[12,13].

1.5 Masala qo’yilishi

Berilgan masala vazifasi, har bir individual yechimni cheklangan avtomat shaklida kodlash, shuningdek olingan yechimni o'rganishda "aqlli chumoli" muammosini hal qilish uchun genetik algoritmni ishlab chiqishdan iborat.

I– bob bo’yicha xulosa

Bu bobda NP sinf masalalari tushunchalari, genetik algoritm, avtomatlashgan dasturlash, “Aqlli chumoli haqida” gi ma’lumotlar berilgan.

Dissertatsiya ishida qo’yilayotgan masalaning berilishi tavsiflangan va o’rganib, yechim uchun misollar keltirilgan.

II-BOB. ALGORITM YECHIMI TAVSIFI

2.1 Genetik algoritmning sxemasi

Genetik dasturlashning rivojlangan algoritmi besh qismdan iborat:

· Dastlabki avlodni yaratish;

· Mutatsiyalar;

· O'tish (kesish);

· Keyingi avlodni shakllantirish uchun shaxslarni tanlash;

Fitnes funktsiyasini hisoblash (fitnes funktsiyasi).

Har bir inson, chumolining xatti-harakatini tavsiflovchi ma'lum bir avtomatdir.

Individual xromosomalar davlatning boshlang'ich holatidan va davlatlarning tavsiflaridan iborat. Shtatning ta'rifi chumoli oldida taom borligi yoki u mavjud bo'lmaganligi sababli ikki o'tishning tavsiflarini o'z ichiga oladi. O'tishning tavsifi u olib keladigan davlat raqamidan va ushbu o'tish davrida tanlangan harakatdan iborat[10,11].

2.2. Avtomat tasavvuri

Xromosomalar genetik algoritmlarda bo'lgani singari, biroz bit shaklida ko'rinmaydi

C # dasturlash tilida ob'ekt. Bu ob'ekt quyidgi tasvirlangan tuzilishga ega:

```
public class Automaton {
public Transition[][] transitions;
public int initialState;
public int stateCount;
}
```

Kleene-ning teoremlari bilan berilgan muntazam fotoalbomlarda - Kleene teoremasi mashinada voqealar bir ketma-ketlikda ifodalanishi uchun zarur va yetarli shart bu aniq, ya'ni, kirish simgelerinin ketliklar maxsus majmuini taqdim etadi. Bunday majmui kirish tartibi ko'rinishi bir mos keladigan muntazam voqea deyiladi. Kleene teoremasining davlat mashina navbatdagi tadbir orqali vakili, va faqat ularni mumkin, deb belgilaydi. Shunday qilib, voqealar taqdim tili davlat mashinasini "qilish" mumkin savolning javobi aniq beradi: davlat mashinasi faqat bir muntazam tadbir bo'lishi mumkin. tez-tez amalda, aniq muntazam bilan shug'ullanish kerak kiritish-ketliklar muhim fotoalbomlarda, bir

qator. Misol uchun, ma'lum muntazam cheklangan uzunligi kiritish-ketliklar, har qanday cheklangan soni iborat o'rnatish; har qanday davriy kirish sekanslari to'plami; Bas, so'nggi bir necha ko'chadan uchun biron cheklangan sekansları o'z ichiga olgan, va cheksiz-ketliklar bir ko'pchilik.

Tyuring mashinasi nima qilishi mumkin nima? savoliga bunday aniq va ravshan javob asosi sifatida algoritm aniqlashni Turing mashinasi konsepsiyasini qo'yish imkonini beradi: algoritm mashinada amalga oshirilishi mumkin bo'lgan har qanday jarayon ekanligini algoritmlar to'liq mashinalari, cheksiz xotira to'ldiriladi

2.3. Boshlang’ich populyatsiya

Dastlabki nasl sobit bo'lgan tasodifiy ishlab chiqarilgan avtomatning soni. Avloddagi barcha avtomatlar bir xil oldindan belgilangan raqamlar soni.

Bugungi kunda katta hajmdagi ma'lumotlar katta ma'lumotlar bazasini shakllantiradigan tizim yaratiladi. Katta ma'lumotlar bazalari bilan ishlaydigan ko'plab algoritmlar mavjud.

Big Data katta ma'lumot to'plamini yashiradi. Va uning hajmi juda katta, chunki katta hajmdagi ma'lumotlarni standart dasturiy ta'minot va apparat bilan ishlash juda qiyin. Boshqacha aytganda, Big Data masalasi - katta hajmdagi ma'lumotlarni saqlash va qayta ishlash masalasidir.

Big Data katta axborot miqdori, uning xilma-xilligi va juda tez ma'lumotlarni qayta ishlash zaruriyati bilan bog'liq. Biz optimal yechim olish mumkin bo'lgan genetik algoritmlar usullarida katta ma'lumotlar bazalari bilan ishlash uchun juda ko'p algoritmlari mavjud.

Ayni paytda, genetik algoritmga murojaatlardan biri katta geografik ma'lumotlar doirasida optimal yechimlarni topish uchun grafikalar bilan ishlash masalasidir. Geodata juda katta ma'lumotlar bazasi bo'lgani uchun, zamonaviy texnologiyalarda ushbu algoritm bunday muammolarni optimallashtirish va hal qilish uchun ishlatiladi[12,13].

Genetik Algoritm (ing genetik algoritm.) - tabiatda tabiiy tanlash o'xshash mexanizmlari yordamida, tasodifiy tanlash, birlashtirish va kerakli parametrlarni almashtirish tomonidan optimallashtirish va modellashtirish hal qilish uchun foydalaniladigan intuitiv qidirish algoritmi hisoblanadi. Bu, masalan, meros, mutatsiya, tanlash va krossover kabi tabiiy evolyutsiya usullari yordamida optimallashtirish muammolarni hal qilish mumkin bo'lgan tadrijiy hisoblashning bir turi hisoblanadi. Genetik algoritmning o'ziga xos xususiyati nomzod

yechimlari operatsiya rekombinatsiyani bajaradi. "Kesishgan" operator foydalanishga urg'u bo'lib, u tabiatdagi o’rinma-o’rin urchitishga o'xshaydi.

Bizning davrimizda genetik algoritmlar mashhurlikni kasb etmoqda. Ular turli masalani yechish uchun ishlatiladi.

Algoritm rivojlanishi tez bo’layotganligi uchun dastur spektri juda keng:

• Grafik vazifalar

• Yaratilgan vazifalar

• Rejalashtirish

• "Sun'iy intellekt" ni yaratish

Operatsion printsipi

Genetik algoritm birinchi navbatda evolyutsiya algoritmidir, ya'ni algoritmning asosiy xususiyati evolyutsiyaga o'zaro bog'lanadi. Algoritmning g'oyasi tabiatdan olingani hech kimga sir emas.

Algoritm uch bosqichga bo'linadi:

• O'tish

• Tanlash (tanlash)

• Yangi avlodni shakllantirish

Agar natija bizni qoniqtirmasa, natijalar bizni qoniqtirmagunga qadar yoki quyidagi shartlardan biri sodir bo'lgunga qadar ushbu bosqichlar takrorlanadi:

• Avlodlar soni (tsikllar) oldindan tanlangan maksimal darajaga yetadi

• Mutatsiyalar uchun sarflash vaqti

Yangi aholi yaratish. Ushbu qadamda dastlabki populyatsiya yaratiladi, bu juda yaxshi. Algoritm bu muammoni bartaraf etishi ehtimoli katta. Eng asosiysi, ular "formatga" mos keladi va "reproduktsiyaga moslashtiriladi".

Ko'paytirish. Xo'sh, bu yerda hamma narsa odamlarga o'xshaydi, ularning avlodi bo'lishi uchun ikkita ota-ona tanlab qilinadi. Eng muhimi, avlod (farzand) ota-onasidan ularning xususiyatlaridan meros olishi mumkin. Mutatsiyalar. mutantlar parvarish o'xshash mutatsiyalar oldindan belgilangan operatsiyalar ko'ra jismoniy shaxslarning muayyan miqdordagi tanlangan, va ularni o'zgartirsa bo'ladi.

Tanlash. Bu yerda, biz bir aholi ulushidan tanlash boshlanadi va "davom" ettiriladi[11,12].

2.4 Genetik operatorlar

Mutatsiyalar. Mutatsiyaga uchraganda, to'rtta teng variantlardan biri tasodifiy tanlanadi:

· Dastlabki holatni o'zgartirish - bu holda yangi boshlang'ich holat tasodifiy va mos ravishda tanlangan;

· O'tkazishdagi ishni o'zgartirish - tasodifiy va muvozanatli o'tishni tanlash va unga ta'sir tasodifiy o'zgaradi. Bunday holda barcha mumkin bo'lgan harakatlar teng darajada mumkin;

· O'tishga olib keladigan holatni o'zgartirish - tasodifiy va mos ravishda tanlangan o'tish. Shundan so'ng, o'tishni boshlaydigan davlat tasodifiy tanlangan davlat bilan almashtiriladi;

· O'tkazish shartini o'zgartirish - tasodifan va teng tanlangan holat. Shundan so'ng, bu ahvoldan "Qalamponni oldin oziq-ovqat yeyishdan oldin" va "Qisqichbaqasimon ovqat yo'q" shartlariga mos keladigan o'zgarishlar, joylarni o'zgartirish.

O'tish. Krossover operator ikki shaxslar usulidan qabul qiladi va shuningdek, ikki kishi beradi. O'tish jarayoni quyidagi tarzda amalga oshiriladi. S1 va S2 - ota-ona qushlar P1 va P2, va bolalarni belgilaymiz.

A A.is. sifatida otomat dastlabki holatini bo'lsin yo S1.is = P1.is va S2.is = P2.is yoki S1.is = P2.is va S2.is = P1.is, har ikkisi ham bir xil bo'ladi: Bas, S1 va S2 avlodlari ikki narsadan biri to'g'ri bo'ladi .

"Tashkil" O'tishlarning S2-S1 va avlodlari Otomata ikki teng ehtimol variantlarining biri tomonidan amalga oshirilishi -can, biz tasvirlab.

Biz P1 kabi "ovqatlanishga chumoli oldin" Kirish o'zgaruvchining qiymatiga mashina P1 soni o'tishni i bildirmoq (i, 1), P1 "chumoli oldin hech oziq-ovqat" qiymati esa (i, 0). Biz shunga o'xshash ma'noga egamiz

P2 (i, 0) va P2 (i, 1). So'ngra, davlatdan bir o'tish uchun i avlodlari, S1 va S2 avtomatah- soni to'rt amal biri bo'ladi:

· yoki S1(i, 0) = P1(i, 0), S1(i, 1) = P2(i, 1) va
S2(i, 0) = P2(i, 0), S2(i, 1) = P1(i, 1);
· yoki S1(i, 0) = P2(i, 0), S1(i, 1) = P1(i, 1) va

S2(i, 0) = P1(i, 0), S2(i, 1) = P2(i, 1);
· yoki S1(i, 0) = P1(i, 0), S1(i, 1) = P1(i, 1) va
S2(i, 0) = P2(i, 0), S2(i, 1) = P2(i, 1);
· yoki S1(i, 0) = P2(i, 0), S1(i, 1) = P2(i, 1) va
S2(i, 0) = P1(i, 0), S2(i, 1) = P1(i, 1).

Hamma to’rtta variant ehtimolligi katta.

Keyingi avlodni shakllantirish. Elitizm keyingi avlodni shakllantirishning asosiy strategiyasidir. Joriy avlodni qayta ishlash jarayonida, barcha fidokorlardan tashqari barcha shaxslar tashlab ketiladi. Tirik qolgan shaxslarning nisbati har bir avlod uchun doimiy bo'lib, algoritm parametrlaridan biridir.

Bu odamlar keyingi avlodga o'tadilar. Shundan so'ng, talab qilinadigan o'lchamga quyidagicha to'ldiriladi: to'ldirilguncha, hozirgi avloddan ikkita shaxs tanlanadi va ular kesib o'tishlari yoki mutatsiyaga uchraydilar. Mutatsiyalar yoki kesishmalar natijasida olingan ikkala shaxs yangi avlodga qo'shiladi. Bundan tashqari, agar etarli darajada ko'p avlodlar uchun fitnes o'sish bo'lmasa, avlodning "kichik" va "katta" mutatsiyalaridan foydalaniladi. "Kichik" avlodlar mutatsiyasiga ega mutatsion operator barcha insonlarga, eng yaxshi mutatsiyalarning 10 foizidan tashqari, qo'llaniladi. "Katta" mutatsiyaga ega bo'lgan har bir kishi mutatsiyaga uchraydi yoki o'zgartiriladi. "Kichkina" va "katta" mutatsiyalar oldidan avlodlar soni algoritmni ishlatish vaqtida sobit bo'lsa-da, turli xil uchish uchun farq bo'lishi mumkin[9,10].

2.5 Fitnes-funksiya

Mashina bilan belgilanadigan 200 chumoli harakat orqali yeb bo'lgan oziq-ovqat miqdori va T - - ishlayotgan soni qaysi chumoli o'tgan oziq-ovqat birligidan yeydi F / 200 - Function Fittest (mashina) F + (T 200) ga teng bo'ladi. U modellashtirish va saqlangan holda hisoblanadi. Shunday qilib, har bir individual fitnes funksiyasi uchun bir marta hisoblanadi.

2.6 Algoritmning to’g’irlanadigan parametrlari

Genetik dasturlash algoritmida quyidagi parametrlar o'zgartirilishi mumkin:

· Avlod miqdori;

· Keyingi avlod uchun hijrat shaxslar ulushi;

Shtatlar soni;

· Mutatsiyalarning mumkinligi;

"Kichik" avlodlarning mutatsiyasiga o'tish vaqti;

· "Katta" mutatsion avlodga o’tish vaqti.

Hisoblash tajribalariga ko’ra, chumoli barcha oziq-ovqatni yeyish imkonini beradigan, qidiruv vaqti mashinasini sezilarli darajada ta'sir qilishi mumkin, bu parametrlarini ayrim o'zgartirishlar paydo qilinadi.

2.7.Natija

Hisoblash tajribalari natijasida 197 ta qadam uchun 83 dona olma yeb yeb bo’lingan deb hisoblanadi va shu bilan bir mashinani qurish muvaffaqiyatli tugatiladi.

II – bob bo’yicha xulosa

Ushbu bobda dissertatsiyada ishida berilgan masala yechimi, genetik algoritmlarni qo’llash, qo’llanilgan algoritmlarga avtomat dasturlash orqali dastur tuzish sxemasi, masala yechimi tadbiqi sifatida populyatsiya misoli olinishi va uni yechish usuli, genetik operatorlar tushunchasi, foydalanilgan algoritmlarning to’g’irlanadigan parametrlari va kelib chiqqan natija tavsiflari keltirib o’tilgan.

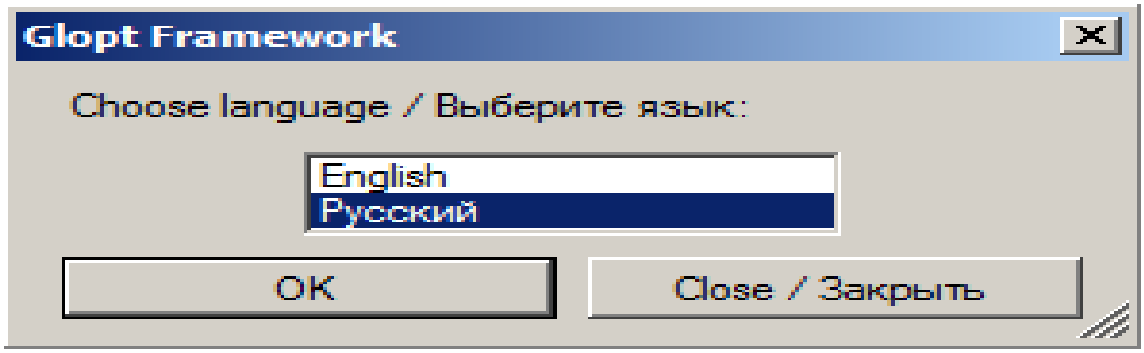
III BOB. YECHIMNI DASTURIY TA’MINOT KO’RINISHIDA TADBIQ ETISH

3.1 Global optimizatsiya freymvorki

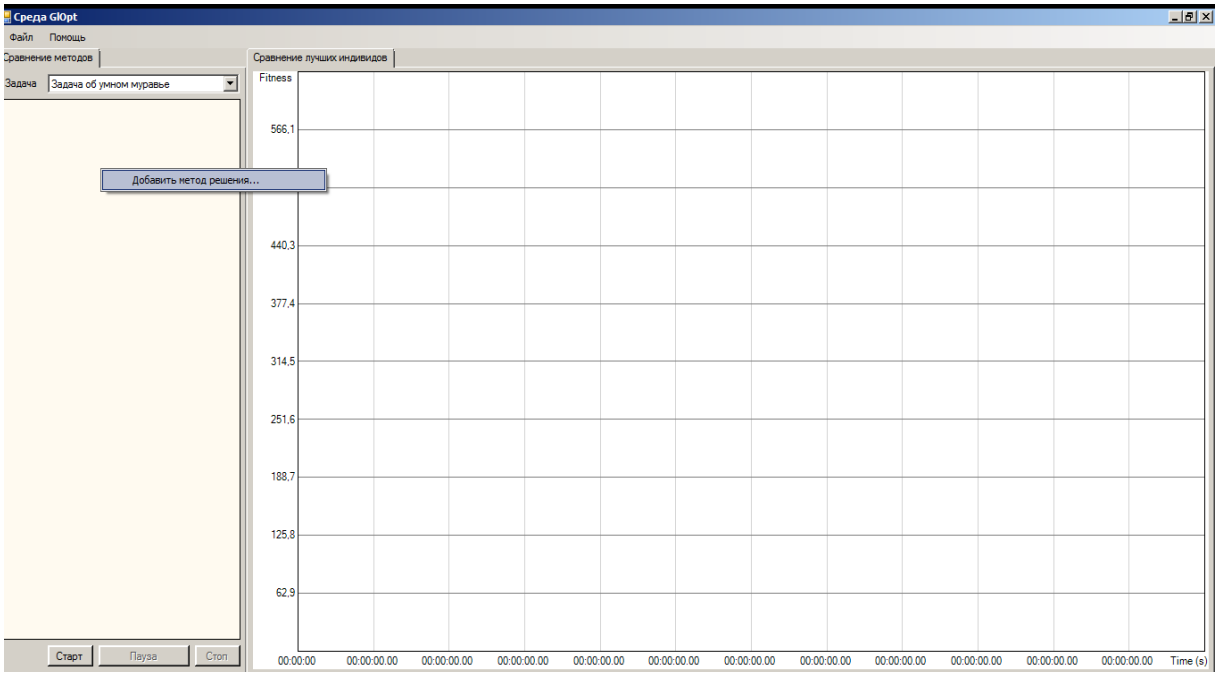
Ushbu muammoni hal qilish uchun Peterburgdagi ITMOda ishlab chiqilgan optimallashtirish global tizimidan texnika fanlari nomzodi Tsarev Fedor Nikolayevich rahbarligida topshiriqning muhim bo'lmagan qismlarini amalga oshirishning murakkabligini kamaytirish uchun foydalanganmiz.

3.2 Foydalanuvchiga qo'llanma

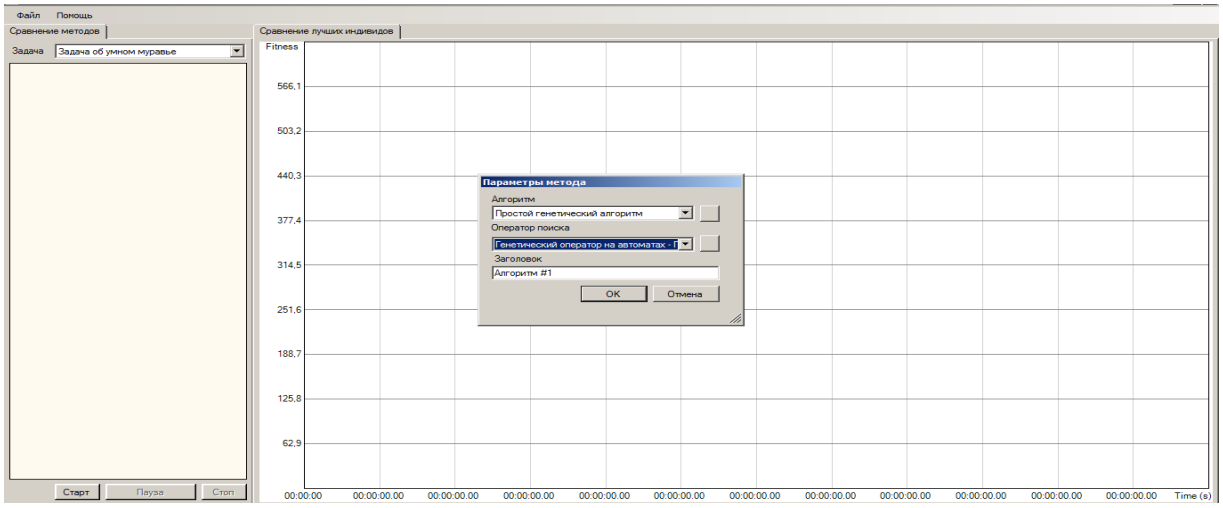
Dasturni boshlashda foydalanuvchi tilni tanlashni kerak:



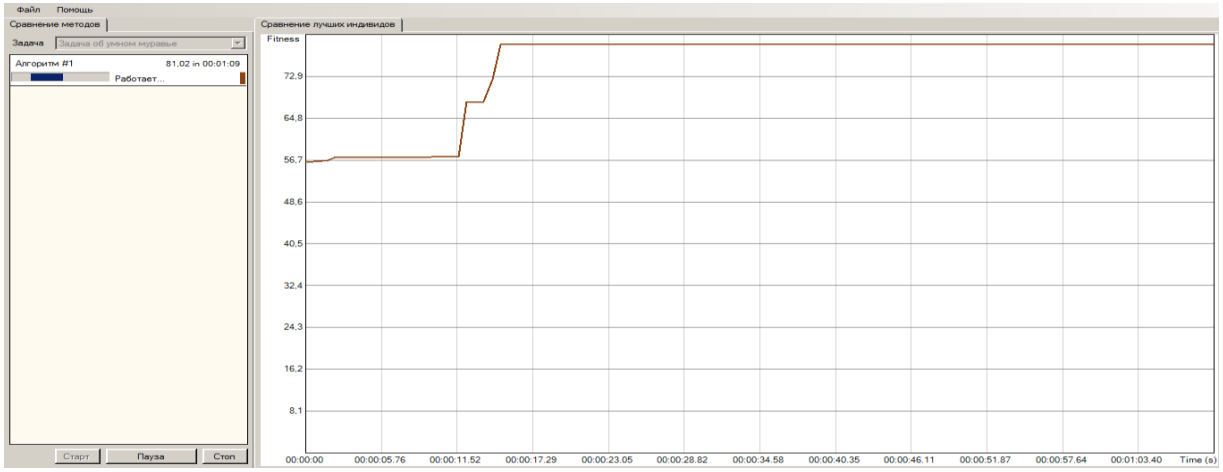
Til tanlagandan so'ng, vazifa va yechim usulini tanlash kerak bo'lgan joy ochiladi:



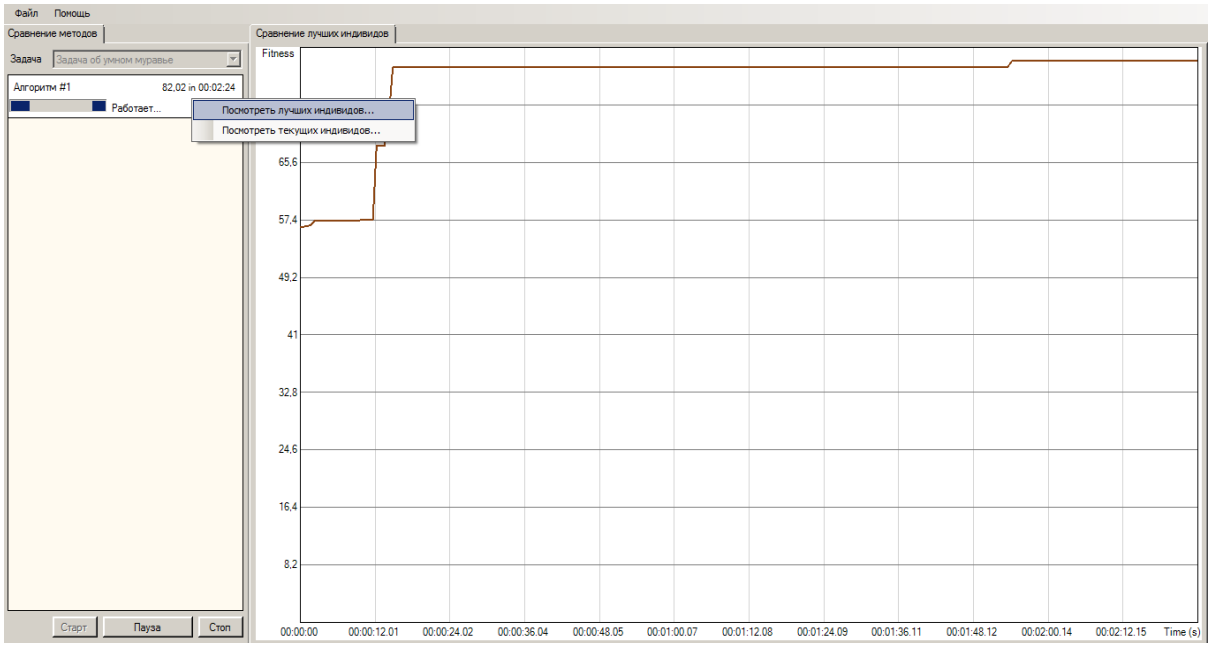
Bu yerda siz hal qilinadigan muammoni hal qilish usulini tanlashingiz va algoritmni ishga tushirishingiz mumkin:

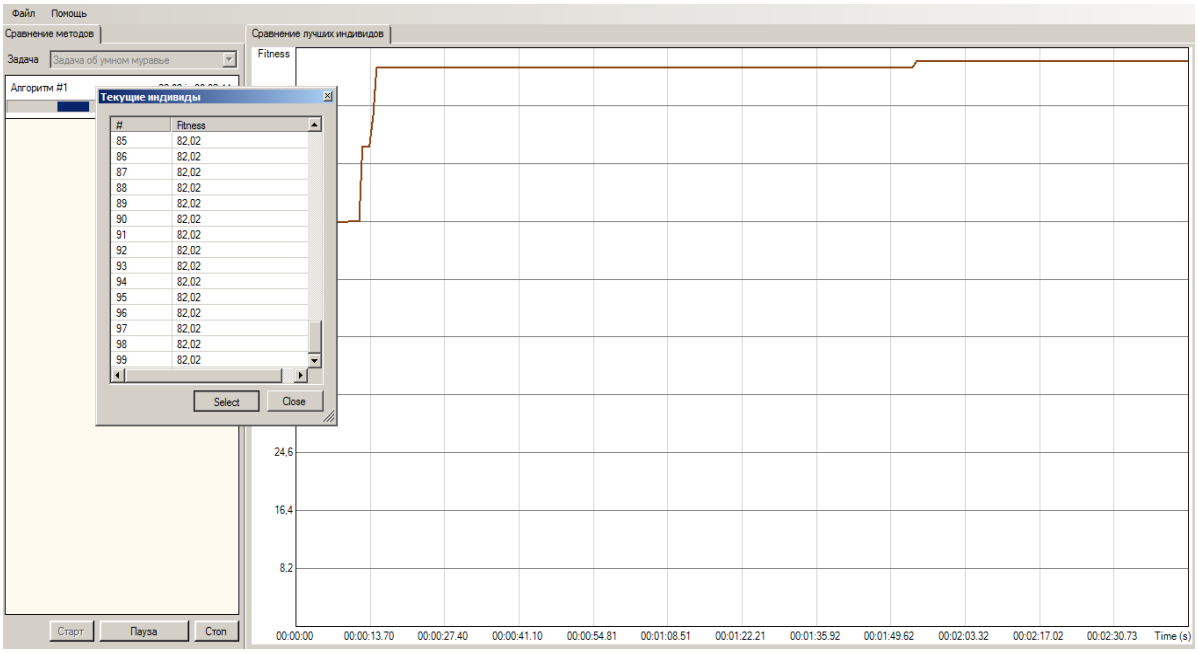


Algoritm ishlayotganda, eng yaxshi yechimlarni ishlab chiqaradigan fitnes funktsiyasining vaqtga bog'liqligining grafigi ko'rsatiladi va shuningdek, eritmaning eng yaxshi individualining fitness funktsiyasining qiymati ko'rsatiladi:

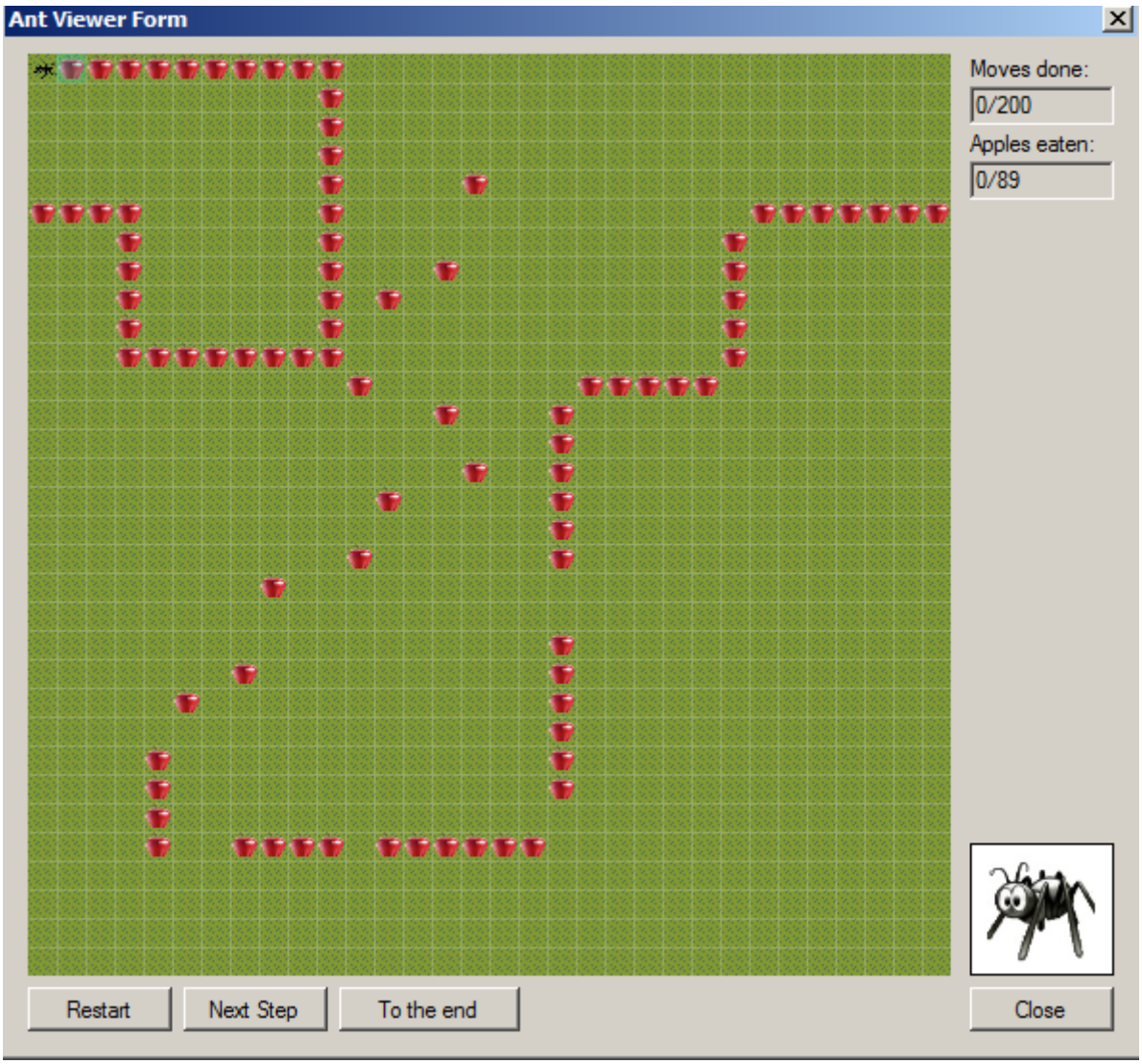


Ish algoritmini o'ng tugmani bosib, siz eng yaxshi kishilarni ko'rishingiz mumkin:

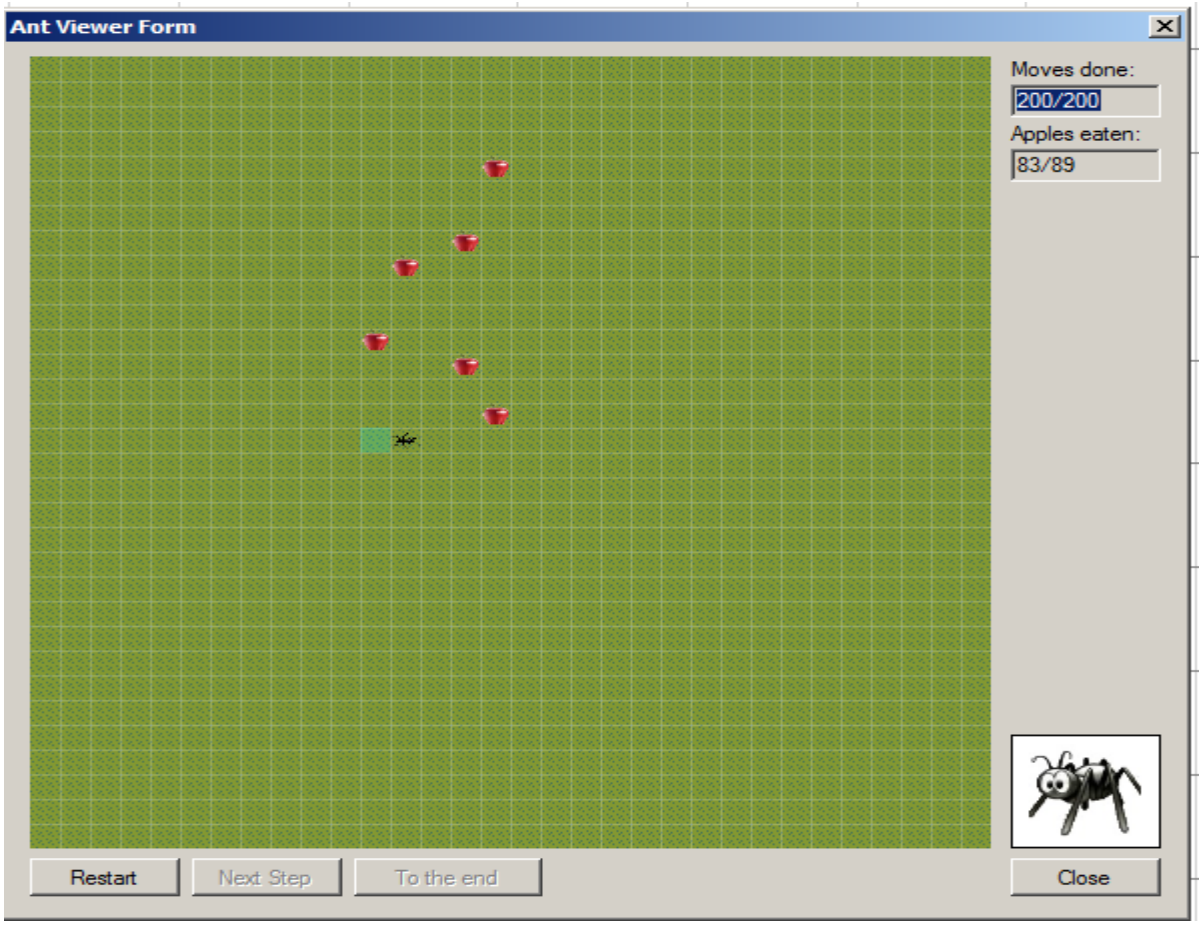




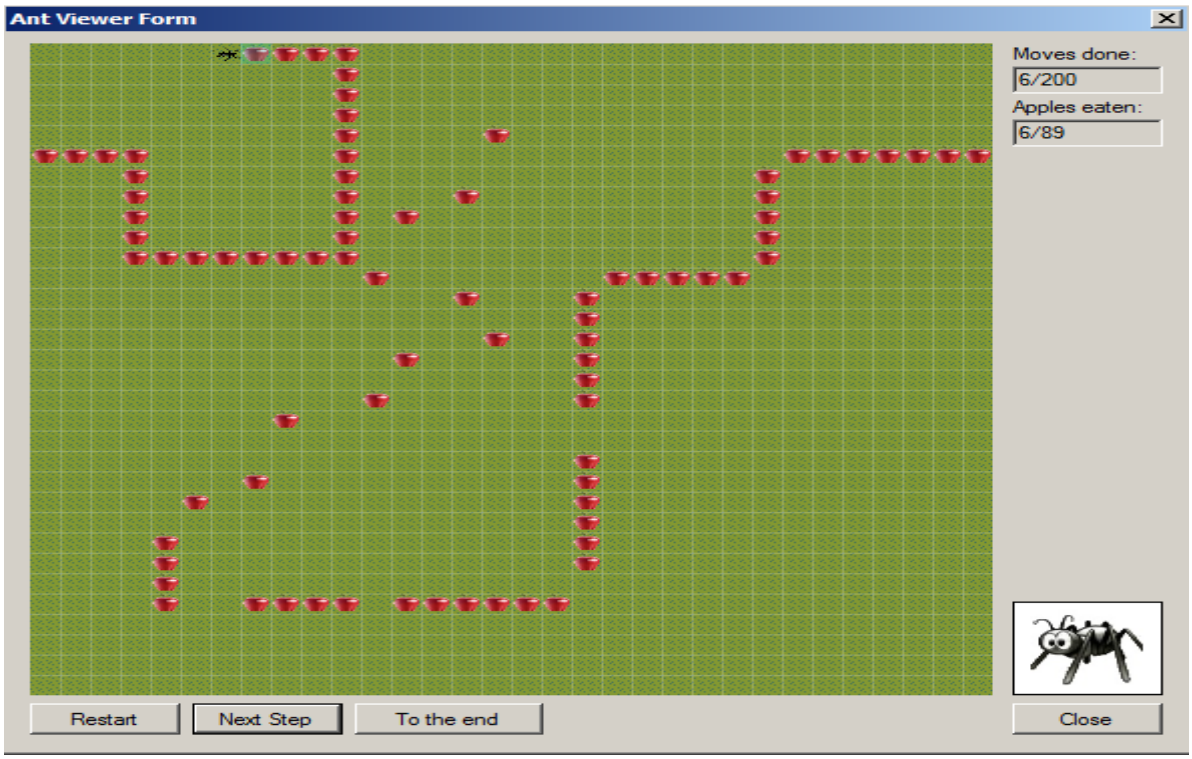
Keyinchalik, "Aqlli chumolilar haqida" muammosining grafik tasviri ochiladi:



Bu yerda "Keyingi qadam" tugmachasini bosib, uning cheklangan avtomatining vakili bo'yicha qadamlar ustida chumolini chizishingiz mumkin:



Shu bilan bir qatorda, "oxirigacha" tugmasini bosish orqali natijani ko'rsatishim mumkin:



Shundan so'ng, siz qarich ishni boshga qaytarishingiz yoki shaklni yopishingiz va algoritmni ishlashni kutishingiz mumkin.

III– bob bo’yicha xulosa

Uchinchi bobda, algoritm ishlashi amaliy tadbiqi keltirib o’tilgan bo’lib, 3.1 bo’limda bu ishni amalga oshirishda foydalanilgan global optimizatsiya freymvorki to’g’risida ma’lumot berib o’tilgan, uni tadbiq etish dastur kodi esa ilovada keltirilgan. 3.2. bo’limda esa dasturdan foydalanish ketma-ketligi, ya’ni foydalanuvchiga qo’llanma berilgan.

XULOSA

1 - bobda NP sinf masalalari tushunchalari, genetik algoritm, avtomatlashgan dasturlash, “Aqlli chumoli haqida” gi ma’lumotlar berilgan.

Dissertatsiya ishida qo’yilayotgan masalaning berilishi tavsiflangan va o’rganib, yechim uchun misollar keltirilgan.

2 - bobda dissertatsiyada ishida berilgan masala yechimi, genetik algoritmlarni qo’llash, qo’llanilgan algoritmlarga avtomat dasturlash orqali dastur tuzish sxemasi, masala yechimi tadbiqi sifatida populyatsiya misoli olinishi va uni yechish usuli, genetik operatorlar tushunchasi, foydalanilgan algoritmlarning to’g’irlanadigan parametrlari va kelib chiqqan natija tavsiflari keltirib o’tilgan.

3 - bobda, algoritm ishlashi amaliy tadbiqi keltirib o’tilgan bo’lib, 3.1 bo’limda bu ishni amalga oshirishda foydalanilgan global optimizatsiya freymvorki to’g’risida ma’lumot berib o’tilgan, uni tadbiq etish dastur kodi esa ilovada keltirilgan. 3.2. bo’limda esa dasturdan foydalanish ketma-ketligi, ya’ni foydalanuvchiga qo’llanma berilgan.

Ushbu dissertatsiya ishida biz genetik algoritm deb ataladigan bir qator murakkab algoritmlarni sinab ko'rdik, dasturiy ta'minotni qurish uchun avtonom yondashuvni ko'rib chiqdik, umumjahon optimallashtirishning odatiy muammolarini o'rganib chiqdik va ushbu muammoga dasturiy ta'minot shaklida yechim topish uchun algoritm ishlab chiqdik.

FOYDALANILGAN ADABIYOTLAR

Normativ-huquqiy hujjatlar va prezident asarlari

1. O’zbekiston Respublikasining “Ta’lim to’g’risida”gi 1997-yil 29-avgustdagi 464-1-sonli Qonuni.

2. O’zbekiston Respublikasining “Kadrlar tayyorlash milliy dasturi”, 1997- yil.

3. O’zbekiston Respublikasining “Axborotlashtirish to’g’risida”gi 2003-yil 11-dekabrdagi Qonuni.

4. O’zbekiston Respublikasi Prezidentining 2017-yil 30 iyundagi PF-5099 son “Respublikada axborot texnologiyalari sohasini rivojlantirish uchun shart-sharoitlarni tubdan yaxshilash chora-tadbirlari to’g’risida”gi Farmoni.

5. O’zbekiston Respublikasi Prezidentining 2012-yil 24 iyuldagi PF-4456-son “Oliy malakali ilmiy va ilmiy-pedagog kadrlar tayyorlash va attestatsiya o’tkazish tizimini yanada takomillashtirish to’g’risida”gi farmoni.

6. Vazirlar Mahkamasining 2007-yil 10 sentyabrdagi 190-son “O’zbekiston Respublikasi Oliy ta’lim tizimida magistratura faoliyatini yanada takomillashtirish, uning samaradorligini oshirish chora-tadbirlari to’g’risida”gi qarori.

7. I.A.Karimov. Yuksak ma‘naviyat – yengilmas kuch. Toshkent: Manaviyat-2008.

8. I.A.Karimov. “O’zbekiston Mustaqillikka erishish ostonasida”. Toshkent: O’zbekiston – 2011.

Asosiy adabiyotlar

9. Д. Хопкрофт, Р. Мотвани, Д. Ульман «Введение в теорию автоматов», 2002

10. Л. Гладков, В. Курейчик, В. Курейчик «Генетические алгоритмы», 2006

Dissertatsiya ishi bo’yicha chop etilgan maqola va tezislar

11. Касымов С.Н., Рахимбаева Р.М. “О кодировании конечных автоматов для генерации с помощью генетических алгоритмов” Ташкент УЗМУ 2018 г. 96-с.

12. Касымов Н.Х., Касымов С.Н., «Универсальные эффективные компактные расширения над разбиениями натурального ряда» Ташкент УзМУ 2018 г. 98-с.

13. Касымов С.Н., Рахимбаева Р.М. “Генетический алгоритм” НавПДИ 2018 г. 6-с.

Internet saytlari

1. www.Lex.uz - O‘zbekiston Respublikasi qonun hujjatlar sayti.
2. www.informika.ru – Rossiya o‘quv yurtlarining ma’lumotlar bazasi sayti.
3. http://ru.wikipedia.org
4. www.5fan.ru
5. www.MachineLearning.ru

ILOVALAR

1.Yechimning dastur kodi

```
    #region Usings
1.  using System;
2.  using System.Collections.Generic;
3.  using System.ComponentModel;
4.  using Genetic.Operators;
5.  using NewBrain;
6.  using NewBrain.Attributes;
7.  using NewBrain.Plugins;
8.
9.  #endregion
10.
11.namespace Genetic
12.{
13.    [SearchOperatorType(typeof (IGeneticSearchOperator))]
14.    public abstract class GeneticOptimizationAlgorithm : Algorithm
15.    {
16.            [Configurable]
17.            [Description("Avlod o'lchami")]
18.        public int GenerationSize { get; set;}
19.
20.        /// <summary>
21.        /// </summary>
22.        protected GeneticOptimizationAlgorithm()
23.        {
24.            GenerationSize = 100;
25.           Generation = null;
26.        }
27.
28.        protected Random Random { get; private set; }
29.
30.        /// <summary>
31.        /// </summary>
32.        protected List<Individual> Generation { get; set; }
33.
34.        protected IGeneticSearchOperator GeneticSearchOperator
35.        {
36.           get { return SearchOperator as IGeneticSearchOperator; }
37.        }
38.
39.        /// <summary>
40.        /// </summary>
```

```
41.         public int GenerationNumber { get; private set; }
42.
43.         /// <summary>
44.         /// </summary>
45.         protected override void Initialize()
46.         {
47.             Random = new Random();
48.             Generation = new List<Individual>(GenerationSize);
49.             for (int i = 0; i < GenerationSize; i++)
50.                 Generation.Add(SearchOperator.Create(Random));
51.         }
52.
53.         /// <summary>
54.         /// </summary>
55.         protected override void NextIteration()
56.         {
57.             GenerationNumber++;
58.             NextGeneration();
59.             if (GenerationNumber >= 500)
60.             {
61.                 GenerationNumber = 0;
62.                 BigMutation();
63.             }
64.         }
65.         /// <summary>
66.         /// </summary>
67.         protected abstract void NextGeneration();
68.
69.         protected abstract void BigMutation();
70.     }
71.}
72.
73.#region Usings
74.
75.using System;
76.using NewBrain;
77.
78.#endregion
79.
80.namespace Genetic.Operators
81.{
82.     public interface IGeneticSearchOperator : ICreateOperator
83.     {
84.         Individual Mutate(Individual oldIndividual, Random random);
```

```
85.     Individual[] Crossover(Individual[] individuals, Random
    random);
86.   }
87.}
88.#region Usings
89.
90.using System;
91.using System.ComponentModel;
92.using ArtificialAnt.IndividualInfo;
93.using ArtificialAnt.ProblemInfo;
94.using Genetic.Operators;
95.using NewBrain;
96.using NewBrain.Attributes;
97.
98.#endregion
99.
100.  namespace Genetic
101.  {
102.     [IndividualType(typeof (Automation))]
103.     public abstract class SearchOperatorOnAutomation :
    SearchOperator
104.     {
105.         public SearchOperatorOnAutomation()
106.         {
107.                 StatesCount = 8;
108.         }
109.
110.                 [Configurable]
111.                 [Description("Holatlar soni: ")]
112.         public int StatesCount { get; set; }
113.
114.         public override Individual Create(Random random)
115.        {
116.           Automation automation = new Automation(0, StatesCount);
117.           for (int i = 0; i < StatesCount; i++)
118.              for (int j = 0; j < 2; j++)
119.                 automation.SetTransition(i, j,
120.                                          new
    Automation.Transition(random.Next(StatesCount),
    GetRandomAction(random)));
121.
122.           return automation;
123.        }
124.
```

```
125.        protected SimpleAntProblem.AntActions
    GetRandomAction(Random random)
126.        {
127.            SimpleAntProblem.AntActions action =
    SimpleAntProblem.AntActions.Left;
128.            switch (random.Next(3))
129.            {
130.                case 0:
131.                    action = SimpleAntProblem.AntActions.Left;
132.                    break;
133.                case 1:
134.                    action = SimpleAntProblem.AntActions.Move;
135.                    break;
136.                case 2:
137.                    action = SimpleAntProblem.AntActions.Right;
138.                    break;
139.            }
140.            return action;
141.        }
142.    }
143.
144.    [IndividualType(typeof (Automation))]
145.    public class GeneticSearchOperatorOnAutomation :
    SearchOperatorOnAutomation, IGeneticSearchOperator
146.    {
147.        public override string Title
148.        {
149.            get { return Properties.Resources.GeneticOperOnAuto; }
150.        }
151.
152.        public override string Description
153.        {
154.            get { return Properties.Resources.GeneticOperOnAuto; }
155.        }
156.
157.        #region IGeneticSearchOperator Members
158.
159.        public Individual Mutate(Individual oldIndividual, Random
    random)
160.        {
161.            Automation aa = oldIndividual as Automation;
162.            if (aa == null)
163.                return oldIndividual;
164.            int state = random.Next(aa.StatesCount);
165.            int c = random.Next(2);
```

```
166.
167.            int iss = aa.InitialState;
168.            if (random.Next(2) == 1)
169.                iss = random.Next(aa.StatesCount);
170.
171.            Automation res = new Automation(iss, aa.StatesCount);
172.
173.            for (int i = 0; i < res.StatesCount; i++)
174.            {
175.                Automation.Transition t = aa.GetTransition(i, 1);
176.                res.SetTransition(i, 1, new
    Automation.Transition(t.EndState, t.Action));
177.                t = aa.GetTransition(i, 0);
178.                res.SetTransition(i, 0, new
    Automation.Transition(t.EndState, t.Action));
179.            }
180.
181.            int es = random.Next(res.StatesCount);
182.            res.SetTransition(state, c, new Automation.Transition(es,
    GetRandomAction(random)));
183.
184.            return res;
185.        }
186.
187.        public Individual[] Crossover(Individual[] individuals,
    Random random)
188.        {
189.            Automation aa1 = individuals[0] as Automation;
190.            Automation aa2 = individuals[1] as Automation;
191.
192.            if (aa1 == null || aa2 == null)
193.                return individuals;
194.
195.            Automation[] s = new Automation[2];
196.            for (int i = 0; i < 2; i++)
197.                s[i] = new Automation(0, aa1.StatesCount);
198.
199.            if (random.Next(2) == 1)
200.            {
201.                s[0].InitialState = aa2.InitialState;
202.                s[1].InitialState = aa1.InitialState;
203.            }
204.            else
205.            {
206.                s[0].InitialState = aa1.InitialState;
```

```
207.                s[1].InitialState = aa2.InitialState;
208.            }
209.
210.            for (int i = 0; i < aa1.StatesCount; i++)
211.            {
212.                int flag = random.Next(4);
213.                switch (flag)
214.                {
215.                    case 0:
216.                        s[0].SetTransition(i, 0, aa2.GetTransition(i, 0));
217.                        s[0].SetTransition(i, 1, aa2.GetTransition(i, 1));
218.                        s[1].SetTransition(i, 0, aa1.GetTransition(i, 0));
219.                        s[1].SetTransition(i, 1, aa1.GetTransition(i, 1));
220.                        break;
221.                    case 1:
222.                        s[0].SetTransition(i, 0, aa1.GetTransition(i, 0));
223.                        s[0].SetTransition(i, 1, aa1.GetTransition(i, 1));
224.                        s[1].SetTransition(i, 0, aa2.GetTransition(i, 0));
225.                        s[1].SetTransition(i, 1, aa2.GetTransition(i, 1));
226.                        break;
227.                    case 2:
228.                        s[0].SetTransition(i, 0, aa2.GetTransition(i, 0));
229.                        s[0].SetTransition(i, 1, aa1.GetTransition(i, 1));
230.                        s[1].SetTransition(i, 0, aa2.GetTransition(i, 0));
231.                        s[1].SetTransition(i, 1, aa1.GetTransition(i, 1));
232.                        break;
233.                    case 3:
234.                        s[0].SetTransition(i, 0, aa1.GetTransition(i, 0));
235.                        s[0].SetTransition(i, 1, aa2.GetTransition(i, 1));
236.                        s[1].SetTransition(i, 0, aa1.GetTransition(i, 0));
237.                        s[1].SetTransition(i, 1, aa2.GetTransition(i, 1));
238.                        break;
239.                }
240.            }
241.
242.            return s;
243.        }
244.
245.        #endregion
246.    }
247. }
248. #region Usings
249.
250. using System.Collections.Generic;
251. using System.ComponentModel;
```

```
252. using Genetic;
253. using NewBrain;
254. using NewBrain.Attributes;
255. using SimpleGeneticOptimizationAlgorithm.Properties;
256.
257. #endregion
258.
259. namespace SimpleGeneticOptimizationAlgorithm
260. {
261.     /// <summary>
262.     /// </summary>
263.     public class SimpleGeneticOptimizationAlgorithm : GeneticOptimizationAlgorithm
264.     {
265.         public SimpleGeneticOptimizationAlgorithm()
266.         {
267.             ElitePart = 0.1;
268.             MutationProbability = 0.1;
269.         }
270.
271.         [Configurable]
272.             [Description("Yuqori turuvchi juftliklar soni")]
273.         public double ElitePart { get; set; }
274.
275.         [Configurable]
276.             [Description("Mutatsiya ehtimolligi")]
277.         public double MutationProbability { get; set; }
278.
279.         public override string Title
280.         {
281.             get { return Properties.Resources.SimpleGenetic; }
282.         }
283.
284.         public override string Description
285.         {
286.             get { return ""; }
287.         }
288.
289.         public override string HtmlDescription
290.         {
291.             get
292.             {
293.                 return Resources.simple;
294.             }
295.         }
```

```
296.
297.        /// <summary>
298.        /// </summary>
299.        protected override OptimizationDirection
    OptimizationDirection
300.        {
301.            get { return OptimizationDirection.Maximize; }
302.        }
303.
304.        /// <summary>
305.        /// </summary>
306.        protected override void NextGeneration()
307.        {
308.            int gSize = Generation.Count;
309.            var newGeneration = new List<Individual>(gSize);
310.           int eliteSize = (int) (ElitePart*GenerationSize);
311.
        newGeneration.AddRange(Generation.GetRange(0, eliteSize));
312.            for (int i = 0; i < (gSize - ElitePart)/2; i++)
313.            {
314.                Individual a1, a2;
315.                a1 = Generation[Random.Next(gSize)];
316.                a2 = Generation[Random.Next(gSize)];
317.                Individual[] s = GeneticSearchOperator.Crossover(new[]
    {a1, a2}, Random);
318.                newGeneration.AddRange(s);
319.            }
320.            if (newGeneration.Count < gSize)
321.
    newGeneration.Add(GeneticSearchOperator.Mutate(Generation[Rand
    om.Next(gSize)], Random));
322.
323.            for (int i = 0; i < newGeneration.Count; i++)
324.            {
325.                if (Random.NextDouble() < MutationProbability)
326.                    newGeneration[i]                                    =
    GeneticSearchOperator.Mutate(newGeneration[i], Random);
327.            }
328.
329.            newGeneration.Sort();
330.            newGeneration.Reverse();
331.            Generation = newGeneration;
332.            CurrentIndividuals = Generation;
333.            if          (BestIndividual          ==          null          ||
    BestIndividual.CompareTo(Generation[0]) < 0)
```

```
334.                 BestIndividual = Generation[0];
335.             }
336.
337.             protected override void BigMutation()
338.             {
339.                 lock (Generation)
340.                 {
341.                     for (int i = 0; i < Generation.Count; i++)
342.                         Generation[i]                                    =
    GeneticSearchOperator.Create(Random);
343.                     Generation.Sort();
344.                     Generation.Reverse();
345.                 }
346.             }
347.         }
348.     }
349.
350.     #region Usings
351.
352.     using ArtificialAnt.ProblemInfo;
353.     using NewBrain;
354.
355.     #endregion
356.
357.     namespace ArtificialAnt.IndividualInfo
358.     {
359.         /// <summary>
360.         /// </summary>
361.         public class Automation : Individual
362.         {
363.             /// <summary>
364.             /// </summary>
365.             private readonly Transition[,] _transitions;
366.
367.             public Automation(int initialState, int statesCount)
368.             {
369.                 InitialState = initialState;
370.                 StatesCount = statesCount;
371.                 _transitions = new Transition[statesCount,2];
372.             }
373.
374.             /// <summary>
375.
376.         /// </summary>
377.             public int InitialState { get; set; }
```

```
378.
379.        /// <summary>
380.        /// </summary>
381.        public int StatesCount { get; protected set; }
382.
383.        public Transition GetTransition(int index, int condition)
384.        {
385.            return _transitions[index, condition];
386.        }
387.
388.        public void SetTransition(int index, int condition, Transition
    transition)
389.        {
390.            _transitions[index, condition] = transition;
391.        }
392.
393.        #region Nested type: Transitions
394.
395.        /// <summary>
396.        /// </summary>
397.        public class Transition
398.        {
399.            public Transition(int endState,
    SimpleAntProblem.AntActions action)
400.            {
401.                EndState = endState;
402.                Action = action;
403.            }
404.
405.            /// <summary>
406.            /// </summary>
407.            public int EndState { get; private set; }
408.
409.            /// <summary>
410.            /// </summary>
411.            public SimpleAntProblem.AntActions Action { get;
    protected set; }
412.
413.            public override string ToString()
414.            {
415.                return string.Format("-{0}->{1}", Action, EndState);
416.            }
417.        }
418.
419.        #endregion
```

```
420.     }
421. }
422.
423. #region Usings
424.
425. using System;
426. using System.Windows.Forms;
427. using ArtificialAnt._Internal.Mover;
428. using ArtificialAnt._Internal.Phenotype;
429. using ArtificialAnt.IndividualInfo;
430. using ArtificialAnt.Properties;
431. using NewBrain;
432. using NewBrain.ComponentModel;
433.
434. #endregion
435.
436. namespace ArtificialAnt.ProblemInfo
437. {
438.    [IndividualType(typeof (Automation))]
439.         [IndividualViewer(typeof(AntViewer))]
440.    public class SimpleAntProblem : Problem
441.    {
442.       #region AntActions enum
443.       /// <summary>
444.       /// </summary>
445.       public enum AntActions : byte
446.       {
447.          Left,
448.          Right,
449.          Move
450.       }
451.
452.       #endregion
453.
454.       /// <summary>
455.       /// </summary>
456.       public override OptimizationDirection OptimizationDirection
457.       {
458.          get { return OptimizationDirection.Maximize; }
459.       }
460.
461.       public override string HtmlDescription
462.       {
463.          get
464.          {
```

```
465.                 return Resources.artAnt;
466.             }
467.         }
468.
469.         public override string Title
470.         {
471.             get { return Properties.Resources.ArtAntProblem; }
472.         }
473.
474.         public override string Description
475.         {
476.             get { return ""; }
477.         }
478.
479.         /// <summary>
480.         public override double EvaluateIndividual(Individual
    individual)
481.         {
482.             var a = individual as Automation;
483.             if (a == null)
484.                 return double.NaN;
485.
486.             var mover = new SimpleMover(a);
487.             mover.Reset();
488.             int apples = 0;
489.             int lastStep = 0;
490.             for (int i = 0; i < Ant.MaxStepsCount; ++i)
491.             {
492.                 if (mover.Move())
493.                 {
494.                     apples++;
495.                     lastStep = i;
496.                 }
497.                 if (apples == Ant.FoodCount)
498.                     break;
499.             }
500.             return apples - 1.0*lastStep/Ant.MaxStepsCount;
501.         }
502.     }
503.
504.         internal class AntViewer : IIndividualViewer
505.         {
506.         /// <summary>
507.         /// </summary>
508.                 public void ViewIndividual(Individual individual)
```

```
509.                {
510.                    Automation a = individual as Automation;
511.                    if (a == null)
512.                        throw new ArgumentException();
513.
514.                    using (var form = new AntViewerForm())
515.                    {
516.                        form.fieldControl.Mover = new
    SimpleMover(a);
517.                        form.ShowDialog();
518.                    }
519.                }
520.            }
521.    }
522.    using System;
523.    using System.Windows.Forms;
524.
525.    namespace ArtificialAnt
526.    {
527.        public partial class AntViewerForm : Form
528.        {
529.            public AntViewerForm()
530.            {
531.                InitializeComponent();
532.            }
533.
534.            private int apples = 0;
535.            private int moves = 0;
536.
537.            private void restartButton_Click(object sender,
    EventArgs e)
538.            {
539.                fieldControl.Mover.Reset();
540.                fieldControl.Invalidate();
541.                apples = moves = 0;
542.                UpdateState();
543.            }
544.
545.            private void UpdateState()
546.            {
547.                UpdateTextBoxes();
548.                UpdateButtons();
549.            }
550.
551.            private void UpdateButtons()
```

```
552.            {
553.                this.restartButton.Enabled = (moves != 0);
554.                this.nextButton.Enabled = (moves < 200);
555.                this.toEndButton.Enabled = (moves < 200);
556.            }
557.
558.            private void UpdateTextBoxes()
559.            {
560.                txtApples.Text = string.Format("{0}/{1}",
    apples, 89);
561.                txtMoves.Text = string.Format("{0}/{1}",
    moves, 200);
562.            }
563.
564.
565.            private void nextButton_Click(object sender,
    EventArgs e)
566.            {
567.                if (fieldControl.Mover.Move())
568.                    apples++;
569.                moves++;
570.                fieldControl.Invalidate();
571.                UpdateState();
572.            }
573.
574.            private void closeButton_Click(object sender,
    EventArgs e)
575.            {
576.                this.Close();
577.            }
578.
579.            private void toEndButton_Click(object sender,
    EventArgs e)
580.            {
581.                while (++moves != 200)
582.                    apples += fieldControl.Mover.Move() ? 1 :
    0;
583.
584.                fieldControl.Invalidate();
585.                UpdateState();
586.            }
587.
588.        private void fieldControl_Load(object sender, EventArgs e)
589.        {
590.
```

```
591.         }
592.           }
593.   }
594.
595.   using System;
596.   using System.Collections.Generic;
597.   using System.ComponentModel;
598.   using System.Drawing;
599.   using System.Data;
600.   using System.Drawing.Drawing2D;
601.   using System.Text;
602.   using System.Windows.Forms;
603.   using ArtificialAnt._Internal.Mover;
604.   using ArtificialAnt._Internal.Phenotype;
605.   using ArtificialAnt.IndividualInfo;
606.   using ArtificialAnt.Properties;
607.
608.   namespace ArtificialAnt.AntViewer
609.   {
610.           public partial class FieldControl : UserControl
611.           {
612.                   public FieldControl()
613.                   {
614.                           InitializeComponent();
615.                   }
616.
617.                   internal SimpleMover Mover { get; set; }
618.                   internal SimpleAnt Ant { get { return (SimpleAnt)
      Mover.Ant; } }
619.                   private float GridSize { get { return Width / 32f; } }
620.
621.                   protected override void OnPaint(PaintEventArgs e)
622.                   {
623.                           base.OnPaint(e);
624.
625.                           using (var p = new Pen(Color.FromArgb(50,
      255, 255, 255), 1))
626.                           {
627.                                   for (int i = 0; i <= 32; ++i)
628.                                   {
629.                                           e.Graphics.DrawLine(p, 0, GridSize
      * (i + 1), Width, GridSize * (i + 1));
630.                                           e.Graphics.DrawLine(p, GridSize *
      (i + 1), 0, GridSize * (i + 1), Height);
631.                                   }
```

```
632.                     }
633.
634.                     if (Mover == null)
635.                         return;
636.
637.                     DrawApples(e.Graphics);
638.
639.                     DrawAnt(e.Graphics,  Mover.Ant.CurrentCell.X,
    Mover.Ant.CurrentCell.Y, Mover.Ant.CurrentDirection);
640.                 }
641.
642.                 private void DrawApples(Graphics graphics)
643.                 {
644.                     for (int i = 0; i < 32; ++i)
645.                         for (int j =0; j < 32; ++j)
646.                         {
647.                             if (Ant.CurrentField[i, j])
648.                                 DrawApple(graphics, i, j);
649.                         }
650.                 }
651.
652.                 private void DrawAnt(Graphics graphics, int col, int
    row, Ant.Direction direction)
653.                 {
654.                     int left = 55*(int) direction;
655.                     graphics.DrawImage(Resources.ant,           new
    RectangleF(GridSize * (col), GridSize * (row), GridSize, GridSize),
    new RectangleF(left, 0, 55, 55), GraphicsUnit.Pixel);
656.
657.                     int dx = 0, dy = 0;
658.                     switch (direction)
659.                     {
660.                         case
    _Internal.Phenotype.Ant.Direction.Up:
661.                             dx = 0;
662.                             dy = -1;
663.                             break;
664.                         case
    _Internal.Phenotype.Ant.Direction.Right:
665.                             dx = 1;
666.                             dy = 0;
667.                             break;
668.                         case
    _Internal.Phenotype.Ant.Direction.Down:
669.                             dx = 0;
```

```
                                    dy = 1;
                                    break;
                                case _Internal.Phenotype.Ant.Direction.Left:
                                    dx = -1;
                                    dy = 0;
                                    break;
                            }

                            int newx = (col + 32 + dx)%32;
                            int newy = (row + 32 + dy)%32;

                            Brush b = new Pen(Color.FromArgb(50, 0, 255, 255)).Brush;
                            graphics.FillRectangle(b, new RectangleF(GridSize * (newx), GridSize * (newy), GridSize, GridSize));
                        }

                        private void DrawApple(Graphics graphics, int col, int row)
                        {
                            graphics.DrawImage(Resources.apple, GridSize * (col) + 1, GridSize * (row), GridSize - 2, GridSize - 2);
                        }

        private void FieldControl_Load(object sender, EventArgs e)
        {

        }
          }
    }
    #region Usings

    using ArtificialAnt._Internal.Phenotype;

    #endregion

    namespace ArtificialAnt._Internal.Mover
    {
        internal interface IMover
        {
            bool Move();
            void Restart(Ant ant);
        }
```

```
709.  }
710.  #region Usings
711.
712.  using System;
713.  using ArtificialAnt._Internal.Phenotype;
714.  using ArtificialAnt.IndividualInfo;
715.  using ArtificialAnt.ProblemInfo;
716.
717.  #endregion
718.
719.  namespace ArtificialAnt._Internal.Mover
720.  {
721.     internal class SimpleMover
722.     {
723.        private readonly Automation _automation;
724.        private SimpleAnt _ant;
725.                 internal Ant Ant { get { return _ant; } }
726.        private int _current;
727.
728.        public SimpleMover(Automation automation)
729.           : this(automation, null)
730.        {
731.        }
732.
733.        public SimpleMover(Automation automation, SimpleAnt ant)
734.        {
735.           _automation = automation;
736.           Restart(ant);
737.        }
738.
739.        public void Restart(SimpleAnt ant)
740.        {
741.           _ant = ant ?? new SimpleAnt();
742.           _current = _automation.InitialState;
743.        }
744.
745.        public void Reset()
746.        {
747.           Restart(null);
748.        }
749.
750.        public bool Move()
751.        {
752.           int food = _ant.F()[0] ? 1 : 0;
```

```
753.            Automation.Transition t =
    _automation.GetTransition(_current, food);
754.            int ends = t.EndState;
755.            if (ends >= 0 && ends < _automation.StatesCount)
756.            {
757.                SimpleAntProblem.AntActions action = t.Action;
758.                _current = ends;
759.                switch (action)
760.                {
761.                    case SimpleAntProblem.AntActions.Left:
762.                        _ant.L();
763.                        break;
764.                    case SimpleAntProblem.AntActions.Right:
765.                        _ant.R();
766.                        break;
767.                    case SimpleAntProblem.AntActions.Move:
768.                        _ant.M();
769.                        break;
770.                }
771.                return ((food == 1) && (action ==
    SimpleAntProblem.AntActions.Move));
772.            }
773.            throw new Exception(string.Format("ends fail: {0}", ends));
774.        }
775.    }
776. }
777. #region Usings
778.
779. using System;
780. using System.Collections.Specialized;
781. using ArtificialAnt.Properties;
782.
783. #endregion
784.
785. namespace ArtificialAnt._Internal.Phenotype
786. {
787.    internal abstract class Ant
788.    {
789.        #region Direction enum
790.
791.        public enum Direction : byte
792.        {
793.            Up = 0,
794.            Right,
795.            Down,
```

```
796.                     Left
797.         }
798.
799.         #endregion
800.
801.         public static int MaxStepsCount
802.         {
803.             get { return Settings.Default.MaxStepsCount; }
804.         }
805.
806.         public static int FoodCount
807.         {
808.             get { return Settings.Default.FoodCount; }
809.         }
810.
811.         public static StringCollection Field
812.         {
813.             get { return Settings.Default.Field; }
814.         }
815.
816.         public abstract Direction CurrentDirection { get; }
817.         public abstract Cell CurrentCell { get; protected set; }
818.
819.         protected static Direction TurnRight(Direction direction)
820.         {
821.             return (Direction) (((byte) direction + 1)%4);
822.         }
823.
824.         protected static Direction TurnLeft(Direction direction)
825.         {
826.             return (Direction) (((byte) direction + 3)%4);
827.         }
828.
829.         public abstract bool[] F();
830.
831.         public abstract void L();
832.         public abstract void R();
833.         public abstract void M();
834.
835.         #region Nested type: Cell
836.
837.         public sealed class Cell
838.         {
839.             public Cell(int x, int y)
840.             {
```

```
841.             X = x;
842.             Y = y;
843.         }
844.
845.         public int X { get; private set; }
846.         public int Y { get; private set; }
847.
848.         public Cell Next(Direction direction)
849.         {
850.             switch (direction)
851.             {
852.                 case Direction.Left:
853.                     return new Cell((X + 31)%32, Y);
854.                 case Direction.Up:
855.                     return new Cell(X, (Y + 31)%32);
856.                 case Direction.Right:
857.                     return new Cell((X + 1)%32, Y);
858.                 case Direction.Down:
859.                     return new Cell(X, (Y + 1)%32);
860.                 default:
861.                     throw                                    new
    ArgumentOutOfRangeException("direction");
862.             }
863.         }
864.     }
865.
866.     #endregion
867. }
868. }
869. namespace ArtificialAnt._Internal.Phenotype
870. {
871.     internal abstract class AbstractAnt : Ant
872.     {
873.         private Cell cell;
874.         private Direction direction;
875.
876.         protected AbstractAnt()
877.         {
878.             direction = Direction.Right;
879.             cell = new Cell(0, 0);
880.         }
881.
882.         public override Direction CurrentDirection
883.         {
884.             get { return direction; }
```

```
885.         }
886.
887.         public override Cell CurrentCell
888.         {
889.             get { return cell; }
890.             protected set { cell = value; }
891.         }
892.
893.         public override void L()
894.         {
895.             direction = TurnLeft(direction);
896.         }
897.
898.         public override void R()
899.         {
900.             direction = TurnRight(direction);
901.         }
902.     }
903. }
904. #region Usings
905.
906. using System.Diagnostics;
907.
908. #endregion
909.
910. namespace ArtificialAnt._Internal.Phenotype
911. {
912.     internal class SimpleAnt : AbstractAnt
913.     {
914.         public static readonly bool[,] __Field = ReadField();
915.
916.         private readonly bool[,] _field;
917.
918.         private readonly bool[] _ret;
919.
920.         public SimpleAnt()
921.         {
922.             _field = (bool[,]) __Field.Clone();
923.             _ret = new bool[1];
924.         }
925.
926.         public bool[,] CurrentField
927.         {
928.                         get { return _field; }
929.         }
```

```
930.
931.        private static bool[,] ReadField()
932.        {
933.            Debug.Assert(Field.Count == 32);
934.            Debug.Assert(Field[0].Length == 32);
935.
936.            bool[,] field = new bool[32,32];
937.            for (int i = 0; i < 32; ++i)
938.            {
939.                string s = Field[i];
940.                for (int j = 0; j < 32; ++j)
941.                    field[j, i] = s[j] == '*';
942.            }
943.            return field;
944.        }
945.
946.        public override bool[] F()
947.        {
948.            Cell nextCell = CurrentCell.Next(CurrentDirection);
949.            _ret[0] = _field[nextCell.X, nextCell.Y];
950.            return _ret;
951.        }
952.
953.        public override void M()
954.        {
955.            Cell cur = CurrentCell.Next(CurrentDirection);
956.            CurrentCell = cur;
957.            _field[cur.X, cur.Y] = false;
958.        }
959.    }
960. }
```