KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

Tomas Armoška

GAMYBINIŲ TVARKARAŠČIŲ SUDARYMO OPTIMUMO KRITERIJŲ TYRIMAS

Magistro darbas

Vadovas
doc. dr. N. Listopadskis

KAUNAS, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. N. Listopadskis
2011 06 02

GAMYBINIŲ TVARKARAŠČIŲ SUDARYMO OPTIMUMO KRITERIJŲ TYRIMAS

Taikomosios matematikos magistro baigiamasis darbas

Vadovas
doc.dr.N.Listopadskis
2010 06 01

Recenzentas

2010 06 01

Atliko
FMMM 5/2 gr. stud.
T.Armoška
2010 05 30

KAUNAS, 2011

KVALIFIKACINĖ KOMISIJA

Pirmininkas: **Leonas Saulis, profesorius (VGTU)**

Sekretorius: **Eimutis Valakevičius, docentas (KTU)**

Nariai: **Algimantas Jonas Aksomaitis, profesorius (KTU)**

Vytautas Janilionis, docentas (KTU)

Vidmantas Povilas Pekarskas, profesorius (KTU)

Rimantas Rudzkis, habil. dr., vyriausiasis analitikas (DnB NORD Bankas)

Zenonas Navickas, profesorius (KTU)

Arūnas Barauskas, dr., vice-prezidentas projektams (UAB „Baltic Amadeus“)

Armoška T. Shop scheduling analyses by optimality criteria / supervisor doc.dr.Narimantas Listopadskis; Department of Applied mathematics, Faculty of Fundamental Sciences, Kaunas University of Technology. – Kaunas, 2011. –p.

SUMMARY

The purpose of this work is to analyze the scheduling problem as well as optimality criteria's which is used in it. To realize it in program and do the comparable analyze for optimality criteria.

In this work it was used *Simulated Annealing, Tabu-search and Genetic algorithm* methods. Also, it was used these three optimality criteria's: *Makespan, Maximum Lateness* and *Total weighted completion time.*

To realize it in programmable way, I used C++ builder program. I chosen this program because of its advantages: variety of its solving problems, very good interface, ability to write arithmetic and logical functions and so on.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLĖLIŲ SĄRAŠAS

ĮVADAS

Šiuolaikinėse gamyklose veikia daug įvairiu gamybiniu mašinų, su sudėtingais nustatymais, kiekvienam gamybos žingsniui reikia daug įvairių priemonių ir mašinų. Gamybos koordinatoriai privalo rasti būdus, kaip naudojant turimus resursus pasiekti efektyviausius rezultatus. Iš čia ir kilo mokslo šaka – gamybiniu tvarkaraščiu sudarymas.

Gamybiniu tvarkaraščių sudarymo problema priskiriama NP-sunkumo klasei sudėtingumo teorijoje (Nedeterminuotas polinomas).

Standartinėje tvarkaraščių sudarymo uždavinyje yra *n* darbų su skirtingais procesais turi būti atliktas vienoje mašinoje arba *m* paralelinių mašinų. Darbai atsiranda tam tikrais laiko momentais, ir jie turi būti priskirti prie tam tikros mašinos ir būti atliekamas toje mašinoje iki atlikimo. Visos mašinos gali dirbti tik su vienu darbu vienu metu.

Šiame darbe nagrinėjami ***nuoseklaus fabriko*** ($Fm \mid prmu \mid C_{max}$) tvarkaraščių sudarymo uždaviniai bei juose naudojamų optimalumo kriterijų variantai. Programiškai realizuojami gamybinių tvarkaraščių sudarymo algoritmai ir atliekama optimalumo kriterijų lyginamoji analizė. Konkrečiai naudojau tokius kriterijus: atlikimo laikas, didžiausio vėlavimo laikas ir bendras svorinis atlikimo laikas.

Pasirinkau C++Builder programą. Tokį pasirinkimą nulėmė daugybė C++Builder sprendžiamų problemų, puikus grafinis rezultatų pateikimas, galimybė užrašyti aritmetines ir logines išraiškas bei komentarus, kurie padeda geriau suprasti panaudotas duomenų struktūras ar atliekamus veiksmus, patogi vartotojo aplinka ir geras C++Builder aprašymas. Taip pat labai svarbu skaičiavimų tikslumas ir greitis.

1. BENDROJI DALIS

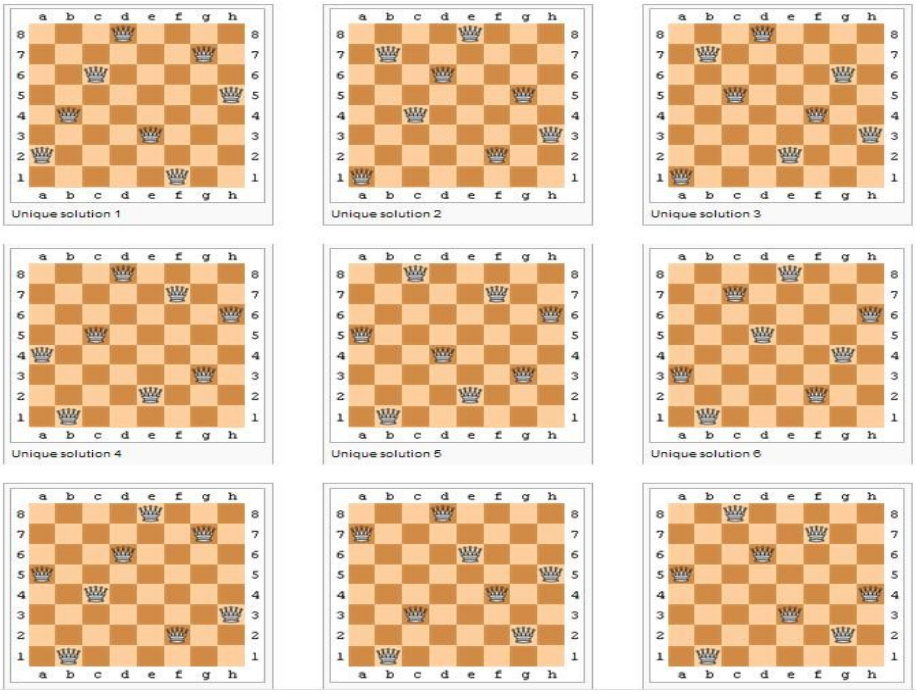
1.1 KOMBINATORINIO OPTIMIZAVIMO UŽDAVINIAI

Kombinatorinė optimizacija yra optimizacijos šaka. Jos pagrindas yra optimizacijos problemos, kurių realiųjų sprendinių grupė yra diskreti arba galima suprastinti į diskrečiąją, o tikslas yra surasti našiausia sprendimą. Keletas kombinatorinės optimizacijos pavyzdžių:

Mašinų maršrutų problema (Vehicle routing problem). Mašinų maršrutų problema (MMP) – tai siekimas patenkinti kažkokį kieki klientų su tam tikru kiekiu transporto priemonių. Problema buvo pasiūlyta 1959 metais Dantzigo ir Ramserio, ir išliko aktuali transporto, paskirstymo ir logistikos srityse. Dažniausiai pagrindinis uždavinio formulavimas būna tai, kad reikia išvežioti kažkokius resursus pas klientus, kurie yra davę užsakymus, iš pagrindinio sandėlio, ir padaryti tai sunaudojant mažiausiai išteklių. Plačiau ši problema panagrinėta (žr. [3] nuorodą).

Atsargų pjaustymo problema (Cutting stock problem). Ši problema kyla daugelyje pramonės industrijos šakų. Ši problema sprendžia, kaip iš kažkokio kiekio medžiagų, pagaminti kitokios formos medžiagas, našiausiai. Klasikinis atvejis – iš stačiakampio popieriaus lapo reikia išpjaustyti kokias nors figūras.

Aštuonių karalienių problema(Eight queens puzzle). Tai problema, kur reikia aštuonias šachmatų karalienes sudėlioti ant standartinės šachmatų lentos (8x8) taip, kad jos viena kitos nekirstu standartiniais šachmatų ėjimais. Ši problema yra kaip pavyzdys labiau apibendrytos problemos **n karalienių problema,** kur reikia n karalienių išdėstyti $n^x n$ lentoje, o sprendimas egzistuoja tik kai n=1, arba n≥4. Taip pat iš šios problemos išplaukia ir panašiu problemų su skirtingom figūrom.



1.1 pav. keletas karalienių sprendimo būdų.

Yra ir daugybė kitų problemų, kuriu plačiau neaptarinėsiu:

- **Keliaujančio prekybininko problema**
- **Mažiausio besisukančio medžio problema**
- **Kuprines problema**

Ir daugelis kitų.

1.3 skyrelyje aptarsiu savo magistriniame darbe nagrinėjama kombinatorinės optimizacijos problemą – tvarkaraščių sudarymo problemą.

1.2 SUDĖTINGUMO TEORIJA

Sudėtingumo teorija – tai kompiuterių mokslo šaka, kuri nagrinėja problemas, susijusias su resursais reikalingais, kad algoritmai veiktų, bei sprendžia iškilusius sunkumus, susijusius su jų trūkumais. Sudėtingumo teorijoje egzistuoja *sudėtingumo klasės* – grupė problemų susijusių savo sudėtingumo teorijos sprendimo būdu. Dažniausiai sudėtingumo klasė turi tokia aprašymo formą:

Grupė problemų galima išspręsti su *M* abstrakčiu mašinų naudojant *O(f(n))* kiekį resurso *R* (n – įvestas dydis). Toliau apžvelgsiu keletą sudėtingumo teorijos klasių.

Klasė P

Sudėtingumo teorijoje, ***P***, taip žinoma kaip ***PTIME*** *ar* ***DTIME***, yra viena iš fundamentaliųjų sudėtingumo klasių. Tai klasė visų problemų, kurių sprendimo laikas nustatomas tos problemos dydžio, pagal *determinuotą* algoritmą, polinominę funkcija. ***P*** reiškia Poliniminis laikas.

Klasė NP

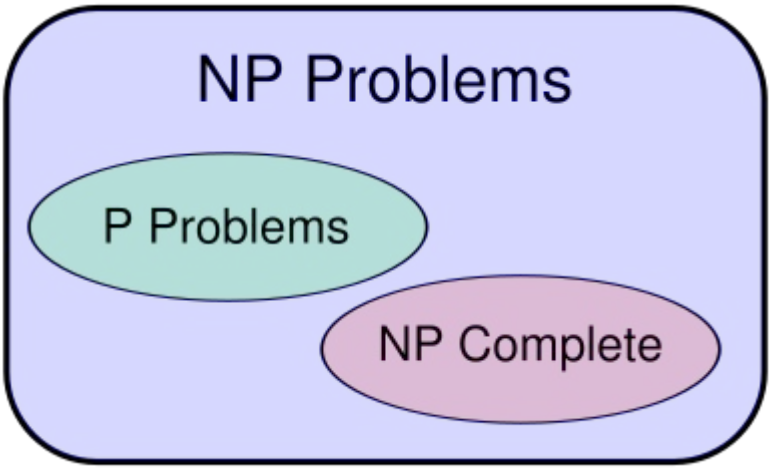
Sudėtingumo klasė ***NP***, tai klasė kurios visų problemų, įrodomos polinominiame laike arba sprendžiama polinominiame laike pagal *nedeterminuota* algoritmą. ***NP*** reiškia Nedeterminuotas Polinominis laikas. Jeigu yra problemos sprendimas, mes galime patikrinti jos teisingumą ar neteisingumą polinominiame laike. Kitas būdas pažvelgti į tai: problema yra ***NP*** klasėje, kai žingsneliu skaičius, reikalingas išspręsti šiai problemai, yra apibrėžtas kokia nors riba, susijusia su problemos dydžiu, bet taip pat egzistuoja galimybė atspėti sprendinį daugiau nei vienu variantu. Visos klasės ***P*** problemos priklauso klasei ***NP***, bet ne atvirkščiai. Kartais problemų neįmanoma išspręsti kitaip nei jos ieškant panaudojus „jėga“. Nėra žinoma ar šios problemos turi apibrėžtus sprendinius polinominiame laike.

Klasė NP-COPLETE

Sudėtingumo klasė ***NP*-COPLETE**, tai klasė problemų, kurios turi dvi savybes:

- bet koks duotas problemos sprendimas gali būti patikrintas palyginti greitai (polinominiame laike); problemų grupė su šia savybe yra **NP** klasė.

- jeigu problema gali būti išspręsta palyginti greitai(polinominiame laike), tai tada galima ir kiekviena **NP** problemą.



1.2 pav. Sudėtingumo teorijos klasės.

Yra manoma, kad klasė ***P*** ir klasė ***NP*** nėra lygios. Paprastai tariant, klasės ***P*** problemos yra greitai išsprendžiamos, o klasės ***NP*** problemų sprendiniai yra greitai patikrinami. Bet galiausiai, mokslininkai negali įrodyti nei šių klasių lygumo, nei nelygumo.

1.3 TVARKARŠČIŲ SUDARYMO PROBLEMA

Standartinėje tvarkaraščių sudarymo uždavinyje yra *n* darbų su skirtingais apdirbimo laikais turi būti atliktas vienoje mašinoje arba *m* paralelinių mašinų. Darbai atsiranda tam tikrais laiko momentais, ir jie turi būti priskirti prie tam tikros mašinos ir būti atliekamas toje mašinoje iki atlikimo. Visos mašinos gali dirbti tik su vienu darbu vienu metu.

Įvesties duomenys yra darbų aibė $\sigma = \{J_1, J_2, \ldots J_n\}$. Kiekvienas darbas J_i atsiranda sistemoje r_i laiku ir turi „svorį“ w_i, kuris parodo darbo svarbumą lyginant su kitais darbais (pvz., šis dydis gali reikšti darbo išlaikymo sistemoje kainą). Darbo baigimo laikas žymimas C_i. Išvesties informacija yra tvarkaraštis, kuris nurodo kiekvieno darbui kada ir kurioje mašinoje jis turi būti pradėtas atlikinėti.

Tvarkaraščių sudarymo uždavinyje problema apsirašo sistema $\alpha \mid \beta \mid \gamma$. α – aprašo mašinų įvairovę ir turi tik vieną įrašą. β laukas aprašo sistemos savybes ir charakteristikas ir gali neturėti nei

vieno įrašo, vieną įrašą arba daugybę įrašų. γ aprašo tikslą kurį reikia minimizuoti ir dažniausiai turi vieną įrašą.

Problemos įvairovė. Mes galime keisti algoritmą, kad jis suteiktu darbam pirmumo teisę (kiti darbai praleistų eilėje), sustabdyti darbo vykdymą ir pratęsti jį vėliau, kad ir kitoje mašinoje. Taip pat galima įtraukti mašinų santykį, kuriam nurodytas kiekvienos mašinos darbo atlikimo laikas. Netgi galime pridėti nustatymą, kad būtų įmanoma keisti mašinos darbo atlikimo greitį bet kuriuo momentu, tačiau didėjant greičiui, didėja ir reikalinga galia mašinai. Ryšys tarp mašinos greičio ir galios priklauso nuo turimos mašinos, bet daugeliu atveju tai yra sąryšis s^{α}, kur kažkoks dydis $\alpha > 1$.

Tam tikrose sistemose darbai nėra nepriklausomi vienas nuo kito, o tam tikri darbai gali būti pradėti tik po kažkurių kitų darbu pabaigimo.

Optimalumo kriterijus. Optimalumo kriterijų yra daug. Optimalumo kriterijus – tai kriterijus pagal kurį yra sudarinėjamas tvarkaraštis.

Dažniausiai naudojamas kriterijus yra didžiausias galimas baigimo laiko $max\ c_i$ (makespan), laikas kuriuo turi būti užbaigtas paskutinis darbas. Dar tai vadinama *atlikimo laiku.* Atveju, kai darbai atsiranda kas kažkiek tai laiko, maksimalaus atlikimo laiko minimizavimo problema tampa ekvivalenti maksimalaus mašinos apkrovimo minimizavimui: apkrovimo balansavimas. Aš savo bakalauriniame darbe dar naudojau maksimalaus vėlavimo kriterijų, bei bendrą svorinį atlikimo laiką.

Maksimalus vėlavimas (L_{max}, Maximum Lateness) – tai didžiausias iš vėlavimu tarp visų darbų, kuris nustatomas:

$$L_j = C_j - d_j \tag{1.1}$$

Kur C_j – darbo *j* užbaigimo laikas, d_j – darbo *j* planuojamas užbaigimo laikas.

Bendras svorinis atlikimo laikas (∑ wj*Cj, Total weight completion time) – suma visų darbo atlikimo laiko padauginti iš to darbo svorio.

1.4 NUOSEKLUS FABRIKAS

Tarkime iš eilės stovi *m* mašinų. Kiekvienas darbas turi būti atliktas kiekvienoje iš *m* mašinų. Kiekvienas darbas per mašinas turi praeiti tokia pačia tvarka, t.y pirmiausia turi pradėti mašinoje *1*, tada mašinoje *2*, ir taip toliau. Kai darbas atliekamas vienoje mašinoje, tai jis atsiduria eilėje prie kitos mašinos. Dažniausiai visos eilė operuoja pagal *Pirmas Įeina Pirmas Išeina* (FIFO) maršrutą, t.y darbas negali aplenkti kito darbo kol laukia eilėje. Jeigu galioja FIFO maršrutas, tai darbo fabrikas vadinsis permutaciniu darbo fabriku, o jo β kriterijus turės įrašą *prmu.*

Nuoseklus fabrikai su begaline tarpine sandėliavimo vieta. Kaip realu pavyzdį galėčiau pateikti kokiu nors labai mažų objektu gaminimą (pvz sagų), nors ir realiai sandėliavo vieta yra ribota, bet ji neskaičiuojama. Duotas permutacinis tvarkaraštis $j_1, j_2,... j_n$ mašinai *m* nuosekliame fabrike. Darbo j_k baigimo laikas mašinoje *i* gali būti apskaičiuotas šiomis formulėmis:

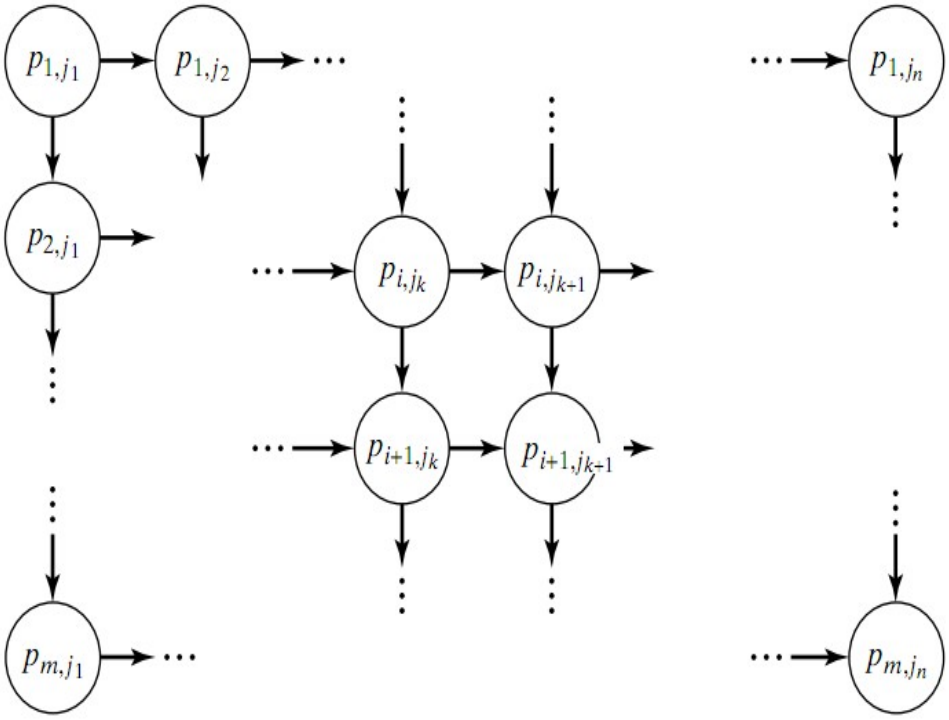
$$C_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1} \qquad i = 1,..,m \qquad \textbf{(1.2)}$$

$$C_{1,j_k} = \sum_{l=1}^{k} p_{1,j_l} \qquad k = 1,..,n \qquad \textbf{(1.3)}$$

$$C_{i,j_k} = \max\left(C_{i-1,}\ C_{i,j_{k-1}}\right) + p_{i,k_k} \qquad i = 2,..,m;\ k = 2,..,n \qquad \textbf{(1.4)}$$

Kur $p_{i,j}$ – apdirbimo laikas darbo *i* ant mašinos *j*. $C_{i,jk}$ – darbo j_k baigimo laikas ant mašinos *i*.

Taip pat darbo baigimo laiko (makespan) reikšmė gali būti nustatyta ir pagal *kritini kelią*, iš tiesioginio grafo(direct graph), kuris atitinka tvarkaraštį. Duotai sekai $j_1, ..., j_n$ šis tiesioginis grafas sudarytas taip: kiekvienai operacijai, sakykim darbo j_k mašinoje *i*, yra mazgas (i, j_k), turintis svorį lygu apdirbimo laikui darbo j_1 mašinoje *i.* Mazgas (i, j_k), $i = 1,...,m-1$ ir $k\ 1,...,n-1$, turi išeinančius lankus į mazgus $(i+1, j_k)$ ir (i, j_{k+1}).Mazgai, atitinkantys mašiną *m* turi tik viena išeinantį lanką, kaip ir mazgai atitinkantys darbą j_n. Mazgas (m, j_n) neturi išeinančių lankų. Bendras svoris iš didžiausio svorio kelio iš mazgo $(1, j_1)$ į mazgą (m, j_n) atitnka darbo baigimo laiką permutaciniame tvarkaraštį $j_1, ..., j_n$.



1.3 pav. Tiesioginis grafas nuoseklaus fabriko tvarkaraščiui.

Gaunamas įdomus rezultatas, lyginant du *m* mašinų permutacinius nuoseklaus fabriko tvarkaraščius su *n* darbų. Tegul $p_{ij}^{(1)}$ ir $p_{ij}^{(2)}$ žymi apdirbimo laiką darbo *j* mašinoje *i* atitinkamai pirmame ir antrame nuoseklaus fabriko tvarkaraščiuose. Tarkime:

$$p_{ij}^{(1)} = p_{i+1-i,j}^{(2)}$$

Paprasčiau tariant, tai reiškia, kad pirma mašina pirmame nuoseklaus fabriko tvarkaraštyje atitinka paskutinę mašiną antrame nuoseklaus fabriko tvarkaraštyje; antra mašina pirmame nuoseklaus fabriko tvarkaraštyje atitinka priešpaskutinę antrame nuoseklaus fabriko tvarkaraštyje, ir taip toliau.

Toliau einanti Lema tinka šiems dviem nuoseklaus fabriko tvarkaraščiams:

1 Lema: Darbų $j_1, ..., j_n$ išsidėstymas pirmame nuoseklaus fabriko tvarkaraštyje yra lygiai toks pat kaip darbų $j_n, ..., j_1$ išsidėstymas antrame nuoseklaus fabriko tvarkaraštyje.

Įrodymas. Jeigu pirmas nuoseklaus fabriko tvarkaraštis atsispindi seka $j_1, ..., j_n$ tiesioginiame grafe (1,1 pav), tai antras nuoseklaus fabriko tvarkaraštis seka $j_n, ..., j_1$ atsispindi tame pačiame tiesioginiame grafe, tik visi lankai rodo atvirkščia tvarka. Didžiausias svoris iš visų galimų nepasikeičia, nes einant ta pačia seka iš kitos pusės svorio skirtumo nėra.

Ši lema patvirtina *atgręžimo* taisykle(*reversibility)*: darbo baigimo laikas nepasikeičia jei darbai nuosekliame fabrike apsikeičia vietomis ir atliekami atvirkščia seka.

Panagrinėsime nuoseklaus fabriko tvarkaraščius su dviem mašinom ($m = 2$) ir su begaline tarpine sandėliavimo vieta, pagal atlikimo laiko optimumo kriterijų ($F2||C_{max}$). Yra *n* darbų, ir darbo *j*

apdirbimo laikas žymėsime p_{1j} pirmoje mašinoje, ir p_{2j} antroje mašinoje. Tai viena pirmųjų problemų kuri buvo nagrinėjama Operacijų teorijoje ir išsivystę į klasikinius tyrimus tvarkaraščių teorijoje, kuriems vadovavo S.M Jonsonas (S.M Johnson). Taisyklė, pagal kurią minimizuojamas atlikimo laiko kriterijus dažnai vadinama Jonsono taisyklė.

Optimali tvarkaraščio darbų seka gali būti aprašyta šitaip: padalinkime darbus į du rinkinius, kur pirmam rinkiny būtų visi darbai $p_{1j} > p_{2j,}$ o antrame rinkinyje būtų visi darbai $p_{1j} < p_{2j}$. Darbai kur $p_{2j} = p_{2j}$ gali būti priskirti bet kuriam rinkiniui. Darbai pirmame rinkinyje eina pirmiausiai, ir sudėliojami didėjančia tvarka pagal p_{1j} (SPT);darbai antrame rinkinyje seka mažėjančia tvarka pagal p_{2j} (LPT). Ryšiai gali būti pakeisti savo nuožiūra. Šitoks tvarkaraštis vadinamas **SPT(1)-LPT(2)** tvarkaraščiu. Žinoma, daugelis tvarkaraščių gali būti šitaip generuojami.

1 Teorema.

SPT(1)-LPT(2) tvarkaraštis yra optimalus F2||C_{max} tvarkaraščiui.

Įrodymas. Įrodysime prieštaros metodu. Tarkim kad egzistuoja kitos rūšies tvarkaraštis kuris yra optimalus. Tokiame tvarkaraštyje turi egzistuoti tokia gretimų darbų pora, tarkim darbo *j*, kurį seka darbas *k*, kuriuos tenkina viena iš šių sąlygų:

(1) Darbas *j* priklauso antram rinkiniui ir darbas *k* priklauso pirmam rinkiniui;

(2) Darbas *j* ir *k* priklauso pirmam rinkiniui ir $p_{1j} > p_{1k}$.

(3) Darbas *j* ir *k* priklauso antram rinkiniui ir $p_{1j} > p_{2k}$.

Dabar tereikia įrodyti, kad bet kuriuo iš šių atveju atlikimo laikas sumažėja apkeitus *j* ir *k* darbus vietomis. Tarkime kad pradiniame tvarkaraštyje(prieš sukeitimą vietomis) darbas *l* atitinka tvarkaraščio darbą einantį prieš darbą *j*, ir atitinkamai darbas *m* atitinka darbą einantį po darbo *k*. Tarkim C_{ij} atitinka darbo *j* mašinoje *i* baigimo laiką pradiniame tvarkaraštyje, o C'_{ij} apibrėžia darbo *j* mašinoje *i* baigimo laiką tvarkaraštyje, kai darbų sukeitimas vietomis jau atliktas. Darbu *j* ir *k* akivaizdžiai neįtakoja darbo *m* paleidimo laiko pirmoje mašinoje, nes *m* darbo paleidimo laikas lygus $C_{1l}+ p_{1j}+ p_{1k}$. Tačiau mus domina sužinoti, kada antroji mašina pasidaro prieinama darbui *m*. Pradiniame tvarkaraštyje tai bus laikas, kai bus užbaigtas darbas *k* antroje mašinoje, t.y C_{2k} ,o po pakeitimo darbu *j* ir *k* , tai būtu darbo *j* baigimo laikas antroje mašinoje, t.y C'_{2j}.

Dabar užteks parodyti kad $C_{2k} \geq C'_{2j}$ prie bent iš vienos iš prieš tai pateiktų sąlygų.

Baigimo laikas darbo *k* mašinoje 2 pradiniame tvarkaraštyje (prieš sukeitimą vietomis) yra toks:

$$C_{2k} = \max (\max (C_{2l} , C_{1l} + p_{1j})+ p_{2j} , C_{1l} + p_{1j} + p_{1k}) + p_{2k} =$$
$$= \max (C_{2l} + p_{2j} + p_{2k} , C_{1l} + p_{2j} + p_{1k} + p_{2k} , C_{1l} + p_{1j} + p_{1k} + p_{2k}),$$

o baigimo laikas darbo *j* mašinoje 2 po sukeitimo darbu *j* ir *k* vietomis yra toks:

$C'_{2j} = \max (C_{2l} + p_{2j} + p_{2k} , C_{1l} + p_{2j} + p_{1k} + p_{2k} , C_{1l} + p_{1j} + p_{1k} + p_{2j})$.

Pagal (1) sąlygą $p_{1j} > p_{2j}$ ir $p_{1k} < p_{2k}$. Akivaizdu, kad pirmosios sąlygos maksimumo C_{2k} ir C'_{2j} išraiškoje yra vienodos. Antroji sąlyga sąlygos C'_{2j} maksimumo išraiškoje yra mažesnė nei trečioji sąlyga C_{2k} išraiškoje, bei trečioji sąlyga sąlygos C'_{2j} maksimumo išraiškoje yra mažesnė nei antroji sąlyga C_{2k} išraiškoje.

Tai pagal (1) sąlyga $C_{2k} \geq C'_{2j.}$

Pagal (2) sąlyga- $p_{1j} < p_{2j}$, $p_{1j} < p_{2k}$ ir $p_{1k} < p_{2k}$.Taigi dabar antroji ir trečioji sąlyga iš C'_{2j} maksimumo išraiškos yra mažesnė nei antroji ir atitinkamai trečioji sąlyga C_{2k} maksimumo išraiškoje.

Taigi taip pat ir pagal (2) sąlyga $C_{2k} \geq C'_{2j.}$

Sąlyga (3) įsirodo taip pat kaip (2), tik panaudojus *atgręžimo* lema.

Įrodymas baigtas.

Tokie SPT(1)-LPT(2) tvarkaraščiai yra nevieninteliai optimalus tvarkaraščiai $F2||C_{max}$ tvarkaraščiams. Optimalių tvarkaraščių klasė yra sunkiai apibūdinama ir labai priklausoma nuo duomenų.

Nelaimei, SPT(1)-LPT(2) tvarkaraščio struktūra negalime sugeneruoti optimalaus tvarkaraščio nuoseklaus fabriko tvarkaraščiams su daugiau nei dviem mašinoms. Bet sumažinti atlikimo laiką permutacinaime nuoseklaus fabriko tvarkaraštyje su tam tikru kiekiu mašinų gali būti suformuluota kaip Sumaišytu programų visuma (Mixed Integer Program – MIP).

Pateiksiu dar vieną teorema be įrodymo, kuri labai svarbi.

2 teorema. Permutacinio nuoseklaus fabriko tvarkaraščiui, kur darbo *j* apdirbimo laikas kiekvienoje mašinoje yra vienodas ir lygus p_j, t.y $p_{1j} = p_{2j} = p_{3j} = \ldots = p_{mj} = p_j$ $(Fm|prmu, p_{1j} = p_j|C_{max})$ *atlikimo laikas yra lygus:*

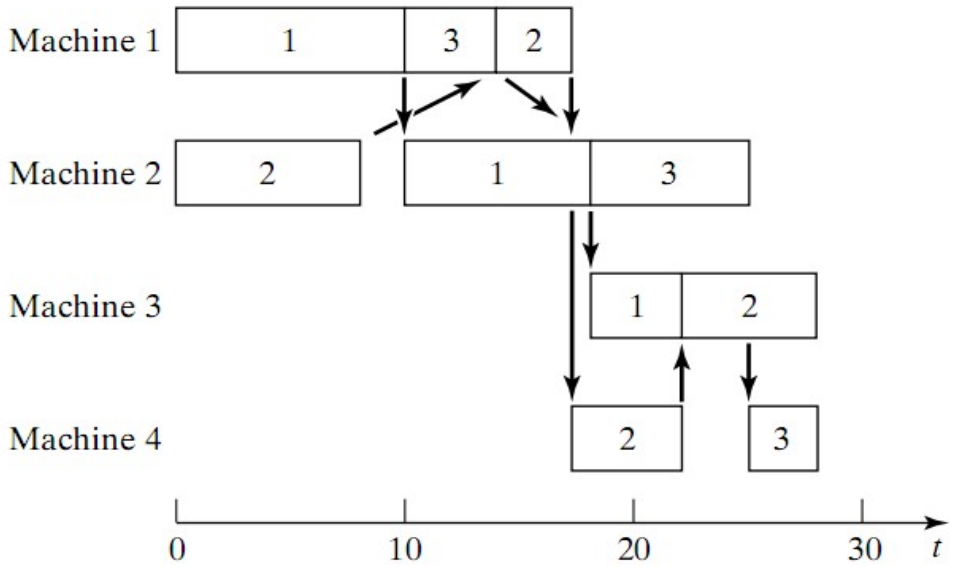
$$C_{max} = \sum_{j=1}^{n} p_j + (m-1)\max(p_1, \ldots, p_n) \quad \textbf{(1.5)}$$

Ir yra nepriklausomas nuo tvarkaraščio.

1.4 KITI FABRIKAI

Darbo fabrikas

Šioje sistemoje, kurioje yra *m* mašinų, kiekvienas darbas turi savo atskirą maršrutą, kurį turi įvykdyti. Yra atskiros Darbo fabrikas sistemos, iš kurių vienoje darbai gali pabūti kiekvienoje mašinoje daugiausiai vieną kartą, ir kitose sistemoje, kurioje darbai gali pabūti mašinose daugiau nei kartą. Dažniausiai naudojama darbo fabriko tvarkaraštį su *m* mašinų pagal mažiausią baigimo laiko optimumo kriterijų žymėsime $J_m||C_{max}$.



1.4 pav Ganto diagrama J4$||C_{max}$

Panagrinėsime $J2||C_{max}$, t.y dvi mašinos ir n darbų. Kai kurie darbai turi būti apdirbti pirmiausia pirmoje mašinoje, o tada antroje mašinoje, atitinkamai likusieji darbai turi būti apdirbti pirmiausia antroje mašinoje, ir tik tada pirmoje. Apdirbimo laikas darbo *j* mašinoje 1(2) yra p_{1j}(p_{2j}). Tikslas yra sumažinti atlikimo laiką.

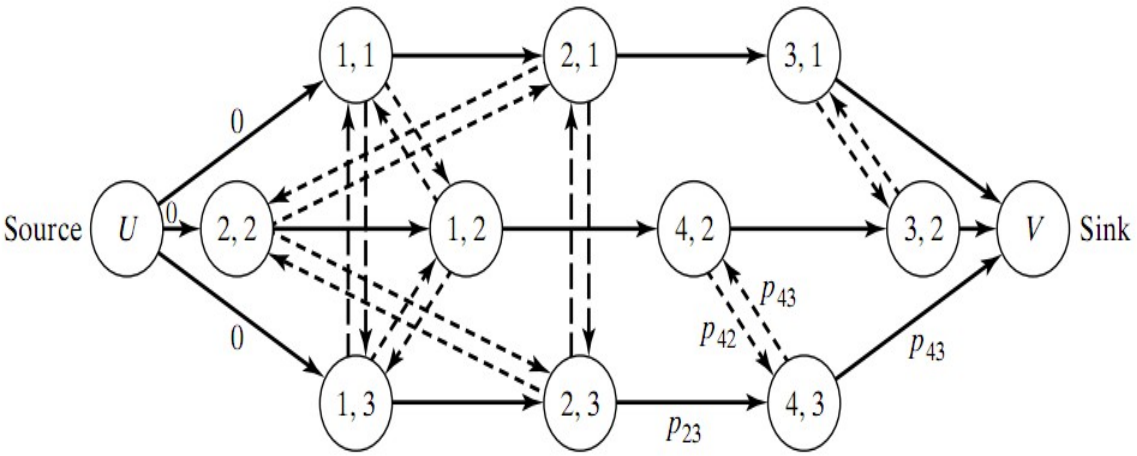
Ši problema gali būti sumažinta iki $F2||\ C_{max}$ Tokiu būdu; Tegul $J_{1,2}$ žymi rinkinį darbų, kurie pirmiausia turi būti atlikti mašinoje 1, ir $J_{2,1}$ žymi rinkinį darbų, kurie pirmiausia turi būti atlikti mašinoje 2. Pastebėkime, kad darbas iš $J_{1,2}$ rinkinio baigia savo apdirbimą mašinoje 1, atidėdami jo apdirbimo pradžia mašinoje 2 mes neįtakojame bendro atlikimo laiko tol, kol mašina 2 yra ištisai užimta. Atitinkamai tą pati galima pasakyti ir apie $J_{2,1}$ darbų rinkinį. Vadinasi darbai iš $J_{1,2}$ rinkinio turi didesnį prioritetą negu darbai iš $J_{2,1}$ rinkinio mašinoje 1, atitinkamai darbai iš $J_{2,1}$ rinkinio turi didesnį prioritetą negu darbai iš $J_{1,2}$ rinkinio mašinoje 2. Pirmąją iš dviejų seką galime apibūdinti $J_{1,2}$ rinkinį kaip $F2||C_{max}$ problemą su 1 mašina kaip pirma ir 2 mašina antroji, ir antrąją seką galime apibūdinti $J_{2,1}$

rinkinį kaip F2||C_{max} problemą su 2 mašina kaip pirma ir 1 mašina antroji. Tai veda į SPT(1)-LPT(2) pritaikymą kiekvienam iš šių rinkinių, su prioritetais kaip įvardinta aukščiau.

Ši dviejų mašinų darbo fabriko tvarkaraščių problemų yra viena iš keleto problemų, kurioms įmanoma rasti polinominį algoritmą optimaliam tvarkaraščiui rasti.

Toliau šnekėsime apie Jm|| C_{max} problemą be recirkuliacijų.

Sumažinimui atlikimo laikui darbo fabriko tvarkaraštyje be recirkuliacijų, labai patogu naudoti tiesioginį grafą. Tarkime turime tiesioginį grafą G su mazgų rinkiniu N ir dviejų rūsių lankais A ir B. Mazgai N atitinka visas operacijas (*i,j*) kurios turi būti atliktos *n* darbuose. Taip vadinamos *jungiamieji (conjuctive,* vienspalviai*)* lankai žymi darbų kelius. Jeigu lankas (*i,j*) -> (*k,j*) priklauso A, tai darbas *j* turi būti apdirbtas mašinoje *i* prieš apdirbant mašinoje *k*, t.y operacija (*i,j*) atliekama anksčiau už (*k,j*).Dvi operacijos kurios priklauso skirtingiem darbam, bet turi būti atlikti toje pačioje mašinoje yra sujungti dviem, taip vadinamiem *skiriamaisiais (disjunctive,* punktyriniai) lankais skirtingomis kryptimis ir priklauso B grupei. Toks grafas žymimas G = (N,A,B).



1.5 pav. Tiesioginis grafas darbo fabriko tvarkaraščiui.

Galimas sąrašas atitinka vieno skiriančio lanko pasirinkimą nuo kiekvienos poros taip, kad išplaukiantis tiesioginis grafas būtų ciklinis. Tai leidžia suprasti, kad pasirinkimas skiriančių lankų nuo klikos (*klika* – terminas, grafų teorijoje, reiškiantis grafe du sujungtus mazgus, šiuo atveju reiškia du skiriamuosius ženklus) turi būti ciklinis. Toks pasirinkimas nustato seką, kurioje operacijos daro lanką, kad būtų įvykdytos tos mašinos. Kad pasirinkimas nuo klikos turi būti ciklinis gali būti aprašytas taip: Jei buvo ciklas klikos viduje, įmanoma operacijų seka atitinkamose mašinose nebūtų buvusi galima.

Galbūt nėra akivaizdu, kodėl neturi būti jokio ciklo, suformuoto jungiamųjų lankų ir skiriančių arkų nuo skirtingų klikų. Tačiau, toks ciklas taip pat reikštu situacija, kuri yra neįmanoma. Pavyzdžiui, tarkim (h, j) ir (i, j) reiškia dvi nuoseklias operacijas, kurios priklauso darbui j ir tarkim (i, k) ir (h, k) reiškia dvi nuoseklias operacijas, kurios priklauso darbui k. Jei tvarkaraštyje operacija (i, j) įvyksta pirma operacijos (i, k) ant mašinos i, ir operacija (h, k) įvyksta pirma operacijos (h, j) ant mašinos h, tai grafas turi savyje ciklą su keturiais lankais, dviem jungiamais lankais ir dviem skiriančiais lankais nuo skirtingų klikų. Toks sąrašas yra fiziškai neįmanomas. Susumuodamas, jei D reiškia poaibį išrinktų skiriančių lankų ir grafo G (D) yra apibrėžtas komplekto jungiamųjų lankų ir poaibio D, tai D atitinka galimą sąrašą, tada ir tik tada, jei G (D) neturi savyje jokių nukreiptų ciklų.

Galimo tvarakraščio atlikimo laikas yra nustatytas ilgiausio kelio G (D) iš šaltinio U į nuotaką V. Šis ilgiausias kelias susideda iš komplekto, kuris operacijų pradiniu laiku 0 ir pabaigos atlikimo laiku. Kiekviena operacija šiame kelyje buvo nedelsiant sekta ar prie kitos operacijos toje pačioje mašinoje ar prie kitos operacijos to paties darbo kitoje mašinoje. Problema mažinanti atlikimo laiką yra sumažinta iki pasirinkimo rasti skiriančių lankų, kuris mažina ilgiausio kelio ilgį (kuris yra, *kritiškas* kelias). Yra kelios matematinės programinės formuluotės darbo fabrikui be cirkuliacijos iš naujo, apimant daug programinių sveikojo skaičiaus formuluočių. Tačiau, formuluotė, dažniausiai panaudota, yra vadinamoji skirianti programinė formuluotė.

Ši skirianti programinė formuluotė yra artimai susieta su skiriančiu darbo fabriko grafo pavaizdavimu. Kad pristatytumėte skiriančią programinę formuluotę, tarkim y_{ij}, reiškia pradinį operacijos laiką (i, j). Prisiminkite, kad komplektas N reiškia visų operacijų komplektą (i, j), ir komplektą visų nukreipimo apribojimų komplektas (i, j) — »(k, j), kurie reikalingi, kad darbas j būtų apdirbtas mašinoje i anksčiau, negu tai bus apdirbama mašinoje k. Sekanti matematinė programa mažina atlikimo laiką.

Tikslas, sumažinti atlikimo laiką.

$y_{kj} - y_{ij} \geq p_{ij}$	visiem *(i,j)→ (k,j)€ A*	
$C_{max} - y_{ij} \geq p_{ij}$	visiem *(i,j)€ N*	
$y_{ij} - y_{il} \geq p_{il}$ $y_{il} - y_{ij} \geq p_{ij}$	visiem *(i,l)* ir *(i,j),*	i = 1,...,m
$y_{ij} \geq 0$	visiem *(i,j)€ N*	

Šioje formuluotėje, pirmas apribojimų komplektas garantuoja, kad operacija (k, j) negali prasidėti anksčiau, negu operacija (i, j) yra užbaigta. Trečią apribojimų komplektą pavadiname skiriančiais apribojimais; jie garantuoja, kad tam tikras užsakymas egzistuoja tarp operacijų skirtingų

darbo vietų, kurios turi būti apdirbtos toje pačioje mašinoje. Dėl šitų apribojimų ši formuluotė vadinama *skiriančia programine formuluote*.

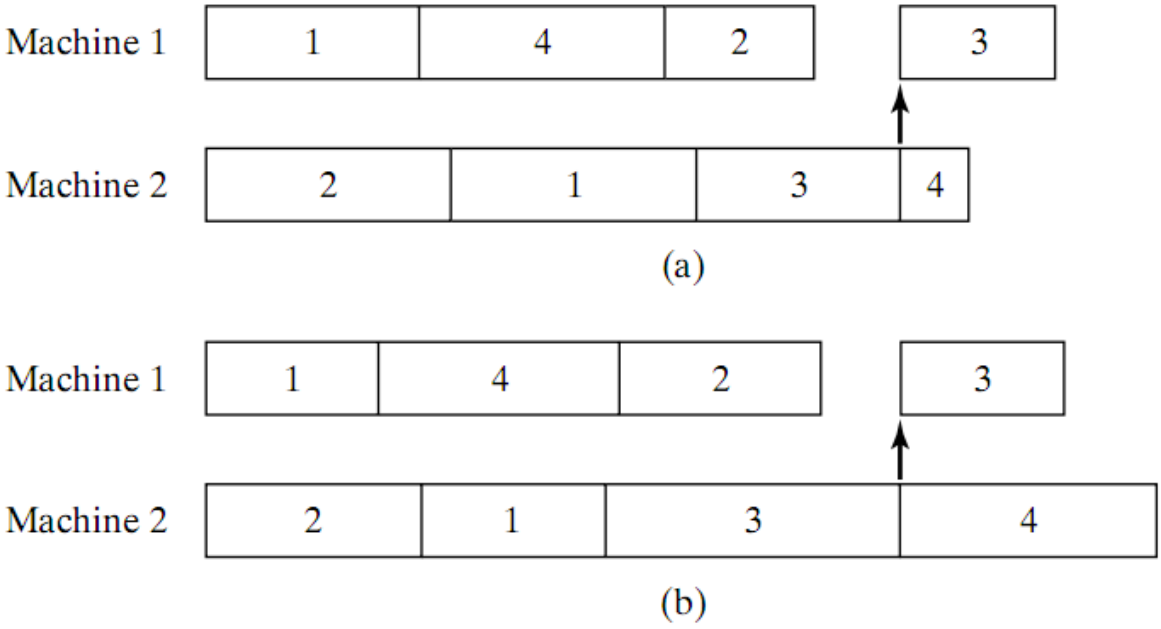
Atviras fabrikas

Tarkime turime *m* mašinų. Kiekvienas darbas turi pabūti kiekvienoje iš *m* mašinų, tik tam tikro darbo apdorojimo laikas ant tam tikros mašinos gali būti lygus nuliui. Nėra jokių apribojimų liečiančių darbo maršrutą per mašinas. Tvarkaraštis leidžia sudaryti maršrutą kiekvienam darbui, taigi kiekvienas darbas gali turėti skirtingą maršrutą.

Nagrinėkime O2 || C_{max}(atviro fabriko tvarkaraštį, su didžiausio atlikimo laiko optimumo kriterijum); tai yra, yra dvi mašinos ir n darbų. Darbas j gali būti apdirbtas iš pradžių ant mašinos 1 ir paskui ant mašinos 2 ar atvirkščiai; galima laisvai nustatyti maršrutus. Atlikimo laikas turi būti sumažintas. Aišku, kad

$$C_{max} \geq max\left(\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\right), \quad \textbf{(1.6)}$$

Kadangi atlikimo laikas negali būti mažesnis, negu darbo krūvis ant bet kurios mašinos.



1.6 pav. Atviro fabriko tvarkaraštis

Toliau panagrinėsime ne užlaikymo tvarkaraščius. Tai yra. jei yra darbas, laukiantis apdirbimo, kai mašina yra laisva, tai mašinai privalo jį apdirbti. Iš to išplaukia, neturintis darbo periodas gali įvykti ant mašinos, tada ir tik tada jei vienas darbas lieka būti apdirbtas ant tos mašinos ir. kai ta mašina yra pasiekiama, šis paskutinis darbas yra tik tada apdirbamas ant kitos mašinos. Gali būti parodoma, kad daugumoje vieni tokie neturinčio darbo periodai gali įvykti ant beveik visų vienoje iš dviejų mašinų (žr. 1,3 iliustraciją). Toks neturinčio darbo periodas gali sukelti nereikalingą atlikimo laiko padidėjimą; jei pasirodo, šis paskutinis darbas yra pats paskutinis darbas užbaigti visą jo apdirbimą, tai neturintis darbo periodas tikrai sukelia atlikimo laiko padidėjimą (žr. 1,3.a iliustraciją). Jei šis paskutinis darbas. užbaigęs jo apdirbimą ant mašinos, kuri buvo neturinti darbo, nėra pats paskutinis darbas paliekantis sistemą, tada atlikimo laikas yra vis dar lygus dviejų darbo krūvių maksimumui (žr. 1,3.b iliustraciją).

Panagrinėkime tokią taisyklę: kada mašina tampa laisva, pradėkite apdirbti tuos darbus, kurie dar nebuvo apdirbti nei vienoje mašinoje ir kurie yra su ilgiausiu apdirbimo laiku ant kitos mašinos. Ši taisyklė yra vadinama kaip *Ilgiausiu Kintamu Apdirbimo Laiku iš pradžių* (*Longest Alternate Processing Time first* - LAPT) taisyklė. Pradiniame laiko momente, kai abi mašinos yra neturinčios darbo, tai gali įvykti, kad tas pats darbas ruošiasi, kad būtų pirmas ant abiejų mašinų. Jei taip atsitinka, tai nėra reikšmės, ant kurios mašinos šis darbas yra apdirbtas iš pradžių. Pagal šią LAPT taisyklę, kad ir kada mašina tampa laisva, darbai, kurie jau užbaigė apdirbimą ant kitos mašinos turi žemą, kuris yra. nulis, prioritetas ant mašinos kurios ką tik baigė darbą. Nėra jokio skirtingumo tarp dvejų darbų prioritetų, kai abu jau buvo apdirbti ant kitos mašinos.

3 teorema. LAPT taisyklė sugeneruoja optimalų tvarkaraštį O2||C_{max} (atviro fabriko tvarkaraštis su dviem mašinom ir didžiausio atlikimo laiko optimumo kriterijus) sistemai. O jo didžiausias atlikimo laikas yra

$$C_{max} = \max\left(\underbrace{max}_{j\in(1,\dots,n)} (p_{1j}, p_{2j}), \sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j} \right), \quad \textbf{(1.7)}$$

Įrodymas. Iš tikrųjų, bendra (ir mažiau apribojantis) tvarkaraščio taisyklė jau garantuoja mažiausia atlikimo laiką. Ši bendresnė taisyklė gali būti priežastimi daugelio skirtingų tvarkaraščių, kurie yra visi optimalūs. Ši optimalių sąrašų klasė apima LAPT tvarkaraščius.

Tarkime, be bendrumo praradimo, kad ilgiausias apdirbimo laikas tarp 2n apdirbimo laiko priklauso operacijai (1, k), kuris yra,

$p_{ij} \leq p_{1k}$, $\quad i = 1,2,\ j = 1,\dots,n$

Bendresnė taisyklė gali būti apibūdinta taip: Jei operacija (l, k) yra ilgiausia operacija, tai darbas k turi būti pradėtas laiku 0 ant mašinos 2. Po to, kai darbas k užbaigė savo apdirbimą mašinoje 2, jo operacija (1, k) turi žemiausią prioritetą dėl apdirbimo ant mašinos 1. Kadangi jo prioritetas yra visada žemesnis negu prioritetas bet kokios kitos operacijos, pasiekiamos mašinai 1, operacijos apdirbimas (1, k) bus atidėtas tiek kiek įmanoma. Tai gali būti apdirbama tiktai mašinoje 1, jei joks kitas darbas nėra pasiekiamas tam, kad apdirbtų mašina 1 (tai gali įvykti jei tai yra paskutinė operacija, kuri būtų padaryta mašinoje 1 arba jei tai yra priešpaskutinė operacija ir paskutinis, operacija nėra pasiekiama. Tai tada ir tik tada darbas yra apdirbamas ant mašinos 2). 2 (n—1) operacijos likusių n—1 darbų gali būti apdirbti ant dviejų mašinų bet kokiame užsakyme; tačiau, nevaržomas tarpinis neveikimas nėra leistas.

Tai kad ši taisyklė yra tvarkaraščio su minimaliu atlikimo laiku taisyklė, gali būti parodyta taip; Jei sugeneruotas tvarkaraštis neturi jokio neturinčio darbo periodo ant kiekvienos mašinos, žinoma, tai optimalu. Tačiau, neturintis darbo periodas gali įvykti ir mašinai 1 ir mašinai 2. Tokiu atveju galimi du variantai.

Atvejis 1: Tarkime, kad neturintis darbo periodas įvyksta mašinoje 2. Jei taip, tai vienintelė operacija turi būti atlikta mašinoje 2, bet ši operacija vis dar turi užbaigti savo apdirbimą ant mašinos 1. Manykime, kad ši operacija priklauso darbui *l*. Kai darbas *l* pradeda apdirbimą mašinoje 2, darbas k

pradeda apdirbimą mašinoje 1 ir $p_{1k} > p_{2l}$. Tokiu-būdu atlikimo laikas yra nustatytas baigimo darbo k ant mašinos 1, ir joks neturintis darbo periodas neįvyko ant mašinos 1. Tokiu būdu sąrašas yra optimalus.

Atvejis 2: Tarkime, kad neturintis darbo periodas įvyksta ant mašinos 1. Neturintis darbo periodas mašinoje 1 gali įvykti tiktai tada, kai mašina 1 yra laisva po užbaigimo visų jos operacijų išskyrus operaciją (l, k), ir operacija (2, k) darbo k yra tame punkte, kai vis dar apdirbama mašinoje 2. Šiuo atveju, atlikimo laikas yra lygus $p_{2k} + p_{1k}$, ir sąrašas yra optimalus.

Įrodymas baigtas.

Egzistuoja ir kita taisyklė, kuri iš pirmo žvilgsnio yra tokia taisyklė, kuri suteikia, kuriai nors mašinai kai tik ji atsilaisvina, didžiausią prioritetą darbui su didžiausiu bendru likusiu baigimo laiku abiejuose mašinose. Tačiau, kaip pasirodo, yra atveju, kai ši taisyklė sugeneruoja neoptimalų tvarkaraštį, netgi, jei tvarkaraštyje egzistuoja tik dvi mašinos. Faktas yra tai, kad prioriteto lygis priklauso *tik* nuo likusio apdirbimo laiko kitoje mašinoje.

Darbo fabrikas su pirmenybės savybę pagal baigimo laiko kriterijų.

Darbo fabriko tvarkaraščiai su pirmenybės savybe dažniausiai yra lengvesni sugeneruoti. Palyginimui $Om||C_{max}$, $Om|prmp|C_{max}$(prmp – pirmenybės savybė) yra išsprendžiama polinominiame laike.

Iš fakto, kad baigimo laiko reikšmė yra sumažinama dviejų mašinų modelyje net ir bė pirmenybės savybės, išplaukia kad LAPT bė pirmenybių savybes yra optimali ir $Om|prmp|C_{max}$.

Yra lengva nustatyti apatini rėžį darbo atlikimo laikui, su m mašinų ($m \geq 3$), kai pirmenybių savybė yra galiojanti:

$$C_{max} \geq max\left(\underbrace{max}_{j\in(1,\dots,n)} \sum_{j=1}^{m} p_{ij}, \underbrace{max}_{i\in(1,\dots,m)} \sum_{i=1}^{n} p_{ij} \right), \quad \textbf{(1.8)}$$

Tai yra – atlikimo laikas yra bent jau tokio dydžio, koks yra didžiausias apkrovimas kurioje nors iš mašinų ir yra bent jau tokio dydžio koks turi būti padarytas apdirbimas kiekvienam iš n darbų. Pasirodo, yra gan paprasta sugeneruoti tvarkarašti, kurio atlikimo laikas yra apatinio rėžio.

Kad išnagrinėti kaip toks algoritmas veikia, tarkime turime m * n matrica P, kuri susidaro iš apdirbimo laikų p_{ij}. Eilute i arba stulpelis j yra vadinamas *tvirtu* jei jo suma yra lygi apatiniam rėžiui, arba *silpna* kitu atveju. Tarkime kad įmanoma surasti tokia matrica kuri yra netuščia ir su lygiai viena reikšme tvirtoj eilutėj, ir viena reikšme tvirtam stulpely, ir daugiausiai viena reikšme silpnoj eilutėj ir viena reikšme silpnam stulpely. Tokia sudėtis vadinama *mažėjančiu* rinkiniu. Šis sutrumpinimas naudojamas tam kad sukonstruoti tvarkaraštį, kurio dydis būtų Δ, kažkokiam tinkamai pasirinktam Δ.

Šitam daliniame tvarkaraštyje mašina i dirba darbą j tokį laiką, kuris lygus min(p_{ij},Δ) kiekvienam elementui p_{ij} mažėjančiame rinkinyje. Pradinėje matricoje P reikšmės atitinkančios mažėjantį rinkinį yra sumažinamos į max(0, p_{ij}-Δ), o gauta matrica pažymima P‘. Jei Δ parenkamas tinkamai, alikimo laikas $C‘_{max}$ kuris atitinka nauja matrica P‘ yra lygus $C‘_{max}$ - Δ. Ši Δ reikšmė turi būti parenkama labai atsargiai. Pirmiausia, akivaizdu, kad Δ turi būti mažesnis nei kiekviena p_{ij} mažėjančiame rinkinyje kuris yra tvirtame stulpelyje ar eilutėje, kitu atveju egzistuos eilutė ar stulpelis, kuris yra aiškiai didesnis nei $C‘_{max}$. Jei del kažkokios priežasties p_{ij} yra elementas mažėjančiame rinkinio silpnoje eilutėje, tarkim eilutėje i, tai būtina kad

$$\Delta \leq p_{ij} + C_{max} - \sum_k p_{ik,} \tag{1.9}$$

Kur $C_{max} - \sum p_{ik}$ yra kiekis silpnu laikų eilutėje i. Panašiai, jei p_{ij} yra elementas silpname stulpelyje j, tada

$$\Delta \leq p_{ij} + C_{max} - \sum_k p_{kj,} \tag{1.10}$$

Kur $C_{max} - \sum p_{kj}$ yra kiekis silpnu laikų stulpelyje j. Jeigu eilute i arba stulpelis j neturi elementu mažėjančiame rinkinyje, tai

$$\Delta \leq C_{max} - \sum_j p_{ij,}$$

arba

$$\Delta \leq C_{max} - \sum_i p_{ij.}$$

Jeigu Δ pasirenkamas kiek galima didelis pagal šitas sąlygas, tada arba P‘ turės bent vienu griežtai teigiamu elementu daugiau negu P, arba P‘ turės bent viena tvirta eilute daugiau negu P. Iš to išplaukia, kad negali būti daugiau nei r+m+n iteracijų, kur r yra kiekis griežtai teigiamu elementu pradinėje matricoje.

Paaiškėjo, kad visada įmanoma rasti mažėjanti rinkinį neneigiamai matricai P. Ši savybė yra rezultatas gautas iš Birkofo ir Neumano teoremos(Birkhoff and von Neumann) skirtos stochastinėms

matricoms ir permutacinėms matricom. Tačiau šios teoremos įrodymas yra per didelis šiam magistriniam darbui.

1.5 ALGORITMAI

Simulated Annealing

Dirbtinis atrinkimas (*simulated annealing)* – tai procesas, kuris kilęs iš medžiagų mokslo bei fizikos sričių. Pradžioje jis buvo sukurtas kaip modelis, apibūdinantis fizinio atrinkimo procesą determinuotiems procesams.

Dirbtinio atrinkimo procesas atlieka tam tikra iteracijų kiekį. Iteracijoje k, yra susiformavęs tam tikras tvarkaraštis S_k, taip pat turime kol kas geriausią rastą tvarkaraštį S_0. Tegul $G(S_k)$ ir $G(S_0)$ apibūdina atitinkamas reikšmes, pagal kurių prioritetą rūšiuojam tvarkaraštį. Aišku ,kad $G(S_k) \geq G(S_0)$. Geriausio tvarkaraščio reikšmė $G(S_k)$ dažnai vadinama įkvepiantysis kriterijus. Algoritmas, ieškantis geriausio tvarkaraščio, keliauja nuo vieno tvarkaraščio prie kito. Iteracijoje k algoritmas ieško tvarkaraščio S_k aplinkoje. Pirmiausia, iš šios aplinkos, parenkamas taip vadinamas *kandidatas* tvarkaraštis S_c. Šis parinktasis tvarkaraštis kandidatas gali būti parenkamas atsitiktinai arba kokia nors organizuota tvarka. Jeigu $G(S_c) < G(S_k)$, žingsnelis atliekamas $S_{k+1} = S_c$. Jeigu $G(S_c) < G(S_0)$, tada S_0 yra prilyginama S_c. Tačiau, jeigu $G(S_c) \geq G(S_k)$, žingsnelis į S_c padaromas tik su tikimybe $1 - P(S_k, S_c)$, kur

$$P(S_k, S_c) = e^{\left(\frac{G(S_k) - G(S_k)}{\beta_k}\right)} \qquad \textbf{(1.11)}$$

Parametras $\beta_1 \geq \beta_2 \geq \beta_3 \geq \ldots > 0$ yra kontroliavimo parametras, kuris vadinamas *vėsinimo parametras* arba *temperatūra.* Dažnai β_k pasirenkamas a^k, kur a yra tarp 0 ir 1.

Simulated Annealing algoritmas:

pirmas žingsnis:

Nustatome k = 1 ir pasirenkame β_1.

Pasirenkame koki nors tvarkaraštį S_1 naudodami kokį nors būdą.

Nustatome $S_0 = S_1$.

antras žingsnis:

Pasirenkame tvarkaraštį kandidatą S_c iš S_k aplinkos.

Jei $G(S_0) < G(S_c) < G(S_k)$, nustatome $S_{k+1} = S_c$ ir einam į 3čia žingsnelį.

Jei $G(S_c) < G(S_0)$, nustatome $S_0 = S_{k+1} = S_c$ ir einam į 3čia žingsnelį.

Jei $G(S_c) > G(S_k)$, sugeneruojame atsitiktinį skaičių U_k iš *Uniform(0,1)* pasiskirstymo; Jei $U_k < P(S_k, S_c)$, nustatome $S_{k+1} = S_c$, kitu atveju nustatome $S_{k+1} = S_k$ ir einam i 3čia žingsnį.

trečias žingsnis:

Nustatome $\beta_{k+1} \leq \beta_k$.

Nustatome k = k+1.

Jei k = N, nutraukiame algoritmą, kitu atveju einam į 2rą žingsnį.

Tabu paieška

Tabu paieška (tabu-search) daugeliu aspektu yra labai panaši į dirbtinio atrinkimo metodą tuo, kad ji irgi keliauja nuo vieno tvarkaraščio prie kito, ieškant potencialiai geresniu tvarkaraščių. Taip pat kaip ir dirbtinio atrinkimo metode, tvarkaraščio aplinkoje ieškoma kandidatų, tai taip pat galima daryti atsitiktinai arba kokia nors tvarka. Pagrindinis skirtumas tarp dirbtinio atrinkimo ir tabu paieškos yra mechanizme, kuris naudojamas atrinkti arba atmesti kandidatą tvarkaraštį. Tabu paieškoje tai nebe tikimybinis atrinkimas, o, galima sakyti, deterministinės prigimties. Bet kuriame proceso bėgyje, yra tam tikras mutacijų sąrašas, kuris neleidžia atrinkti tam tikro tvarkaraščio. Mutacija tabu paieškoje gali būti, pavyzdžiui, tam tikri darbai negali būti sukeisti vietomis. Tabu paieška dažniausiai turi fiksuotą skaičių mutacijų (dažniausiai 5-9) priklausomai nuo užduoties. Tokia sistema leidžia patikrinti daugiau tvarkaraščių kandidatų per kiekvieną iteracija, nes mutacijų apribojimai neleidžia kartotis tikrinamiems tvarkaraščiams, kas dažnai gali atsitikti tarkim dirbtinio atrinkimo metode.

Tabu paieškos algoritmas:

pirmas žingsnis:

Nustatome k = 1.

Pasirenkame koki nors tvarkaraštį S_1 naudodami kokį nors būdą.

Nustatome $S_0 = S_1$.

antras žingsnis:

Pasirenkame tvarkaraštį kandidatą S_c iš S_k aplinkos.

Jei $S_k \rightarrow S_c$ yra leidžiamas pagal mutacijų sąrašą, nustatome $S_{k+1} = S_k$ ir einam į 3čia žingsnelį.

Jei Jei $S_k \rightarrow S_c$ yra neleidžiamas pagal mutacijų sąrašą, nustatome $S_{k+1} = S_c$ ir įvedame atvirkštinę mutaciją į viršų mutacijos sąraše.

Pastumiame visus įrašus mutacijų sąraše viena pozicija į apačią, o patį paskutinį įrašą ištriname.

Jei $G(S_c) < G(S_0)$, nustatome $S_0 = S_c$ ir einam į 3čia žingsnelį.

trečias žingsnis:

Nustatome k = k+1.

Jei k = N, nutraukiame algoritmą, kitu atveju einam į 2rą žingsnį.

Genetinis algoritmas

Genetinis algoritmas (*Genetic algorithm*) yra daug labiau bendrinis ir abstraktus negu dirbtinis atrinkimas ar tabu paieška. Iš tikro, dirbtinis atrinkimas ir tabu paieška yra genetinio algoritmo atskiras atvejis.

Genetinis algoritmas, savo paieškos sistema skiriasi nuo dirbtinio atrinkimo ar tabu paieškos vienu svarbiu akcentu. Kiekviename iteracijos žingsnelyje į kitą iteracija perkeliamas tam tikras kiekis sugeneruotu tvarkaraščių, tuo tarpu dirbtiniame atrinkime ir tabu paieškoje buvo perkeliamas tik vienas tvarkaraštis. Taigi dirbtinis atrinkimas ir tabu paieška gali būti traktuojamas kaip atskira klasė genetinio algoritmo su populiacijos dydžiu 1. Genetiniame algoritme taip pat skiriasi ir sudaromoji tvarkaraščių aplinka, kuri remiasi nebe vienu tvarkaraščiu, o jau yra sudaroma iš keleto.

Toliau pateikiamas labai supaprastintas genetinio algoritmo algoritmas.

Genetinio algoritmo algoritmas:

pirmas žingsnis:

Nustatome k = 1.

Pasirenkame pradinę seką sudaryta iš L narių naudodami kokį nors būdą $S_{1,1} \dots S_{1,L}$.

antras žingsnis:

Pasirenkame 2 geriausius tvarkaraščius iš $S_{k,1} \dots S_{k,L}$, ir pavadinkim juos S_k^{+} ir S_k^{++}.

Pasirenkame 2 blogiausius tvarkaraščius iš $S_{k,1} \dots S_{k,L}$, ir pavadinkim juos S_k^{-} ir S_k^{--}.

Sugeneruokime 2 naujus tvarkaraščius S_k^{*} ir S_k^{**} (palikuonius) sudarytus iš tvarkaraščių S_k^{+} ir S_k^{++}(tėvų).

Pakeiskime tvarkaraščius S_k^{-} ir S_k^{--}tvarkaraščiais S_k^{*} ir S_k^{**}.

Palikime visus kitus tvarkaraščius ir einam į 3čia žingsnelį.

trečias žingsnis:

Nustatome k = k+1.

Jei k = N, nutraukiame algoritmą, kitu atveju einam į 2rą žingsnį.

2. TIRIAMOJI DALIS

Tvarkaraščių sudarymo pagal optimumo kriterijus analizė

Šioje dalyje pateiksiu analizuotus pavyzdžius. Sutrumpinimų lentelėse reikšmės:

- p_{ij} – procesing time - proceso laikas. Tai proceso laikas kurį tam tikras darbas *j* užtrunka ant tam tikros mašinos *i.*
- w_j – wight – svoris. Tai prioriteto faktorius, rodantis darbo *j* svarba nuo kitu darbų. Pavyzdžiui svoris gali reikšti darbo išlaikymo sistemoje kainą.
- d_j – due date - numatytas laikas. Tai laikas, per kurį tam tikras darbas *j* turi būti baigtas.
- C_{max} - makespan – atlikimo laikas. Viso tvarkaraščio baigimo laikas.
- L_{max} – Lateness – vėlavimas. Parodo didžiausia vėlavimą tarp darbo atlikimo laiko ir numatyto darbui baigti laiko.
- $\sum w_j * C_j$ - bendras svorinis atlikimo laikas.

Programoje, naudojant *Dirbtinio atrinkimo* metodą, parametre β_k (*temperatūroje*) naudoju $0{,}5^k$, kur k – iteracijų skaičius.

Kandidato tvarkaraščio iš prieš tai buvusio tvarkaraščio atrinkimui naudoju tokia sistemą:

- pradedant pirmuoju darbu, sukeičiu jį su šalia esančiu darbu.

-Kitam žingsnelį, antrąjį darbą sukeičiu su trečiuoju, ir t.t.

-Jei prieinama iki paskutinio darbo, pradeda nuo pradžių, tik pirmąjį darbą sukeičia su trečiuoju, ir taip tęsiame procesą.

-Jei tvarkaraštis kandidatas priimamas, visa algoritmą pradedam iš pradžių.

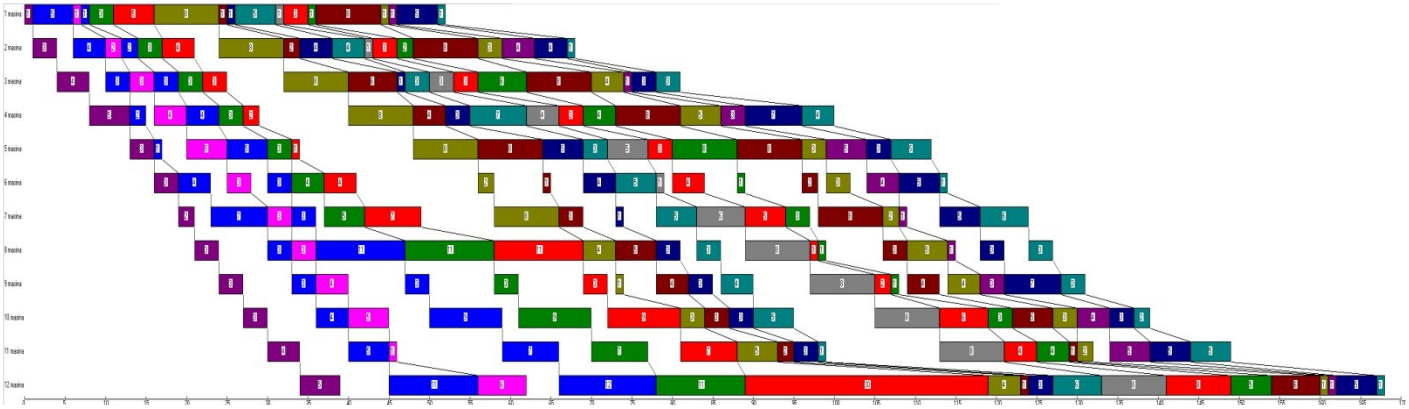
Pradiniai duomenys pateikti 3.1 lentelėje:

3.1 lentelė

Pradiniai modelio duomenys (1 eksperimentas)

Darbo nr (j)	w_j	d_j	p_{1j}	p_{2j}	p_{3j}	p_{4j}	p_{5j}	p_{6j}	p_{7j}	p_{8j}	p_{9j}	p_{10j}	p_{11j}	p_{12j}
1	10	25	1	2	3	4	5	3	3	11	3	9	7	12
2	12	32	5	4	3	2	1	4	7	11	3	9	7	30
3	8	34	3	3	3	3	3	4	5	11	3	9	7	11
4	14	12	1	2	6	4	8	1	3	5	4	3	5	4
5	6	44	8	8	8	8	8	2	8	4	1	3	5	4
6	11	12	1	3	4	5	3	3	2	3	3	3	4	5
7	9	19	1	4	1	3	5	4	1	3	3	3	3	3
8	4	28	5	4	3	7	3	5	5	3	4	5	1	6
9	10	17	1	1	3	4	5	1	6	8	8	8	8	8
10	10	25	1	2	3	4	5	3	3	3	4	5	1	6
11	12	32	5	4	3	2	1	4	7	3	3	4	5	11
12	8	34	3	3	3	3	3	4	5	1	2	6	4	8
13	14	12	1	2	6	4	8	1	3	1	1	3	4	5
14	6	44	8	8	8	8	8	2	8	3	4	5	1	6
15	11	12	1	3	4	5	3	3	2	5	4	3	2	1
16	9	19	1	4	1	3	5	4	1	1	3	4	5	1
17	4	28	5	4	3	7	3	5	5	3	7	3	5	5
18	10	17	1	1	3	4	5	1	6	3	3	2	5	1

- Tvarkaraštis, gautas sudėliojant darbus naudojant dirbtinio atrinkimo (Simulated annealing) algoritmą, teikiant prioritetą atlikimo laikui (Cmax, makespan):



3.1 pav. Ganto diagrama, Sim.anneal.(Cmax)

Metodas: Dirbtinis atrinkimas (Simulated anealing), sudaro tvarkaraštį pagal Cmax

Kriterijai:

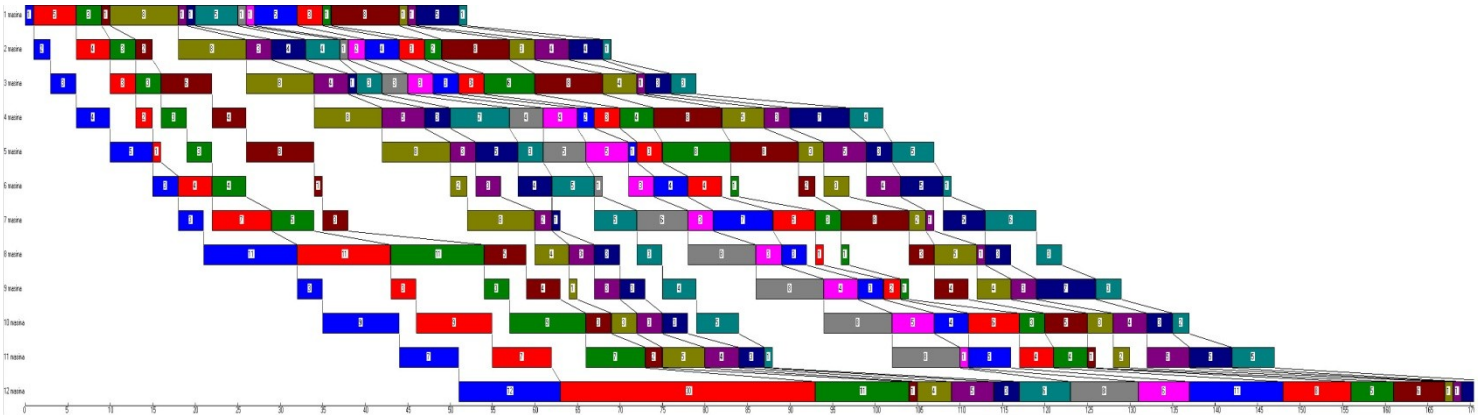
Atlikimo laikas (Cmax, makespan): 168

Didžiausias vėlavimo laikas (Lmax, maximum lateness): 151

Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): 20085

Darbų seka: 6 11 10 1 3 2 5 4 7 8 9 12 13 14 15 16 17 18

- Tvarkaraštis, gautas sudėliojant darbus naudojant dirbtinio atrinkimo (Simulated annealing) algoritmą, teikiant prioritetą didžiausiam vėlavimo laikui (Lmax, maximum lateness):



3.2 pav. Ganto diagrama, Sim.anneal.(Lmax)

Metodas: Dirbtinis atrinkimas (Simulated anealing), sudaro tvarkaraštį pagal Lmax

Kriterijai:

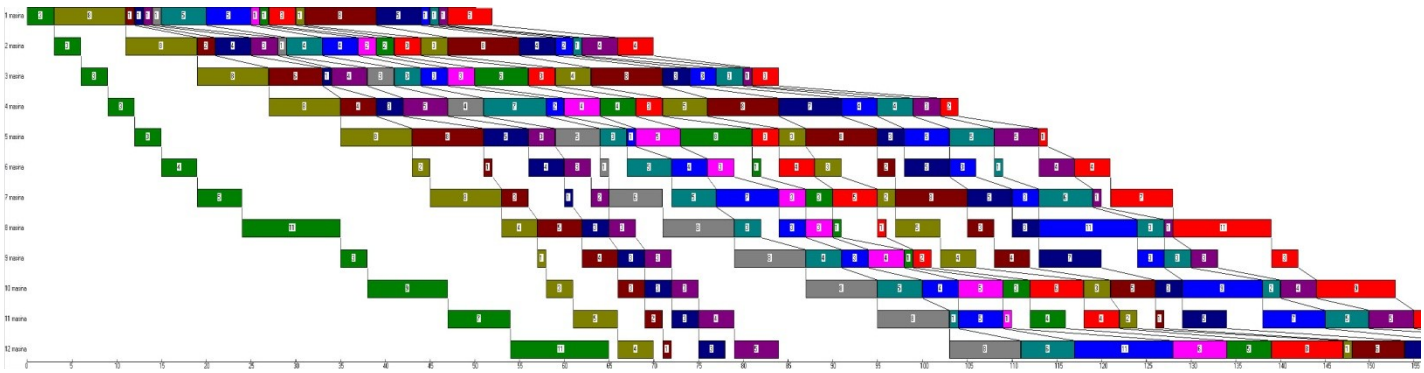
Atlikimo laikas (Cmax, makespan): 175

Didžiausias vėlavimo laikas (Lmax, maximum lateness): 142

Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): 22172

Darbų seka: 1 2 5 6 7 8 9 10 11 12 13 14 15 16 4 3 18 17

- Tvarkaraštis, gautas sudėliojant darbus naudojant dirbtinio atrinkimo (Simulated annealing) algoritmą, teikiant prioritetą bendram svoriniui atlikimo laikui (Sum(wj*Cj), total wighted completion time):



3.3 pav. Ganto diagrama, Sim.anneal.(.($\sum w_j*C_j$)

Metodas: Dirbtinis atrinkimas (Simulated anealing), sudaro tvarkaraštį pagal wC

Kriterijai:

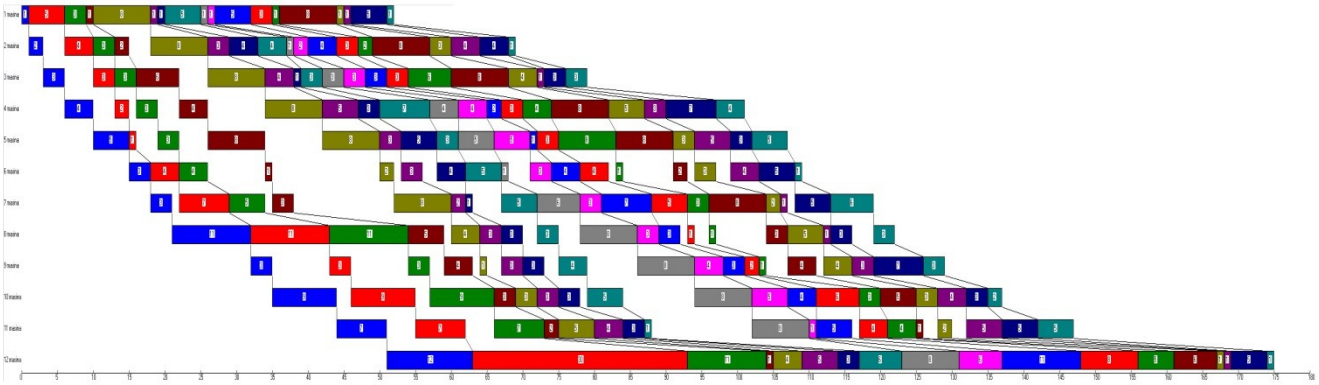
Atlikimo laikas (Cmax, makespan): 203

Didžiausias vėlavimo laikas (Lmax, maximum lateness): 171

Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): 19363

Darbų seka: 3 5 4 7 6 9 8 11 10 13 12 15 14 17 1 18 16 2

- Tvarkaraštis, gautas sudėliojant darbus naudojant Tabu paiškos (Tabu-search) algoritmą, teikiant prioritetą atlikimo laikui (Cmax, makespan):



3.4 pav. Ganto diagrama, Tabu-search.(Cmax)

Metodas: Tabu paieška (Tabu-Search), sudaro tvarkaraštį pagal Cmax

Kriterijai:

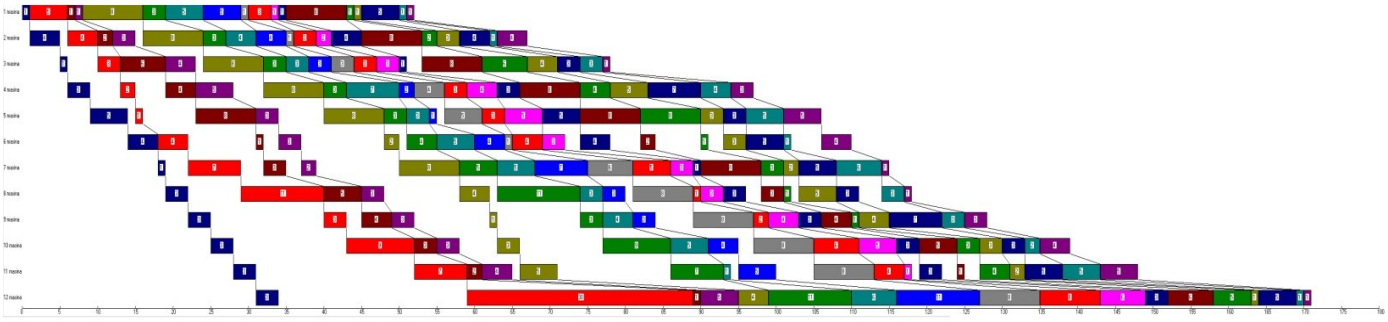
Atlikimo laikas (Cmax, makespan): 173

Didžiausias vėlavimo laikas (Lmax, maximum lateness): 158

Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): 22276

Darbų seka: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

- Tvarkaraštis, gautas sudėliojant darbus naudojant Genetinį (*Genetic algorithm*) algoritmą, teikiant prioritetą didžiausiam vėlavimo laikui (Lmax, maximum lateness):



3.5 pav. Ganto diagrama, Genetic.(Lmax)

Metodas: Genetinis(genetic), sudaro tvarkaraðti pagal Lmax

Kriterijai:

Atlikimo laikas (Cmax, makespan): 171

Didþiausias vėlavimo laikas (Lmax, maximum lateness): 153

Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): 21766

Darbu seka: 7 2 4 6 5 3 8 11 9 12 10 7 14 13 15 17 18 16

Rezultatas:

3.2 lentelė

Galutiniai modelio duomenys(1 eksperimentas)

Metodas	Cmax	Lmax	∑ wj*Cj
Sim.anneal.(Cmax),	168	151	20085
Sim.anneal.(Lmax),	175	142	22172
Sim.anneal.(∑ wj*Cj),	203	171	19363
Tabu-search(Cmax),	173	158	22276
Tabu-search (Lmax),	175	155	21929
Tabu-search (∑ wj*Cj),	178	183	20114
Genetic (Cmax),	156	190	20465
Genetic (Lmax),	171	153	21766
Genetic (∑ wj*Cj),	186	197	18034

Žali langeliai parodo geriausia rezultatą tam kriterijui.

3.3 lentelė

Pradiniai modelio duomenys(2 eksperimentas)

Darbo nr (j)	w_j	d_j	p_{1j}	p_{2j}	p_{3j}	p_{4j}	p_{5j}	p_{6j}	p_{7j}
1	10	25	4	5	3	3	7	8	9
2	12	32	5	4	3	2	1	4	7
3	8	34	3	3	3	3	3	4	5
4	14	12	1	2	6	4	8	1	3
5	6	44	6	8	7	9	8	2	8
6	11	12	1	3	4	5	3	3	2
7	9	19	1	4	1	3	5	4	1
8	4	28	5	4	3	7	3	5	5
9	10	17	1	1	3	4	5	1	6

Toliau rezultatus pateiksiu be ganto diagramų.

3.4 lentelė

Galutiniai modelio duomenys(2 eksperimentas)

Metodas	Cmax	Lmax	∑ wj*Cj
Sim.anneal.(Cmax),	69	43	3762
Sim.anneal.(Lmax),	79	39	4035
Sim.anneal.(∑ wj*Cj),	72	40	3550
Tabu-search(Cmax),	69	43	3550
Tabu-search (Lmax),	77	51	4495
Tabu-search (∑ wj*Cj),	72	60	3764
Genetic (Cmax),	69	50	3600
Genetic (Lmax),	60	41	3831
Genetic (∑ wj*Cj),	69	44	3442

Žali langeliai parodo geriausia rezultatą tam kriterijui.

3.5 lentelė

Pradiniai modelio duomenys(3 eksperimentas)

Darbo nr (j)	w_j	d_j	p_{1j}	p_{2j}	p_{3j}	p_{4j}	p_{5j}	p_{6j}	p_{7j}
1	5	10	5	4	6	7	3	1	6
2	4	10	3	4	2	5	1	6	4
3	6	10	7	5	3	1	5	9	1
4	3	10	6	3	2	5	4	1	3
5	7	10	1	2	5	4	6	3	2
6	9	10	9	7	1	3	7	6	7

3.6lentelė

Galutiniai modelio duomenys(3 eksperimentas)

Metodas	Cmax	Lmax	∑ wj*Cj
Sim.anneal.(Cmax),	56	46	1456
Sim.anneal.(Lmax),	57	45	1446
Sim.anneal.(∑ wj*Cj),	62	52	1430
Tabu-search(Cmax),	56	46	1486
Tabu-search (Lmax),	57	45	1478
Tabu-search (∑ wj*Cj),	63	53	1254
Genetic (Cmax),	55	45	1457
Genetic (Lmax),	59	43	1606
Genetic (∑ wj*Cj),	57	47	1316

Žali langeliai parodo geriausia rezultatą tam kriterijui.

3.7 lentelė

Pradiniai modelio duomenys(4 eksperimentas)

Darbo nr (j)	w_j	d_j	p_{1j}	p_{2j}	p_{3j}	p_{4j}	p_{5j}	p_{6j}	p_{7j}
1	5	24	5	1	7	9	3	1	6
2	7	15	1	6	8	7	6	3	7
3	4	34	9	1	7	5	6	8	4
4	3	54	1	3	6	7	8	9	1
5	8	32	1	3	5	7	9	6	1
6	5	41	3	6	4	8	9	1	5
7	3	10	7	1	6	8	9	1	3

3.8lentelė

Galutiniai modelio duomenys(4 eksperimentas)

Metodas	Cmax	Lmax	∑ wj*Cj
Sim.anneal.(Cmax),	88	67	2326
Sim.anneal.(Lmax),	97	61	2523
Sim.anneal.(∑ wj*Cj),	94	84	1820
Tabu-search(Cmax),	86	71	2204
Tabu-search (Lmax),	107	60	2681
Tabu-search (∑ wj*Cj),	94	83	1947
Genetic (Cmax),	88	75	2523
Genetic (Lmax),	90	62	2107
Genetic (∑ wj*Cj),	97	80	1795

Žali langeliai parodo geriausia rezultatą tam kriterijui.

3.9 lentelė

Pradiniai modelio duomenys(5 eksperimentas)

Darbo nr (j)	w_j	d_j	p_{1j}	p_{2j}	p_{3j}	p_{4j}	p_{5j}	p_{6j}	p_{7j}
1	2	41	5	4	9	8	7	1	3
2	3	42	4	6	3	1	9	8	7
3	4	40	1	9	8	7	6	1	7
4	6	46	4	3	4	9	1	6	5
5	5	52	7	2	1	6	9	7	3
6	2	39	1	3	6	7	9	8	5
7	1	47	8	7	6	3	5	9	7

3.10lentelė

Galutiniai modelio duomenys(5 eksperimentas)

Metodas	Cmax	Lmax	∑ wj*Cj
Sim.anneal.(Cmax),	75	34	1412
Sim.anneal.(Lmax),	75	25	1336
Sim.anneal.(∑ wj*Cj),	82	35	1190
Tabu-search(Cmax),	75	25	1316
Tabu-search (Lmax),	76	35	1342
Tabu-search (∑ wj*Cj),	80	40	1210
Genetic (Cmax),	74	33	1238
Genetic (Lmax),	75	25	1316
Genetic (∑ wj*Cj),	81	40	1113

Žali langeliai parodo geriausia rezultatą tam kriterijui.

3.11 lentelė

Pradiniai modelio duomenys(6 eksperimentas)

Darbo nr (j)	w_j	d_j	p_{1j}	p_{2j}	p_{3j}	p_{4j}	p_{5j}	p_{6j}	p_{7j}
1	4	28	19	12	13	19	18	22	18
2	5	29	19	18	18	11	16	15	23
3	6	22	25	26	22	19	17	14	22
4	4	26	18	14	16	12	33	15	11
5	2	31	18	21	25	26	23	27	19
6	8	18	11	19	18	17	15	22	27
7	9	22	30	15	12	16	26	25	24

3.12lentelė

Galutiniai modelio duomenys(6 eksperimentas)

Metodas	Cmax	Lmax	∑ wj*Cj
Sim.anneal.(Cmax),	264	238	7619
Sim.anneal.(Lmax),	271	240	7992
Sim.anneal.(∑ wj*Cj),	282	256	6675
Tabu-search(Cmax),	271	245	7995
Tabu-search (Lmax),	276	248	7814
Tabu-search (∑ wj*Cj),	297	268	7084
Genetic (Cmax),	249	244	7738
Genetic (Lmax),	265	221	7303
Genetic (∑ wj*Cj),	280	252	6724

Žali langeliai parodo geriausia rezultatą tam kriterijui.

3.13 lentelė

Pradiniai modelio duomenys(7 eksperimentas)

Darbo nr (j)	w_j	d_j	p_{1j}	p_{2j}	p_{3j}	p_{4j}	p_{5j}	p_{6j}
1	6	6	1	9	6	1	7	9
2	4	8	1	4	5	6	9	1
3	5	4	3	5	4	6	7	9
4	6	9	7	2	1	6	4	8
5	7	7	3	1	4	8	9	5
6	1	6	5	1	3	6	7	9

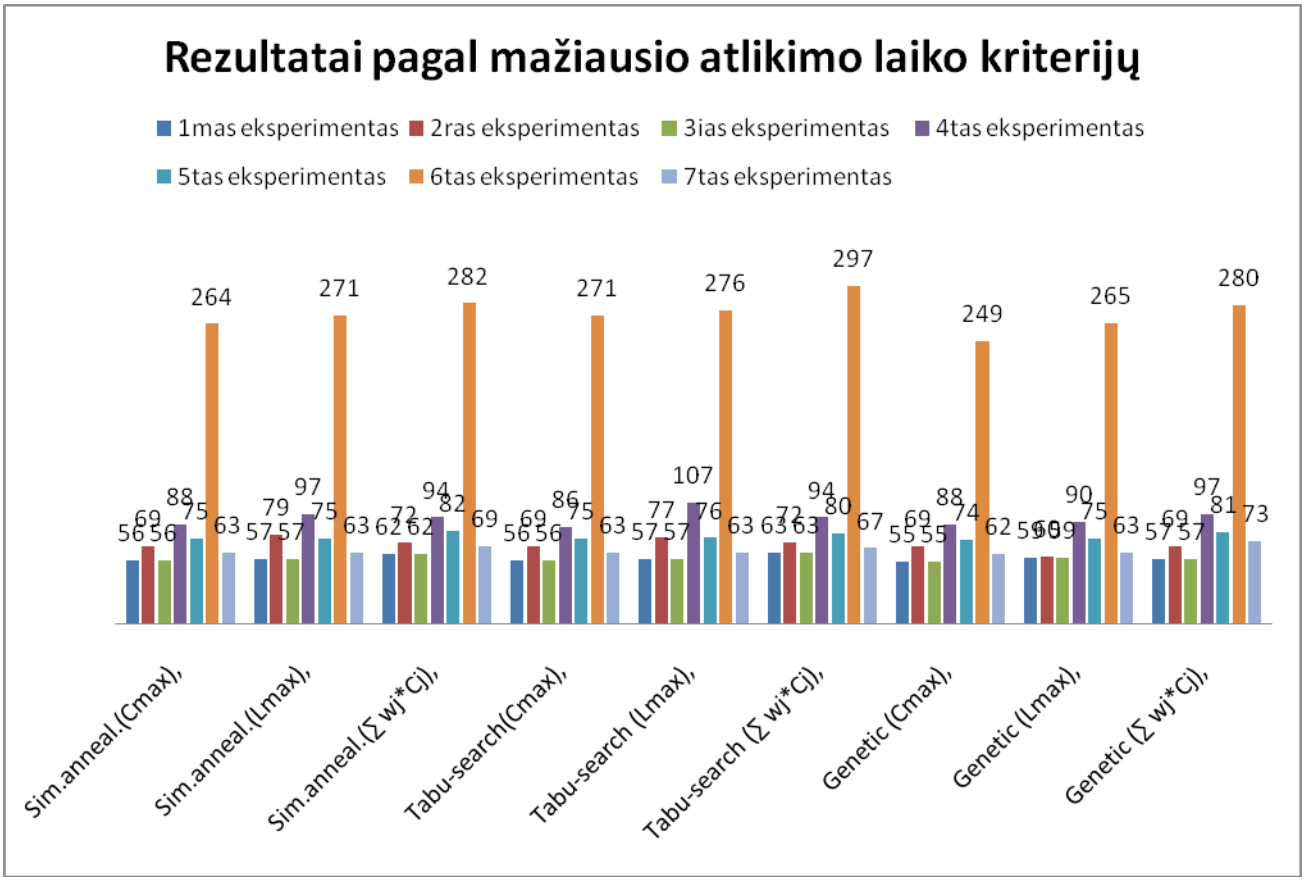
3.14lentelė

Galutiniai modelio duomenys(7 eksperimentas)

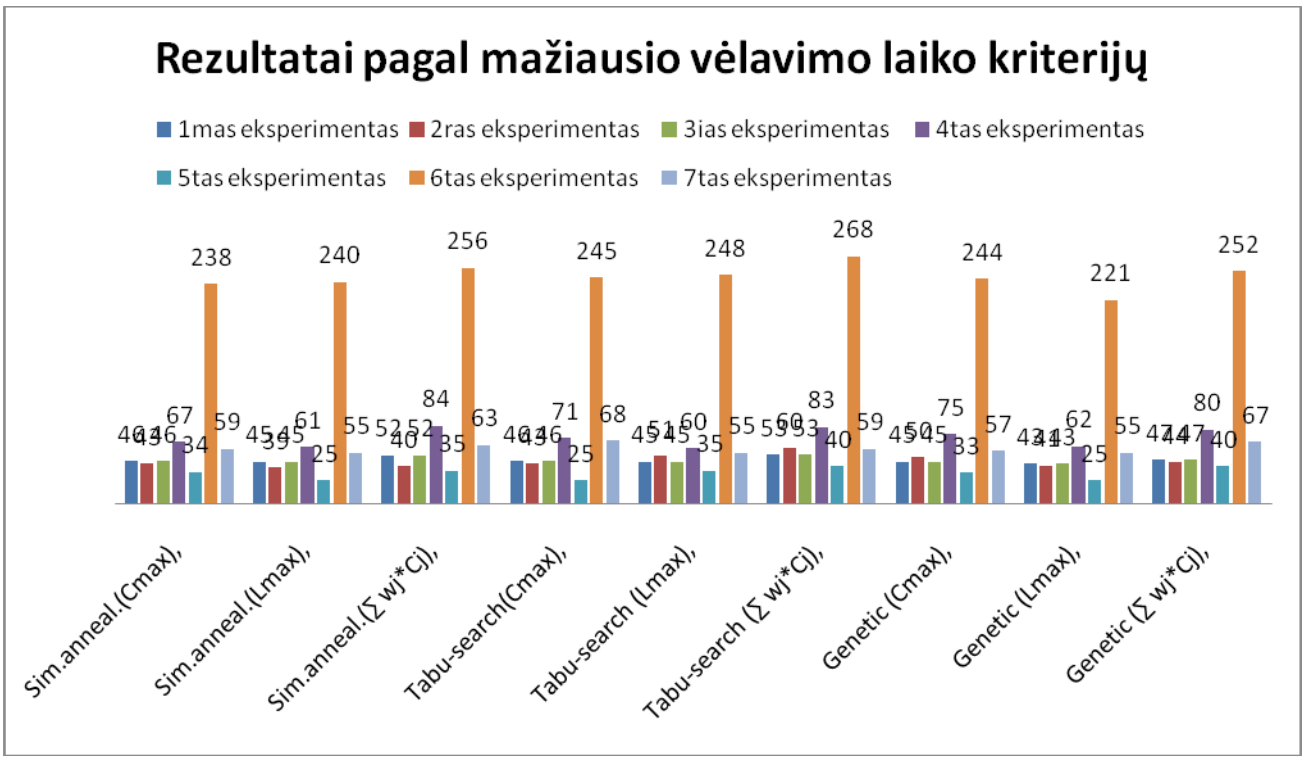
Metodas	Cmax	Lmax	∑ wj*Cj
Sim.anneal.(Cmax),	63	59	1518
Sim.anneal.(Lmax),	63	55	1527
Sim.anneal.(∑ wj*Cj),	69	63	1297
Tabu-search(Cmax),	63	68	1518
Tabu-search (Lmax),	63	55	1492
Tabu-search (∑ wj*Cj),	67	59	1265
Genetic (Cmax),	62	57	1526
Genetic (Lmax),	63	55	1451
Genetic (∑ wj*Cj),	73	67	1236

Žali langeliai parodo geriausia rezultatą tam kriterijui.

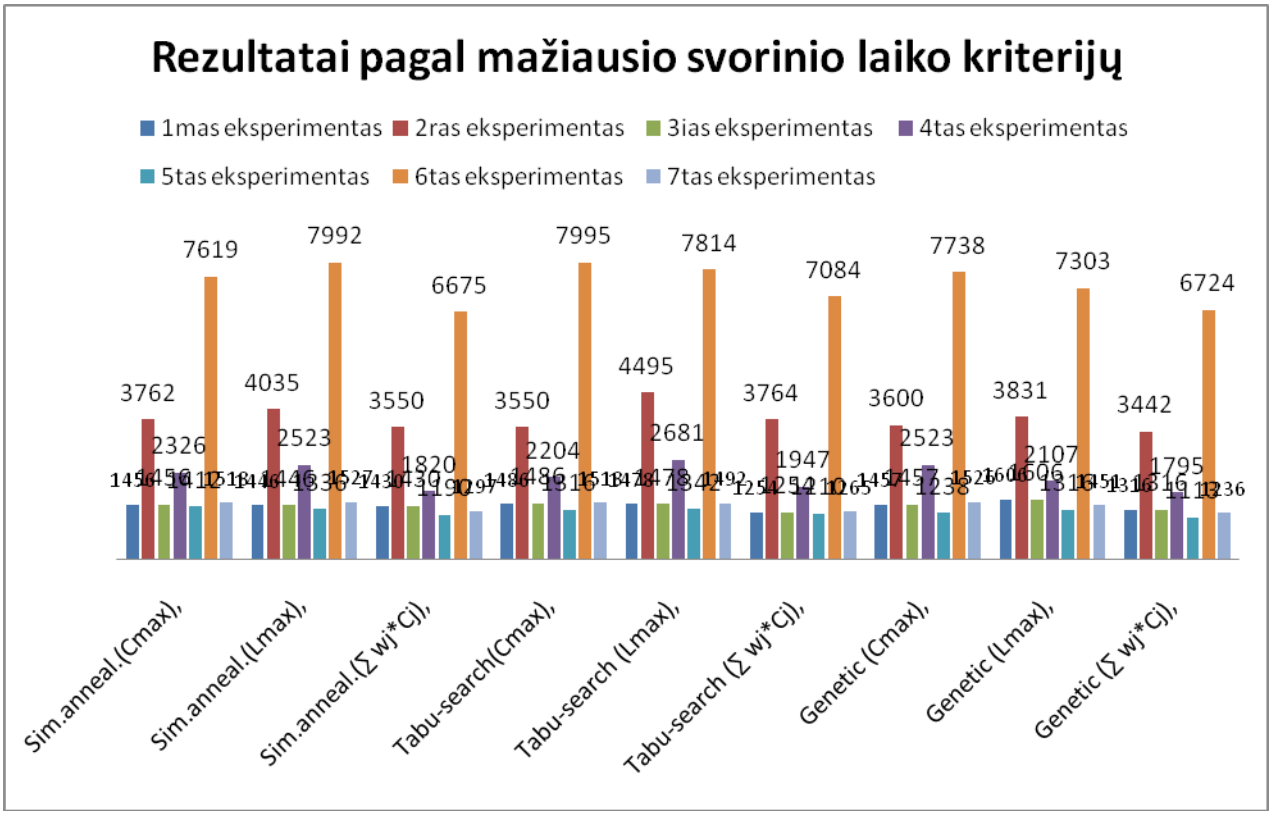
Toliau pateiksiu visų eksperimentų rezultatus grafiškai pagal kriterijus.



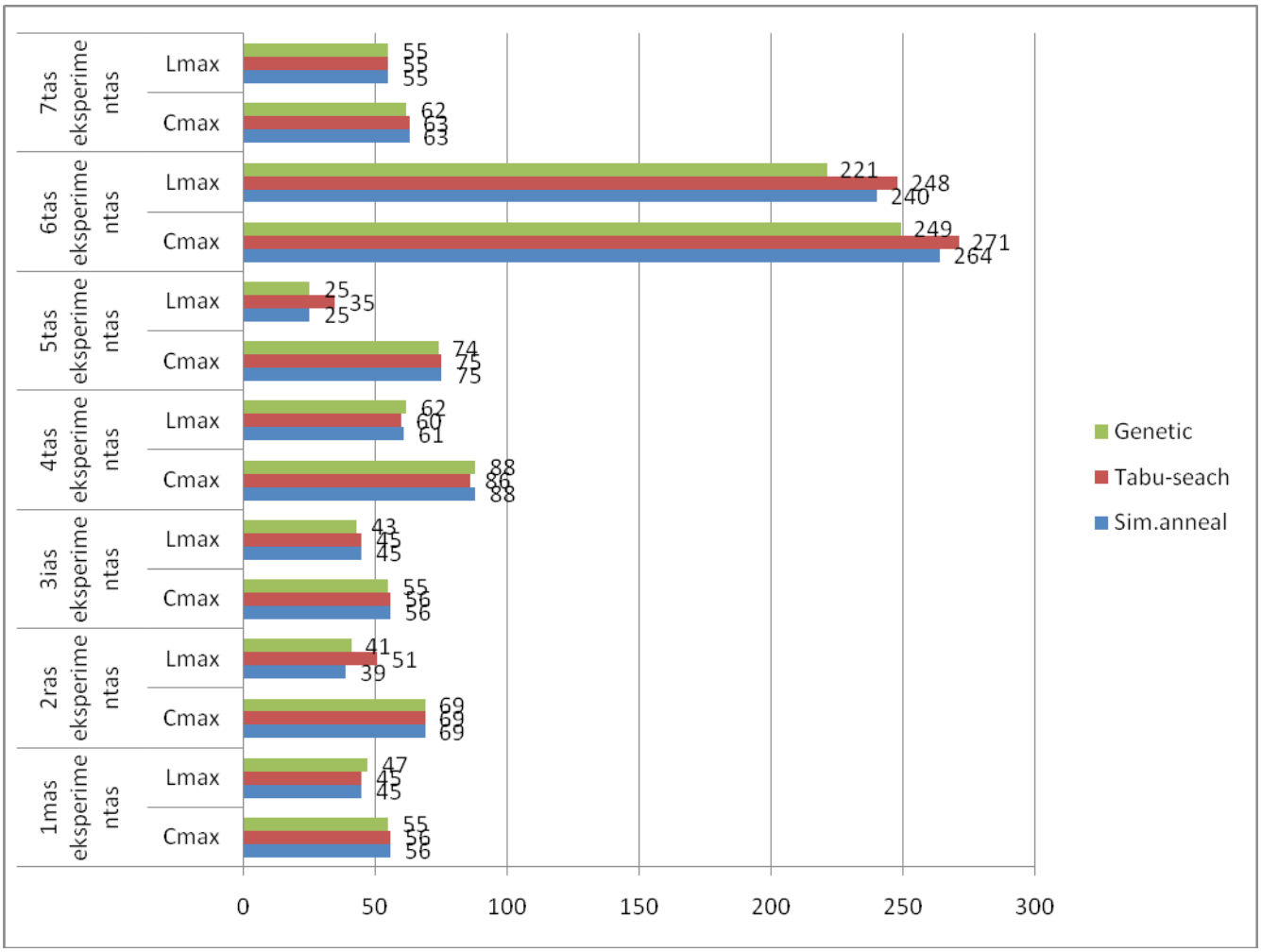
3.6 pav. Eksperimentų rezultatų diagramos (Cmax)



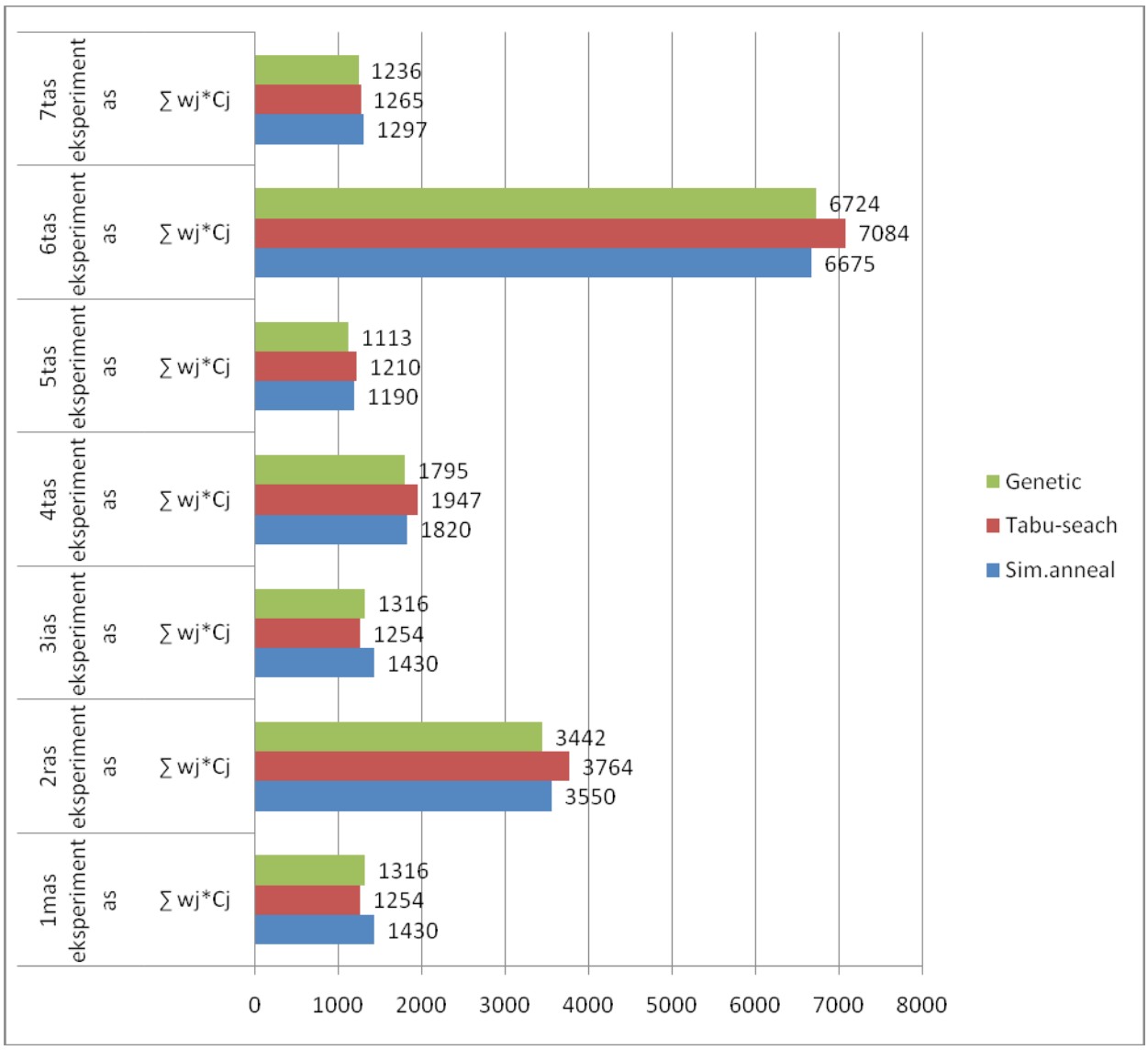
3.7 pav. Eksperimentų rezultatų diagramos (Lmax)



3.8 pav. Eksperimentų rezultatų diagramos (∑ wj*Cj)



3.9 pav. Eksperimentų rezultatų diagramos liginant algoritmus(Cmax ir Lmax)



3.10 pav. Eksperimentų rezultatų diagramos liginant algoritmus(∑ wj*Cj)

IŠVADOS

- Dažniausiai geriausius rezultatus pateikia Genetinis algoritmas, nesvarbu koks būtų kriterijus.
- Tabu-paieškos algoritmas pateikdavo prasčiausius tvarkaraščius.
- Kuo tvarkaraštyje mažiau darbų ir mašinų, tuo mažiau iteracijų reikia ir visi algoritmai randa beveik vienodus tvarkaraščius vertinant pagal optimumo kriterijus.
- Sudėtingiausias yra bendras svorinis atlikimo laiko kriterijus – tvarkaraščiams pagal šį kriterijų sudarinėti reikia daugiausiai iteracijų.
-Našiausias optimumo kriterijus yra mažiausio vėlavimo kriterijus – sudarinėjant tvarkaraščius pagal šį kriterijų, kiti kriterijai gaunami taip pat neblogi.

Ateityje dar būtų galima šia analizę plėsti:

- Įvedant daugiau algoritmų tvarkaraščiam sudarinėti.
- Naudojant daug daugiau duomenų, atitinkamai ir daug daugiau iteracijų.
- Dirbtinio atrinkimo metode keisti *temperatūros* parametrą, bei tvarkaraščio kandidato iš prieš tai buvusio tvarkaraščio aplinkos paieškos algoritmą.
- Tabu paieškoje optimaliai suderinti mutacijų sąrašus.
- Genetiniam algoritme rasti geriausia palikuoniu atrinkimo būdą.

ŽODYNAS

- Scheduling – tvarkaraščio sudarymas
- Cutting stock problem – Pjaustymo kiekio problema
- Flow shop – nuoseklus fabrikas
- Job shop – darbo fabrikas
- Open shop – atviras fabrikas
- *First In First Out* (FIFO) – „Pirmas Įeina Pirmas Išeina“. Maršruto tipas.
- Makespan – atlikimo laikas
- Processing time – proceso laikas. Tai proceso laikas kurį tam tikras darbas užtrunka ant tam tikros mašinos.
- Release date – Paleidimo laikas. Tai tam tikro darbo atvykimo laikas į sistemą. Tai yra laikas, kada darbas gali pradėti procesą.
- Due date – numatytas laikas. Tai laikas, per kurį tam tikras darbas turi būti baigtas.
- Slope heuristic – nuožulni euristika
- Lateness– vėlavimas.
- Simulated annealing – dirbtinis atrinkimas
- Node - mazgas
- Direct graph - tiesioginis grafas
- Mixed Integer Problem – Sumaišytų programų visuma

LITERATŪRA

1. Nuoroda į knyga „The Truck Dispatching Problem“, parašyta G. B. Dantzig ir J. H. Ramser 1959metais. Prieiga per intrenetą: <http://www.jstor.org/pss/2627477 >

2. Michael L. Pinedo „Scheduling Theory,algorithms, and systems“ third edition, New York 2008.

3. International Journal of Production Research Vol. 43, No. 14, 15 July 2005, 2895–2929 „Flowshop-scheduling problems with makespan criterion: a review“.

4. Blonskis, J., V. Bukšnaitis, A. Misevičius, D. Rubliauskas. C++ Builder grafika. Smaltija, Kaunas, 2004

5. Blonskis, J., V. Bukšnaitis, V. Jusas, R. Marcinkevičius, J. Smolinskas. Programavimo C++ Builder pavyzdžiai. Smaltija, Kaunas, 2002

6. Glover, F. Tabu Search: A Tutorial / Interfaces 20:4, July-August 1990, p. 74-94.

7. Goldberg, D. E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989, 372 p.

PRIEDAS

Unit.h

```
//---------------------------------------------------------------------------

#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <jpeg.hpp>

#include <fstream>
#include <iomanip>
using namespace std;

const char *tvark = "tvark.txt";
const char *duom = "duom.txt";
const char *rez = "rez.txt";
const char *rez1 = "rez1.txt";
const char *iter = "iter.txt";

const int Cn = 50;
const int Cm = 50;
const int xx = 20;
const int yy = 30; //masteliai


//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:          // IDE-managed Components
      TButton *Button1;
      TButton *Button2;
      TButton *Button3;
      TButton *Button4;
      TButton *Button5;
      TButton *Button6;
      TMemo *Memo1;
      TMemo *Memo2;
      TEdit *Edit1;
      TImage *Image1;
      TButton *Button7;
      TButton *Button8;
      TButton *Button9;
```

```
        TButton *Button10;
        TButton *Button11;
        TButton *Button12;
        TEdit *Edit2;
        TEdit *Edit3;
        TLabel *Label1;
        TImage *Image2;
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall Button3Click(TObject *Sender);
        void __fastcall Button4Click(TObject *Sender);
        void __fastcall Button5Click(TObject *Sender);
        void __fastcall Button6Click(TObject *Sender);
        void __fastcall Button7Click(TObject *Sender);
        void __fastcall Button8Click(TObject *Sender);
        void __fastcall Button9Click(TObject *Sender);
        void __fastcall Button10Click(TObject *Sender);
        void __fastcall Button12Click(TObject *Sender);
        void __fastcall Button11Click(TObject *Sender);
private:                // User declarations
        //paisimo funkcijos
        int J[Cn][Cm];
        int P[Cn][Cm];
        int G[Cn][Cm];
        int nr[Cn];
        int ilgis;
        int plotis;
        //baigiasi paisimo funkcijos

        int m; //masinu kiekis
        int d; //darbu kiekis


        int Cmax;
        int Lmax;
        int wC;
        int iter;


        int K[Cn];
        int Ind[Cn];
        int wj[Cn]; //svoris
        int Cj[Cn]; //darbo baigimo laikas
        int Lj[Cn]; // velavimas


        int dj[Cn]; //numatytas laikas
        int pj[Cn][Cm]; //pricesavimo laikai

        void Nuskaityti();
        void Nuskaityti1();
        void Paisyti();
        void Slope();
```

```
        void Gupta();
        void Surasyti();
        void Sim();
        void SimL();
        void SimwC();
        void Tabu();
        void TabuL();
        void TabuwC();
        void Gen();
        bool MutacijuSar();
        void Pri(int A[][99],int B[],int eil);    //prisikirt masyvo B reiksmes masyvui A
        void Pri1(int B[],int A[][99],int eil);
        void Pri2(int A[][99],int B[][99],int eil,int eill);


        String spalva(int sk);

public:                                 // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

Unit.cpp

```
#include <vcl.h>
#include <math.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
ofstream fr(rez);
fr.close();

ofstream fr11("iter.txt");
fr11.close();

}

//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
 Nuskaityti();
 Slope();
 Nuskaityti1();
 Paisyti();
 Memo2->Lines->LoadFromFile(tvark);
 Memo1->Lines->LoadFromFile(rez);
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
Close();
}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
void TForm1::Nuskaityti1()
{
      ilgis = 0;

      ifstream fd(tvark);
      fd >> m >> d;
      for (int i = 1; i <= m ; i++)
            {
             fd >> nr[i];

             for (int j = 1; j <= d ; j++)
               {
               fd >> P[i][j];
               P[i][j] = 50 + P[i][j] * 20;
               fd >> G[i][j];
               G[i][j] = 50 + G[i][j] * 20;
               fd >> J[i][j];

                 if ( G[i][j] > ilgis)
                    ilgis = G[i][j];
               }
            }
      fd.close();
}
//---------------------------------------------------------------------------
String TForm1::spalva(int sk)
{
      while ( sk > 10 )
            sk = sk - 10;

      if (sk == 1)
            { return "clBlue"; }

      if (sk == 2)
            { return "clRed";  }
      if (sk == 3)
            { return "clGreen";  }
```

```
      if (sk == 4)
           { return "clMaroon";  }
      if (sk == 5)
           { return "clOlive";  }
      if (sk == 6)
           { return "clPurple";  }
      if (sk == 7)
           { return "clNavy";  }
      if (sk == 8)
           { return "clTeal";  }
      if (sk == 9)
           { return "clGray";  }
      if (sk == 10)
           { return "clFuchsia";  }
      return "clSilver";

}
//---------------------------------------------------------------------------
void TForm1::Paisyti()
{
 Image1->Picture = NULL;
 String sp;
 int y = 0;
 for (int i = 1; i <= m ; i++)
 {
      Image1->Canvas->TextOut(0,y+7,IntToStr(nr[i])+" masina");

      for (int j = 1; j <= d ; j++)
      {
      sp = spalva(J[i][j]);
      Image1->Canvas->Brush->Color = StringToColor(sp);
      Image1->Canvas->Rectangle(P[i][j], y, G[i][j],y+30);
      Image1->Canvas->Brush->Color = clWhite;
      Image1->Canvas->TextOut((P[i][j]+G[i][j])/2-5,y+7,IntToStr((G[i][j]-P[i][j])/20)+" ");

       if (i > 1)
       {
       Image1->Canvas->MoveTo(P[i][j], y);
       Image1->Canvas->LineTo(G[i-1][j], y-20);
       }

      }
      y = y + 50;
      Image1->Canvas->Brush->Color = clWhite;
 }
 y = y - 15;
 int a = 5;
 int x = 50;
 int xx = 10;
 Image1->Canvas->MoveTo(50, y);
 Image1->Canvas->LineTo(50, y-a);
 Image1->Canvas->TextOut(50-5,y,IntToStr(xx-10));
```

```
  while (x <= ilgis)
      {
      Image1->Canvas->TextOut(x+200-5,y,IntToStr(xx));
      Image1->Canvas->TextOut(x+100-5,y,IntToStr(xx-5));

      Image1->Canvas->MoveTo(x, y);
      Image1->Canvas->LineTo(x+200, y);

      Image1->Canvas->MoveTo(x+100, y);
      Image1->Canvas->LineTo(x+100, y-a);

      Image1->Canvas->MoveTo(x+200, y);
      Image1->Canvas->LineTo(x+200, y-a);
      x = x + 200;
      xx = xx + 10;
      }


  ilgis = x+15;
  plotis = y+15;

  Image1->Height = plotis;
  Image1->Width = ilgis ;
}
//---------------------------------------------------------------------------
void TForm1::Nuskaityti()
{
  ifstream fd(duom);
      fd >> m >> d;
      fd.ignore(97); //ignoruoja aprasimo eilute

      for (int i = 1; i <= d ; i++)
        {

        fd >> wj[i]; fd >> dj[i];
        for (int j = 1; j <= m ; j++)
          { fd >> pj[j][i]; }

        }
      fd.close();
}
//---------------------------------------------------------------------------
void TForm1::Slope()
{
  int Aj[99]; //slope indeksas

  int max; int ind; ind = 0;

  for (int i = 1; i <= d ; i++)
    {
    Aj[i]=0;
    for (int j = 1; j <= m ; j++)
    {
```

```
     Aj[i] = Aj[i] - (( m - ( 2 * j - 1 )) * pj[j][i]);//skaiciuoja slope indeksa
    }
   }
  for (int i = 1; i <= d ; i++)
   {

   max = -9999;
   for (int j = 1; j <= d ; j++)
    {
     if ( max < Aj[j])
       {
       ind = j;
       max = Aj[j];

       }
    }
   Aj[ind] = -9999;
   Ind[i] = ind;   //randame darbu eiliskuma
   }
   Surasyti();


  ofstream frr(rez,ios::app);
  frr << "Metodas: Nuožulni euristika (Slope heuristic), sudaro tvarkaraštį pagal Cmax"<<endl;
  frr << "Kriterijai:"<<endl;
  frr<< "Atlikimo laikas (Cmax, makespan): "<< Cmax << endl;
  frr<< "Didžiausias vėlavimo laikas  (Lmax, maximum lateness): "<< Lmax << endl;
  frr<< "Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): "<< wC << endl;
  frr<< "Darbų seka: ";
  for (int i =1; i <= d;i++)
       frr<< Ind[i] << " ";
  frr<< endl<<endl;
  frr.close();

}
//---------------------------------------------------------------------------
void TForm1::Gupta()
{
  double SI[99]; //Guptos indeksas
  int e[99];
  int min;int max; int ind; ind = 0;

  for (int i = 1; i <= d ; i++)
    {
    SI[i]=0;

    if ( pj[1][i] < pj[m][i])
      e[i] = 1;
      else e[i] = -1;

    min = 9999;

    for (int j = 1; j <= m-1 ; j++)
```

```
    {
     if ( min > ( pj[j][i] + pj[j+1][i]))
       min = pj[j][i] + pj[j+1][i];

    }
    SI[i] =  e[i]/ min;     //skaiciuojam gupta indeksa
   }
  for (int i = 1; i <= d ; i++)
   {

   max = -9999;
   for (int j = 1; j <= d ; j++)
     {
      if ( max < SI[j])
        {
        ind = j;
        max = SI[j];

        }
     }
   SI[ind] = -9999;
   Ind[i] = ind;   //randame darbu eiliskuma
   }
  Surasyti();


  ofstream frr(rez,ios::app);
  frr << "Metodas: Gupta euristika (Gupta heuristic), sudaro tvarkaraštį pagal Cmax"<<endl;
  frr << "Kriterijai:"<<endl;
  frr<< "Atlikimo laikas (Cmax, makespan): "<< Cmax << endl;
  frr<< "Didžiausias vėavimo laikas (Lmax, maximum lateness): "<< Lmax << endl;
  frr<< "Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): "<< wC << endl;
  frr<< "Darbų seka: ";
  for (int i =1; i <= d;i++)
       frr<< Ind[i] << " ";
  frr<< endl<<endl;
  frr.close();

}
//---------------------------------------------------------------------------
void TForm1::Surasyti()

{
  int jj;
  K[0] = 0;
  ofstream fr(tvark);
  fr << m << "   "<< d <<endl;
  fr << 1 << endl;

  for (int j = 1; j <= d ; j++)
    {
    K[j] = 0;
    jj = Ind[j];
```

```
   fr << K[j-1] << " "  << K[j-1] + pj[1][jj]<< " "<< jj << " " <<endl;
   K[j] = K[j-1] + pj[1][jj];
   //fr << K[j] << endl;
   }

  for (int i = 2; i <= m ; i++)
   {
   fr << i << endl;
   for (int j = 1; j <= d ; j++)
     {
     jj = Ind[j];

     if ( K[j-1] < K[j])
       {
       fr << K[j] << " "  << K[j] + pj[i][jj]<< " "<< jj << " " <<endl;
       K[j]= K[j] + pj[i][jj];
       }
     else
       {
       fr << K[j-1] << " "  << K[j-1] + pj[i][jj]<< " "<< jj << " " <<endl;
       K[j]= K[j-1] + pj[i][jj];
       }
      Cj[jj]=K[j];              //darbo atlikimas
      Lj[jj]= Cj[jj] - dj[jj];           // darbo velavimas

     }
   }
  Lmax = -999;
  wC = 0;
  for (int j = 1; j <= d ; j++)
    {
    if (Lmax < Lj[j])
      Lmax = Lj[j];
    wC = wC + Cj[j]*wj[j];

    }

  for (int j1 = 1; j1 <= d ; j1++)
  {
   fr<< endl;
   fr << " Cj:  "<< Cj[j1] << ", Lj:  "<< Lj[j1];
  }

  Cmax = K[d];
  fr<<endl<< " Sum(wj*Cj):  "<< wC;


  fr.close();

}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
```

```
  Nuskaityti();
  Gupta();
  Nuskaityti1();
  Paisyti();
  Memo2->Lines->LoadFromFile(tvark);
  Memo1->Lines->LoadFromFile(rez);

}

//---------------------------------------------------------------------------
void TForm1::Sim()
{
  int k; k = 1;
  double a;  a = 0.5;
  a = StrToFloat(Edit2->Text);
  double U; U = 0.8;
  U = StrToFloat(Edit3->Text);

  iter = StrToInt(Edit1->Text);
  int S[999][99];

  int laik;

  int j; j = 1;

  int krit; int kritl;
  bool sedi; sedi = false;



  for (int i = 1; i <=d ; i++)
    {
    S[1][i]=i;
    S[0][i]=i;
    }
  Pri1(Ind,S,1);
  Surasyti();
  krit = Cmax;
  kritl = Cmax;
  Pri2(S,S,2,1);

  for (int i = 2; i <= iter; i++)
  {

  sedi = false;

  laik = S[i][j];
  S[i][j] = S[i][j+k];
  S[i][j+k] = laik;
  Pri1(Ind,S,i);
  Surasyti();
  if (Cmax < krit)
       {
```

```
    krit = Cmax;        //naujas kriterijus
    Pri2(S,S,0,i);      //priskiriam S0
    Pri2(S,S,i+1,i);
    kritl = Cmax;
    sedi = true;
    }
else
    {
    if (Cmax < kritl)
        {
        Pri2(S,S,i+1,i);
        kritl = Cmax;
        sedi = true;
        }
     else
        {
        if (U <= exp ((kritl-Cmax)/(pow(a,i))))
            {
            Pri2(S,S,i+1,i);

            }
        else
            Pri2(S,S,i+1,i-1);

        }
    }



if (sedi)
{
    j = 1;
    k = 1;
}
else
    k++;

if ((j+k)> d)
    {
    k = 1;
    j++;
    }

 if ((j+k)> d)
    {
    j = 1;
    k = 1;
    }
ofstream fr11("iter.txt",ios::app);
fr11<<krit<<endl;
fr11.close();

}
```

```
 Pri1(Ind,S,0);
 Surasyti();
 Nuskaityti1();
 Paisyti();




 ofstream frr(rez,ios::app);
 frr << "Metodas: Dirbtinis atrinkimas (Simulated anealing), sudaro tvarkaraštį pagal Cmax"<<endl;
 frr << "Kriterijai:"<<endl;
 frr<< "Atlikimo laikas (Cmax, makespan): "<< Cmax << endl;
 frr<< "Didžiausias vėlavimo laikas (Lmax, maximum lateness): "<< Lmax << endl;
 frr<< "Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): "<< wC << endl;
 frr<< "Darbų seka: ";
 for (int i =1; i <= d;i++)
     frr<< Ind[i] << " ";
 frr<< endl<<endl;
 frr.close();

  ofstream fr(rez1);
  fr<< iter <<endl;
  fr<< krit <<endl;
  for (int i = 1; i <=d ; i++)
   {
   fr<< S[0][i]<< " ";
   }
  fr<<endl;

  for (int i = 1; i <=d ; i++)
   {
   fr<< Ind[i]<< " ";
   }
   fr<<endl;

  for (int i = 1; i <=d ; i++)
   {
   fr<< Cj[i]<< " ";
   }
   fr<<endl;

   fr<<endl;

  fr.close();



}
//---------------------------------------------------------------------------
void TForm1::Pri(int A[][99],int B[],int eil)
{
```

```
 for (int i = 1; i <=d ; i++)
 {
  A[eil][i]=B[i];
 }

}
//---------------------------------------------------------------------------
void TForm1::Pri1(int B[],int A[][99],int eil)
{
 for (int i = 1; i <=d ; i++)
 {
  B[i]=A[eil][i];
 }

}
//---------------------------------------------------------------------------


void TForm1::Pri2(int A[][99],int B[][99],int eil,int eill)
{
 for (int i = 1; i <=d ; i++)
 {
  A[eil][i]=B[eill][i];
 }

}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button4Click(TObject *Sender)
{
  Nuskaityti();
  Sim();
  Memo2->Lines->LoadFromFile(rez1);
  Memo1->Lines->LoadFromFile(rez);

}

//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
  Nuskaityti();
  SimL();
  Memo2->Lines->LoadFromFile(rez1);
  Memo1->Lines->LoadFromFile(rez);

}

//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
void TForm1::SimL()
{
  int k; k = 1;
  double a; a = 0.5; double U; U = 0.8;
```

```
iter = StrToInt(Edit1->Text);
int S[999][99];


int laik;

int j; j = 1;


int krit; int kritl;
bool sedi; sedi = false;



for (int i = 1; i <=d ; i++)
  {
  S[1][i]=i;
  S[0][i]=i;
  }
Pri1(Ind,S,1);
Surasyti();
krit = Lmax;
kritl = Lmax;
Pri2(S,S,2,1);


for (int i = 2; i <= iter; i++)
{

sedi = false;

laik = S[i][j];
S[i][j] = S[i][j+k];
S[i][j+k] = laik;
Pri1(Ind,S,i);
Surasyti();

if (Lmax < krit)
     {
     krit = Lmax;        //naujas kriterijus
     Pri2(S,S,0,i);      //priskiriam S0
     Pri2(S,S,i+1,i);
     kritl = Lmax;
     sedi = true;
     }
 else
     {
     if (Lmax < kritl)
          {
          Pri2(S,S,i+1,i);
          kritl = Lmax;
          sedi = true;
```

```
            }
        else
            {
            if (U <= exp ((kritl-Lmax)/(pow(a,i))))
                {
                Pri2(S,S,i+1,i);

                }
            else
                Pri2(S,S,i+1,i-1);

            }
        }

    if (sedi)
        j = 1;
    else
        j++;

    if ((j+k)> d)
        {
        j = 1;
        k++;
        }

     if ((j+k)> d)
        {
        j = 1;
        k = 1;
        }

}

Pri1(Ind,S,0);
Surasyti();
Nuskaityti1();
Paisyti();

ofstream frr(rez,ios::app);
frr << "Metodas: Dirbtinis atrinkimas (Simulated anealing), sudaro tvarkaraštį pagal Lmax"<<endl;
frr << "Kriterijai:"<<endl;
frr<< "Atlikimo laikas (Cmax, makespan): "<< Cmax << endl;
frr<< "Didžiausias vėlavimo laikas (Lmax, maximum lateness): "<< Lmax << endl;
frr<< "Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): "<< wC << endl;
frr<< "Darbų seka: ";
for (int i =1; i <= d;i++)
    frr<< Ind[i] << " ";
frr<< endl<<endl;
frr.close();


 ofstream fr(rez1);
 fr<< iter <<endl;
```

```
  fr<< krit <<endl;
  for (int i = 1; i <=d ; i++)
   {
   fr<< S[0][i]<< " ";
   }
  fr<<endl;

  for (int i = 1; i <=d ; i++)
   {
   fr<< Ind[i]<< " ";
   }
   fr<<endl;

  for (int i = 1; i <=d ; i++)
   {
   fr<< Cj[i]<< " ";
   }
   fr<<endl;

   fr<<endl;

  fr.close();

}

//---------------------------------------------------------------------------

void __fastcall TForm1::Button6Click(TObject *Sender)
{
 Nuskaityti();
 SimwC();
 Memo2->Lines->LoadFromFile(rez1);
 Memo1->Lines->LoadFromFile(rez);

}

//---------------------------------------------------------------------------
void __fastcall TForm1::Button10Click(TObject *Sender)
{
 Nuskaityti();
 Gupta();
 Nuskaityti1();
 Paisyti();
 Memo2->Lines->LoadFromFile(tvark);
 Memo1->Lines->LoadFromFile(rez);
 Nuskaityti();
 SimL();
 Memo2->Lines->LoadFromFile(rez1);
 Memo1->Lines->LoadFromFile(rez);
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Button12Click(TObject *Sender)
{
  Nuskaityti();
  SimL();
  Memo2->Lines->LoadFromFile(rez1);
  Memo1->Lines->LoadFromFile(rez);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button11Click(TObject *Sender)
{
  Nuskaityti();
  Slope();
  Nuskaityti1();
  Paisyti();
  Memo2->Lines->LoadFromFile(tvark);
  Memo1->Lines->LoadFromFile(rez);
}
//---------------------------------------------------------------------------
void TForm1::SimwC()
{
  int k; k = 1;
  double a; a = 0.5; double U; U = 0.8;

  iter = StrToInt(Edit1->Text);
  int S[999][99];
  int laik;
  int j; j = 1;
  int krit; int kritl;
  bool sedi; sedi = false;
  for (int i = 1; i <=d ; i++)
    {
    S[1][i]=i;
    S[0][i]=i;
    }
  Pri1(Ind,S,1);
  Surasyti();
  krit = wC;
  kritl = wC;
  Pri2(S,S,2,1);

  for (int i = 2; i <= iter; i++)
  {

  sedi = false;

  laik = S[i][j];
  S[i][j] = S[i][j+k];
  S[i][j+k] = laik;
  Pri1(Ind,S,i);
  Surasyti();

  if (wC < krit)
       {
```

```
    krit = wC;       //naujas kriterijus
    Pri2(S,S,0,i);     //priskiriam S0
    Pri2(S,S,i+1,i);
    kritl = wC;
    sedi = true;
    }
 else
    {
    if (wC < kritl)
          {
          Pri2(S,S,i+1,i);
          kritl = wC;
          sedi = true;
          }
     else
          {
          if (U <= exp ((kritl-wC)/(pow(a,i))))
               {
               Pri2(S,S,i+1,i);
               //sedi = true;
               }
          else
               Pri2(S,S,i+1,i-1);

          }
    }

 if (sedi)
    j = 1;
 else
    j++;

 if ((j+k)> d)
    {
    j = 1;
    k++;
    }

  if ((j+k)> d)
    {
    j = 1;
    k = 1;
    }
}

 Pri1(Ind,S,0);
 Surasyti();
 Nuskaityti1();
 Paisyti();

 ofstream frr(rez,ios::app);
frr << "Metodas: Dirbtinis atrinkimas (Simulated anealing), sudaro tvarkaraštį pagal wC"<<endl;
 frr << "Kriterijai:"<<endl;
```

```
  frr<< "Atlikimo laikas (Cmax, makespan): "<< Cmax << endl;
  frr<< "Didžiausias vėlavimo laikas (Lmax, maximum lateness): "<< Lmax << endl;
  frr<< "Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): "<< wC << endl;
  frr<< "Darbų seka: ";
  for (int i =1; i <= d;i++)
      frr<< Ind[i] << " ";
  frr<< endl<<endl;
  frr.close();


   ofstream fr(rez1);
   fr<< iter <<endl;
   fr<< krit <<endl;
   for (int i = 1; i <=d ; i++)
    {
    fr<< S[0][i]<< " ";
    }
   fr<<endl;

   for (int i = 1; i <=d ; i++)
    {
    fr<< Ind[i]<< " ";
    }
    fr<<endl;

   for (int i = 1; i <=d ; i++)
    {
    fr<< Cj[i]<< " ";
    }
    fr<<endl;

    fr<<endl;

   fr.close();

}
//---------------------------------------------------------------------------
void TForm1::Tabu()
{
  int k; k = 1;

  iter = StrToInt(Edit1->Text);
  int S[999][99];

  int laik;

  int j; j = 1;

  int krit; int kritl;
  bool sedi; sedi = false;
```

```
for (int i = 1; i <=d ; i++)
  {
  S[1][i]=i;
  S[0][i]=i;
  }
Pri1(Ind,S,1);
Surasyti();
krit = Cmax;
kritl = Cmax;
Pri2(S,S,2,1);

for (int i = 2; i <= iter; i++)
{
sedi = false;
laik = S[i][j];
S[i][j] = S[i][j+k];
S[i][j+k] = laik;
Pri1(Ind,S,i);
Surasyti();
if (Cmax < krit)
     {
     krit = Cmax;        //naujas kriterijus
     Pri2(S,S,0,i);      //priskiriam S0
     Pri2(S,S,i+1,i);
     kritl = Cmax;
     sedi = true;
     }
 else
     {
     if (Cmax < kritl)
          {
          Pri2(S,S,i+1,i);
          kritl = Cmax;
          sedi = true;
          }
      else
          {
          if (MutacijuSar())
               {
               Pri2(S,S,i+1,i-1);
               }
          else
               Pri2(S,S,i+1,i-1);
          }
     }
j = rand() % (d-1) + 1;
k = 1;
}
Pri1(Ind,S,0);
Surasyti();
Nuskaityti1();
Paisyti();
ofstream frr(rez,ios::app);
```

```
 frr << "Metodas: Tabu paieska (Tabu-Search), sudaro tvarkarašti pagal Cmax"<<endl;
 frr << "Kriterijai:"<<endl;
 frr<< "Atlikimo laikas (Cmax, makespan): "<< Cmax << endl;
 frr<< "Didžiausias vėlavimo laikas (Lmax, maximum lateness): "<< Lmax << endl;
 frr<< "Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): "<< wC << endl;
 frr<< "Darbu seka: ";
 for (int i =1; i <= d;i++)
     frr<< Ind[i] << " ";
 frr<< endl<<endl;
 frr.close();

  ofstream fr(rez1);
  fr<< iter <<endl;
  fr<< krit <<endl;
  for (int i = 1; i <=d ; i++)
   {
   fr<< S[0][i]<< " ";
   }
  fr<<endl;

  for (int i = 1; i <=d ; i++)
   {
   fr<< Ind[i]<< " ";
   }
   fr<<endl;

  for (int i = 1; i <=d ; i++)
   {
   fr<< Cj[i]<< " ";
   }
   fr<<endl;

   fr<<endl;

  fr.close();



}
//---------------------------------------------------------------------------
void TForm1::TabuL()
{
  int k; k = 1;


  iter = StrToInt(Edit1->Text);
  int S[999][99];


  int laik;

  int j; j = 1;
```

```
int krit; int kritl;
bool sedi; sedi = false;



for (int i = 1; i <=d ; i++)
  {
  S[1][i]=i;
  S[0][i]=i;
  }
Pri1(Ind,S,1);
Surasyti();
krit = Lmax;
kritl = Lmax;
Pri2(S,S,2,1);


for (int i = 2; i <= iter; i++)
{

sedi = false;

laik = S[i][j];
S[i][j] = S[i][j+k];
S[i][j+k] = laik;
Pri1(Ind,S,i);
Surasyti();

if (Lmax < krit)
     {
     krit = Lmax;        //naujas kriterijus
     Pri2(S,S,0,i);      //priskiriam S0
     Pri2(S,S,i+1,i);
     kritl = Lmax;
     sedi = true;
     }
 else
     {
     if (Lmax < kritl)
          {
          Pri2(S,S,i+1,i);
          kritl = Lmax;
          sedi = true;
          }
      else
          {
          if (MutacijuSar())
               {
               Pri2(S,S,i+1,i);

               }
          else
```

```
                Pri2(S,S,i+1,i-1);

            }
        }

j = rand() % d + 1;
k=-1;

}

Pri1(Ind,S,0);
Surasyti();
Nuskaityti1();
Paisyti();

ofstream frr(rez,ios::app);
frr << "Metodas: Tabu paieska (Tabu-Search), sudaro tvarkarašti pagal Lmax"<<endl;
frr << "Kriterijai:"<<endl;
frr<< "Atlikimo laikas (Cmax, makespan): "<< Cmax << endl;
frr<< "Didžiausias vėlavimo laikas (Lmax, maximum lateness): "<< Lmax << endl;
frr<< "Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): "<< wC << endl;
frr<< "Darbu seka: ";
for (int i =1; i <= d;i++)
    frr<< Ind[i] << " ";
frr<< endl<<endl;
frr.close();


 ofstream fr(rez1);
 fr<< iter <<endl;
 fr<< krit <<endl;
 for (int i = 1; i <=d ; i++)
  {
  fr<< S[0][i]<< " ";
  }
 fr<<endl;

 for (int i = 1; i <=d ; i++)
  {
  fr<< Ind[i]<< " ";
  }
  fr<<endl;

 for (int i = 1; i <=d ; i++)
  {
  fr<< Cj[i]<< " ";
  }
  fr<<endl;

  fr<<endl;

 fr.close();
```

```
}

//---------------------------------------------------------------------------

void __fastcall TForm1::Button7Click(TObject *Sender)
{
 Nuskaityti();
 Tabu();
 Memo2->Lines->LoadFromFile(rez1);
 Memo1->Lines->LoadFromFile(rez);

}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button8Click(TObject *Sender)
{
 Nuskaityti();
 TabuL();
 Memo2->Lines->LoadFromFile(rez1);
 Memo1->Lines->LoadFromFile(rez);
}
//---------------------------------------------------------------------------
void TForm1::TabuwC()
{
 int k; k = 1;


 iter = StrToInt(Edit1->Text);
 int S[999][99];


 int laik;

 int j; j = 1;


 int krit; int kritl;
 bool sedi; sedi = false;



 for (int i = 1; i <=d ; i++)
   {
   S[1][i]=i;
   S[0][i]=i;
   }
 Pri1(Ind,S,1);
 Surasyti();
 krit = wC;
 kritl = wC;
 Pri2(S,S,2,1);

 for (int i = 2; i <= iter; i++)
 {
```

```
sedi = false;

laik = S[i][j];
S[i][j] = S[i][j+k];
S[i][j+k] = laik;
Pri1(Ind,S,i);
Surasyti();

if (wC < krit)
    {
    krit = wC;        //naujas kriterijus
    Pri2(S,S,0,i);      //priskiriam S0
    Pri2(S,S,i+1,i);
    kritl = wC;
    sedi = true;
    }
 else
    {
    if (wC < kritl)
          {
          Pri2(S,S,i+1,i);
          kritl = wC;
          sedi = true;
          }
     else
          {
          if (MutacijuSar())
               {
                Pri2(S,S,i+1,i);

                }
          else
                Pri2(S,S,i+1,i-1);

          }
    }

j = rand() % d + 1;
k=-1;
ofstream fr11("iter.txt",ios::app);
fr11<<krit<<endl;
fr11.close();


}

Pri1(Ind,S,0);
Surasyti();
Nuskaityti1();
Paisyti();

ofstream frr(rez,ios::app);
```

```
 frr << "Metodas: Tabu paieska (Tabu-Search), sudaro tvarkarašti pagal cW"<<endl;
 frr << "Kriterijai:"<<endl;
 frr<< "Atlikimo laikas (Cmax, makespan): "<< Cmax << endl;
 frr<< "Didžiausias vėlavimo laikas (Lmax, maximum lateness): "<< Lmax << endl;
 frr<< "Bendras svorinis atlikimo laikas (Sum(wj*Cj), total wighted completion time): "<< wC << endl;
 frr<< "Darbu seka: ";
 for (int i =1; i <= d;i++)
      frr<< Ind[i] << " ";
 frr<< endl<<endl;
 frr.close();


  ofstream fr(rez1);
  fr<< iter <<endl;
  fr<< krit <<endl;
  for (int i = 1; i <=d ; i++)
   {
   fr<< S[0][i]<< " ";
   }
  fr<<endl;

  for (int i = 1; i <=d ; i++)
   {
   fr<< Ind[i]<< " ";
   }
   fr<<endl;

  for (int i = 1; i <=d ; i++)
   {
   fr<< Cj[i]<< " ";
   }
   fr<<endl;

   fr<<endl;

  fr.close();

}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button9Click(TObject *Sender)
{
 Nuskaityti();
 TabuwC();
 Memo2->Lines->LoadFromFile(rez1);
 Memo1->Lines->LoadFromFile(rez);

}
//---------------------------------------------------------------------------

typedef struct Chrom
  {
   short int bit[6];
     int fit;
```

```
  }chrom;

void *evpop(chrom popcurrent[4]);
int x(chrom popcurrent);
int y(int x);
void *pickchroms(chrom popcurrent[4]);
void *crossover(chrom popnext[4]);
void *mutation(chrom popnext[4]);

void main()
{
  int num;
  int i,j;
  scanf("%d",&num);
  chrom popcurrent[4];
  chrom popnext[4];
  if(num<1)
  evpop(popcurrent);

  for(i=0;i<num;i++)
{
    for(j=0;j<4;j++)
      popnext[j]=popcurrent[j];

    pickchroms(popnext);
    crossover(popnext);
    mutation(popnext);
    for(j=0;j<4;j++)
      popcurrent[j]=popnext[j];

   }
 flushall();
}
void *evpop(chrom popcurrent[4])
{
 int i,j,value;
  int random;
  for(j=0;j<4;j++)
   {
      for(i=0;i<6;i++)

      {
         random=rand();
         random=(random%2);
         popcurrent[j].bit[i]=random;
      }

      value=x(popcurrent[j]);
      popcurrent[j].fit=y(x(popcurrent[j]));

     }
```

```
 return(0);
}

int x(chrom popcurrent)
{
 int z;
  z=(popcurrent.bit[0]*1)+(popcurrent.bit[1]*2)+(popcurrent.bit[2]*4)+(popcurrent.bit[3]*8)+(popcurrent.bit[4]*16);
  if(popcurrent.bit[5]==1)
  z=z*(-1);
   return(z);
 }

int y(int x)
{
 int y;
  y=-(x*x)+5;
  return(y);
}

void *pickchroms(chrom popcurrent[4])
{

 int i,j;
  chrom temp;

   for(i=0;i<3;i++)
     for(j=0;j<3;j++)
       {
          if(popcurrent[j+1].fit>popcurrent[j].fit)
            {
             temp=popcurrent[j+1];
             popcurrent[j+1]=popcurrent[j];
             popcurrent[j]=temp;

            }
       }

    for(i=0;i<4;i++)
 printf("\nSorting:popnext[%d] fitness=%d",i,popcurrent[i].fit);
 printf("\n");
    flushall();
 return(0);
}

void *crossover(chrom popnext[4])
{
 int random;
  int i;
  random=rand();
  random=((random%5)+1);
  for(i=0;i<random;i++)
    {
     popnext[2].bit[i]=popnext[0].bit[i];
```

```
        popnext[3].bit[i]=popnext[1].bit[i];
      }

 for(i=random;i<6;i++)
     {
       popnext[2].bit[i]=popnext[1].bit[i];
       popnext[3].bit[i]=popnext[0].bit[i];
      }

      for(i=0;i<4;i++)
       popnext[i].fit=y(x(popnext[i]));
      for(i=0;i<4;i++)



  return(0);
   }

void *mutation(chrom popnext[4])
{

   int random;
   int row,col,value;
   random=rand()%50;

   if (random==25)
   {
    col=rand()%6;
      row=rand()%4;

     if (popnext[row].bit[col]==0)
      popnext[row].bit[col]=1 ;

      else if (popnext[row].bit[col]==1)
           popnext[row].bit[col]=0;

       popnext[row].fit=y(x(popnext[row]));
       value=x(popnext[row]);

   }

   return(0);
}
```