VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

# Euristiniai algoritmai NP-pilniems uždaviniams spręsti ir jų realizacija GRID‘ui

## Heuristic Algorithms for NP-complete Problems and their realization for GRID

Magistro baigiamasis darbas

| | | |
|---|---|---|
| Atliko: | Irmantas Venckus | (parašas) |
| Darbo vadovas: | Prof. hab.dr. Antanas Žilinskas | (parašas) |
| Recenzentas: | Audrius Varoneckas | |

Vilnius – 2009

## Santrauka

Darbe nagrinėjami euristiniai algoritmai NPC aibės uždaviniams spręsti ir jų taikymas lygiagrečiųjų ir paskirstytųjų skaičiavimų (angl. GRID) tinkle. NPC aibės uždaviniai, taikant įprastus algoritmus, nėra išsprendžiami per polinominį laiką, todėl jiems taikomi euristiniai algoritmai, kurie pasižymi gebėjimu, per priimtiną laiko tarpą, rasti geros kokybės sprendinius, bet didėjant uždavinių apimtims, euristinių algoritmų vykdymo laikas taip pat sparčiai ilgėja. Norint gauti geresnės kokybės sprendinius, reikia daugiau kompiuterinių išteklių. Darbe detaliau nagrinėjami trys populiarūs euristiniai algoritmai: genetinis, modeliuoto atkaitinimo ir skruzdžių kolonijų. Visi šie algoritmai buvo pritaikyti keliaujančio prekeivio uždaviniui (angl. Traveling salesman problem) spręsti GRID skaičiavimo tinkle. Atlikti bandymai su 20 didelės apimties TSPLIB bibliotekos testinių pavyzdžių leidžia teigti, kad kompiuterinių išteklių problemą, sėkmingai galima išspręsti euristinius algoritmus vykdant GRID skaičiavimo tinkle. Gauti rezultatai rodo, kad euristinių algoritmų efektyvumas, juos vykdant GRID skaičiavimo tinkle yra labai aukštas. Daugelyje bandymų pavyko rasti optimalius sprendinius, o kitais atvejais rasti sprendiniai nedaug skyrėsi nuo optimalių. Darbo autorius euristinių algoritmų bandymams siūlo naudoti „DAG“ tipo GRID užduotis. Tokio tipo užduotys leidžia ta patį bandymą atlikti skirtinguose skaičiavimo klasteriuose tuo pačiu metu, tokiu būdu galima įvykdyti daug pakartotinų bandymų per trumpą laiko tarpą.

**Raktiniai žodžiai:** Euristiniai, algoritmai, euristiniai algoritmai, keliaujančio prekeivio uždavinys, genetinis algoritmas, modeliuotas atkaitinimas, skruzdžių kolonijų algoritmas, lygiagrečiųjų ir paskirstytųjų skaičiavimų tinkas

## Summary

The main goal of this work is to implement and test heuristic algorithms for GRID computing network to solve NP-complete problems. The problems of NP-complete set are not solved in polynomial time. To solve such problems, researchers have to use heuristic algorithms. Heuristic algorithms always give result in polynomial time, but it doesn't mean that result is optimal, also computing time grows together with problem scope, and in this case bigger computing recourses are needed. Three popular heuristic algorithms are included in this works: genetic, simulated annealing and ant colony. All of them were implemented to solve traveling salesman problem in GRID computing network. With mentioned heuristic algorithms 20 TSP instances of TSPLIB library were solved. Experiential results shows that efficiently of heuristic algorithms are high and with 12 tested instances optimal solution was found. Author of this work recommends to use "DAG" type GRID tasks. Such type tasks allows to execute algorithms in different clusters at same time, so in same time researcher can execute a lot of tests and final test will give best results.

**Keywords:** heuristic, algorithm, heuristic algorithms, traveling salesman problem, genetic algorithm, simulate annealing, ant colonies, GRID

# Turinys

# Įvadas

Euristiniai algoritmai yra aktuali pastarųjų metų informatikos mokslų sritis. Vienas pagrindinių tokių algoritmų tikslų – spręsti sudėtingus NPC aibės uždavinius, kurie reikalauja daug kompiuterinių išteklių arba yra neišsprendžiami taikant kitus algoritmus. Euristiniai algoritmai negarantuoja gautų sprendinių optimalumo, tuo jie skiriasi nuo tiksliųjų algoritmų, kurie visada duoda optimalų sprendinį, ir nuo apytikslių algoritmų, kurie garantuoja, kad gautas sprendinys neviršys iš karto žinomos paklaidos. Euristinių algoritmų sukurta daug ir įvairių: atkaitinimo modeliavimo (angl. Simulated anealing), genetiniai algoritmai (angl. Genetic algorithms), išbarstytoji paieška (angl. Scatter search), paieška su draudimais (angl. Tabu search), kintamos aplinkos paieškos (angl. Varialbe neigborhood search), skruzdžių kolonijų elgsenos imitavimo (angl. Ant colony optimization) ir kt.

Lygiagrečiųjų ir paskirstytųjų skaičiavimų infrastruktūros kūrimas ( angl. GRID ) – tai viena naujausių sparčiai besivystančių technologijų. Jos esmė – turimų kompiuterinių išteklių sujungimas į vieną bendrą sistemą, taip sukuriamas galingas superkompiuteris, kurio dalys geografiškai yra toli viena nuo kitos, o kompiuteriai bendrauja internetu. Skaičiavimo tikle yra naudojami tik tie kompiuteriniai ištekliai, kurių realus jų savininkas nenaudoja. Tokiu būdu turima kompiuterinė technika ne tik yra maksimaliai panaudojama, bet ir praplečiamos jos galimybės – padidėja kompiuterių atminties ištekliai.

Darbo tikslas – pritaikyti euristinius algoritmus darbui GRID skaičiavimo tinkle ir atlikti ekspermentinius bandymus, kurie padėtų nustatyti ar efektyvu ir tikslinga euristnius algoritmus spręsti jame. Sėkmingai pritaikius euristinius algoritmus GRID skaičiavimo tinklui, būtų išspręsta nemaža kylančių problemų dalis:

- Euristiniai algoritmai labai greitai sprendžia nedidelės apimties NPC aibės uždavinius, bet didėjant uždavinių apimčiai sprendimo laikas taip pat auga, todėl reikia daugiau kompiuterinių išteklių. Pritaikius euristinius algoritmus darbui GRID skaičiavimo tinkle, ši problema būtų išspręsta;
- Kadangi euristiniai algoritmai ne visada randa optimalius sprendimus, tenka atlikti daug pakartotinų bandymų, kad gauti rezultatai būtų pakankamai tikslūs. Atliekant bandymus, sugaištamas tyrinėtojo laikas, kuris auga didėjant bandymų skaičiui. Pritaikius euristinius algoritmus darbui GRID skaičiavimo tinkle, norimą bandymų skaičių būtų galima įvykdyti tuo pačiu metu skirtinguose skaičiavimo mazguose;

Darbe GRID skaičiavimo tinklui bus pritaikomi keli pasirinkti euristiniai algoritmai: genetinis, modeliuoto atkaitinimo ir skruzdžių kolonijų. Juos pritaikius, bus vykdomi ekspermentiniai bandymai, kurių rezultatai leis nustatyti ar GRID užduotis tikslinga ir efektyvu vykdyti GRID skaičiavimo tinkle.

# 1 Euristiniai algoritmai

Euristiniai arba intelektualieji [Mis03a] algoritmai skirti spręsti sudėtingus NP-pilnos (NP-complete) aibės uždavinius. Jie remiasi racionaliaisias žmogaus samprotavimais [MBB06]. Pagrindinis jų tikslas – rasti priimtinos kokybės, bet nebūtinai optimalų sprendinį, per priimtiną skaičiavimo laiką. Euristiniai algoritmai paprastai randa tik lokaliai optimalius sprendinius, todėl jie dar vadinami lokalios paieškos algoritmais. Visi euristiniai algoritmai pasižymi panašiu veikimo principu: sprendimo paieška prasideda nuo pasirinkto pradinio sprendimo, kitas žingsnis – taikant aplinkos funkciją (aplinkos funkcija pagal pasirinktą metodiką transformuoja duotą sprendinį į kitą) gauti kitą sprendinį, kuris pasižymi skirtingomis sąvybėmis negu prieš tai buvęs sprendinys. Jei gautas naujas sprendinys pasirodo esąs geresnis, pagal pasirinktus kriterijus, negu ęsantis geriausias sprendinys, jis tampa geriausiu sprendiniu. Algoritmo rezultatas – geriausias sprendinys, rastas iteracinio algoritmo vykdymo metu [MBB+04].

1) Sukonstruojamas pradinis algoritmo sprendinys;
2) Taikant aplinkos funkciją, gaunami nauji sprendiniai, kurie pasižymi kitokiomis savybėmis, negu prieš tai buvę sprendiniai;
3) Jei naujasis sprendinys pasižymi geresnėmis sąvybėmis negu buvę geriausi sprendiniai, tada naujasis sprendinys tampa geriausiu sprendiniu ir jis naudojamas tolesnėse algoritmo veikimo iteracijose. Jei naujasis sprendinys nepasižymi geresnėmis savybėmis, jokie pakeitimai neatliekami ir tolesnėse iteracijose naudojamas senasis sprendinys;
4) Jei tenkinama baigimo sąlyga, algoritmas baigia darbą, priešingu atveju – grįžtama į antrą algoritmo žingsnį;

1 pav. Bendra lokaliosios paieškos metaparadigma.

Populiariausi euristiniai (intelektualieji) algoritmai yra šie: atkaitinimo modeliavimo (angl. Simulated annealing), genetiniai algoritmai (angl. Genetic algorithms), išbarstytoji paieška (angl. Scatter search), paieška su draudimais (angl. Tabu search), kintamos aplinkos paieškos (angl. Variable neighborhood search), skruzdžių kolonijų elgsenos imitavimo (angl. Ant colony optimization) ir kt.

## 1.1 Genetiniai algoritmai

Genetiniai algoritmai paremti natūralios atrankos principu. Pagrindinės naudojamos sąvokos yra perimtos iš biologijos: individas, populiacija, individo reikšmė, chromosoma, genas. Individas genetiniuose algoritmuose atitinka sprendinį, populiacija – sprendinių aibę, individo reikšmė – tikslo

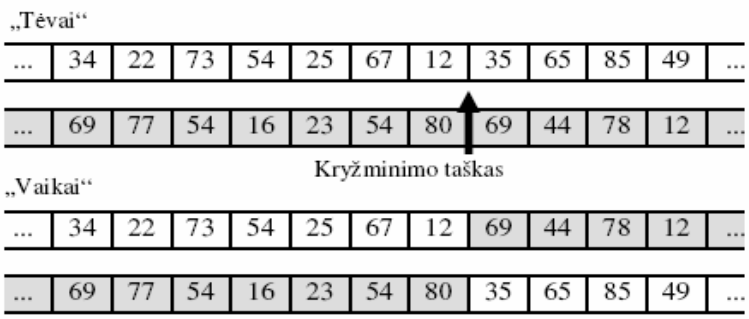
funkcijos reikšmę. Algoritmo esmė – atrinkti tik tuos individus, kurie turi didžiausią tikslo funkcijos reikšmę (stipriausius individus), kitaip tariant algoritmas iteracijų metu imituoja natūralią atranką, kuri paplitusi gamtoje, tokiu būdu iš aibės galimų sprendinių atrenkamas tik geriausias. Sprendiniai genetiniuose algoritmuose yra koduojami simbolių eilutėmis, kas biologijoje atitinka chromosomas, o vienas simbolis iš chromosomos eilutės atitinka geną. Algoritmo vykdymo metu operuojama chromosomų aibėmis ir jų genais, jiems taikomi kryžminimo ir mutacijos operatoriai. Pats algoritmas yra iteracinis, kurį sudaro tokie žingsniai [Ree02]:

1) Atsitiktinai arba naudojant pasirinktą algoritmą sugeneruojama pradinė populiacija iš N individų;
2) Apskaičiuojama visų populiacijos individų tikslo funkcijos reikšmė;
3) Taikant kryžminimo ir mutacijos operatorius iš esamų populiacijos individų gaunami nauji individai;
4) Apskaičiuojamos naujų individų tikslo funkcijos reikšmės, jei gautos reikšmės yra geresnės pagal pasirinktus kriterijus, individai tampa populiacijos nariais, blogiausi populiacijos nariai pašalinami (natūrali atranka).
5) Tikrinama ar pasiekti tikslai, kurių siekia algoritmas, jei taip – algoritmas baigia darbą, jei ne – grįžtama į antrą algoritmo žingsnį

2 pav. Genetinio algoritmo veikimas

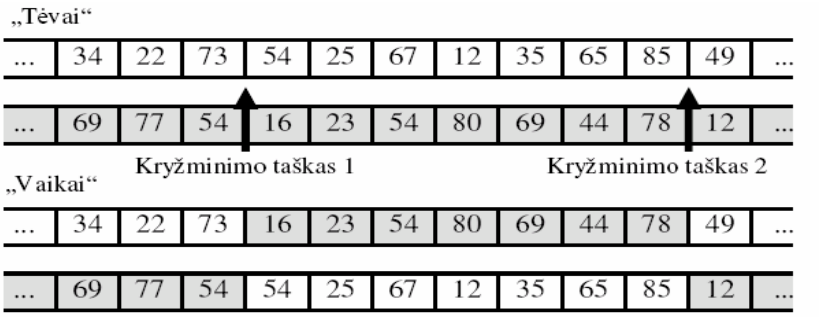
Galimi du genetinio algoritmo operatoriai: mutacija ir kryžminimas. Paprasčiausios mutacijos metu atsitiktinis chromosomos genas sukeičiamas su kitu atsitiktiniu genu, taip gaunamas naujas individas, kuris pasižymi kitokiomis savybėmis. Mutacijos operatoriaus veikimas priklauso nuo sprendžiamo uždavinio ir jo chromosomų kodavimo būdo.

Kryžminimo operatoriaus veikimas taip pat priklauso nuo sprendžiamo uždavinio ir chromosomų kodavimo būdo. Paprasčiausias kryžminimo operatorius gali būti vieno taško arba dviejų taškų. Vieno taško kryžminimo metu parenkamas taškas, per kurį dviejų chromosomų genų intervalai yra sukeičiami vietomis.




3 pav. Individų kryžminimas parenkant vieną kryžminimo tašką

Panašiai kryžminimas vyksta ir dviejų taškų atveju, tik tada parenkami tu kryžminimo taškai, taip nustatomi genų intervalai, kurie bus sukeičiami vietomis.



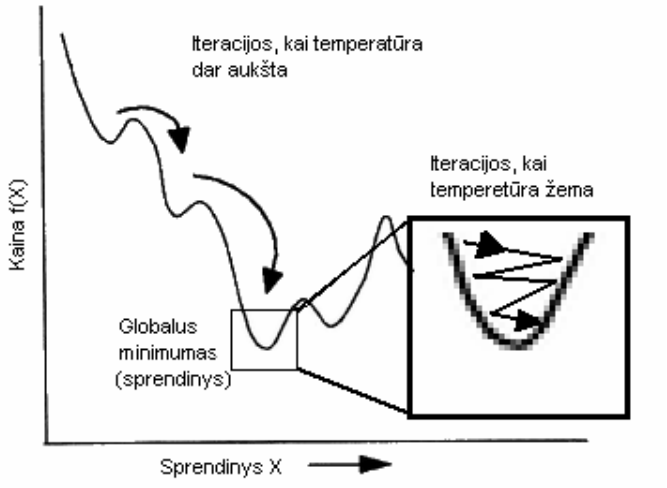

4 pav. Individų kryžminimas parenkant du kryžminimo taškus

## 1.3 Atkaitinimo modeliavimas

Atkaitinimo modeliavimo algoritmas remiasi energetinių procesų, vykstančių sistemose, sudarytose iš didelio skaičiaus dalelių, imitavimu. Pagrindinė algoritmo idėja – imituoti kūno atkaitinimą, tai nurodo ir algoritmo pavadinimas. Algoritmo pradžioje kūnas yra įkaitinamas iki aukštos temperatūros ir kiekviename iteracijos žingsnyje jis yra palaipsniui vėsinamas. Vienas pirmųjų algoritmo veikimo principus aprašė S. Kirkpatrick (1983) [KGV83]. Algoritmo veikimo paradigma:

1) Sugeneruojamas pradinis sprendinys s;
1) Pasirenkami pradiniai parametrai (temperatūra ir uždavinio stabdymo sąlyga);
2) Atsitiktinai pakeičiamas pradinis sprendinys s ir gaunamas sprendinys s';
3) Jei s' tenkina tikimybinį patenkinimo kriterijų, kuris priklauso nuo temperatūros, tada s = s';
4) Pagal pasirinktą aušinimo formulę, atnaujinama temperatūra T.

5 pav. Atkaitinimo modeliavimo algoritmo veikimas




6 pav. Atkaitinimo modeliavimo algoritmo veikimas

Atkaitinimo modeliavimo algoritme svarbu teisingai parinkti pradinę ir galutinę temperatūras. Jei bus parinkta didelė pradinė temperatūra, teks atlikti daug iteracijų ir algoritmo veikimas bus gana ilgas, nes bus nagrinėjama daug nereikalingų sprendinių. Kita vertus, parinkus per mažą pradinę temperatūrą, algoritmas gali papulti į lokalų minimumą, todėl jis neras geriausio galimo sprendinio (globalaus minimumo). Taip pat svarbu parinkti tinkamą temperatūros keitimo algoritmą. Praktikoje naudojami keli algoritmai [BL99] [HJJ02]:

- geometrinė formulė: ($t_k = \alpha t_{k-1}$ ; $t_k$ – esama temperatūros reikšmė; $k$=1, 2, ...; $t_0$=const; $0{,}8 \leq \alpha \leq 0{,}99$)
- Lundy-Mees formulė ( $t_k = t_{k-1}(1+ \beta t_{k-1})$; $k$=1, 2, ...; $t_0$=const; $\beta << t_0$) .
- Naujausiose atkaitinimo modeliavimo algoritmų versijose temperatūra greičiau keičiama periodiškai nei monotoniškai mažinama. Tai vadinama reatkaitinimu (angl. Reannealing): vietoj tiesioginio atkaitinimo taikoma tam tikra „atkaitinimų“ ir „kaitinimų“ seka.

## 1.3 Skruzdžių kolonijos

Daug metų buvo stebima skruzdžių kolonijų elgsena ir jų gebėjimas rasti trumpiausią maršrutą iki maisto šaltinio. Buvo nustatyta, kad skruzdės, eidamos link maisto šaltinio, palieka chemines medžiagas – feromonus. Jei maršrutas ilgas, jį aplanko mažiau skruzdžių, todėl feromonai jame išgaruoja greičiau nei trumpame maršrute. Taip ilgesnis maršrutas skruzdėms tampa mažiau patrauklus ir pasirenkamas trumpesnis maršrutas, turintis daugiau feromonų. Toks skruzdžių elgsenos imitavimas buvo panaudotas optimizavimo uždaviniams spręsti.

Skruzdžių kolonijų optimizavimo (ACO) principai pirmą kartą buvo aprašyti Marco Dorigo darbe 1992 metais [Mon09]. Paprastai skruzdžių kolonijų optimizavimo algoritmas taikomas kombinatoriniams optimizavimo uždaviniams, tokiems kaip TSP uždavinys, tvarkaraščių sudarymo ir kiti, spręsti. Bendra algoritmo veikimo paradigma yra tokia:

1) Nustatomi pradiniai algoritmo parametrai;
2) Sukuriamas pasirinktas skaičius skruzdžių;
3) Kiekviena skruzdė konstruoja sprendinį – konstruojant sprendinį, sprendinio elementai pasirenkami tikimybiškai. Tikimybė, kad bus pasirinktas tam tikras elementas, priklauso nuo jam priskirto feromonų kiekio;
4) Priklausomai nuo to, kokios kokybės sprendinys buvo gautas, kiekvienam sprendinio elementui atnaujinamas feromonų kiekis;
5) Visiems galimiems elementams sumažinamas tam tikras feromonų kiekis (garavimas). Taip išvengiama lokalių optimumų;
6) Tikrinama, ar pasiektas tikslas, jei ne, grįžtama į antrą žingsnį.

7 pav. Skruzdžių kolonijų algoritmo veikimas

# 2 Uždaviniai sprendžiami euristiniais metodais

## 2.1 Keliaujančio prekeivio uždavinys

Keliaujančio prekeivio uždavinys (*TSP-Traveling Salesperson Problem*) – viena iš plačiausiai nagrinėjamų užduočių, skirtų spręsti optimizavimo uždavinius. Keliaujantis prekeivis turi aplankyti N skirtingų miestų taip, kad viso maršruto svoris būtų kaip galima mažesnis. Visi maršrutų svoriai yra iš anksto žinomi. Maršruto svoriu gali būti įvardintas jo ilgis, kaina, trukmė. Kiekvienas miestas turi būti aplankytas tik vieną kartą ir prekeivis turi baigti savo kelionę tame pačiame mieste, iš kurio jis pradėjo keliauti. Keliaujančio prekeivio uždavinio tikslas yra minimizuoti viso maršruto svorinę funkciją $\omega$ (maršruto kainą, ilgį ar laiką):

$$\sum_{i=1}^{N} \omega_{i,i-1} \rightarrow \min$$

### 2.1.1 Įrodymas, kad TSP uždavinys priklauso NPC uždavinių sudėtingumo klasei

Kadangi euristiniai algoritmai taikytini tik uždaviniams iš NPC uždavinių aibės, įrodysime, kad TSP priklauso tokiai uždavinių sudėtingumo aibei ir kitais algoritmais per priimtiną skaičiavimo laiką jo išspręsti negalima.

Pirma įrodysime, kad TSP priklauso NP sudėtingumo klasei. Tarkime turime maršrutą K, kuris bus sertifikatas. Patikrinimo algoritmas nustato ar naujo maršruto miestai pasikartoja tik po vieną kartą, apskaičiuoja tikslo funkcijos reikšmę ir gautą reikšmę palygina su reikšme k. Visa tai patikrinimo algoritmas padaro per polinominį laiką, todėl TSP priklauso NP uždavinių sudėtingumo klasei.

Dabar įrodysime, kad TSP priklauso NPC uždavinių klasei. Kad tai padarytume, reikia įrodyti, kad HAM-CYCLE $\leq$ pTSP. Sakykime, kad G = (V, E) yra Hamiltono ciklas. TSP maršrutą suformuosime taikydami tokį algoritmą: sudarysime pilną grafą G' = (V, E'), kur E' = {(i, j):i, j iš V ir j$\neq$i}, o maršruto svorį apskaičiuosime šitaip: $\omega$ (i, j) = 0, jei (i, j) priklauso E, $\omega$ (i, j) = 1, jei (i , j) nepriklauso E. Taikydami tokį algoritmą, naują TSP maršrutą sudarysime per polinominį laiką, nes grafo G' suformavimo ir $\omega$ (i, j) apskaičiavimo sudėtingumas yra $O(V^2)$.

Dabar parodysime, kad grafas G turi Hamiltono ciklą tada ir tik tada, kai G' turi maršrutą, kurio kaina nedidesnė negu 0. Sakykime, kad G turi Hamiltono ciklą h. Visos h briaunos priklauso grafui G, bet taikant ω (i, j) funkciją, visų h briaunų kaina grafe G' bus lygi 0, taigi G' turi maršrutą su kaina 0. Tarkime, kad G' turi maršrutą h', kurio kaina yra ≤ 0. Visų G' briaunų galima kaina yra 0 arba 1, bet, kad tenkintume pradinę sąlygą, h' briaunų kaina gali būti tik 0, taigi reiškia, kad visos h' briaunos priklauso ir grafui G, o tai reiškia, kad h' yra Hamiltono ciklas grafe G. Taigi HAM-CYCLE yra redukuojamas į pTSP per polinominį laiką. Kadangi HAM-CYCLE priklauso NP-sudėtingų uždavinių klasei, tai ir TSP priklauso tokiai klasei, o kadangi TSP priklauso NP uždavinių klasei, tai reiškias TSP yra iš NPC uždavinių klasės. Įrodyta.

### 2.1.2 Keliaujančio prekeivio uždavinio sprendimas naudojant genetinius algoritmus

Naudojant genetinius algoritmus keliaujančio prekeivio uždaviniui spręsti, maršrutas turi būti koduojamas. Galimi keli maršruto (chromosomos) kodavimo būdai [BB00] [LKM+99]:

- Kelio kodavimas;
- Dvejetainis kodavimas;
- Matricos kodavimas;
- Tvarkos kodavimas.

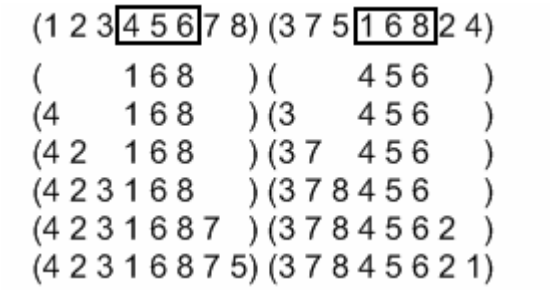
Dėl paprastumo ir aiškumo tyrime bus naudojamas kelio kodavimas (path representation). Chromosoma 1-2-5-4-3-1 reiškia, kad visi miestai bus aplankyti chromosomos genų išsidėstymo tvarka.

Svarbiausias genetinio algoritmo operatorius – kryžminimo. Pritaikius kryžminimo operatorių, nauji individai turi išlaikyti pradinio uždavinio reikalavimus – miestai maršrute turi pasikartoti tik vieną kartą. Keliaujančio prekeivio uždaviniui su kelio kodavimu taikomi keli skirtingi kryžminimo operatoriai:

- Dalinio atvaizdavimo;
- Tvarkos;
- Ciklo;
- Godaus lankų sukeitimo [SY98].

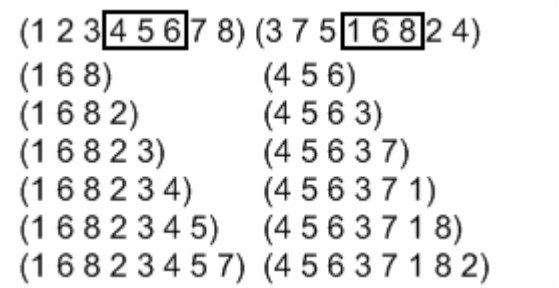
Dalinio atvaizdavimo kryžminimo metu parenkami du kryžminimo taškai. Viduriniai tėvų chromosomų genų intervalai vaikų chromosomose išlieka nepakeisti ir tose pačiose pozicijose, o prie tų intervalų nuo chromosomos pradžios pridedami kiti genai iš tėvų chromosomų. Kad nebūtų

pažeisti pradiniai uždavinio reikalavimai, yra sudaroma pakeitimų lentelė, kuri nurodo, kaip genai turi būti sukeisti kryžminimo metu (4↔1, 5↔6,6↔8). Jei keitimo metu chromosoma tokį geną jau turi, imamas kitas atsitiktinis genas, kurio ji dar neturi. 7 pav. pavaizduota, kaip dalinio atvaizdavimo operatorius atlieka kryžminimą.



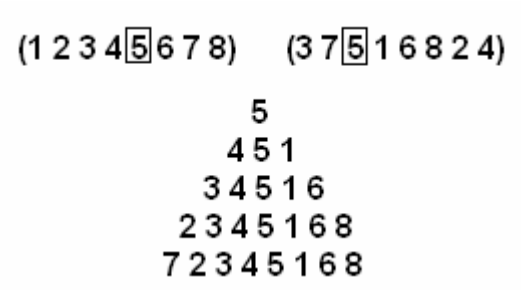

8 pav. Dalinio atvaizdavimo kryžminimas

Tvarkos kryžminimo metu parenkami du kryžminimo taškai. Viduriniai tėvų chromosomų genų intervalai vaikų chromosomose išlieka nepakeisti, jie tampa pirmais vaikų chromosomų genais, o prie šių genų, pagal jų išsidėstymą tėvinėse chromosomose, iš eilės prijungiami kiti tėvinių chromosomų genai, kurie dar nebuvo pradiniame genų intervale. 8 pav. pavaizduota, kaip veikia tvarkos kryžminimo operatorius.




9 pav. Tvarkos kryžminimas

Godaus lankų sukeitimo kryžminimo operatorius atsitiktinai pasirenka vieną miestą ir nustato miesto poziciją tėvų chromosomose. Prie pasirinkto miesto iš kairės pusės pridedami visi pirmo tėvo genai (lankai) tokia eilės tvarka, kokia jie yra išdėstyti einant į nuo pasirinkto miesto pozicijos į kairę, o iš dešinės visi antro tėvo genai (lankai) tokia eilės tvarka, kokia jie yra išdėstyti einant į nuo pasirinkto miesto pozicijos į dešinę. Jei miesto negalima pridėti prie vaiko chromosomos, nes toks miestas jau yra įtrauktas, kitas miestas pasirenkamas atsitiktinai iš likusių laisvų miestų. Kaip godaus lankų sukeitimo kryžminimo operatorius atlieka dviejų tėvinių chromosomų kryžminimą pavaizduota 10 paveiksle.




10 pav. Godus lankų sukeitimo kryžminimas

### 2.1.3 Keliaujančio prekeivio uždavinio sprendimas naudojant modeliuoto atkaitinimo algoritmą

Keliaujančio prekeivio uždavinį galima spręsti ir naudojant modeliuoto atkaitinimo algoritmą. Algoritmo vykdymo žingsniai yra tokie:

1) Sudaromas atsitiktinis maršrutas, kuris tenkina keliaujančio prekeivio uždavinio sąlygas. Šis sprendinys tampa „geriausiu“. Parenkama pradinė algoritmo temperatūra;
2) Tikimybiškai (atsitiktinai) pasirenkame maršrutą s', kuris yra s kaimynas;
3) Jei maršrutas s' tenkina tikimybinį patvirtinimo kriterijų (kuris priklauso nuo pradinės temperatūros), tada geriausias maršrutas = maršrutas s';
4) Atnaujiname temperatūrą pagal temperatūros atnaujinimo funkciją, nustatytą šiam procesui;
5) Jei nepatenkinama baigimo sąlyga, grįžtam į antrą žingsnį.

11 pav. Modeliuoto atkaitinimo algoritmas TSP uždaviniui

### 2.1.4 Keliaujančio prekeivio uždavinio sprendimas naudojant skruzdžių kolonijų algoritmą

Sprendžiant keliaujančio prekeivio uždavinį, naudojant skruzdžių kolonijų algoritmą, svarbu nustatyti pradinius algoritmo parametrus. Algoritmui reikalingi tokie pradiniai parametrai:

- Skruzdžių kiekis – kuo daugiau skruzdžių, tuo didesnė tikimybė, kad optimalus sprendimas bus rastas. Bet pasirinkus per didelį skaičių skruzdžių algoritmas dirbs ilgai;
- Parametras $\alpha$ – šiuo parametru galima reguliuoti feromonų kiekio svarbą skruzdei renkantis miestus, kuriuos ji aplankys;
- Parametras $\beta$ – šiuo parametru galima reguliuoti atstumo tarp miestų svarbą skruzdei renkantis miestus, kuriuos ji aplankys;

- Parametras ρ – šiuo parametru nurodomas feromonų garavimo koeficientas.

Tikimybė, kad skruzdės įtrauks miestą į savo maršrutą, yra apskaičiuojama naudojantis Dorigo [DG97] pateikta formule:

$$p_k(i,j) = \begin{cases} \dfrac{\tau_{ij}^{\alpha} * d_{ij}^{\beta}}{\sum_{g \in J_k(i)} \tau_{ig}^{\alpha} * d_{ig}^{\beta}} \\ 0 \end{cases}$$

kur:

- $p_k(i,j)$ – tikimybė, kad skruzdė *k* iš miesto *i* nueis į miestą *j*;
- $j \in J_k(i)$ – Aibė miestų, kurių skruzdė *k* dar neaplankė;
- α – feromonų kiekio įtaka, skruzdei renkantis naują miestą;
- β – atstumų tarp miestų įtaka, skruzdei renkantis naują miestą.

Kai skruzdė aplanko visus miestus, dydžiu $\Delta\tau_k$ atnaujinamas feromonų kiekis lankams, kurie sudaro pasirinktą maršrutą. $\Delta\tau_k$ apskaičiuojamas naudojantis šia formule: $\Delta\tau_k = \frac{1}{L_k}$, kur $L_k$ – skruzdės *k* surasto maršruto ilgis. Kai visos skruzdės baigia darbą, reikia atnaujinti visų galimų maršrutų lankų feromonų kiekį, tokiu būdu imituojamas feromonų garavimas ir išvengiama lokalių minimumų. Tai galima padaryti esamą feromonų kiekį padauginus iš dydžio $(1-\rho)$, kur ρ – feromonų garavimo koeficientas.

```
Procedure ACOTSP
        Parametrų iniciajavimas;
        While (sustojimo sąlyga) do
                konstruotiMaršrutus();
                atnaujintiFeromonus();
        end;
end;
```

12 pav. Skruzdžių kolonijos (ACO) algoritmas

## 2.2 Kuprinės uždavinys

Kuprinės uždavinys (angl. Knapsack problem) – taip pat vienas labiausiai paplitusių optimizacijos uždavinių. Uždavinio esmė – suskirstyti duotų objektų aibę į atskirus poaibius, kurie turi tenkinti iškeltus reikalavimus. Skirtingoms uždavinių variacijoms reikalavimai gali būti skirtingi.

Formalus uždavinio apibrėžimas yra toks: tarkime turime N nedalomų skirtingų daiktų, kurių svoris - $c_j$, o vertė - $a_j$, $j=1,...,n$. Į kurinę telpa tik tam tikras kiekis daiktų, kurių bendra svorių suma negali viršyti svorio b. Reikia rasti daiktų rinkinį, kuris butų geriausias daiktų vertės atžvilgiu, bet neviršytų leistino daiktų svorio. Nagrinėjamas uždavinys išreiškiamas šiuo matematiniu modeliu:

$$max \sum_{j=1}^{n} c_j x_j ,$$

$x_j$ šiame modelyje yra 0 arba 1, jei daiktas įtraukiamas į rinkinį tada 1, jei ne – 0.

### 2.2.1 Įrodymas, kad kuprinės uždavinys priklauso NPC uždavinių sudėtingumo klasei

Kadangi euristiniai algoritmai taikomi tik uždaviniams iš NPC uždavinių aibės, įrodysime, kad kuprinės uždavinys priklauso tokiai uždavinių sudėtingumo aibei ir kitokio tipo algoritmais per priimtiną skaičiavimo laiką jo išspręsti negalima.

Tarkime turime baigtinę daiktų aibę U, operaciją $s(u) \in N+$, kuri skaičiuoja pasirinkto daiktų rinkinio svorį ir operaciją $v(u) \in N+$, kuri skaičiuoja pasirinkto daiktų rinkinio vertę. Taip pat žinome leistiną daiktų rinkinio svorį B ir daiktų rinkinio vertę K, kurią turime pasiekti. Reikia nustatyti, ar egzistuoja toks daiktų aibės U rinkinys u, kad s(u) <= B ir v(u) >= K.

Pirma įrodysime, kad kuprinės uždavinys priklauso NP sudėtingumo klasei. Tam, kad patikrintume ar daiktų aibės U rinkinys u tenkina iškeltas sąlygas, turime įvykdyti operaciją s(u), operaciją v(u) ir gautus rezultatus atitinkamai palyginti su dydžiais B ir K. Visa tai patikrinimo algoritmas padaro per polinominį laiką, todėl kuprinės uždavinys priklauso NP uždavinių sudėtingumo klasei.

Dabar įrodysime, kad keliaujančio prekeivio uždavinys priklauso NP-sudėtingų uždavinių klasei. Kad tai padarytume, reikia įrodyti, kad PARTITION $\leq$ pKNAPSACK. Sakykime, kad kiekvienam aibės U rinkiniui u galioja tokios sąlygos: s(u)=v(u) ir B=K=1/2*s(u). Tokiu atveju,

keliaujančio prekeivio uždavinys yra suvedamas į aibių padalinimo uždavinį (PARTITION problem), kuris priklauso NP- sudėtingų uždavinių klasei, todėl ir kuprinės uždavinys priklauso NP-sudėtingų uždavinių klasei. Kadangi kuprinės uždavinys priklauso NP uždavinių klasei ir priklauso NP-sudėtingų uždavinių aibei, tai kuprinės uždavinys priklauso ir NPC uždavinių klasei. Įrodyta.

### 2.2.2 Kuprinės uždavinio sprendimas

Kuprinės uždavinys gali būti sprendžiamas daugeliu metodų, bet jie nėra vienodai efektyvūs. Vienas iš sprendimo būdų – naudoti dinaminį programavimą. Dinaminio sprendimo idėja:

Tarkime, nagrinėjame $J$-tąjį daiktą, o kuprinėje dar yra $K$ talpos vienetų, galime pasielgti dvejopai:

a) Jeigu daiktą imame, tuomet galime iš $J - 1$ daiktų paimti tiek, kad neviršytų $K - D_j$

b) Jeigu neimame, tuomet galime iš $J - 1$ daiktų paimti tiek, kad neviršytų $K$

Iš šių dviejų atvejų, pasirenkime vertingesnį.

13 pav. Kuprinės uždavinio sprendimas taikant dinaminį programavimą

Taigi, problemą $J$ daiktų atveju galime išspręsti, jeigu žinome problemos su $J - 1$ daiktų sprendimą. Matematiškai tai atrodo taip: $F(j, k) = max(F(j - 1, k), F(j - 1, k - D_j) + V(j))$. Sprendimą galima rasti ieškant nuo apačios, t.y. pirma apskaičiuojant $F$ reikšmes su mažesnėmis $J$ reikšmėmis. Taip pat, būtina atkreipti dėmesį, kad būtina turėti tam tikras „ribines“ reikšmes. Šiuo atveju tai būtų: $F(0, k) = 0$.

Kuprinės uždaviniui spręsti taip pat galima naudoti modeliuoto atkaitinimo, genetinius ir kitus euristinius algoritmus.

### 2.2.3 Kuprinės uždavinio sprendimas naudojant genetinius algoritmus

Kuprinės uždavinys gali būti sprendžiamas naudojant genetinius algoritmus. Genetinio algoritmo veikimą sprendžiant kuprinės uždavinį galime pavaizduoti tokiu pavyzdžiu. Tarkime turime tokius daiktus:

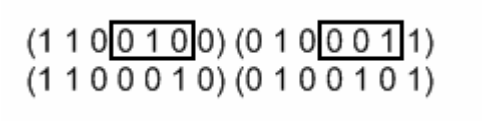
| Daiktai | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Svoris | 5 | 8 | 3 | 2 | 7 | 9 | 4 |
| Vertė | 7 | 8 | 4 | 10 | 4 | 6 | 4 |

Kuprinės talpa 22 svorio vienetai, tikslas patalpinti tokius daiktus, kurie duotų didžiausią vertę. Kuprinės turinį koduosime dvejetainiu būdu, todėl individo chromosoma bus saugoma tokiu pavidalu:

Daiktai: 1 2 3 4 5 6 7

Kuprinės turinys (chromosoma): 0 1 0 0 1 0 0

Šiuo atveju kuprinės turinį sudaro 2 ir 5 daiktai, bendras kuprinės svoris 15, o bendra vertė 12. Saugant kuprinės duomenis tokiu pavidalu nesunku pritaikyti kryžminimo ir mutacijos operatorius. Tarkime turime du individus, kuriems pritaikysime kryžminimo operatorių: 1 1 0 0 1 0 0 (svoris 19, vertė 20) ir 0 1 0 0 0 1 1 (svoris 18, vertė 21). Kryžminimą atliksime per du kryžminimo taškus:




Po kryžminimo gaunami du nauji individai: 1 1 0 0 0 1 0 (svoris 22, vertė 21) ir 0 1 0 0 1 0 1 (svoris 19, vertė 16). Kad algoritmas nepakliūtų į lokalų minimumą, papildomai reikia pritaikyti mutacijos operatorių. Tarkime, kad pirmam vaikui pritaikomas sukeitimo (angl. exchange) mutacijos operatorius: 6 genas sukeičiamas su 4 ir gaunamas naujas individas 1 1 0 1 0 0 0, kurio svoris 15, o vertė 25. Šis individas turi geresnes charakteristikas negu jo tėvai, todėl bus įtrauktas į populiaciją.

### 2.2.4 Kuprinės uždavinio sprendimas naudojant modeliuoto atkaitinimo algoritmą

Modeliuoto atkaitinimo algoritmo taikymas kuprinės uždaviniui spręsti – panašus į keliaujančio prekeivio uždavinio sprendimą modeliuoto atkaitinimo algoritmu. Pagrindiniai algoritmo žingsniai pateikti žemiau esančioje kuprinės uždavinio sprendimo paradigmoje (14 pav.).

1) Atsitiktiniu būdu sudaromas pradinis sprendinys, kuris tenkina uždavinio sąlygas;
2) Parenkama pradinė algoritmo temperatūra;
2) Taikant mutacijos operatorių, gauname sprendinį s', kuris yra s kaimynas;
3) Jei sprendinys s' tenkina tikimybinį patvirtinimo kriterijų (kuris priklauso nuo pradinės temperatūros), tada geriausias sprendinys = sprendinys s';
4) Atnaujiname temperatūrą pagal temperatūros atnaujinimo grafiką, nustatytą šiam procesui;
5) Jei nepatenkinama baigimo sąlyga, grįžtam į antrą žingsnį.

14 pav. Modeliuoto atkaitinimo algoritmas kuprinės uždaviniui

Kuprinės uždavinio sprendimą modeliuoto atkaitinimo algoritmu galėtume iliustruoti paprastu pavyzdžiu. Tarkime turime 22 svorio vienetų kuprinę ir tokią daiktų aibę:

| Daiktai | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Svoris (kg) | 5 | 8 | 3 | 2 | 7 | 9 | 4 |
| Vertė (lt) | 7 | 8 | 4 | 10 | 4 | 6 | 4 |

Žemiau išvardinti algoritmo veiksmai vienos iteracijos metu:

1) Sugeneruojamas pradinis sprendinys. Tarkime buvo sugeneruotas toks sprendinys 0 1 0 0 1 0 0. Šiuo atveju kuprinės turinį sudaro 2 ir 5 daiktai, bendras kuprinės svoris 15kg, o bendra vertė 12Lt;

2) Parenkama pradinė algoritmo temperatūra, tarkime t = 20° ir aušinimo koeficientas $t_{const}$ = 0.99;

3) Pradinis sprendinys yra modifikuojamas pasirinktu būdu ir gaunamas naujas sprendinys. Tarkime, kad buvo gautas toks sprendinys 0 1 1 0 1 0 0, kurio svoris yra 18kg, o vertė 16Lt.

4) Jei patvirtinimo sąlyga patenkinama Math.random() < exp((s' - s)/t) [KGV83], tai naujas rastas sprendinys tampa geriausiu;

5) Temperatūra atnaujinama nustatytu grafiku. Tarkim aušinimui naudojame geometrinę formulę, tai t = t * $t_{const}$ = 20 * 0.99 = 19.8°;

6) Jei patenkinama baigimo sąlyga, algoritmas baigia darbą, priešingu atveju grįžtame į 3 žingsnį.

# 3 Lygiagretūs ir paskirstyti skaičiavimo tinklai (GRID)

## 3.1 Užduočių apibrėžimo kalba

GRID‘e atliekamos užduotys aprašomos JDL (Job Description Language) kalba. JDL kalbos paskirtis – apibrėžti GRID užduotį ir nustatyti jos vykdymui reikalingus atributus[Lit08]. Galimi užduoties atributai: užduoties tipas, užduoties nuoroda, kuri bus vykdoma GRID‘e, užduoties rezultatų bylos ir t.t. JDL bylos bendra struktūra pateikiama žemiau:

```
# komentaras
atributas1 = skaičius;
atributas2 = "eilutė";
atributas3 = { "eilute1", "eilute2" };
atributas4 = ( loginė išraiška );
```

Žemiau pateikiami JDL atributai, kurie bus naudojami tyrime:

- **Executable –** nurodoma programa, kuri bus vykdoma GRID‘e;
- **Arguments** – nurodomi programos parametrai, reikalingi užduoties įvykdymui;
- **StdOutput** – nurodoma byla, į kurią bus išvedami rezultatai;
- **StdError** – nurodoma byla, į kurią bus išvedami pranešimai apie klaidas;
- **JobType –** nurodomas užduoties tipas. Galimi užduoties tipai: "Normal", "MPICH", "Interactive", "DAG", "Parametric", "Checkpointable". Tyrime bus naudojamos "DAG" tipo užduotys;
- **InputSandbox** – nurodoma programa ir jos duomenys, kurie nusiunčiami įvykdymui į GRID skaičiavimo tinklą (iki 10 MB);
- **OutputSandbox –** nurodomos bylos, kurios gražinamos su užduoties rezultatais;
- **MyProxyServer –** nurodomas proxy serveris, kuris bus naudojamas užduoties nusiuntimui į skaičiavimo tinklą. Šis atributas būtinas norint skaičiuoti užduotis ilgiau negu 12 valandų;
- **Requirements –** nurodomi reikalavimai, kuriuos turi tenkinti skaičiavimo mazgas;
- **OutputSandboxDestURI –** nurodomas grid ftp serveris, į kurį persiunčiamos rezultatų bylos po užduoties įvykdymo;
- **Nodes –** nurodomos užduoties subužduotys, kurios bus vykdomos;
- **Description –** nurodomi subužduočių atributai;
- **Dependencies –** nurodomos subužduočių priklausomybės.

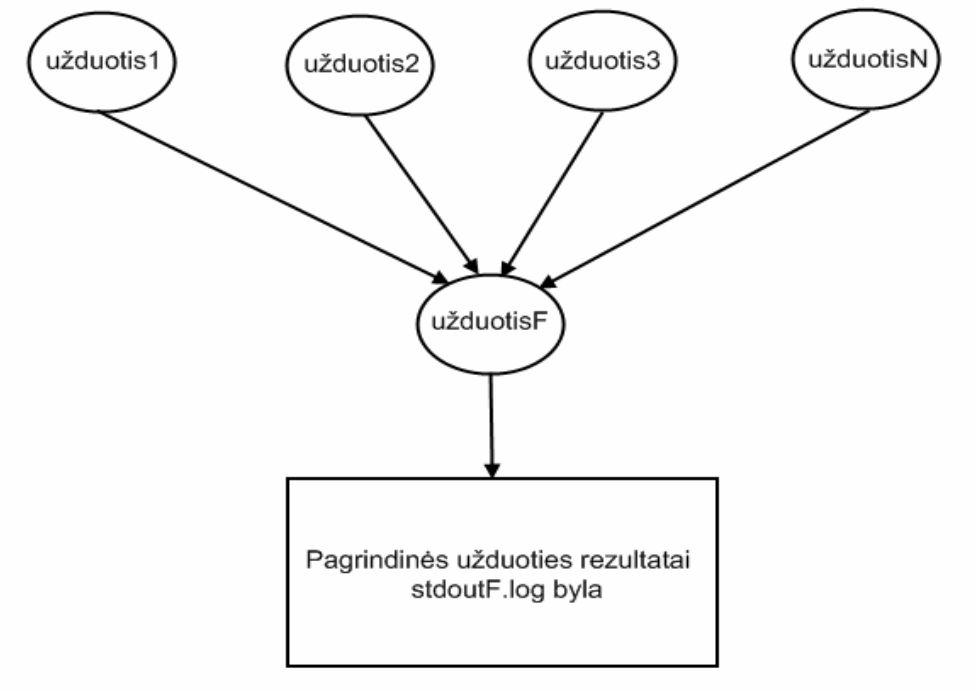
## 3.2 gLite aplinkos komandos

Euristinių algoritmų taikymo NPC uždaviniams tyrimas bus atliekamas „LitGrid“ (http://www.litgrid.lt) skaičiavimo tinkle. „LitGrid“ skaičiavimo tinklas naudoja gLite (Lightweight Middleware for Grid Computing, http://www.glite.org/) programinę įranga, kuri leidžia GRID tinkle vykdyti užduotis, gauti informaciją apie vykdomų užduočių būseną, gauti užduočių rezultatus ir t.t [Lit06]. Žemiau pateikiamos komandos, kurios bus reikalingos tiriamų užduočių vykdymui ir jų rezultatų gavimui „LitGrid“ skaičiavimų tinkle:

- **glite-wms-job-submit** – užduoties paleidimas GRID'e, pavyzdžiui *glite-wms-job-submit -a -o sa sa01.jdl* komanda nusiunčia į GRID tinklą užduotį sa01 ir unikalų užduoties ID įrašo į „sa“ bylą;
- **glite-wms-job-status –** užduoties būsenos informacija, pavyzdžiui *glite-wms-job-status -i sa* grąžins visų užduočių sąrašą, kurių ID buvo užsaugoti į bylą „sa“. Pasirinkus norimą užduotį, į ekraną bus išvedama jos būsena. Galimos kelios reikšmės:
    - **submitted –** užduotis nusiųsta į GRID skaičiavimo tinklą, bet dar nepriimta;
    - **waiting –** užduotis priimta, bet dar nepriskirta skaičiavimo mazgui;
    - **ready –** užduotis priskirta skaičiavimo mazgui, bet dar nenusiųsta į jį;
    - **scheduled –** užduotis nusiųsta į skaičiavimo mazgą ir laukia eilėje;
    - **running –** užduotis vykdoma;
    - **done –** užduotis įvykdyta;
    - **cleared –** vartotojas parsisiuntė užduoties rezultatus ir jie nebesaugomi GRID tinkle.
- **glite-wms-job-output –** užduoties rezultatų susigrąžinimas, pavyzdžiui *glite-wms-job-output -i sa* grąžins visų užduočių sąrašą, kurių ID buvo užsaugoti į bylą „sa“. Pasirinkus norimą užduotį, bus parsiųsta rezultatų byla;
- **glite-gridftp-mkdir –** katalogo sukūrimas grid ftp serveryje**,** pavyzdžiui *glite-gridftp-mkdir gsiftp://grid8.mif.vu.lt/tmp/irve3993* temp direktorijoje sukurs katalogą irve3993, kuris bus naudojamas tarpiniams užduočių rezultatams saugoti;
- **glite-url-copy –** rezultatų parsisiuntimas iš grid ftp serverio, pavyzdžiui *glite-url-copy gsiftp://grid8.mif.vu.lt/tmp/irve3993/stdout1.log* file:/afs/mif.vu.lt/stud/2003/irve3993/out/out1.log.

## 3.3 „DAG“ tipo užduotys

Kad padidintumėme tyrimų galimybes, GRID skaičiavimų tinkle bus naudojamos DAG (angl. directed acyclic graph) tipo užduotys. Tokio tipo užduotys leidžia GRID‘e užduotis skaičiuoti lygiagrečiai skirtinguose skaičiavimo mazguose, taip pat galima nurodyti užduočių priklausomybes. Naudojant šio tipo GRID užduotis, vienu metu galima spręsti daug uždavinių, o jas atlikus – jų rezultatus galima naudoti kitose užduotyse [Pac07].




15 pav. „DAG“ tipo užduoties vykdymas GRID tinkle

Tokio tipo užduotys tyrime turi savo privalumų ir trūkumų.

**Privalumai**:

- TSP užduotys gali būti išskirstytos į skirtingus skaičiavimo mazgus, todėl jų apimtis gali būti didesnė, nei vienos užduoties apimtis;
- Vienu metu galime paleisti N skirtingų užduoties šakų, taip pat galima konstruoti N lygių užduočių hierarchiją, nurodant skirtingas priklausomybes užduoties šakoms;
- Galima konstruoti hibridines užduotis. Kelios užduoties šakos gali būti sprendžiamos naudojant vieno tipo euristinį algoritmą, kitos šakos – kito tipo euristinius algoritmus. Galutinė užduotis sujungs visų šakų rezultatus ir naudojantis trečio tipo algoritmu pateiks galutinį rezultatą;
- Galima pasiimti tarpinius užduoties šakų rezultatus, neatsižvelgiant į galutinės užduoties būseną;

- Skirtingoms užduoties šakoms galima nurodyti skirtingus pradinius algoritmų parametrus;
- Galima naudoti skirtingas programavimo kalbas tam pačiam uždaviniui spręsti.

**Trūkumai**:

- Užduoties paruošimas yra sudėtingas, nes JDL byloje reikia nurodyti visų užduočių priklausomybes ir jų parametrus;
- Bent vienai užduoties šakai nepabaigus užduoties, nuo tos užduoties priklausomos užduotys negali būti vykdomos, jei ta užduotis dėl GRID‘e įvykusios klaidos nebus įvykdyta, tada ir visos kitos nuo tos užduoties priklausomos užduotys nebus įvykdytos.

Žemiau pateiktas DAG tipo užduoties šablonas, kuris bus naudojamas užduotims vykdyti.

```
type = "dag"; # užduoties tipas
MyProxyServer = "grid3.mif.vu.lt";
InputSandbox = {#užduočiai vykdyti reikalingos bylos};
Requirements = Member("VO-balticgrid-S-DEVEL-JRE-1.5.0_12",
other.GlueHostApplicationSoftwareRunTimeEnvironment);
nodes = [
      node1 = [
         description = [# “normal” tipo užduoties aprašymas, pateiktas 17 pav.];
      ];
      nodeN = [
         description = [#“normal” tipo užduoties aprašymas, pateiktas 17 pav.];
      ];
      nodeFinal = [
         description = [#“normal” tipo užduoties aprašymas, pateiktas 17 pav.];
      ];
 ];
dependencies = {
            {node1 , nodeFinal},
            {nodeN , nodeFinal}
   }
```

16 pav. DAG tipo užduoties šablonas

```
Executable = "/bin/sh";
Arguments = "sa01.sh";
StdOutput = "stdout1.log";
StdError = "stderr1.log";
InputSandbox = {root.InputSandbox};
OutputSandbox = {#rezultatų bylos};
OutputSandboxDestURI = {# rezultatų bylų saugojimo vieta grid ftp serveryje};
```

17 pav. “normal” tipo užduoties aprašymas

Euristiniai algoritmai įgyvendinti Java programavimo kalba. Nusiuntus juos į skaičiavimo mazgą, visas bylas reikia sukompiliuoti ir įvykdyti. Tam naudojamos standartinės Java komandos: javac ir java. Jos įvykdomos iš sh programavimo kalba parašyto scenarijaus (angl. script). Scenarijaus bylos turinys pateiktas žemiau.

```
echo "*** Uzduoties pradzia. ***"
javac TSP.java GA.java Sortable.java Sort.java Gene.java City.java Cities.java SA.java
echo "*** Sukompiliuota. Paleidziama... ***"
START=$(date +%s)
java TSP -v -p 0.99 gr96.tsp 400
END=$(date +%s)
DIFF=$(( $END - $START ))
echo "Vykdymo laikas $DIFF sekundes"
echo "*** Uzduoties pabaiga. ***"
```

18 pav. Užduoties paleidimo scenarijaus bylos pavyzdys

# 4 Tyrimo duomenys ir programos

## 4.1 TSPLIB duomenų biblioteka

TSPLIB duomenų bibliotekoje galima rasti skirtingų uždavinių ir jų sprendinių keliaujančio prekeivio uždaviniui spręsti. Tyrinėtojai gali parsisiųsti duomenų bylas savo algoritmams testuoti, taip pat gali nusiųsti savo rezultatus, jei jie yra geresni už iki tol publikuotus sprendinius. Kiekvieną duomenų bylą sudaro duomenys ir jų aprašymas. Aprašymo dalis susideda iš duomenų formato apibūdinimo ir komentarų. Duomenų formato aprašymas susideda iš tokių tyrimui reikalingų atributų:

- „Name“ – duomenų bylos pavadinimas;
- „Type“ – duomenų tipas. Tyrime bus naudojami simetrinio tipo TSP duomenys (maršruto lankas iš *i* į *j* yra tokios pat vertės kaip ir lankas iš *j* į *i*);
- „Comment“ – komentaras apie duomenis;
- „Dimension“ – miestų skaičius;
- „Edge_Weight_Type“ – nusako kokiu formatu pateikiami miestų duomenys, tyrime naudojami GEO ir EUC_2D duomenų formatai;
- „Tour_Section“ – nusako, nuo kur prasideda duomenų dalis.

TSP tipo miestų duomenys yra pateikiami paprastomis koordinatėmis arba geografinėmis koordinatėmis: geografine ilguma ir platuma. Kai duomenys pateikiami paprastomis koordinatėmis, skaičiuojamas euklidinis atstumas. Kai miesto koordinatės yra pateikiamos geografine ilguma ir platuma, atstumas tarp miestų bus apskaičiuojamas naudojantis žemiau pateiktu algoritmu. Iš pradžių geografinė ilguma ir platuma iš laipsnių konvertuojama į radianus:

```
double PI = 3.141592;
double deg = Math.floor(ilguma);
double min = ilguma - deg;
latitude = PI*(deg + 5*min/3)/180;
deg = Math.floor(platuma);
min = platuma - deg;
longitude = PI*(deg + 5*min/3)/180;
```

19 pav. Geografinės ilgumos ir platumos laipsnių konvertavimas į radianus

Sekančiame žingsnyje apskaičiuojamas atstumas kilometrais tarp dviejų miestų *i* ir *j*:

```
double rrr=6378.388;
```

```
double q1 = Math.cos(longitudeI - longitudeJ);
double q2 = Math.cos(latitudeI - latitudeJ);
double q3 = Math.cos(latitudeI + latitudeJ);
distancesIJ = (int) Math.round( (rrr*Math.acos(0.5*((1.0+q1)*q2-(1.0-q1)*q3))+1.0) );
```

20 pav. Atstumo kilometrais tarp dviejų miestų apskaičiavimas

## 4.2 TSP uždavinio tyrimo programa

Tyrimui naudojama aplikacija buvo įgyvendinta java programavimo kalba. Visa aplikacija susideda iš tokių klasių:

- TSP.java – pagrindinė aplikacijos klasė, atsakinga už programos paleidimą;
- GA.java – genetinį algoritmą įgyvendinanti klasė. Reikalingi parametrai: populiacijos dydis, individų skaičius, kurie dalyvaus kryžminime ir individų skaičius, kurie dalyvaus mutacijoje;
- Gene.java – įgyvendintas GSX (Greedy sub tour crossover) kryžminimo operatorius, taip pat įgyvendintas 2opt mutacijos operatorius [SY98].
- Cities.java – klasė, kuri saugo visus iš duomenų bylos nuskaitytus miestus, taip pat pagal pateiktus algoritmus, skaičiuoja rasto maršruto atstumą;
- City.java – klasė, kuri saugo informaciją apie miestą, pavyzdžiui jo koordinates;
- SA.java – klasė, kuri įgyvendina modeliuoto atkaitinimo algoritmą. Reikalingi parametrai: pradinė temperatūra ir aušinimo koeficientas. Temperatūra aušinama naudojantis geometrine formule;
- ACO.java – klasė, kuri įgyvendina skruzdžių kolonijų algoritmą. Reikalingi parametrai: maksimalus iteracijų skaičius, skruzdžių kiekis, $\alpha$ parametras, $\beta$ parametras ir $\rho$ parametras;
- Ant.java – klasė, kuri įgyvendina skruzdės elgseną. Ši klasė turi metodus naujų maršrutų sudarymui ir feromonų kiekio atnaujinimui;

Paleidžiant aplikaciją GRID tinkle, reikia nurodyti jos tipą. Reikšmė „node“ reiškia, kad programa yra nepriklausoma nuo kitų GRID užduoties aplikacijų, todėl gali dirbti standartiškai, reikšmė „final“ nusako, kad užduotis yra priklausoma, todėl jai reiks nuskaityti kitų programų rezultatų bylas ir su tais duomenimis atlikti skaičiavimus.

GRID skaičiavimo tinkle, bet kokie rezultatai, kurie išvedami į ekraną, yra patalpinami vartotojo nurodytoje byloje. „Final“ tipo programa nuskaito visas kitų mazgų rezultatų bylas (visų jų pavadinimai yra stdoutN.log, kur N užduoties identifikatorius) ir iš jų sudaro pradinius algoritmų

maršrutus. Taip galima sujungti visų mazgų rezultatus į vieną visumą ir atlikti skaičiavimus su geresniais pradiniais duomenis. Tikėtina, paskutinio mazgo programa nuskaitytus maršrutus sugebės pagerinti dar kartą ir pateiks geresnius rezultatus.

# 5 TSP uždavinio eksperimentiniai rezultatai

Siekiant ištirti euristinių algoritmų efektyvumą, sprendžiant NPC sudėtingumo aibės uždavinius GRID skaičiavimo tinkle, buvo atlikti eksperimentiniai bandymai su keliaujančio prekeivio ir kuprinės uždaviniais. Keliaujančio prekeivio uždavinio testavimui buvo naudojami testiniai duomenys iš TSPLIB duomenų bibliotekos [Rei08]. Buvo pasirinkta 20 didesnės apimties testinių pavyzdžių (skaičius prie pavyzdžio pavadinimo nurodo miestų skaičių):

- gr96.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 55209 km.
- a280.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 2579 km.
- ali535.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 202339 km.
- bier127.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 118282 km.
- gr666.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 294358 km.
- ch130.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 6110 km.
- ch150.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 6528 km.
- d198.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 15780 km.
- d493.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 35002 km.
- d657.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 48912 km.
- d1291.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 50801 km.
- fl417.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 11861 km.
- fl1400.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 20127 km.
- pcb442.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 50778 km.
- rat575.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 6773 km.
- eil101.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 629 km.
- d2103.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 80450 km.
- d1655.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 62128 km.
- tsp225.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 3916 km.
- lin318.TSP, optimalaus maršruto tikslo funkcijos reikšmė – 42029 km.

Kiekvienam iš išvardintų testinių pavyzdžių po 10 kartų buvo pritaikyti tokie algoritmai:

- Genetinis algoritmas;
- Modeliuoto atkaitinimo algoritmas;
- Skruzdžių kolonijų algoritmas.

Kiekvienam bandymui buvo sukurta viena „DAG“ tipo užduotis, kuri turėjo 10 nepriklausomų mazgų (angl. node). Visi užduoties mazgai buvo įvykdomi lygiagrečiai, skirtinguose GRID skaičiavimo tinklo mazguose. Efektyvumas buvo nustatinėjamas remiantis tokiais kriteriais:

- Vidutinis santykinis tikslo funkcijos nuokrypis nuo žinomos optimalios reikšmės – $\bar{\delta}$ ($\bar{\delta}=100(\bar{z}-z_{opt})/z_{opt}$) [%], kur $\bar{z}$ yra bandymuose gautų tikslo funkcijų reikšmių aritmetinis vidurkis, gautas atlikus 10 bandymų, o $z_{opt}$ – optimali žinoma tikslo funkcijos reikšmė;
- Sprendimų, esančių „1% optimalumo intervale“ ($\bar{\delta} \leq 1$), skaičius $C_{1\%}$;
- Sprendimų, esančių „5% optimalumo intervale“ ($\bar{\delta} \leq 5$), skaičius $C_{5\%}$, taikoma tik skruzdžių kolonijų algoritmui, nes jo efektyvumas dažnai neleidžia pasiekti „1%“ optimumo;
- Bandymų skaičius $C_{opt}$, kai gauta tikslo funkcijos reikšmė nėra blogesnė nei geriausia žinoma testinio uždavinio tikslo funkcijos reikšmė;
- Vidutinis užduoties vykdymo GRID skaičiavimo tinkle laikas $t_{GRID}$.
- Geriausio bandymo tikslo funkcijos reikšmė $z_{best}$.

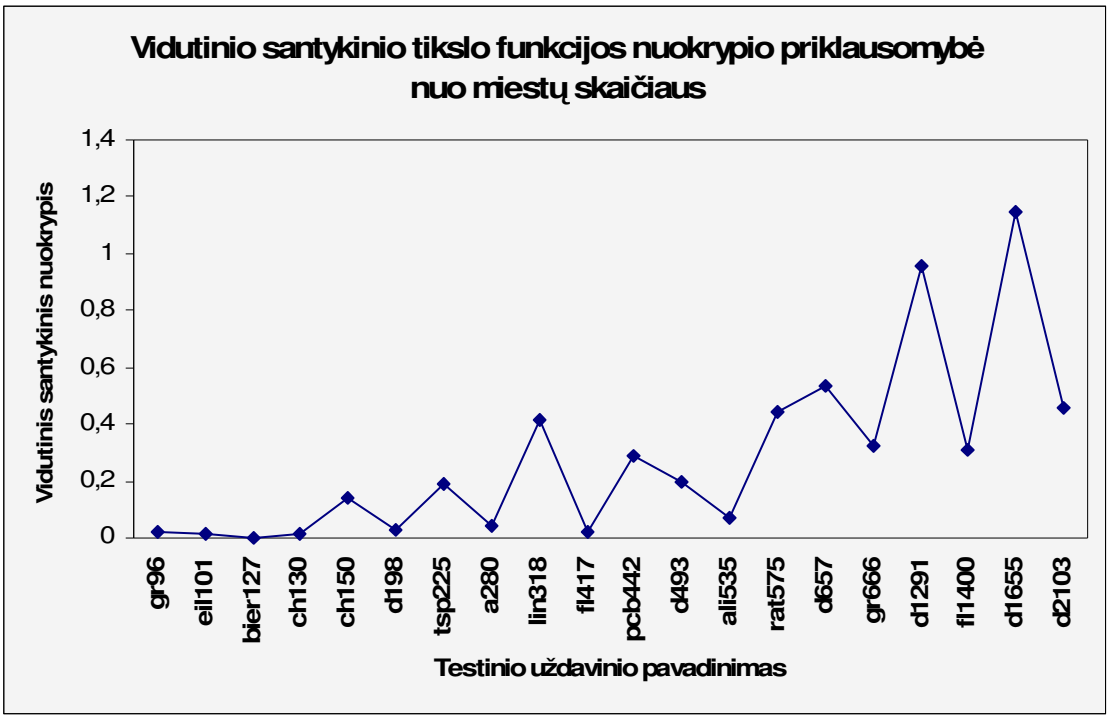
## 5.1 Genetinio algoritmo tyrimas

1 lentelė, genetinio algoritmo taikymo TSP testiniams uždaviniams spręsti GRID skaičiavimo tinkle rezultatai

| Testinio uždavinio pavadinimas | $z_{opt}$ (km.) | $z_{best}$ (km.) | Populiacijos dydis | Max generacijų skaičius | $\bar{\delta}$ | $C_{opt}$ | $C_{1\%}$ | $t_{GRID}$ (sek.) |
|---|---|---|---|---|---|---|---|---|
| gr96 | 55209 | **55209** | 200 | 300 | 0,0239 | 8 | 10 | 1 |
| eil101 | 629 | **629** | 300 | 600 | 0,01589 | 9 | 10 | 5,7 |
| bier127 | 118282 | **118282** | 400 | 700 | 0 | 10 | 10 | 3,4 |
| ch130 | 6110 | **6110** | 300 | 700 | 0,01145 | 9 | 10 | 3,4 |
| ch150 | 6528 | **6528** | 350 | 1000 | 0,14399 | 5 | 10 | 5 |
| d198 | 15780 | **15780** | 600 | 1500 | 0,02471 | 1 | 10 | 21,1 |
| tsp225 | 3916 | **3916** | 600 | 1500 | 0,18896 | 2 | 10 | 36 |
| a280 | 2579 | **2579** | 600 | 1500 | 0,03877 | 7 | 10 | 61,7 |
| lin318 | 42029 | **42029** | 1000 | 15000 | 0,4178 | 1 | 9 | 446 |
| fl417 | 11861 | **11861** | 1500 | 15000 | 0,0177 | 5 | 10 | 2598 |
| pcb442 | 50778 | 50825 | 2000 | 15000 | 0,29067 | 0 | 10 | 4064 |
| d493 | 35002 | 35032 | 2000 | 10000 | 0,19913 | 0 | 10 | 7190 |
| ali535 | 202339 | 202375 | 3000 | 30000 | 0,07116 | 0 | 10 | 9749 |
| rat575 | 6773 | 6790 | 2000 | 15000 | 0,44588 | 0 | 10 | 16133 |
| d657 | 48912 | 49097 | 500 | 10000 | 0,53279 | 0 | 10 | 4065 |
| gr666 | 294358 | 294553 | 5000 | 30000 | 0,32395 | 0 | 10 | 14519 |
| d1291 | 50801 | 51044 | 3000 | 40000 | 0,95726 | 0 | 10 | 20764 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| fl1400 | 20127 | 20138 | 3000 | 30000 | 0,30605 | 0 | 10 | 27543 |
| d1655 | 62128 | 62412 | 500 | 15000 | 1,14666 | 0 | 4 | 9896 |
| d2103 | 80450 | 80521 | 2000 | 30000 | 0,45879 | 0 | 9 | 28414 |
| | | | | Vidurkis: | **0,28077** | | | |

Atlikus genetinio algoritmo bandymus su testiniais TSP bibliotekos pavyzdžiais, gauti geri rezultatai: 10 TSP bibliotekos pavyzdžių pavyko išspręsti, tai yra, buvo rastas geriausias galimas maršrutas. Taip pat visais atvejais buvo rastas sprendinys, esantis 1% optimalumo intervale ($C_{1\%}$). Vidutinio santykinio tikslo funkcijos nuokrypio nuo žinomos optimalios reikšmės vidurkis taip pat mažas - **0,28077**, tai rodo labai gerą genetinio algoritmo efektyvumą. Buvo pastebėta, kad didėjant miestų skaičiui, rasti sprendiniai prastėja, tai rodo ir 21 paveikslas.




21pav. Vidutinio santykinio tikslo funkcijos nuokrypio priklausomybė nuo miestų skaičiaus, genetinis algoritmas

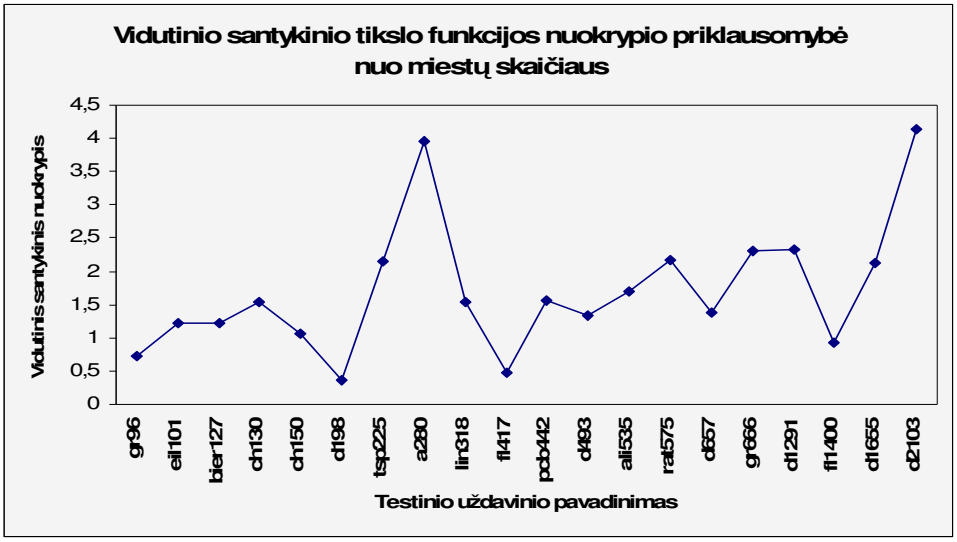
## 5.2 Modeliuoto atkaitinimo algoritmo tyrimas

2 lentelė. Modeliuoto atkaitinimo algoritmo taikymo TSP testiniams uždaviniams spręsti GRID skaičiavimo tinkle rezultatai

| Testinio uždavinio pavadinimas | $z_{opt}$ (km) | $z_{best}$ (km) | Aušinimo koeficientas | Pradinė temperatūra | $\bar{\delta}$ | $C_{opt}$ | $C_{1\%}$ | $t_{GRID}$ (sek) |
|---|---|---|---|---|---|---|---|---|
| gr96 | 55209 | **55209** | 0.99 | 1000 | 0,71944 | 2 | 8 | 8 |
| eil101 | 629 | 631 | 0.99 | 1 | 1,22416 | 0 | 3 | 0 |
| bier127 | 118282 | **118282** | 0.99 | 1000 | 1,21709 | 2 | 7 | 14 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ch130 | 6110 | **6110** | 0.99 | 50 | 1,54173 | 1 | 5 | 5 |
| ch150 | 6528 | 6556 | 0.99 | 50 | 1,0723 | 0 | 7 | 18 |
| d198 | 15780 | 15810 | 0.99 | 100 | 0,35804 | 0 | 10 | 39 |
| tsp225 | 3916 | 3958 | 0.99 | 5 | 2,14504 | 0 | 0 | 34 |
| a280 | 2579 | 2655 | 0.999 | 1 | 3,95889 | 0 | 0 | 109 |
| lin318 | 42029 | 42311 | 0.99 | 100 | 1,52704 | 0 | 1 | 122 |
| fl417 | 11861 | 11879 | 0.99 | 80 | 0,4637 | 0 | 10 | 245 |
| pcb442 | 50778 | 51417 | 0.99 | 800 | 1,55815 | 0 | 0 | 305 |
| d493 | 35002 | 35233 | 0.99 | 400 | 1,3322 | 0 | 1 | 409 |
| ali535 | 202339 | 204789 | 0.99 | 10000 | 1,68511 | 0 | 1 | 745 |
| rat575 | 6773 | 6860 | 0.99 | 50 | 2,16152 | 0 | 0 | 510 |
| d657 | 48912 | 49346 | 0.99 | 400 | 1,36858 | 0 | 1 | 720 |
| gr666 | 294358 | 299802 | 0.99 | 250 | 2,3 | 0 | 0 | 594 |
| d1291 | 50801 | 51451 | 0.99 | 200 | 2,3275 | 0 | 0 | 3076 |
| fl1400 | 20127 | 20213 | 0.99 | 160 | 0,92512 | 0 | 5 | 3730 |
| d1655 | 62128 | 63113 | 0.99 | 200 | 2,1306 | 0 | 0 | 4908 |
| d2103 | 80450 | 82513 | 0.99 | 200 | 4,12939 | 0 | 0 | 8658 |
| | | | | Vidurkis: | **1,70728** | | | |

Atlikus modeliuoto atkaitinimo algoritmo bandymus su testiniais TSP bibliotekos pavyzdžiais, gauti rezultatai rodo, kad šis algoritmas savo efektyvumu nusileidžia genetiniam algoritmui. Tik su trim testiniais pavyzdžiais buvo gautas optimalus maršrutas, o sprendiniai, esantys 1% optimalumo intervale ($C_{1\%}$), buvo rasti tik su 12 testinių pavyzdžių. Vidutinio santykinio tikslo funkcijos nuokrypio nuo žinomos optimalios reikšmės vidurkis taip pat blogesnis – 1,70728.




22 pav. Vidutinio santykinio tikslo funkcijos nuokrypio priklausomybė nuo miestų skaičiaus, modeliuoto atkaitinimo algoritmas

## 5.3 Skruzdžių kolonijų algoritmo tyrimas

3 lentelė, skruzdžių kolonijų algoritmo taikymo TSP testiniams uždaviniams spręsti GRID skaičiavimo tinkle rezultatai

| Testinio uždavinio pavadinimas | $z_{opt}$ (km) | $z_{best}$ (km) | Skruzdžių kiekis | α | β | ρ | $\bar{\delta}$ | $C_{opt}$ | $C_{5\%}$ | $t_{GRID}$ (sek) |
|---|---|---|---|---|---|---|---|---|---|---|
| gr96 | 55209 | 57612 | 100 | 1 | 12 | 0.1 | 7,26439 | 0 | 1 | 170 |
| eil101 | 629 | 661 | 100 | 1 | 12 | 0.1 | 7,34499 | 0 | 0 | 180 |
| bier127 | 118282 | **118282** | 100 | 1 | 16 | 0.1 | 0,41702 | 1 | 10 | 284 |
| ch130 | 6110 | 6452 | 100 | 3 | 15 | 0.2 | 7,20294 | 0 | 0 | 533 |
| ch150 | 6528 | 6682 | 300 | 3 | 15 | 0.2 | 5,57444 | 0 | 3 | 723 |
| d198 | 15780 | 16443 | 300 | 3 | 15 | 0.2 | 5,15462 | 0 | 3 | 909 |
| tsp225 | 3916 | 4242 | 300 | 3 | 20 | 0.2 | 9,22369 | 0 | 0 | 1462 |
| a280 | 2579 | 2858 | 100 | 1 | 12 | 0.1 | 12,5514 | 0 | 10 | 2144 |
| lin318 | 42029 | 47258 | 300 | 3 | 20 | 0.2 | 14,1421 | 0 | 0 | 2612 |
| fl417 | 11861 | 12766 | 300 | 1 | 12 | 0.2 | 9,11221 | 0 | 0 | 3985 |
| pcb442 | 50778 | 55425 | 300 | 1 | 12 | 0.2 | 10,9014 | 0 | 0 | 6687 |
| d493 | 35002 | 36224 | 300 | 1 | 15 | 0.1 | 4,50431 | 0 | 8 | 4890 |
| ali535 | 202339 | 222497 | 300 | 1 | 12 | 0.1 | 12,3387 | 0 | 0 | 7353 |
| rat575 | 6773 | 7750 | 300 | 1 | 12 | 0.2 | 15,2266 | 0 | 0 | 9662 |
| d657 | 48912 | 54488 | 300 | 1 | 15 | 0.1 | 12,4284 | 0 | 0 | 12268 |
| gr666 | 294358 | 331683 | 300 | 3 | 15 | 0.2 | 13,5849 | 0 | 0 | 18923 |
| d1291 | 50801 | 55662 | 300 | 1 | 12 | 0.2 | 10,0018 | 0 | 0 | 31069 |
| fl1400 | 20127 | 24233 | 300 | 1 | 12 | 0.2 | 11,2335 | 0 | 0 | 41062 |
| d1655 | 62128 | 69586 | 300 | 3 | 15 | 0.2 | 12,5385 | 0 | 0 | 55864 |
| d2103 | 80450 | 85595 | 300 | 1 | 12 | 0.2 | 15,1421 | 0 | 0 | 64932 |
| | | | | | | | Vidurkis: | **9,794398** | | |

Atlikus skruzdžių kolonijų algoritmo bandymus su testiniais TSP bibliotekos pavyzdžiais, gauti rezultatai rodo, kad šis algoritmas savo efektyvumu nusileidžia ir genetiniam algoritmui, ir modeliuoto atkaitinimo algoritmui. Vidutinio santykinio tikslo funkcijos nuokrypio nuo žinomos optimalios reikšmės vidurkis - **9,794398**. Bandymų metu buvo išspręstas tik vienas testinis uždavinys, taip pat algoritmo veikimo laikas žymiai ilgesnis nei kitų algoritmų. Žemiau pateiktame paveiksle (23 pav.) parodyta, kaip bandymų metu augant miestų skaičiui prastėja vidutinė santykinė tikslo funkcijos reikšmė.


23 pav. Vidutinio santykinio tikslo funkcijos nuokrypio priklausomybė nuo miestų skaičiaus, skruzdžių kolonijų algoritmas

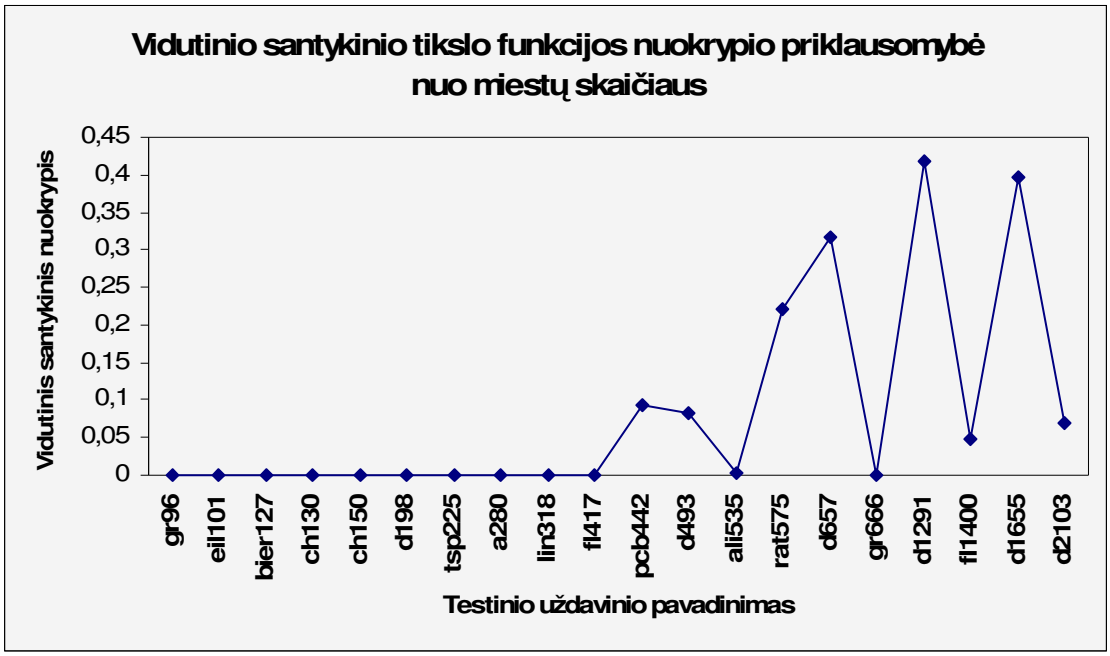
## 5.4 Genetinio algoritmo taikymas, kai pradinė populiacija sudaryta iš kito bandymo rezultatų

Siekiant pagerinti genetinio algoritmo rezultatus buvo nuspręsta atlikti bandymą, kai genetinio algoritmo pradinė populiacija yra sudaroma iš kitų genetinio algoritmo rezultatų. Tai galima nesunkiai atlikti GRID skaičiavimo tinkle. Buvo vykdoma „DAG“ tipo užduotis su nurodytomis užduočių priklausomybėmis. Iš pradžių būdavo įvykdoma 10 nepriklausomų mazgų su genetiniu algoritmu, o gauti rezultatai būdavo išsaugomi GRID ftp serveryje. Kai visi nepriklausomi mazgai baigdavo darbą, startuodavo 10 kitų užduoties mazgų, kurie naudodavo pirmų mazgų rezultatus. Kiekvienam testiniam uždaviniui buvo įvykdyta po 10 tokių bandymų. Rezultatai pateikti 4 lentelėje.

4 lentelė, genetinio algoritmo rezultatai, kai populiacija sudaroma iš kitų mazgų rezultatų

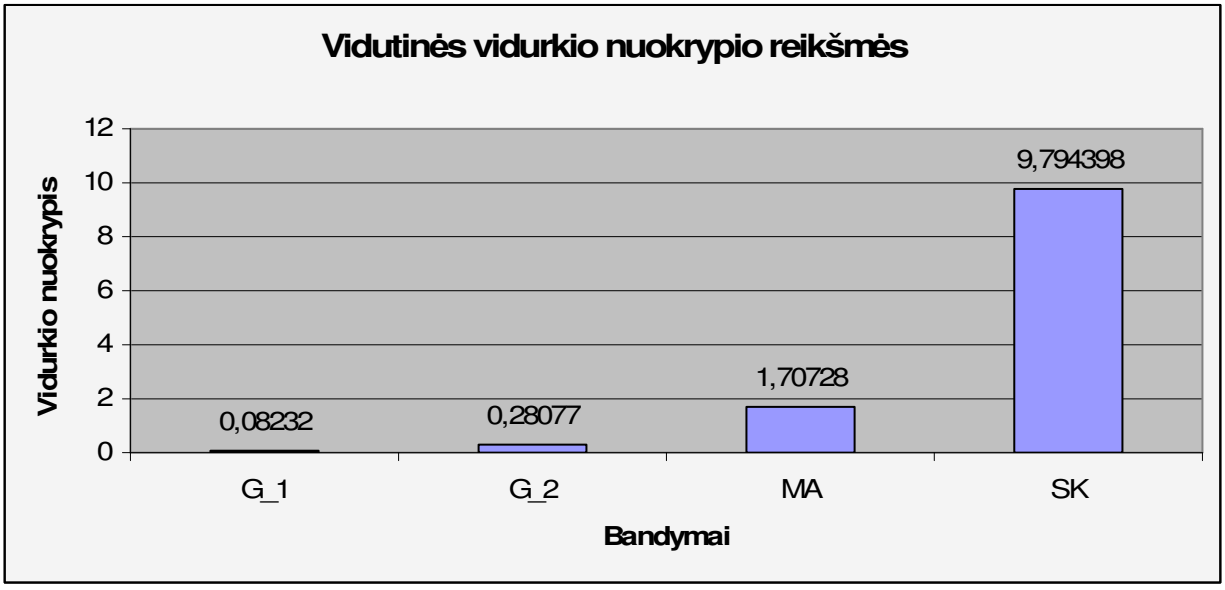
| Testinio uždavinio pavadinimas | $z_{opt}$ (km.) | $z_{best}$ (km.) | Populiacijos dydis | Max generacijų skaičius | $\bar{\delta}$ | $C_{opt}$ | $C_{1\%}$ | $t_{GRID}$ (sek.) |
|---|---|---|---|---|---|---|---|---|
| gr96 | 55209 | **55209** | 10 | 1000 | 0 | 10 | 10 | 0 |
| eil101 | 629 | **629** | 10 | 1000 | 0 | 10 | 10 | 0 |
| bier127 | 118282 | **118282** | 10 | 1000 | 0 | 10 | 10 | 0 |
| ch130 | 6110 | **6110** | 10 | 1000 | 0 | 10 | 10 | 0 |
| ch150 | 6528 | **6528** | 10 | 1000 | 0 | 10 | 10 | 0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| d198 | 15780 | **15780** | 10 | 1000 | 0 | 10 | 10 | 0 |
| tsp225 | 3916 | **3916** | 10 | 1000 | 0 | 10 | 10 | 0 |
| a280 | 2579 | **2579** | 10 | 1000 | 0 | 10 | 10 | 0 |
| lin318 | 42029 | **42029** | 10 | 1000 | 0 | 10 | 10 | 1 |
| fl417 | 11861 | **11861** | 10 | 1000 | 0 | 10 | 10 | 0 |
| pcb442 | 50778 | 50825 | 10 | 1000 | 0,09256 | 0 | 10 | 1 |
| d493 | 35002 | 35030 | 10 | 1000 | 0,08142 | 0 | 10 | 0 |
| ali535 | 202339 | **202339** | 10 | 1000 | 0,00177 | 9 | 10 | 1 |
| rat575 | 6773 | 6781 | 10 | 1000 | 0,22146 | 0 | 10 | 2 |
| d657 | 48912 | 49097 | 10 | 1000 | 0,31648 | 0 | 10 | 1 |
| gr666 | 294358 | **294553** | 10 | 1000 | 0 | 10 | 10 | 1 |
| d1291 | 50801 | 51013 | 10 | 1000 | 0,41849 | 0 | 10 | 14 |
| fl1400 | 20127 | 20135 | 10 | 1000 | 0,0472 | 0 | 10 | 5 |
| d1655 | 62128 | 62256 | 10 | 1000 | 0,39692 | 0 | 10 | 4 |
| d2103 | 80450 | 80498 | 10 | 1000 | 0,07022 | 0 | 10 | 5 |
| | | | | Vidurkis: | **0,08232** | | | |



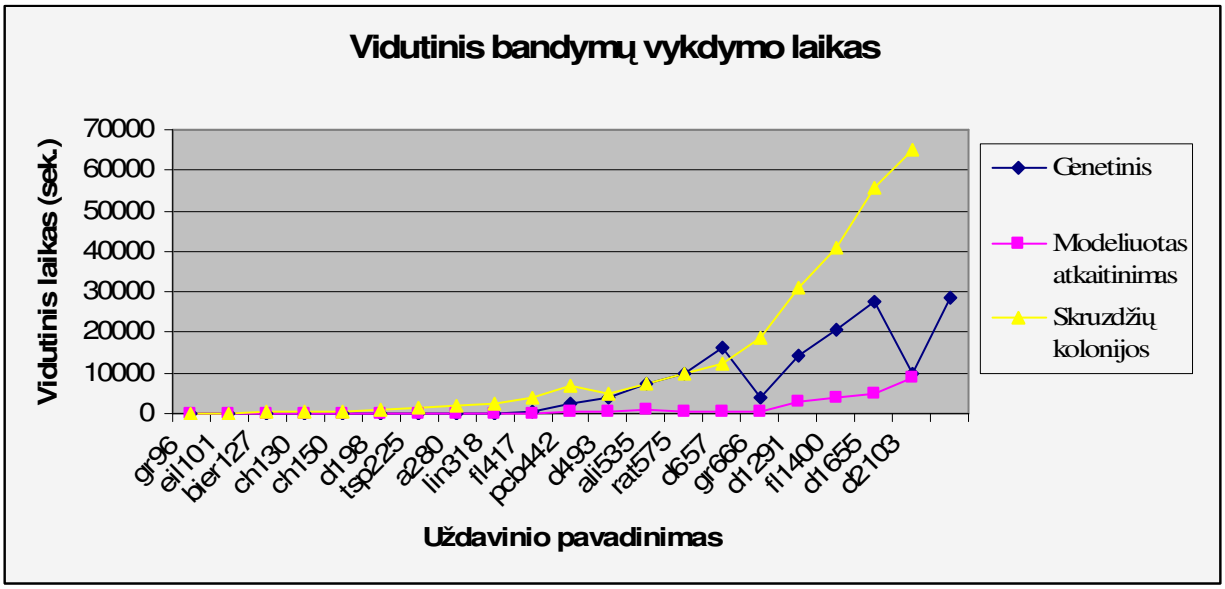

25 pav. Vidutinio santykinio tikslo funkcijos nuokrypio priklausomybė nuo miestų skaičiaus, genetinis algoritmas, kai pradinė populiacija sudaroma iš kitų bandymų rezultatų

Gauti rezultatai rodo, kad pradinę populiaciją sudarant iš kokybiškų sprendinių, gaunamas dar geresnis rezultatas. Šiuo bandymu 12 TSP bibliotekos pavyzdžių pavyko išspręsti, tai yra, buvo gautas geriausias galimas maršrutas. Visi bandymų rezultatai pateko į 1% optimalumo intervalą ($C_{1\%}$). Vidutinio santykinio tikslo funkcijos nuokrypio nuo žinomos optimalios reikšmės vidurkis – **0,08232**. Visų bandymų vidutinės vidurkio nuokrypio reikšmės palygintos 26 paveiksle.




26 pav. Vidutinės vidurkio nuokrypio reikšmės (G_1 – genetinis algoritmas, kai pradinė populiacija sudaroma iš kitų bandymų rezultatų, G_2 – paprastas genetinis algoritmas, MA – modeliuoto atkaitinimo algoritmas, SK – skruzdžių kolonijų algoritmas)

Paveiksle (27 pav.) „Vidutinis bandymų vykdymo laikas“ palyginti vidutiniai algoritmų vykdymo laikai. Matyti, kad skruzdžių kolonijų algoritmas ne tik parodė blogiausius rezultatus rastų maršrutų atžvilgiu, bet ir dirbo ilgiausiai. Efektyviausias laiko prasme buvo modeliuoto atkaitinimo algoritmas, ne daug nuo jo atsiliko genetinis algoritmas.




27 pav. Vidutinis bandymų vykdymo laikas

# 6 Kuprinės uždavinio eksperimentiniai bandymai

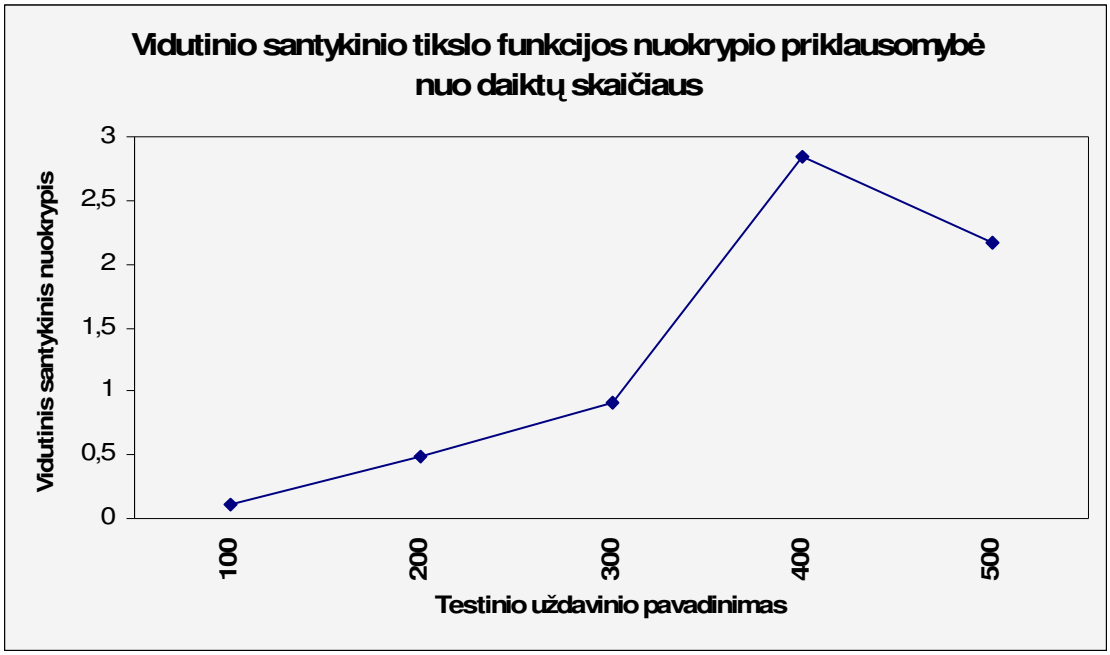
Kuprinės uždavinio testavimui GRID skaičiavimo tinkle, buvo naudojami atsitiktinai sugeneruoti duomenys, nes nebuvo rasta oficialios kuprinės uždavinio testinių duomenų bibliotekos. Atsitiktinai buvo sugeneruoti 5 skirtingų imčių testinių daiktų rinkiniai su skirtingais svoriais. Kiekvienam bandymui buvo sukurta viena „DAG“ tipo užduotis, kuri turėjo 10 nepriklausomų mazgų (angl. node). Visi užduoties mazgai buvo įvykdomi lygiagrečiai, skirtinguose GRID skaičiavimo tinklo mazguose. Efektyvumas buvo nustatinėjamas remiantis tokiais kriteriais:

- Vidutinis santykinis tikslo funkcijos nuokrypis nuo bandymų metu rastos geriausios reikšmės – $\bar{\delta}$ ($\bar{\delta}=100(z_{opt}-\bar{z})/z_{opt}$) [%], kur $\bar{z}$ yra bandymuose gautų tikslo funkcijų reikšmių aritmetinis vidurkis, gautas atlikus 10 bandymų, o $z_{opt}$ – bandymų metu rasta geriausia tikslo funkcijos reikšmė;
- Sprendimų, esančių „1% optimalumo intervale“ ($\bar{\delta} \le 1$), skaičius $C_{1\%}$;
- Bandymų skaičius $C_{opt}$, kai gauta tikslo funkcijos reikšmė nėra blogesnė nei geriausia bandymų metu rasta tikslo funkcijos reikšmė;
- Vidutinis užduoties vykdymo GRID skaičiavimo tinkle laikas $t_{GRID}$;
- Geriausio bandymo tikslo funkcijos reikšmė $z_{best}$.

5 lentelė, genetinio algoritmo taikymas kuprinės uždavimui

| Bandymo numeris | Daiktų skaičius | Bandymų metu rasta geriausia reikšmė | $C_{1\%}$ | $C_{opt}$ | $\bar{\delta}$ | $t_{GRID}$ |
|---|---|---|---|---|---|---|
| 1 | 100 | 5000 | 10 | 2 | 0,11200 | 8 |
| 2 | 200 | 8969 | 9 | 2 | 0,48535 | 46 |
| 3 | 300 | 12274 | 7 | 1 | 0,91412 | 79 |
| 4 | 400 | 15737 | 1 | 1 | 2,84743 | 129 |
| 5 | 500 | 19162 | 2 | 1 | 2,172529 | 210 |
| | | | | Vidurkis: | 1,306286 | |

Kuprinės uždavinys taip pat buvo sėkmingai sprendžiamas naudojant genetinį algoritmą ir jį vykdant GRIG skaičiavimo tinkle. Kadangi duomenys yra testiniai, sunku spręsti apie genetinio algoritmo efektyvumą. Kaip ir keliaujančio prekeivi uždaviniui, didėjant testinių uždavinių apimčiai buvo gauti prastesni rezultatai. Tai parodyta 28 paveikslėlyje.




28 pav. Vidutinio santykinio tikslo funkcijos nuokrypio priklausomybė nuo daiktų skaičiaus, genetinis algoritmas

# 7 GRID užduočių vykdymo efektyvumo tyrimas

Paleidus užduotį GRID skaičiavimo tinkle, užduotis įgyja keletą būsenų, kol yra įvykdoma. Kad nustatytume visą GRID užduoties vykdymo laiką skaičiavimo tinkle, pakartosime genetinio algoritmo bandymą su DAG tipo užduotimis, kurios turi po 10 nepriklausomų mazgų. Detali užduoties vykdymo ataskaita bus gaunama naudojant gLite komandą *glite-wms-job-status -i bandymoId -o rezultatų byla -v 3*. Bandymo metu bus atsižvelgta į tokius GRID užduoties vykdymo rezultatus:

- Laikas sekundėmis, kuris buvo reikalingas užduočiai iš būsenos „Submitted“ pereiti į būseną „Waiting“;
- Laikas sekundėmis, kuris buvo reikalingas užduočiai iš būsenos „Waiting“ pereiti į būseną „Running“;
- Laikas sekundėmis, kuris buvo reikalingas užduočiai iš būsenos „Running“ pereiti į būseną „Done“;
- Bendras GRID užduoties vykdymo laikas sekundėmis nuo būsenos „Submitted“ iki būsenos „Done“;
- Bendras visų mazgų algoritmų vykdymo laikas sekundėmis. Galima teigti, kad tai bus laikas, kuris būtų reikalingas pakartoti tą patį bandymą, leidžiant jį asmeniniame kompiuteryje.

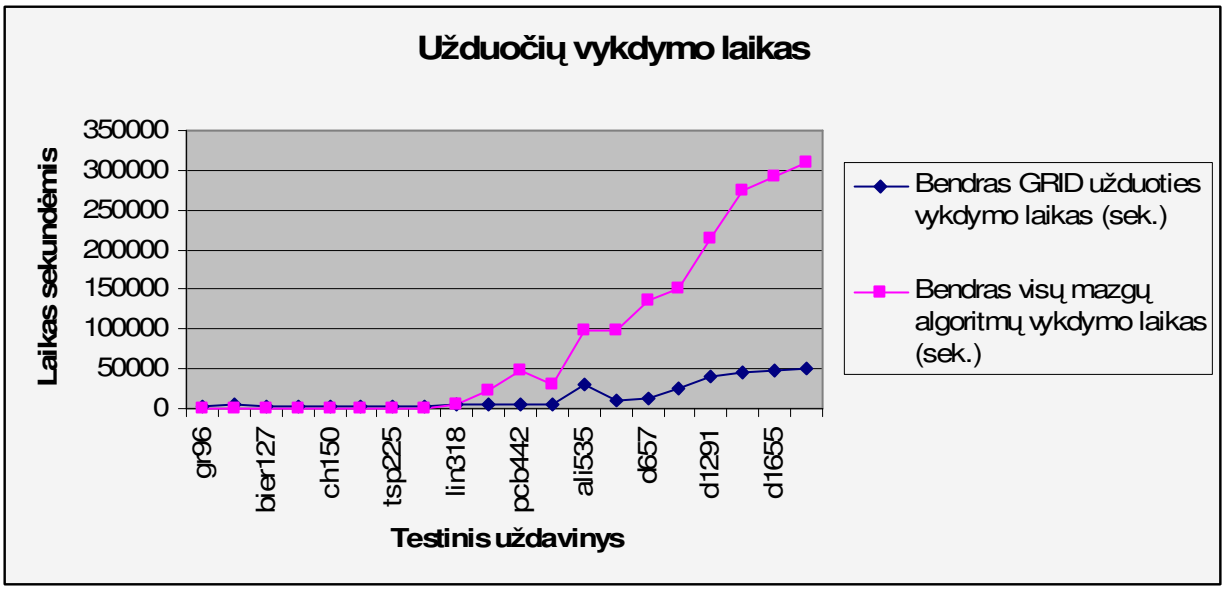
Bandymo rezultatai pateikti 6 lentelėje.

6 lentelė. GRID užduočių būsenos ir jų vykdymo laikas

| Testinio uždavinio pavadinimas | Iš „Submitted“ į „Waiting“ (sek.) | Iš „Waiting“ į „Running“ (sek.) | Iš „Running“ į „Done“ (sek.) | Bendras GRID užduoties vykdymo laikas (sek.) | Bendras visų mazgų algoritmų vykdymo laikas (sek.) |
|---|---|---|---|---|---|
| gr96 | 2 | 920 | 1216 | 2138 | 12 |
| eil101 | 2 | 1991 | 3425 | 5418 | 63 |
| bier127 | 2 | 821 | 941 | 1764 | 40 |
| ch130 | 2 | 944 | 914 | 1860 | 42 |
| ch150 | 2 | 992 | 938 | 1932 | 54 |
| d198 | 2 | 1140 | 1190 | 2332 | 219 |
| tsp225 | 3 | 996 | 1208 | 2207 | 432 |
| a280 | 2 | 675 | 1950 | 2627 | 630 |
| lin318 | 2 | 1326 | 3430 | 4758 | 4402 |
| fl417 | 2 | 2945 | 3115 | 6062 | 23012 |
| pcb442 | 2 | 896 | 3253 | 4151 | 46921 |
| d493 | 2 | 1639 | 4184 | 5825 | 30191 |

| | | | | | |
|---|---|---|---|---|---|
| ali535 | 2 | 112 | 29474 | 29588 | 97495 |
| rat575 | 4 | 884 | 10202 | 11090 | 98302 |
| d657 | 2 | 802 | 11294 | 12098 | 134953 |
| gr666 | 2 | 807 | 24945 | 25754 | 150420 |
| d1291 | 2 | 839 | 39120 | 39961 | 213758 |
| fl1400 | 2 | 932 | 45294 | 46228 | 275433 |
| d1655 | 2 | 1034 | 16043 | 47234 | 291203 |
| d2103 | 2 | 1246 | 47920 | 49168 | 310234 |

Atlikus bandymą matyti, kad ne visada tikslinga bandymus atlikinėti GRID skaičiavimo tinkle. Užduoties vykdymas, jos valdymas, rezultatų parsiuntimas ir t.t GRID skaičiavimo tinkle reikalauja papildomo laiko. Bet jei sprendžiama užduotis yra pakankamai didelės apimties, tada bandymų atlikimas GRID skaičiavimo tinkle tampa tikslingu. Bandymų rezultatai rodo (29 paveikslėlis), kad bandymą (10 pakartotinų algoritmo paleidimų) GRID skaičiavimo tinkle tikslinga vykdyti, kai miestų skaičius yra didesnis nei 417.




29 pav. GRID užduoties ir algoritmų vykdymo laikai

## Išvados

Šiame darbe buvo nagrinėjami euristiniai algoritmai, skirti spręsti NPC aibės uždavinius ir jų taikymas GRID skaičiavimo tinkle. Buvo tiriami trys skirtingi euristiniai algoritmai: genetinis, modeliuoto atkaitinimo ir skruzdžių kolonijų. Genetinis algoritmas buvo realizuotas keliaujančio prekeivio ir kuprinės uždaviniams, kurie priklauso NPC sudėtingumo aibei, spręsti. Modeliuoto atkaitinimo ir skruzdžių kolonijų algoritmai – tik keliaujančio prekeivio uždaviniui. Sėkmingai pritaikius tyrinėtus algoritmus GRID skaičiavimo tinklui jame buvo atlikti 109 eksperimentiniai bandymai, iš kurių 4 nepavyko, dėl GRID skaičiavimo tinkle įvykusių klaidų. Kiekvieno bandymo DAG tipo užduotis turėjo po 10 mazgų, kurie vykdė pasirinktus euristinius algoritmus, taigi viso buvo atlikta 1050 sėkmingų euristinių algoritmų bandymų. Atlikus bandymus, buvo padarytos tokios išvados:

- Atliekant tyrimus, efektyviausia naudoti „DAG“ tipo užduotis, nes:
    a. Vienu metu galima atlikinėti daug bandymų skirtinguose GRID skaičiavimo tinklo mazguose. Tai leidžia efektyviau išnaudoti laiką, kuris reikalingas paleisti visas bandymo užduotis;
    b. Galima nurodyti užduočių priklausomybes. Tokiu būdu norimus užduoties mazgų rezultatus galima naudoti vėliau vykdomuose užduoties mazguose. Tai buvo sėkmingai išbandyta su genetinio algoritmo taikymu TSP uždaviniui, kai jo pradinė populiacija buvo sudaroma iš anksčiau įvykdytų mazgų rezultatų. Tokio bandymo metu buvo gauti geriausi rezultatai.
- Pritaikius euristinius algoritmus GRID skaičiavimo tinklui, buvo išspręsta kompiuterinių resursų problema. Užduotys gali būti vykdomos ilgą laiko tarpą, nenaudojant tyrinėtojo kompiuterinių resursų. Todėl visiems tyrinėtojams, kurie turi prieigą prie GRID skaičiavimo tinklo, atsiveria galimybė atlikinėti didelės apimties bandymus.
- Bandymai rodo, kad užduotis vykdyti GRID skaičiavimo tinkle tikslinga, kai miestų skaičius didesnis nei 417 miestų. Tokios apimties užduoties bandymas asmeniniame kompiuteryje trunka apie 2000 sekundžių. Taip pat buvo pastebėta, kad GRID užduoties įvykdymo greitis priklauso nuo skaičiavimo mazgo apkrovimo. Skaičiavimo mazgų apkrovimą galima patikrinti įvykdant gLite komandą *lcg-infosites --vo litgrid ce*.
- Atlikus visus eksperimentinius bandymus matyti, kad:
    a. Efektyviausias iš tirtų euristinių algoritmų buvo genetinis algoritmas, rezultatai gauti taikant šį algoritmą buvo optimaliausi;

b. Modeliuoto atkaitinimo algoritmas dirbo greičiau nei genetinis algoritmas, bet gauti sprendiniai neprilygo genetinio algoritmo sprendiniams;
c. Prasčiausiai tiek laiko, tiek sprendinių kokybės prasme veikė skruzdžių kolonijų algoritmas;
d. Genetinio algoritmo rezultatus galima dar pagerinti, jei kelis kartus pakartotume procedūrą, kuri buvo taikyta 5.4 skyrelyje.

Reikia paminėti, kad užduotis vykdant GRID skaičiavimo tinkle iškyla ir tam tikrų problemų:

- GRID skaičiavimo tinklas neturi patogios vartotojo sąsajos. Visi bandymai buvo atlikti prisijungus prie MIF kompiuterinio tinklo ir naudojantis komandine eilute.
- Užduoties paleidimas, jos paruošimas ir rezultatų pasiėmimas tampa sudėtingesnis, kai užduotys vykdomos GRID skaičiavimo tinkle.
- Skaičiavimų mazgų programinė įranga ir jos versijos skiriasi, todėl skirtingi skaičiavimo mazgai, gali apteikti skirtingus rezultatus.

## Šaltinių sąrašas


[LKM+99] P. Larranaga , C.M.H. Kuijpers, R.H. Murga, I. Inza ir S. Dizdarevic. Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators, Kluwer Academic Publishers, Netherlands. 1999

[Don99] Dr. Z. Dong. Genetic Algorithm Applications, Sumio Kiyooka, 1999

[BB00] Kylie Bryant Arthur Benjamin. Genetic Algorithms and the Traveling Salesman Problem, 2000

[HH04] Randy L. Haupt Sue Ellen Haupt. PRACTICAL GENETIC ALGORITHMS SECOND EDITION, 2004

[Mis03a] Alfonsas Misevičius. Intelektualieji optimizavimo metodai, INFORMACIJOS MOKSLAI, 2003, pp. 166-172

[MBB+04] Alfonsas Misevičius, Tomas Blažauskas, Jonas Blonskis, Jonas Smolinskas. AN OVERVIEW OF SOME HEURISTIC ALGORITHMS FOR COMBINATORIAL OPTIMIZATION PROBLEMS, INFORMACINĖS TECHNOLOGIJOS IR VALDYMAS, 2004

[KGV83] S. Kirkpatrick; C. D. Gelatt; M. P. Vecchi. Optimization by Simulated Annealing, Science, New Series(220), 1983, pp. 671-680

[BL99] Bradley J. Buckham, Casey Lambert. Simulated Annealing Applications, Mechanical Engineering Department, University of Victoria, 1999

[HJJ02] Darrall Henderson, Sheldon H. Jacobson and Alan W. Johnson, The Theory and Practice of Simulated Annealing, Springer - Handbook of Metaheuristics, Kluwer, 2002, pp. 287-320

[Ree02] Colin Reeves, Genetic Algorithms, Springer - Handbook of Metaheuristics, Kluwer, 2002, pp. 55-83

[Lit06] LitGrid tikslai ir uždaviniai. URL:http://www.litgrid.lt/, 67 KB, 2006.11.16

[Lit08] GRID naudotojo pradmenys. URL:http://www.litgrid.lt/, 69 KB, 2008.03.02

[MBB06] Alfonsas Misevičius, Vytautas Bukšnaitis, Jonas Blonskis. Euristiniai algoritmai: tikslai, iššūkiai, metodologija, perspektyvos, INFORMACIJO MOKSLAI(39), 2006

[Pac07] Fabrizio Pacini, Job Description Language Attributes Specification, 2007

[Rei08] G.Reinelt. TSPLIB – A traveling salesman problem library, 2008. [žiūrėta 2009-05-04]. Prieiga per Internetą:

<http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>

[Mon09] Marco A. MONTES DE OCA. About Ant Colony optimisation, 2009. [žiūrėta 2009-05-04]. Prieiga per Internetą: <http://iridia.ulb.ac.be/~mdorigo/ACO/about.html >

[SY98] Hiroaki SENGOKU, Ikuo YOSHIHARA. A Fast TSP Solver Using GA on JAVA, 1998. [žiūrėta 2009-05-04]. Prieiga per Internetą: < http://www.gcd.org/sengoku/docs/arob98.pdf>

[DG97] M. Dorigo L.M. Gambardella. Ant colonies for the traveling salesman problem. BioSystems(43), 1997, pp.73-81

## Santrumpos

GRID – lygiagrečiųjų ir paskirstytųjų skaičiavimų

MA – modeliuotas atkaitinimas

MIF – matematikos ir informatikos fakultetas

TSPLIB- traveling salesman problem library

NPC – NP-complete

ACO – Ant Colony Optimization

TSP – Traveling Salesman Problem

JDL – Job Description Language

# Priedai

## Priedas A: Viena iš „DAG“ tipo užduočių, vykdytų GRID skaičiavimo tinkle

```
[
   type = "dag";
   MyProxyServer = "grid3.mif.vu.lt";
   InputSandbox = {
          "TSP.java", "GA.java", "Sortable.java", "Sort.java", "Gene.java", "City.java", "Cities.java", "SA.java",
"ACO.java", "Ant.java", "gr96.tsp", "sa01.sh"
   };
    Requirements = Member("VO-balticgrid-S-DEVEL-JRE-1.5.0_12",
other.GlueHostApplicationSoftwareRunTimeEnvironment);
   nodes = [
      node1 = [
         description = [
                              Executable = "/bin/sh";
                              Arguments = "sa01.sh";
                              StdOutput = "stdout1.log";
                              StdError = "stderr1.log";
                              InputSandbox = {
                                    root.InputSandbox
                              };
                              OutputSandbox = {"stdout1.log", "stderr1.log"};
                              OutputSandboxDestURI = {
                                    "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stdout1.log",
                                    "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stderr1.log"
                              };
               ];
      ];
          node2 = [
         description = [
                              Executable = "/bin/sh";
                              Arguments = "sa01.sh";
                              StdOutput = "stdout2.log";
                              StdError = "stderr2.log";
                              InputSandbox = {
                                    root.InputSandbox
                              };
                              OutputSandbox = {"stdout2.log", "stderr2.log"};
                              OutputSandboxDestURI = {
                                    "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stdout2.log",
                                    "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stderr2.log"
                              };
                     ];
      ];
          node3 = [
         description = [
                              Executable = "/bin/sh";
                              Arguments = "sa01.sh";
                              StdOutput = "stdout3.log";
                              StdError = "stderr3.log";
                              InputSandbox = {
                                    root.InputSandbox
                              };
                              OutputSandbox = {"stdout3.log", "stderr3.log"};
                              OutputSandboxDestURI = {
                                    "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stdout3.log",
```

```
                        "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stderr3.log"
                    };
            ];
];
    node4 = [
  description = [
                    Executable = "/bin/sh";
                    Arguments = "sa01.sh";
                    StdOutput = "stdout4.log";
                    StdError = "stderr4.log";
                    InputSandbox = {
                        root.InputSandbox
                    };
                    OutputSandbox = {"stdout4.log", "stderr4.log"};
                    OutputSandboxDestURI = {
                        "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stdout4.log",
                        "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stderr4.log"
                    };
            ];
];

    node5 = [
  description = [
                    Executable = "/bin/sh";
                    Arguments = "sa01.sh";
                    StdOutput = "stdout5.log";
                    StdError = "stderr5.log";
                    InputSandbox = {
                        root.InputSandbox
                    };
                    OutputSandbox = {"stdout5.log", "stderr5.log"};
                    OutputSandboxDestURI = {
                        "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stdout5.log",
                        "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stderr5.log"
                    };
            ];
];

    node6 = [
  description = [
                    Executable = "/bin/sh";
                    Arguments = "sa01.sh";
                    StdOutput = "stdout6.log";
                    StdError = "stderr6.log";
                    InputSandbox = {
                        root.InputSandbox
                    };
                    OutputSandbox = {"stdout6.log", "stderr6.log"};
                    OutputSandboxDestURI = {
                        "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stdout6.log",
                        "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stderr6.log"
                    };
            ];
];

    node7 = [
  description = [
                    Executable = "/bin/sh";
```

```
                        Arguments = "sa01.sh";
                        StdOutput = "stdout7.log";
                        StdError = "stderr7.log";
                        InputSandbox = {
                                root.InputSandbox
                        };
                        OutputSandbox = {"stdout7.log", "stderr7.log"};
                        OutputSandboxDestURI = {
                                "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stdout7.log",
                                "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stderr7.log"
                        };
                ];
];

    node8 = [
  description = [
                        Executable = "/bin/sh";
                        Arguments = "sa01.sh";
                        StdOutput = "stdout8.log";
                        StdError = "stderr8.log";
                        InputSandbox = {
                                root.InputSandbox
                        };
                        OutputSandbox = {"stdout8.log", "stderr8.log"};
                        OutputSandboxDestURI = {
                                "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stdout8.log",
                                "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stderr8.log"
                        };
                ];
];

    node9 = [
  description = [
                        Executable = "/bin/sh";
                        Arguments = "sa01.sh";
                        StdOutput = "stdout9.log";
                        StdError = "stderr9.log";
                        InputSandbox = {
                                root.InputSandbox
                        };
                        OutputSandbox = {"stdout9.log", "stderr9.log"};
                        OutputSandboxDestURI = {
                                "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stdout9.log",
                                "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stderr9.log"
                        };
                ];
];

    node10 = [
  description = [
                        Executable = "/bin/sh";
                        Arguments = "sa01.sh";
                        StdOutput = "stdout10.log";
                        StdError = "stderr10.log";
                        InputSandbox = {
                                root.InputSandbox
                        };
                        OutputSandbox = {"stdout10.log", "stderr10.log"};
```

```
                        OutputSandboxDestURI = {
                                "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stdout10.log",
                                "gsiftp://grid8.mif.vu.lt/tmp/irve3993/sa01/stderr10.log"
                        };
                ];
        ];
    ];
    dependencies = { }
];
```

## Priedas B: TSP uždavinio programos sukompiliavimas ir paleidimas GRID skaičiavimo tinkle

```
echo "*** Uzduoties pradzia. ***"
javac TSP.java GA.java Sortable.java Sort.java Gene.java City.java Cities.java SA.java ACO.java Ant.java
echo "*** Sukompiliuota. Paleidziama... ***"
START=$(date +%s)
java TSP -v -p 0.99 gr96.tsp 1000 sa
END=$(date +%s)
DIFF=$(( $END - $START ))
echo "Vykdymo laikas $DIFF sekundes"
echo "*** Uzduoties pabaiga. ***"
exit 0
```

## Priedas C: Užduoties rezultatų byla, gražinta iš GRID skaičiavimo tinklo

*** Uzduoties pradzia. ***
*** Sukompiliuota. Paleidziama... ***
0 199 197 3 196 194 45 193 217 192 44 47 195 191 223 198 132 190 204 189 206 48 50 49 51 52 53 54 55 56 57 58 1 46 224 188 26 187 116 186 117 185 118 184 119 174 120 121 183 181 180 179 178 177 176 175 173 172 171 170 169 122 123 168 167 149 150 151 148 147 146 145 152 153 154 155 156 143 144 142 200 141 140 139 138 137 158 159 160 161 162 136 135 182 134 157 212 133 214 163 164 165 166 213 211 124 125 126 127 128 131 221 129 210 130 85 84 81 82 83 209 86 89 90 91 92 95 94 208 93 220 80 79 78 77 96 97 98 99 100 101 103 219 102 88 87 104 105 106 107 108 109 110 111 112 113 114 222 115 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 215 218 216 76 27 203 28 29 201 205 30 31 34 32 33 25 24 207 35 23 22 21 20 19 202 18 17 16 15 14 13 12 11 10 36 37 38 9 8 39 40 41 43 42 7 6 5 4 2
Distance: 4012.0
Vykdymo_laikas: 44
*** Uzduoties pabaiga. ***