Kauno technologijos universitetas

Matematikos ir gamtos mokslų fakultetas

Korpusinių baldų gamybos optimalių darbo sekų ir detalių srautų paieška

Baigiamasis magistro studijų projektas

Arnoldas Bankauskas
Projekto autorius

doc. dr. Mindaugas Šnipas
Vadovas

prof. Mantas Vilkas
Vadovas

Kaunas, 2020

Kauno technologijos universitetas

Matematikos ir gamtos mokslų fakultetas

Korpusinių baldų gamybos optimalių darbo sekų ir detalių srautų paieška

Baigiamasis magistro studijų projektas

Didžiųjų verslo duomenų analitika (6213AX001)

Arnoldas Bankauskas
Projekto autorius

doc. dr. Mindaugas Šnipas
Vadovas

prof. Mantas Vilkas
Vadovas

doc. dr. Vilma Petrauskienė
Recenzentė

prof. dr. Vaidas Gaidelys
Recenzentas

Kuanas, 2020

Kauno technologijos universitetas

Matematikos ir gamtos mokslų fakultetas

Arnoldas Bankauskas

Korpusinių baldų gamybos optimalių darbo sekų ir detalių srautų paieška

Akademinio sąžiningumo deklaracija

Patvirtinu, kad mano, Arnoldo Bankausko, baigiamasis projektas tema „Korpusinių baldų gamybos optimalių darbo sekų ir detalių srautų paieška“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

Bankauskas Arnoldas. Korpusinių baldų gamybos optimalių darbo sekų ir detalių srautų paieška. Magistro baigiamasis projektas / vadovas doc. dr. Mindaugas Šnipas / vadovas prof. Mantas Vilkas; Kauno technologijos universitetas, Matematikos ir gamtos mokslų fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Matematika (N 001)

Reikšminiai žodžiai: Klasterizacija, optimizavimas, TSP

Kaunas, 2020. 85 p.

Santrauka

Darbo tikslas – masinės gamybos įmonės gamybos pajėgumų, konkurencinio pranašumo ir finansinės naudos didinimas.

Šiame darbe atlikta literatūros analizė, susisteminti organizaciniai ir matematiniai metodai skirti gamybos srautų optimizavimui. Darbe išskirti pridėtinės vertės nekuriantys gamybos veiksniai tokie kaip: įrenginių gedimai, reguliavimas, techninės priežiūra ir kt. Išsikeltas tikslas padidinti gamybos pajėgumus sumažinant įrenginio reguliavimo trukmę pereinant iš vienos gamybinės partijos į kitą. Tam pasitelkti matematiniai metodai sudaryti optimalias partijų sekas ribojančiuose gamybos srautą darbo centruose. Pagrįstas metodas leidžia įgyvendinti išsikeltą tikslą taikant klasterizavimo ir optimizavimo metodus tokius kaip: DBSCAN, HDBSCAN, K-Means, Hierarchical clustering, Branch and Bound, Ant colony, 2-opt ir kt. Metodą sudaro keturi žingsniai: gamybos srautą ribojančių darbo centrų identifikavimas; perstatymų laikų nustatymas remiantis artumo matais, partijų klasterizavimas ir optimalių sekų radimas klasteriuose. Prisitaikytas metodas gamybos įmonėje leidžia sutrumpinti perstatymo laikus iki 90%. Sutrumpinamas pridėtinės vertės nekuriantis darbo laikas ir padidinamas gamybos pajėgumas iki 9%. Sumažinami tiesioginiai darbo kaštai, ir netiesioginės sąnaudos tokias kaip elektra, įrenginių nusidėvėjimas tenkantys vienam gaminiui. Tuo remiantis galima sumažinti gaminio savikaina 2%, kas suteikia konkurencinį pranašumą arba išlaikant tas pačias gaminio pardavimo kainas pelnas padidinimas 2%.

Pagrįstas metodas gali būti taikomas gamybos įmonėse vykdančiose masinę standartizuotų prekių gamybą ir pasižyminčiose gaminamų produktų įvairove ir reikšmingais įrenginių perderinimo laikais. Metodas gali būti taikomas įvairiems įrenginiams mažinant ne tik bendrą gamybos pajėgumą, bet ir atskirų įrenginių pajėgumą, kas suteikia galimybę sumažinti įrenginių darbo laiką.

Arnoldas Bankauskas. Identification of optimal processing sequences and distribution of parts during the production of cabinet furnitures. Master's Final Degree Project / supervisor assoc.prof.dr. Mindaugas Šnipas / supervisor prof. Mantas Vilkas; Faculty of Mathematics and Natural Sciences, Kaunas University of Technology.

Study field and area (study field group): Matematics (N 001)

Keywords: Klasterization, optimisation, TSP

Kaunas, 2020. 85 p.

Summary

Work subject - increasing the production capacity, competitive advantage and financial benefits of a mass production company.

Work methods: scientific, statistical, public information analysis, systematized organizational and mathematical methods for optimizing production flows. The factors of non-value-added production are singled out in the work, such as: equipment failures, adjustment, maintenance, etc. The aim is to increase production capacity by reducing the adjustment time of the machines from one production batch to the next. For this purpose, mathematical methods were used to create optimal batch sequences in production centres that limit the production flow. Well-grounded approach allows to achieve the set goal by applying clustering and optimization methods such as: DBSCAN, HDBSCAN, K-Means, Hierarchical clustering, Branch and Bound, Ant colony, 2-opt, etc. The method consists of four steps: identification of work centres restricting the flow of production; setting conversion times based on proximity measures; batch clustering; finding optimal sequences in clusters. The adapted method allows to reduce the conversion times by up to 90%. Reduces non-value-added working hours and increases production capacity to 9%. Reduces direct labour costs, and indirect costs such as electricity, equipment depreciation per product. On this basis, it is possible to reduce the cost of the product by 2%, which gives a competitive advantage or by maintaining the same sales prices of the product, an increase in profit of 2%.

This method can be applied in production plants engaged in the mass production of standardized goods and characterized by the variety of products produced and significant adjustment times of the equipment. The method can also be applied to various machines by reducing not only the total production capacity, but also the capacity of individual machines, which makes it possible to reduce the operating time of the machines.

TURINYS

Lentelių sąršas

Paveikslų sąrašas

Santrumpos

- TSP – Keliaujančio pirklio uždavinys *(Travel salesman problem);*
- FMS – Lanksčios gamybos sistemos *(Flexible Manufacturing Systems);*
- CNC – Skaitmeninio valdymo kompiuteriai *(Computer Numerical Control);*
- SSP – Darbo sekos ir įrankių keitimo problema *(Job Sequencing and Tool Switching Problem);*
- JSeP – Darbų sekos problema (*Job Sequencing Problem);*
- KTNS – Įrankių laikymas kuris reikalingas greičiausiai *(Keep Tool Needed Soonest);*
- NP – Klasė nedeterministinio polinominio laiko *(for nondeterministic polynomial time);*
- ILP – Integruotas tiesinis programavimas *(Integer linear programming);*
- ILS – Pažingninė vietinė paieška *(Iterated Local Search)*
- 5S – Nuostolių šalinimo principas organizuojant darbo vietą;
- PDCA – Planuoti, vykdyti, tikrinti, dokumentuoti *(Plan, do, check, act)*;
- SMED – Įrankių keitimas per minutę *(Singel Minute Exchange of Die)*;
- TPM – Pilna gamybos priežiūra *(Total productive maintenance)*;
- MRP – Medžiagų poreikio planavimas *(Material Requitement Planning);*
- GA – Genetiniai algoritmai *(Genetic algorithm);*
- PSO – Particle swarm optimization (PSO);
- SA – Simuliuoto atkaitinimo analizė *(Simulated annealing);*
- NNA – Artimiausio kaimyno metodas *(Nearest Neighbor Algorithm)*;
- RNNA – Kartotinas artimiausio kaimyno metodas *(Repeated Nearest Neighbor Algorithm)*;
- AP – Afinizmo propogavimas *(Affinity Propagation);*
- DBSCAN – Tankiu grįstas klasterizavimo metodas su išskirtimis *(Density Based Spatial Clustering of Application with Noise);*
- HDBSCAN – Hierarchinis tankiu grįstas klasterizavimo metodas su išskirtimis *(Herarchical Density Based Spatial Clustering of Application with Noise);*
- CFT – Charakteristikų ypatybių medis *(Characteristic Feature Tree);*
- BIRCH – Balansinis mažinimas arba klasterizavimas naudojant hierarchijas *(Balanced iterative reducing and clustering using hierarchies);*

Įvadas

Gamybos įmonėse produkcijos apimtys priklauso nuo gamybos srautų pralaidumo. Gamybos srauto pralaidumas apibrėžiamas pagamintos produkcijos kiekiu per tam tikrą laiko tarpą. Produkcija laikoma pagaminta tik tada, kai vykdoma gamybos partija pereina visą gamybos srautą, t.y. atliekamos visos operacijos, kurios sudaro srautą. Gamybos srauto pralaidumą riboja gamybinės operacijos, kurių darbo centras(ai) yra labiausiai apkrauti. Gamyboje ši operacija yra vadinama „butelio kakliuku“.

Yra daug organizacinių ir matematinių prieigų, skirtų gamybos pajėgumus ribojančių darbo centrų analizei ir „butelio kakliukų“ eliminavimui. Viena prieiga skirti daugiau žmogiškųjų ar techninių išteklių gamybos centrui. Kita vertus galima eliminuoti vertės nekuriančias veiklas. Dažniausiai, vertės nekuriančios veiklos, būna įrenginio reguliavimas, valymas, įrankių keitimas ir kt.. Šiame darbe bus susitelkta optimalių partijų sekų paieška siekiant sutrumpinti perstatymo laikus.

Tikslas: Padidinti gamybos pajėgumus, sudarant optimalias partijų sekas ribojančiuose gamybos srautą darbo centruose

Uždaviniai:

1. Atlikus literatūros analizę, susisteminti organizacinius ir statistinius metodus, skirtus gamybos srautų optimizavimui;
2. Pagrįsti metodą, leidžiantį padidinti gamybos pajėgumus, sudarant optimalias partijų sekas ribojančiuose gamybos srautą darbo centruose;
3. Parinkti optimalias partijų sekas ribojančiuose gamybos srautą darbo centruose ir įvertinti poveikį gamybos pajėgumams.

Metodai ir duomenys:

Tyrimas atliktas „Ikea industry“ įmonėje. Įmonė gamina korpusinius baldus. Įmonėje gamyba skaidoma į du lygiagrečiai veikiančius srautus: siaurų ir plačių elementų gamybą. Trečiasis gamybos srautas yra pirmuosiuose dviejuose srautuose pagamintos produkcijos komplektavimas ir pakavimas. Gamybos srautus sudaro nuosekliai veikiančios gamybos operacijos (pvz.: pjovimas, briaunavimas, frezavimas, gręžimas).

Pasitelkti šie duomenys:

- „IKEA Industry“ 2019 m. gamybos operacijų istoriniai duomenys;
- 2019 m. gamybos darbo grafikai ir įrenginių našumas;
- 2015-2019 m. Kategorizuoti gamybos prastovų laikai (pridėtinės vertės nekuriantys laikai);
- Gaminių techninės charakteristikos darbo centruose.

Taikyti šie metodai:

- Klasterizavimo metodai: BIRCH, DBSCAN, HDBSCAN, K-Vidurkio, Hierarchiniai klasterizavimo metodai;
- Optimizavimo metodai: Genetiniai, Sumuletyvinio atkaitinimo, Šakų ir rėžių, Godusis, Skruzdžių kolonijos, Atsitiktinės paieškos, 2-opt algoritmai.

Tyrimo praktinis reikšmingumas

Magistro darbe bus pristatytas ir ištirtas metodas, leidžiantis padidinti gamybos pajėgumus, sudarant optimalias partijų sekas ribojančiuose gamybos srautą darbo centruose. Metodą sudaro keturi žingsniai: gamybos srautą ribojančių darbo centrų identifikavimas, perstatymų laikų nustatymas remiantis artumo matais, partijų klasterizavimas ir optimalių sekų parinkimas. „Ikea industry“ sukaupti duomenys rodo, jog šio metodo taikymas leidžia sutrumpinti gamybos trukmę bent 9%. Gamybos sutrumpinimas sudaro prielaidas 2% finansinei naudai.

Darbo struktūra.

Pirmame skyriuje, buvo atlikta literatūros analizė, susisteminami organizaciniai ir statistiniai metodai, skirti gamybos srautų optimizavimui. Antrame skyriuje atlikome pirminių duomenų analizę, apdorojome duomenis ir atrinkome tyrimo objektą, gamybos pajėgumus ribojančius darbo centrus. Pateikta metodologija, nagrinėjanti darbo sekos parinkimą ir taikytų matematinių metodų panaudojimas. Trečiame skyriuje apibendrinami gauti rezultatai, parodomas praktinis gautų rezultatų reikšmingumas ir gaunama nauda.

1 Organizacinių ir matematinių metodų skirtų gamybos srauto optimizavimui apžvalga

1.1 Organizacinių gamybos srauto optimizavimo analizė

Gamybos pramonės šakose plačiai yra naudojamos tokios organizacinės priemonės kaip *LEAN* ar *AGILE* modeliai. Idėja paremta *LEAN* [14] metodologija „Išnaudojant esamus išteklius ir žinias, kurti pridėtinę vertę ir konkurencinį pranašumą“. Esminis *LEAN* principas „Nuolatinis esamų procesų efektyvinimas ir nereikalingų veiklų nuostolių šalinimas“- remiantis šiuo principu galima bet kurią veiklą suskaldyti į pagrindines grupes:

- Efektyvus darbo laikas (laikas, kuriuo metu vykdomas procesas neša pridėtinę vertę įmonei) t.y. kai įrenginys atlieka operacijas: gręžimo ir briaunavimas ir kt. Įrenginio efektyvumą galima pagerinti optimizuojant parametrus kiekvienai apdirbamai detalei ir/ar fiziškai peržvelgiant modernizavimą silpniausių gamybinės linijos mazgų, kurie daro lemiamą įtaką detalių apdirbimo greičiui;
- Veiksniai nenešantys tiesioginės pridėtinės vertės gamybos procese (nors be jų veikla neįmanoma). Analizuojamoje veikloje galima išskirti grupes:
    - Įrenginio reguliavimo laikas pereinant nuo vienos detalės prie kitos;
    - Įrankių (įrangos komponentų) keitimas;
    - Įrangos valymas ir priežiūra;
    - Įrangos testavimo trukmė (atliekama po įrankių keitimo ir/ar jos reguliavimo);
- Pašaliniai veiksniai kurie pasitaiko gamybos procese:
    - Įrenginių gedimai;
    - Žaliavų/ komponentų trūkumai;
    - Veiksniai įtakojantys kokybę;
    - Kiti veiksniai.

LEAN metodikoje teigiama, kad geriausias sprendimas siekiant pagerinti tokius kritinius taškus - yra taikyti tokius metodus kaip :

- *5S* – metodo principai šalinti nuostolius tinkamai organizuojant darbo vietą. Laikomasi penkių pagrindinių principų: rūšiuok, sutvarkyk, išvalyk, standartizuok, laikykis;
- *Kaizen* – terminas kilęs iš japonų kalbos. Termino reikšmė – nuolatinis tobulėjimas. Šis metodas taikomas darbo vietos ir joje vykstančio proceso stebėjimui bei idėjų siūlymas jos/jo tobulinimui. Pasiūlymai koncentruojasi apie kitus *LEAN* metodus ir jų panaudojimą atitinkamai problemai spręsti;

- *PDCA* - problemų sprendimo technika arba pastovaus tobulinimo modelis, susidedantis iš logiškos keturių žingsnių sekos: planuoti (*Plan*), įgyvendinti (*Do*), patikrinti (*Check*), įtvirtinti/priderinti (*Act/Adjust*). Skirta išmokyti darbuotojus sprendžiant problemas kovoti ne su pasekmėmis, o jų priežastimis. Ši metodika dar vadinama Demingo ratu. Metodo taikymo sritis yra rutininis ciklas darbe;
- Standartizuotas darbas – geriausias galimas darbo atlikimo būdas, kuris yra nustatomas, kaip sektinas standartas kitiems. Darbo standartizavimas leidžia užtikrinti, kad veikla visada būtų atliekama vienodu būdu, per vienodą laiką ir su tokiu pačiu rezultatu, nepriklausomai nuo to kas ir kada šį darbą atlieka;
- *SMED* - Antgalio pakeitimas per minutę - metodai, kaip sureguliuoti stakles per mažiau nei dešimt minučių. Ilgalaikis tokių metodų tikslas - sumažinti nustatymo laiką iki nulio, kai staklės sureguliuojamos akimirksniu, nenutraukiant nuolatinio srauto;
- *TPM* - gamybinė priežiūra. Jos tikslas - padidinti įrangos efektyvumą ir tarnavimo laiką, daugiau dėmesio skiriant jos priežiūrai ir įtraukiant į tai operatorius. Įvairiais metodais siekiama, kad įranga niekada nesustotų ir gamybos srautas nenutrūktų;
- *Kanban* - Japoniškas terminas, reiškiantis „signalą“. *Kanban* naudojama detalių gamybai ir perstatymams valdyti;
- kiti.

AGILE-gamyba[14] yra organizacinės metodikos kryptis taikomą gamybos įmonėse. *AGILE*-gamybos principas: gamyba yra terminas, taikomas organizacijai, kuri kuria procesus, įrankius ir mokymus, kad galėtų greitai reaguoti į klientų poreikius ir rinkos pokyčius, tuo pačiu išlaikydama išlaidų ir kokybės kontrolę. Tai siejama tiek su administraciniais, tiek su gamybiniais procesais. *AGILE* - kitas žingsnis gamybos procesų efektyvumo kėlimui po *LEAN* metodologijos taikomų metodų. Tai gamybos metodas, orientuotas į klientų poreikių tenkinimą, tuo pačiu išlaikant aukštus kokybės standartus ir kontroliuojant bendras konkretaus produkto gamybos sąnaudas. Šis požiūris yra skirtas įmonėms, dirbančioms labai konkurencingoje aplinkoje, kai nedideli veiklos rezultatų ir produktų pristatymo skirtumai ilgainiui gali labai pakeisti bendrovės išlikimą ir įmonės reputaciją.

Ši koncepcija glaudžiai susijusi su *LEAN* metodologija. Gamyboje bendrovė siekia sumažinti visas išlaidas, kurios nėra tiesiogiai susijusios su produkto gamyba vartotojui. *AGILE* gamyba gali apimti šią koncepciją, tačiau ji taip pat remiasi idėja, kad klientų poreikius reikia greitai ir veiksmingai įgyvendinti. Priešingai nei *LEAN* metodologijoje, žvelgiama ne tik į nuostolių šalinimą kad procesas būtų greitesnis, bet ir į patį proceso spartinimą. Visos organizacines priemonės taikomos *AGILE*

metodologijoje yra skirtos procesų skatinimui, koreguojant procesus tarp skyrių, tiekėjų, pirkėjų - t.y. apima gamybos ir tiekimo bendradarbiavimo procesus.

Nagrinėjant organizacines metodologijas galima įžvelgti, kad jos neapima procesų pačiuose darbo centruose. T.y. organizacines priemones galima papildyti įtraukiant matematinius metodus.

1.2 Gamybos srauto optimizavimo formuluotė matematiniais metodais

Lanksčiose gamybos sistemose (FMS) veikia mašinos (CNC) skaitmeninio valdymo kompiuteriai. Mašinos gali apdoroti įvairias veiklas ar procesus, kai mašinų įrankių bloke yra užduočiai atlikti reikalingų darbo įrankių. Analizuojamu atveju, apdirbimo įrankiai - tai skirtingo tipo gręžimo galvos, talpinamos į atitinkamus darbo centro įrankių rėmus (blokus). Apdirbimo bloką sudaro C jungčių. Viena jungtis skirta vienai gręžimo galvai. C taip pat žinomas kaip įrankių rėmo naudingumas (pajėgumas), o atliekamų darbų skaičius negali viršyti C jungčių. Jeigu įrankis (gręžimo galva) nėra apdirbimo bloke, tuomet įrankis turi būti pakeistas esamu gręžimo bloke t.y. užimančių vieną ar kelias C jungtį (-is). Vieno įrankio pašalinimas ir kito įtraukimas yra neišvengiamas veiksmas gamybos procese[20]. Įrankių pakeitimas ir reguliavimas - procesas atimantis daug laiko ir nenešantis tiesioginės pridėtinės vertės, todėl FMS sistemoje svarbus klausimas yra surasti darbo seką, sumažinančią bendrą įrankių keitimų skaičių. Ši problema plačiai žinoma kaip darbų sekos ir įrankių keitimo problema (SSP)[16,20] ir yra taikoma įvairiose srityse, tokiose kaip kompiuterio atminties paskirstymo[25], skambučių srautų paskirstymo ir pan. uždaviniuose. Šis uždavinys plačiai nagrinėjamas įvairiose pramonės šakose: tokiose kaip metalo pramonė, chemijos pramonė, draudimo organizacijos, farmacijos bendrovės[1,23] ir pan. Viena iš daugiausiai susiduriančių industrijų su SSP yra elektronikos industrija, kai skirtingų tipų spausdintinės plokštės (PCB) turi būti apdorotos komponentų surinkimo mašinoje. Skirtingi PCB gali būti išversti į darbus, o įvairūs elektroniniai komponentai, kuriuos reikia įkelti į mašinos padavimo sistemą, gali būti išversti į įrankius. Tokią formuluotę pateikė Shirazi ir Frizelle [19], kurie ištyrė realaus pasaulio taikomus metodus sprendžiant SSP septyniose skirtingose aviacijos bendrovėse, CNC gamintojų ir įrankių gamybos pramonėje.

Tangas ir Denardoas [20] nurodė, kad SSP gali būti išskaidoma į dvi dedamąsias, kurios gali būti sprendžiamos kaip atskiros problemos:

1. Darbo sekos problema (JSeP) - tai yra laiko planavimo problema, kurią galima spręsti kaip optimalią darbų seką kompiuteryje.
2. Įrankio pakeitimo problema (TRP) reiškia, kad reikia įkelti reikalingus įrankius su kuriais būtų galima apdoroti numatytą darbų seką (JSeP), siekiant sumažinti bendrą įrankių pakeitimų skaičių.

Y. Crama ir kt.[19] atliekant tyrimus parodė, kad JSeP yra NP-sudėtingumo problema. Keliuose tyrimuose buvo apsvarstyta JSeP ir modeliuojama problema su tiksliu algoritmu. L. Gilbert ir kt.[22] paskelbė optimalų sprendimą JSeP etaloniniams atvejams iki 25 pozicijų. Modeliuojant sprendimus buvo atsižvelgta jų linijinio programavimo (ILP) modelius. Tačiau realiose situacijose sekos taškų skaičius dažnai viršiją šią ribą. Tokiu atveju stengiamasi skaičiavimo sekas apdoroti kitais metodais. Šios problemos sprendimų tyrimai aprašomi įvairiose publikacijose. Dažnais atvejais JSeP yra prilyginama keliaujančio pirklio uždaviniui (TSP). Tai grafų teorijos uždavinys, kai ieškomas mažiausio svorio Hamiltono kelias pilname svoriniame grafe. Tyrimo metu grafų teorijos terminai siejami taip:

- Viršūnės – gaminio detalės apdirbamos darbo centre;
- Briaunos – įrenginio reguliavimo laikas pereinant iš vienos detalės prie kitos;
- Ciklas – detalių gamybos eiliškumas darbo centre.

Iš pirmo žvilgsnio paprasčiausias metodas šio uždavinio sprendimui yra visų galimų kombinacijų perrinkimas. Tačiau, remiantis gamybos asortimentu, išskiriamos ~300 vnt. skirtingų detalių. Detales galima suskirstyti į 4 – 5 savaičių intervalą pagal gaminius. Tai atliekama dėl to, kad vienu kartu visas esamas asortimentas nėra gaminamas. Toks detalių skaičius apsunkina uždavinio sprendimą.

Ciklų skaičius, remiantis grafų teorija, yra:

- Esant asimetrinei matricai grafe N taškų sudarys $(n-1)!$;
- Esant simetrinei matricai grafe N taškų sudarys $\frac{(n-1)!}{2}$ [29]

Remiantis nagrinėjamais duomenimis ir ekspertine nuomone nustatoma, kad detalių perstatymas yra simetriškas t.y. $A \rightarrow B = A \leftarrow B$ - tokiu atveju laikoma jog bendras variantų skaičius yra lygus: $\frac{(n-1)!}{2}$. Augant taškų skaičiui imtyje, galimų sekų skaičius auga eksponentiškai žr. 1 lentelę “TSP Problema, eksponentinis perrinkimo sekų skaičiaus augimas“. Jei per sekundę kompiuteris atliktų vieną milijardą veiksmų, tada ištirti 20-ies taškų sekas prireiktų dvejų metų[29]. Tokiais atvejais perrinkimo metodas yra netinkamas ir šiam uždaviniui spęsti taikomi euristiniai, meta-euristiniai metodai.

1 lentelė. TSP Problema, eksponentinis perrinkimo sekų skaičiaus augimas

Taškai	$\frac{(n-1)!}{2}$	$(n-1)!$
5	60	120
6	360	720
10	1,814,400	3,628,800
11	19,958,400	39,916,800
20	1,216,451,004,088,320,000	2,432,902,008,176,640,000
21	25,545,471,085,854,700,000	51,090,942,171,709,400,000

1.3 Klasterizavimo ir optimizavimo metodų apžvalga

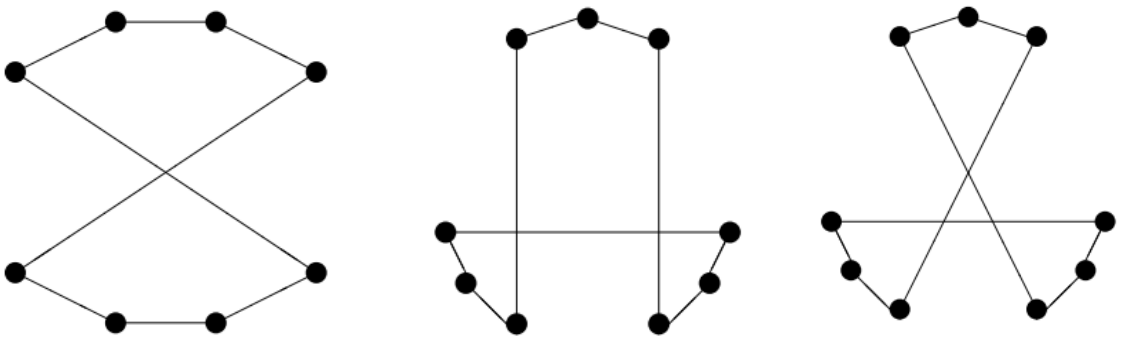
1.3.1 Euristiniai metodai TSP problemai spręsti

Tangas ir Denardoas [20] parodė, kad JSeP, gali būti išspręsta taikant *Greedy* algoritmą (GR) vadinamu KTNS sprendžiama daugianario funkcija. Todėl SSP, kuris yra kombinuotas, gali būti sumažinta iki JSeP ir tai tampa pagrindine problema, kurią reikia spręsti.

G. Gutin ir kt. [18] atlikdami kompiuterinius eksperimentus įrodo, kad GR ir Artimiausio kaimyno algoritmas (NN) yra populiarūs euristiniai metodai sudaryti sekai, yra tinkami Euklido TSP lygtims. Tačiau rezultatai yra labai blogi tiek simetrinėms, tiek asimetrinėms matricoms sprendžiant TSP (STSP ir ATSP). Įrodoma, kad kiekvienam $n \geq 2$ yra ATSP (STSP) atvejis iš n viršūnių, kai GR nustato blogiausią seką. Tokie pat rezultatai nustatyti ir NN. Šiame tyrime taip pat analizuojamas pasikartojantis NN (RNN) t.y. NN metodas atliekamas pradedant nuo visų galimų viršūnių ir atrenkama geriausia seka. Taip pat jie įrodė, kad "ATSP", taikant RNN, visada gaunama geriausia seka, kuri yra ne blogesnė nei mažiausiai $\frac{n}{2} - 1$ kitų sekų, bet tam tikru atveju ji suranda seką, kuri yra ne blogesnė nei ne daugiau kaip $n - 2$ kitų sekų , kai $n \geq 4$. Jie taip pat ištyrė, kad kai kurioms STSP kai $n \geq 4$ atvejų RNN duoda gerus rezultatus (esant ne daugiau kaip $2n - 3$ sekų). Šie rezultatai akivaizdžiai skiriasi nuo ankstesnių Tango ir Denardo [20], kurie įrodė, kad visada gaunami geri rezultatai kai sudaroma (n -2)! sekų.

2-Opt metodas yra turbūt labiausiai paplitęs ir plačiausiai naudojamas vietinės paieškos sistemose. Šios sistemos yra integruotos į MRP. Šis euristinis metodas pasiekia gerus rezultatus Euklido atvejų, tiek laiko, tiek artumo atžvilgiu. Šio metodo principinis veikimas: 2-opt algoritmas iš esmės pašalina dvi kraštines iš sekos ir vėl sujungia dvi sukurtas kraštines. Tai dažnai vadinama "2-opt" judesiu. Yra tik vienas būdas iš naujo prijungti dvi kraštines, kad mes vis dar turėtume tinkamą seką žr. 1 pav. Pašalintų kraštinių jungimo schemos algoritmams 2-opt, 3-opt (1 figūra). Tai atliekama tik tuo atveju, jei naujasis maršrutas bus trumpesnis. Procesas kartojamas tol, kol yra pakeitimų,

išimant dvejas kraštines, kol nerandama trumpesnio atstumo tarp dviejų taškų. 3-opt algoritmas veikia panašiai, bet vietoj to, kad pašanlintume dvi kraštines, pašaliname tris. Tai reiškia, kad mes turime du būdus, kaip iš naujo sujungti tris viršūnes į naują seką žr. 1 pav. "Pašalintų kraštinių jungimo schemos algoritmams 2-opt, 3-opt (2-3 figūros)"



FIGŪRA 1. 2-opt FIGŪRA 2. 3-opt FIGŪRA 3. 3-opt

1 pav. Pašalintų kraštinių jungimo schemos algoritmams 2-opt, 3-opt

D. Catanzaro [5] ištyrė ir pristatė tris ILP formuluotes. Juose pateiktos formuluotės pagerino Laporte Gilbert ir kt.[22] apibrėžtą ribą. Jo atlikti kompiuteriniai eksperimentai rodo, kad ribos, gautos taikant linijinę relaksaciją nagrinėjamų formulių, vidutiniškai pagerėja. Literatūroje Laporte Gilbert ir kt.[22], tuo pat metu siūlo naujas kryptis tolesnio tikslaus sprendimo būdų kūrimui.

Literatūros, apžvelgiančios JSeP sprendimus, labai platus spektras, nagrinėjami ir tobulinami esami ir kuriami nauji tiek euristiniai tiek meta-euristiniai metodai. J. Bard'as [2] savo tyrumose problemą formuluoja kaip nelinijinę sveikojo skaičiaus programą ir sprendžia naudojant dvejopą euristiką. Rezultatai rodo, kad beveik visais atvejais gaunamas optimalus sprendimas. Svarbu tai, kad ypač sutrumpinamas skaičiavimo laikas. Tai rodo algoritmo naudojimo galimybes realaus laiko valdyme. Tačiau tyrime sekos taškų skaičius imtas iki 25. M. Denizelis [9] JSeP pritaikė Lagrandžo skaidymo metodą ir sekos elementų skaičių padidino iki 30 -ies.

1.3.2 Meta euristiniai metodai TSP problemai spręsti

Branch and Bound algoritmai dažnai naudojami rasti kombinatorinius optimizavimo problemų sprendinius. Šis metodas gali būti lengvai taikomas TSP nepriklausomai nuo to, ar tai ATSP ar STSP.

M. Furrer ir T. Mütze [15] pasiūlė sujungtą algoritmą, skirtą įvairiems JSeP tikslams spręsti, susidedantį iš minimizavimo įrankių keitimų ir mašinų sustojimų skaičių per tam tikrą laiką. Šiame darbe pateikiama filtravimo ir susiejimo algoritminė sistema *(Branch and Bound)* dar kitaip vadinama

Šakų ir rėžių metodu. Metodas pagrįstas Godžiuoju algoritmu *(greedy)* su grįžimo modifikacija į praeityje atliktus žingsnius. Šis metodas pateikiamas kaip pirmaujantis tarp metodų sprendžiančių TSP.

Savo darbe T.Volgenant atliko tyrimus pritaikant 1970 m. Heldo ir Karpo pristatytą Lagrandžo sprendinį STSP. Jie panaudojo šį 1-medžio relaksacijos algoritmą naujame *Branch and Bound* metode. Jis skiriasi nuo kitų algoritmų ne tik schema, bet ir kitimo pobūdžiu, norint apskaičiuoti 1-medžio ribas. Be to, jie nustatė heuristinius sprendimus visame skaičiavime, kad pateiktų aukščiausiąsias ribas. Vidutiniškai jų rezultatai nuo Euklido problemų iš literatūros gauna maždaug 60% mažiau šakų, nei geriausiai žinomas algoritmas.

Genetiniai algoritmai (GA). GA veikimo idėja remiasi evoliucijos procesu, kuris vyksta formuojant naujus genus. Paprastasis GA prasideda nuo atsitiktinai sugeneruotų sprendimų kandidatų. Kai kurie (arba visi) kandidatai sujungiami, kad sudarytų palikuonis, o kai kurie iš jų pereina mutacijos procesą. Pasirinkus tinkamiausius kryžminimo ir mutacijos kandidatus, bendras tinkamumas padidėja. GA taikymas TSP apima genų kryžminimą, tinkamumo patikrą, taip pat mutacijas. Kai kurie tyrimai parodė gerus rezultatus. Bet kaip ir Tabu-paieškos, ir GA algoritmo neigiama pusė yra ilga skaičiavimo trukmė.

Peržvelgiant šių tyrimų rezultatus, galima manyti, kad elementų skaičius yra apribojamas dešimtimis vienetų. Bet A.Konak‘as[21] savo publikacijoje „Minimizing the number of tool switching instants in Flexible Manufacturing Systems“ pristatė tyrimą paremtą E. Denard‘o [26] ir M. Denizel‘io [9] atliktais darbais. Tyrime sekos sudarė 210 elementų. Metodika taikė evoliucinius metodus. Atliekant genetines mutacijas, paveldėtojai sugebėjo rasti kokybiškus, bet nebūtinai optimalius sprendinius per priimtiną laiką.

Crama ir kt. [13] pasiūlė keletą euristikos metodų, pagrįstų TSP, ir įvertino jų skaičiavimo rezultatus. Pateiktas kitoks "atstumo" tarp įrankių apibrėžimas. Jie parodė, kad jų euristikos metodai yra pranašesni. Chaves ir kt. [24], pasiūlė hibridinės euristikos metodą, pagrįstą šališku atsitiktinių genetinių algoritmų generavimu remiantis taškų grupavimo paieška. Jie atliko bandymus, ir kai kuriais atvejais, pranešė apie geresnę sprendimo kokybę. Paiva ir kt. [25] siūlomi metaduomenys "Iterated Local Search" (ILS) ir jų rezultatai rodo, kad jų metodas yra pranašesnis už kolegų: Cram‘os ir kt. [13], Yanasse ir kt. [15], Catanzaro ir kt. [16] .

Spiečiaus optimizavimas (PSO) pasirodė kaip efektyvus optimizavimo metodas, kurį 1995 m. sukūrė Kenedis ir Eberhartas[11,13]. PSO - įkvėptas paukščių ir žuvų socialinio elgesio. Jų judėjimas

yra sinchronizuotas ir be susidūrimo kadangi paukščiai skrenda ir palaiko tam tikrą atstumą tarp kaimynų. Rezultatai rodo, kad PSO yra efektyvus algoritmas, kuris sėkmingai sprendžia TSP problemas, naudojant nedidelį skaičių įvertinimų, kad būtų pasiektas geriausias rezultatas.

Simuliuotas atkaitinimas (SA) buvo sėkmingai pritaikytas pateikti apytikslius TSP sprendimus. SA iš esmės yra atsitiktinių imčių vietinės paieškos algoritmas, leidžiantis judėti neigiama kryptimi t.y. palaipsniui mažinti bendrą visos sekos atstumą. Šis metodas pagrįstas tuo, kad renkamas kaimyninis sprendimas - atrenkant geriausią sprendinį.

D. Henderson [30] teigia jog simuliuoto atkaitinimo metodo naudojimas - iš esmės reiškia kopimą į kalną, tačiau su galimybe ir nukristi. Šiame metode, atsiranda nauja kintamasis - temperatūra. Temperatūra yra parenkama pagal tai, kiek geras ar blogas yra vertinimas. Kai temperatūra yra aukšta, blogesnis sprendimas turės didesnę tikimybę būti pasirinktas. Algoritmas dirba taip: jei kaimynas yra geresnis – esamas taškas pakeičiamas kaimynu. Tačiau jei kaimynas yra blogesnis – tikėtina, kad liksime prie dabartinio sprendimo. Tuomet sumažinama esama temperatūra proporcingai sprendinio blogumui. Toliau pasirenkamas kitas dabartinio sprendimo kaimynas ir taip procesas kartojamas.

Procesas vyksta atvėsinant temperatūrą, laikui bėgant vis rečiau yra pasirenkamas blogesnis sprendimas. Pradžioje daromi dideli šuoliai bendrame vaizde, tačiau pabaigoje šokinėjimai vyksta vis mažiau ir mažiau.

Mažiausią svorį turinčio medžio metodas iš esmės yra panašus į Šakų ir rėžių metodą, sudarant atisiktinius medžius ir miškus. Šį algoritmą sukūrė mokslininkas O.Boruvka [31] 1926m. Algoritmas vykdomas etapų seka. Kiekviename etape yra identifikuojamas miškas, susidedantis iš mažiausio kiekvienos viršūnės svorio briaunos. Viršūnių skaičius kiekviename žingsnyje yra sumažinamas per pusę. Mažiausią svorį turinčio medžio metodu sujungtos briaunų, jungiančių visas viršūnes kartu be jokių ciklų turinčių mažiausią galimą bendrą briaunos svorį. Ieškomas besidriekiantis medis, kurio briaunų svorio suma yra kuo mažesnė. Bet kurios briaunos atžvilgiu - miškas, kuris yra iš sujungtų komponentų ir yra mažiausiai apimančio medžio sąjunga. Šis algoritmas dažniausiai naudojamas kaip paprogramė kitų problemų algoritmuose

Sprendžiant kombinuotas optimizavimo problemas, dažnai nepavyksta išspręsti tiksliai (nepavyksta rasti optimalaus optimizavimo taško). Daug tokių problemų atpažįsta aproksimacijos algoritmai, kurie apskaičiuoja apytikslius optimalius sprendimus. Atsitiktinių sekų metodas yra tikimybinis metodas, aproksimacijos algoritmams sudaryti. A.Srinivasan [32] teigia, kad pagrindinė

šio metodo idėja yra naudoti tikimybes, norint rasti optimalų problemos sprendimą apytiksliu pirminės problemos sprendimu. Pagrindinis algoritmo principas toks:

1. Suformuojama išspręsta problema, kaip sveikojo skaičiaus tiesinė funkcija;
2. Apskaičiuojamas optimalus trupmeninis sprendimas;
3. Suapvalinamas trupmeninis sprendimas iki sveikojo sprendinio.

Tyrėjai dažnai bando imituoti gamtą. Vienas iš tokių pavyzdžių yra labai sėkmingas genetinių algoritmų naudojimas. Dar viena įdomi idėja yra imituoti skruzdžių judesius. M. Darigo [10] buvo pirmasis, kuris pritaikė skruzdžių kolonijos algoritmą (ACO) TSP. Šis algoritmas buvo sėkmingas, greitai pateikiantis optimalius mažų problemų sprendimus. Skruzdės, kai tyrinėja naujas sritis, palieka feromonų taką. Šis takas yra skirtas kitoms skruzdėms nukreipti į galimus maisto šaltinius. Skruzdžių kolonijos stiprumas priklauso nuo jų skaičiaus, o tai pat ir optimizavimo. Paprastai tyrimuose pradedama apytiksliai nuo 20 skruzdėlių. Jos pradeda nuo atsitiktinių viršūnių, o paskui pereina į kitas viršūnes. Kartotis tarp viršūnių negalima. Skruzdė, kuri apėjo trumpiausią taką, paliks feromonų taką, atvirkščiai proporcingą kelionių ilgiui. Į šį feromonų taką atsižvelgiama, kai skruzdė pasirenka viršūnę, į kurią keliauja, todėl jos labiau linkusios vaikščioti kelyje su stipriausiu feromono taku. Šis procesas kartojamas tol, kol randamas trumpiausias maršrutas.

Optimizavimo metodų, skirtų TSP spręsti, yra įvairių rūšių. Tinkamų metodų parinkimas priklauso labiausiai nuo dalykinės srities ir bendrų keliamų apribojimų, kur tie metodai bus taikomi. Remiantis turimais įmonės duomenų apribojimais, metodai taikomi ypač didelių duomenų apdirbimui nėra būtini. Taip kaip duomenis reikia klasterizuoti, o optimizavimo procesus atlikti atskirai kiekviename klasteryje. Toks apribojimas keliamas dėl tarpinių sandėlių, kuriuose gali būti saugoma iki 10 gaminių vienu metu. Atsižvelgiant į tai, parenkami metodai, kurie išsiskiria tikslumu ir sparta. Taip pat papildomai peržvelgiami naujausi tyrimai susiję su duomenų klasterizavimų.

1.3.3 Klasterinė analizė, metodai ir jos taikymas

Atsižvelgiant į problematiką, kuri keliama darbe (imties dydis ~300 kintamųjų) papildomai peržvelgiame klasterizavimo metodus, kurie skirstomi į:

- Padalijimo metodai;
- Hierarchiniai metodai;
- Tankių grįsti metodai;
- Tinkleliu grįsti metodai;

Klasterizavimas priskiriamas prie „Unsupervised learning“ t.y. mokymasis be mokytojo. Tokiu atveju klasterizavimo metodų vertinimas yra sudėtingas procesas, kuris gali remtis į nustatytą vertinimo funkciją ar ekspertinę nuomonę. Plačiausiai šiuo metu taikomi klasterizavimo metodai:

2 lentelė. Klasterizavimo metodai, ir jų charakterizavimas

Metodo vardas	**Parametrai**	**Mastelis**	**Naudojimo atvejis**	**Metricos**
K-Means	Klasterių skaičius	Labai didelis `n_samples`, vidutinis `n_clusters` su MiniBatch kodu	Mažas skaičius klasterių, plokščia geometrija, vienodas klasterių dydis	Atstumas tarp taškų
Affinity propogation	Slopinimas,	Nekeičiamo dydžio `n_samples`	Daug klasterių, skirtingas dydis klasterių, ne plokščia geometrija	Grafo atstumas (pvz., Artimiausio kaimyno grafas)
Mean-shift	Pralaidumas	Nekeičiamo dydžio `n_samples`	Daug klasterių, skirtingas klasterių dydis, ne plokščia geometrija	Atstumas tarp taškų
Spectral clustering	Klasterių skaičius	Vidutinis `n_samples`, mažas `n_clusters`	Keli klasteriai, vienodas klasterių dydis, ne plokščia geometrija.	Grafo atstumas (pvz., Artimiausio kaimyno grafas)
Ward hierarchical clustering	Klasterių skaičius	Didelis `n_samples`, ir `n_clusters`	Didelis klasterių skaičius, gali būti jungiamumo apribojimai	Atstumas tarp taškų
Agglomerative clustering	Klasterių skaičius, susiejimo tipas, atstumas	Didelis `n_samples`, ir `n_clusters`	Didelis klasterių skaičius, gali būti jungiamumo apribojimai. Nereikalingi Euklido atstumo matai	Atstumas tarp porų
DBSCAN	Kaimynystės dydis	Labai didelis `n_samples`, vidutinis `n_clusters`	Ne plokščia geometrija, nevienodas klasterių dydis.	Atstumas tarp artimiausių taįkų
HDBSCAN				
Gaussian mixtures	Klasterių skaičius	Nekeičiamo dydžio	Plokščia geometrija, geras tankio įvertinimas	Mahalanobis atstumas iki centro
Birch	Šakojimosi veiksniai, Slenkstis, pasirenkamas globalus klasteris	Didelis `n_samples`, ir `n_clusters`	Didelis duomenų rinkinys, pašalinimas, duomenų mažinimas.	Euklido atstumas tarp taškų

Neplokščių geometrinių grupių naudojimas yra naudingas, kai klasteriai turi tam tikrą formą, t. y. ne plokščią atvaizdą, o standartinis Euklido atstumas nėra tinkama metrika.

K-Vidurkių metodu klasterizuojami duomenys skirstant taškus taip, kad dispersijos būtų lygios. Minimizuojamas inercijos kriterijus arba kvadratinė suma klasteryje. Algoritmas iš anksto nusakomas apsprendžiant klasterių skaičių. P. S. Brandley [3] tyrimose įrodyta, kad K-vidurkių metodas efektyviai dirba su dideliais kiekiais duomenų, dėl šios priežasties K-vidurkių metodas plačiai taikomas įvairiose srityse.

K-vidurkių algoritmas padalina rinkinį iš N elementų į K klasterių, kiekvieną iš jų apibūdina imties vidurkis μ_j. Šios priemonės dažniausiai vadinamos klasteriais "centroidais": K-Vidurkio algoritmas siekia pasirinkti centroidus, kurie mažina inerciją arba suminį kvadratinį vidurkį:

$$\sum_{i=0}^{n} \quad (||x_j - \mu_i||)^2$$

Inercija arba suminis kvadratinis vidurkis pripažintas kaip vidinės darnos klasterių matas. Jis turi įvairių trūkumų:

- Inercija daro prielaidą, kad klasteriai yra išgaubti ir izotropiniai, bet taip būna ne visada. Tai blogai reaguoja į pailgas ar su nelygia forma grupes.
- Inercija tai nenormalizuota metrika: žinoma tik tai, kad mažesnės reikšmės yra geriau ir nuliai yra optimalu. Bet esant daugiadimensiams duomenims, Euklido atstumai tampa išpūsti t.y. vadinama dimensijos persislinkimu „curse of dimensionality“. Tokiu atveju reikia taikyti papildomus metodus dimensijos mažinimui tokius kaip PCA. Išsprendžiamas dimensijos persislinkimas ir padidinamas skaičiavimo greitis.

Algoritmas yra paprastai realizuojamas, jis susideda iš trijų žingsnių:

- Centrinių taškų nustatymas imtyse X kiekvienam klasteriui C.
- Priskiriamas taškas klasteriui, kuris yra nepriskirtas ir artimiausiam centiniam taškui.
- Imant visų įeinančių taškų vidurkį, sukuriamas naujas centrinis taškas. Kartojamas antras žingsnis.

Afinizmą propaguojantis klasterizavimas (AP) - metodas kuria grupes, siunčiant pranešimus tarp taškų porų iki pilno konvergavimo. Duomenų rinkinys apibūdinamas naudojant nedidelį skaičių taškų, kurie yra identifikuojami kaip tie, kurie labiausiai reprezentuoja kitus taškus. Jų atrinkimas vyksta tarp tašku siunčiant pranešimus. Atsižvelgiant į pranešimų atsakus, atnaujinamos grupės t.y. klasteriai. Šis atnaujinimas atitinkamai kartojasi iki pilnos konvergencijos, taigi tuo metu pasirenkami galutiniai pavyzdžiai ir pateikiamas galutinis klasterizavimas.

K.Wang ir kt. [27] savo darbuose paskelbė apie šio metodo limitus: sunku žinoti, kokia parametro "preferencija" vertė gali suteikti optimalų klasterio sprendimą, o svyravimai negali būti automatiškai pašalinami. Šis metodas, be papildomų modifikacijų, sunkiai pritaikomas automatinėms lanksčioms sistemoms. Skaičiavimo trukmė ir resursų poreikis, lyginant su dauguma metodų, yra optimalūs. Šis metodas gali apdoroti didelius kiekius įrašų taip pat ir požymių (dimensijų). Bet jis

nėra integruojamas į dinamines sistemas ir negali būti pritaikytas automatiniams duomenų klasterizavimo procesams.

Hierarchinis klasterizavimas yra bendra klasterizavimo algoritmų grupė, kuri kuria grupes, jas nuosekliai sujungdama arba skaidydama. Ši klasterių hierarchija yra pavaizduota kaip medis (arba dendograma).

Vienas iš hierarchinių klasterizavimo būdų yra Aglomeratyvinis klasterizavimas - atliekamas taikant jungimo metodą. Kiekvienas stebėjimas prasideda savo klasteryje, o grupės nuosekliai sujungiamos. Naudojama metrika nusako sujungimo strategiją:

- *Ward* sumažina kvadratų skirtumus visose klasteriuose. Taikant šią metriką suteikiamas reguliarus dydis.
- *Complete linkage* - sumažina maksimalų atstumą tarp stebėjimo porų klasteriuose. Taikoma kai nėra Euklido metrikos.
- *Average linkage* – sumažina vidutinį atstumą tarp stebėjimo porų klasteriuose. Taikoma kai nėra Euklido metrikos.

I.Davidson [8], nagrinėjant aglomeratyvinio klasterizavimo savybes, nustatė jo tinkamumą didelės apimties duomenims, kai jie naudojami kartu su ryšio matrica. Jeigu matrica nesudaroma, skaičiavimo procesas tiek laiko, tiek atminties resursų atžvilgiu, smarkiai išauga. Taip pat tyrimai parodė, kad hierarchiniai klasterizavimo metodai yra tinkami, kai klasterių dydžiai yra skirtingi (ypač neproporcingi).

DBSCAN Algoritmas peržiūri klasterius, kaip didelio tankio sritis, atskirtas mažo tankio sritimis - t.y. duomenų tankiu grįstas klasterizavimo metodas. Dėl šios priežasties gali būti nustatyti įvairios formos klasteriai. DBSCAN algoritme klasteris yra pagrindinių pavyzdžių rinkinys, kurių kiekvienas yra arti kito. Algoritmui yra du parametrai, `min_samples` (minimalus skaičius elementų klasteryje) ir `eps` (spindulys). Didesni `min_samples` ar žemesni `eps` rodo didesnį tankį, būtiną klasteriui sudaryti. Pagrindinis DBSCAN algoritmo skirtumas nuo paprastų tankių grįstų metodų, yra tai, kad jis randa išskirtis t.y. taškus, kurių neina priskirti nei vienam klasteriui su nustatytais pradiniais parametrais `min_samples` ir `eps`.

S. Chakraborty [6] pateikia DBSCAN pranašumas, lyginant su kitais klasterizavimo metodais dinaminėse duomenų bazėse, kas yra aktualu dirbant su kintančiais duomenimis *„streaming data“*. Jis puikiai tinka labai didelės apimties duomenims t.y. neatima daug atminties resursų ir duomenų kiekis mažai įtakoja skaičiavimo trukmę. Dėl tos priežasties jis naudojamas tiesiai duomenų

kaupyklose *„Data Warehouseses“,* kur įrašų ir jų požymių kiekiai yra ekstremaliai dideli. Šiam metodui S. Chakraborty [6] išskyrė du esminius trūkumus:

- Jis labai jautrus parametrams. Minimalus parametrų pakeitimas padidina klasterizavimo rezultatų skirtumus.
- Norint nustatyti, ar tas objektas yra pagrindinis, reikia aptikti kiekvieną objektą. Tai sudaro daug įvesčių / išvesčių kas didina resursų sąnaudas.

HDBSCAN - Hierarchinis tankiu grįstas klasterizavimo metodas su išskirtimis, ištobulintas J. Pei ir kt.[24]. Šis algoritmas – tai kombinacija iš hierarchinio klasterizavimo ir DBSCAN. Tokiu būdu apjungiamos teigiamos savybės tiek DBSCAN, tiek hierarchinio klasterizavimo algoritmų. Šis algoritmas suderina tvirta bendrąjį ryšį *(Robust Single Linkage)*[7] su plokščių grupių ištraukimu *(flat cluster extraction)*[4]. Šio algoritmo veikimą galima išskirti į pagrindinius 5 žingsnius:

1. Atsižvelgiant į taškų tankį transformuojama erdvė.

Erdvėje atskiriamos dvi grupės: didelio tankio (galimi klasteriai) mažo tankio (galimas triukšmas). Mažo tankio atskyrimas yra atliekamas sudarant naują atstumo matricą. Literatūroje tai vadinama abipusio pasiekiamumo atstumas *(mutual reachability distance):*

$$d_{mreach-k}(a,b) = max\{core_k(a), core_k(b), d(a,b)\}$$

Čia: $d(a,b)$ yra originali atstumo metrika tarp a ir b taškų.

J. Eldrige ir kt.[12] tarpusavio pasiekiamumo matą išreiškia kaip pagrindinę teoriją, rodančią, kad pasiekiamumo atstumas, kaip transformacija, gerai veikia nustatant lygių rinkinių hierarchiją, nepriklausomai nuo tikrojo tankio pasiskirstymo, iš kurių buvo renkami mūsų taškai.

2. Suformuojamas minimalios apimties medis iš svorinio atstumo grafo;
3. Čia svorinį grafą sudaro viršūnės iš duomenų taškų, o briaunos - tai (1.) pateiktas abipusio pasiekiamumo matas. Minimalios apimties medis formuojamas atskiriant silpniausias briaunas iš svorinio grafo ir priskiriant jas prie medžio. Taip sudaroma pilna hierarchija - nuo pilnai sujungtų iki pilnai atskirtų elementų. Ši skaičiavimo vieta yra imli resursams nes kraštinių yra n^2. Todėl naudojamas Godusis algoritmas minimalios apimties medžiams formuoti: Prim‘s algoritmas [17] .
4. Iš sujungtų komponentų, t.y. iš minimalios apimties medžių, suformuojama klasterių hierarchija;

Tai lengviau padaryti atvirkštine tvarka: surūšiuojamos medžio kraštinės pagal atstumą (didėjančia tvarka) ir tada pereinama per medį, sukuriant naują klasterį kiekvienai kraštinei. Sunkiausia dalis, yra nustatyti dvi grupes, kurios jungsis kartu - tai atliekama surandant duomenų struktūrą ir ją apkarpant, panaikinat smulkias dendogramos atšakas.

5. Atsižvelgiant į minimalų klasterio dydį, koncentruojama klasterių hierarchija;

Pirmasis etapas klasterių išgavime - tai didelės ir sudėtingos klasterių hierarchijos sujungimas į mažesnį medį su šiek tiek daugiau duomenų prie kiekvieno mazgo. Dažniausiai klasterio suskaidymas yra vienas ar du taškai, išsiskiriantys iš klasterio ir tai yra pagrindinis aspektas, o ne tai, kad klasteris skirstomas į du naujus klasterius, kuriuos mes norime laikyti viena nuolatine grupe, kuri yra "prarasti taškai". HDBSCAN algoritme taikoma sąvoka minimalaus klasterio dydžio, kuris šiame algoritme išreiškiamas per parametrą `minimum_cluster_size`. Turint minimalaus klasterio dydžio vertę, skaidant hierarchiją, t.y apkarpant hierarchijos medį, kiekvienas klasteris, t.y. jo dydis, vertinamas pagal parametrą `minimum_cluster_size`. Nukirstos šakos sudaro taškus, kurie neatitinka parametro, todėl laikomi išskirtimis, nes nepatenka į sudaromus klasterius. Praėję per visą hierarchiją, gauname mažesnį medį su nedideliu skaičiumi mazgų, kurių kiekvienas turi duomenų apie tai, kaip to mazgo klasterio dydis mažėja skirtingu atstumu.

6. Iš suformuotų medžių ištraukiami stabilūs klasteriai.

Intuityviai norime, kad rinkiniai išliktų ilgesniam laikui. Trumpalaikiai klasteriai dažniausiai yra tik vieno jungiamojo požymio elementai. Sukuriant plokščią grupę, pridedamas dar vienas reikalavimas, kad negali būti pasirenkamas joks palikuonis pasirinkto klasterio.

HDBSCAN algoritmas vertina klasterių patvarumą įvedant metriką $\lambda = \frac{1}{distance}$. Tada mes galime apibrėžti reikšmes λ_{birth} ir λ_{death}. Savo ruožtu, tam tikram grupavimui, kiekvienam taškui p tame klasteryje apibrėžiama reikšmė λ_p. Turint šiuos parametrus galima nustatyti klasterio stabilumą kaip: $\sum_{p \in cluster} (\lambda_p - \lambda_{birth})$.

HDBSCAN gali būti taikomas įvairios didelės apimties duomenų formos klasteriams, taip pat duomenims, kurie apibrėžti dideliu dimensijų skaičiumi.

Birch metodas sukuria medį duotiems duomenims, vadinamą charakteristikos ypatybių medžiu (CFT). CF subklasteriai palaiko reikalingą informaciją klasterizavimui, kuri leidžia nelaikyti visų įvesties duomenų atmintyje. Ši informacija apima:

- Elementų skaičius sub-klasteryje;

- Linijinė suma – n-dimensinis vektorius apimantis suma visų elementų;
- Kvadratinė suma – kvadratinė suma visų elementų L2 regulerizacijos koeeficientų;
- Centroidai – $\frac{Linijinė\ suma}{n\ elementų}$;
- Kvadratiniai normalizuoti centroidai.

Birch algoritmas turi du parametrus: slenkstį ir šakojimą. Šakojimosi veiksnys riboja mazgo pasiskirstymų skaičių, o slenkstis riboja atstumą tarp įvesties pavyzdžio ir esamų poklasių.

Šis algoritmas gali būti laikomas atskiru klasterizavimo atveju arba duomenų mažinimo metodu, nes jis sumažina įvesties duomenis į sub-blokų rinkinį, kuris gaunamas tiesiai iš CFT. Šiuos sumažėjusius duomenis galima toliau apdoroti. T. Zhang ir kt. [28] pristatė šį algoritmą, kaip klasterizavimo sprendimą esant dideliam požymių (dimensijų) skaičiui, didelėms duomenų apimtims. Metodas pasižymėjo savo dinamiškumu sprendžiant klasterizavimo problemas, išnaudojant turimus laiko ir atminties resursus. BIRCH metodas dažniausiai aptinka geriausius klasterizavimo sprendinius, atliekant vieną duomenų skanavimą, t.y. pa žingsninį įrašų perrinkimą. Atliekant papildomus įrašų perrinkimus, klasterizavimo rezultatai tik patikslinami. Šio metodo dėka galima atskirti išskirtis iš bendros duomenų imties.

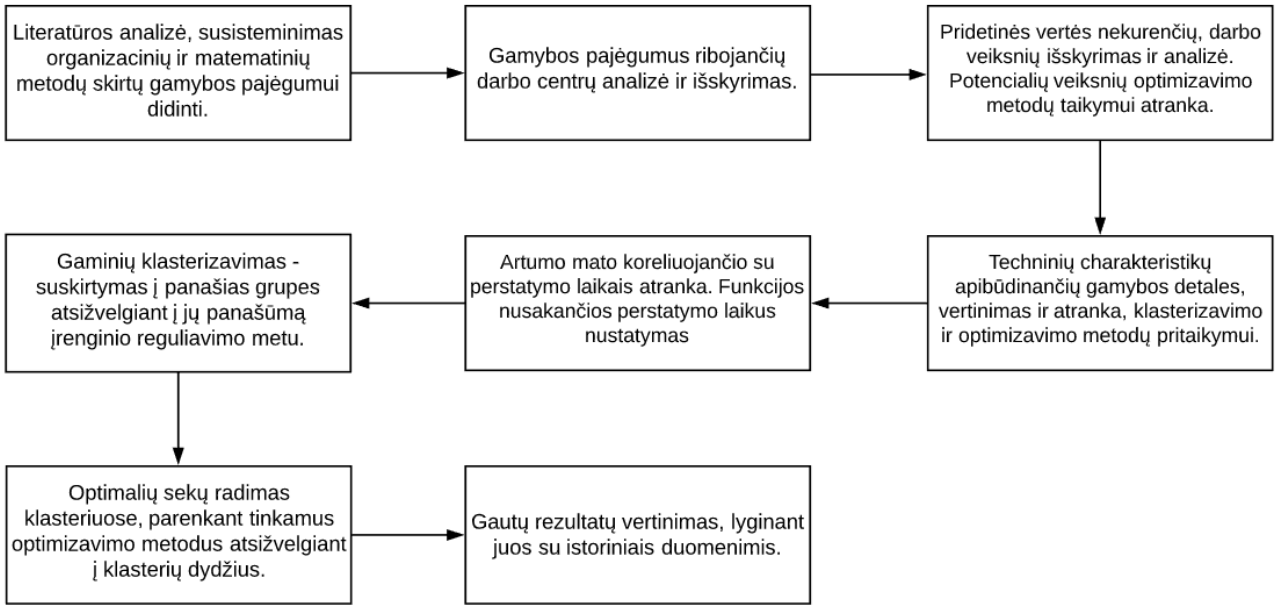
1.4 Literatūros apžvalga (apibendrinimas)

Darbo tema yra gamybinio srauto optimizavimas remiantis turima gamybine / technologine informaciją, bet taikant tik matematinius sprendimus. Literatūros apžvalgos tikslas - patikrinti gamybiniam sektoriuje taikomus organizacinius metodus, jų esmę ir taikymo sritį. Įsitikinta, kad organizaciniai metodai, tokie kaip LEAN ir AGILE, neapima pilno gamybos optimizavimo proceso. Išvada: optimalu taikyti organizacinius metodus papildant matematiniais optimizavimo sprendimais kadangi jų veikimas ir taikymo sritys nepersidengia, o tik papildo vienas kitą. Nagrinėjama problema optimizavime išskiriama kaip atskira šaka JSeP, kuri sprendžiama ją interpretuojant kaip TSP. Tai NP sudėtingumo problema, kuri jautri kintamųjų skaičiui. Peržvelgiant euristinius / meta-euristinius metodus, nustatyta, kad ir geriausiu atveju, maksimalus galimas sekos taškų skaičius siekia ~210. Mūsų duomenyse sekos taškai sudaro ~300. Todėl problemą reikia nagrinėti kombinuojant skirtingus metodus. Atsižvelgiant į gamybinius srautus ir buferinių sandėlių apribojimus, pasirinktas sprendimas gamybą skaldyti porcijomis. Tam tikrose porcijose atlikti TSP sprendimą, taikant optimizavimo algoritmus: GA, RNN, k-opt, Simulated annealing. Bendras gamybos skaldymas turėtų būti atliekamas taikant klasterizavimo metodus, o jų tikslo funkcija TSP sprendinio vertė artėjanti prie 0. Peržvelgiant literatūrą matoma, kad klasterizavimo algoritmai yra jautrūs parametrams: įrašų skaičiui, požymių skaičiui, klasterių formai / dydžiui. Nagrinėjamiems duomenims būdinga: žemas įrašų skaičius, didelis požymių skaičius, neapibrėžtas klasterių dydis ir forma. Remiantis šiais duomenimis, iš populiarių klasterizavimo metodų pritaikomų konkrečiai problemai lieka šie: HDBSCAN, DBSCAN, BIRCH. Kitais klasterizavimo atvejais, dėl didelio požymių (dimensijų) skaičiaus, reikia taikyti dimensijų mažinimo priemones tokias kaip PCA ar kt.

2 Pirminių duomenų analizė ir tyrimo metodų optimalių sekų paieškai pagrindimas

2.1 Tyrimo metodų optimalių sekų paieškai pagrindimo eiga

Šiame skyriuje išdėstyta tyrimo eiga: pirminių duomenų analizė ir atranka, taikomų tyrimo metodų charakterizavimas (tipai, parametrai, modifikacijos). Darbo tikslas - padidinti gamybos pajėgumus, sudarant optimalias partijų sekas, ribojančiuose gamybos srautą darbo centruose. Šiam tikslui įgyvendinti sudaryta veiksmų seka žr. 2 pav. „Tyrimo eigos schema nustatant optimalias gamybos sekas“



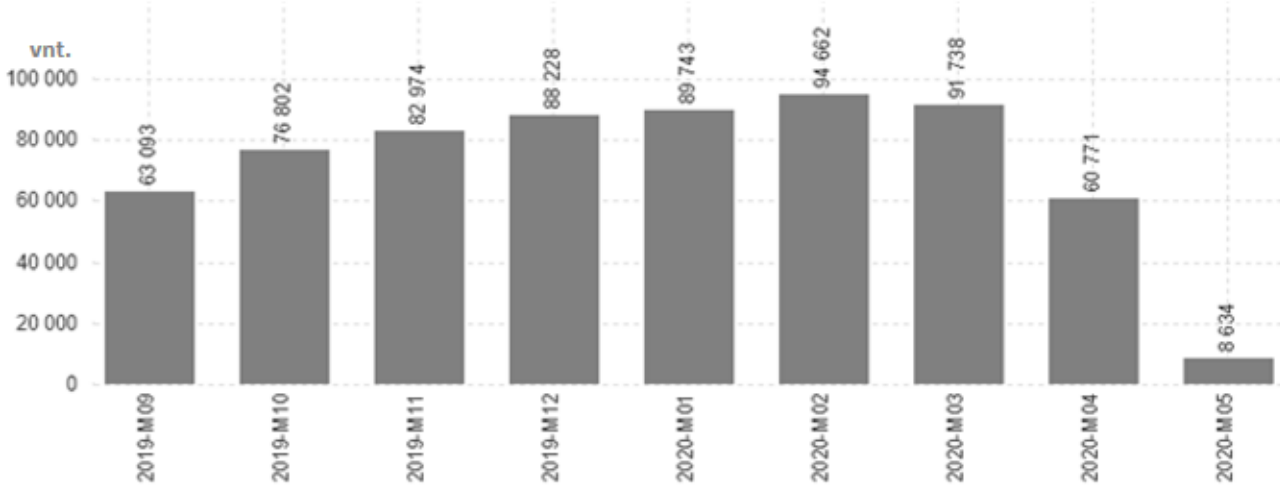
2 pav. Tyrimo eigos schema nustatant optimalias gamybos sekas

Pirmiausia atliekame literatūros analizę, tikslu siekdami informaciją apie organizacinius ir matematinius metodus, skirtus gamybos pajėgumui derinti. Remiantis nagrinėjamos įmonės duomenimis, nustatomi gamybos pajėgumus ribojantys darbo centrai. Susisteminamos operacijos atliekamos darbo centre. Atrenkami pridėtinės vertės nekuriantys veiksniai. Išskiriamas potencialus veiksnys, nekuriantis pridėtinės vertės ir galimas kandidatas optimizavimo metodų taikymui - siekiame sumažinti jo įtaką gamybos pajėgumams. Nustatomi ir aprašomi klasterizavimo ir optimizavimo metodai bei jiems taikomi parametrai (parametrų gardelės). Atliekamas duomenų transformavimas ir jų apdorojimas, siekiant pritaikyti juos optimizavimo metodams (kaip įeitinius duomenis). Atliekamas gaminių (detalių) klasterizavimas ir sudaromi klasteriai. Priklausomai nuo jų dydžio, klasteriai išskirstomi į dvi grupes. Šioms grupėms taikomi skirtingi matematiniai metodai. Rezultatai vertinami lyginant su istorinėmis sekomis taikytomis gamyboje. Remiantis tokia veiksmų

seka buvo įvykdytas tyrimas apie gamybos pajėgumo didinimą, sudarant optimalias partijų sekas ribojančiuose gamybos srautą darbo centruose.

2.1 Trumpas įmonės apibūdinimas

Tam, kad tyrimas būtų sėkmingas, reikia ne tik žinoti organizacinius ir matematinius metodus, taikomus gamybos pajėgumams gerinti, bet ir pažinti konkrečią nagrinėjamą įmonę, žinoti jos poreikius. Tyrimo objektas yra korpusinių baldų gamybos įmonė „IKEA Industry“. Įmonės veikla yra vykdoma ne vienerius metus. Įmonėje dirba apie ~600 darbuotojų. Įmonė nevysto marketingo ar inovatyvių konstrukcinių sprendimų, nesiekia produkcijos patrauklumo. Įmonės gamybos / pardavimų apimtys yra tiesiogiai susiję su jų gaminių konkurencingumu kainos atžvilgiu. Žema kaina ir aukštas gamybos pajėgumas yra pagrindinė varomoji jėga šioje įmonėje. Per mėnesį pagaminama apie 90 000 gaminių. Pažymėtina, kad pastebimas augimas per pastaruosius metus žr. 3 pav. „Įmonės gamybos apimtys 2019m. - 2020m.“



3 pav. Įmonės gamybos apimtys (vnt.) 2019m. - 2020m.

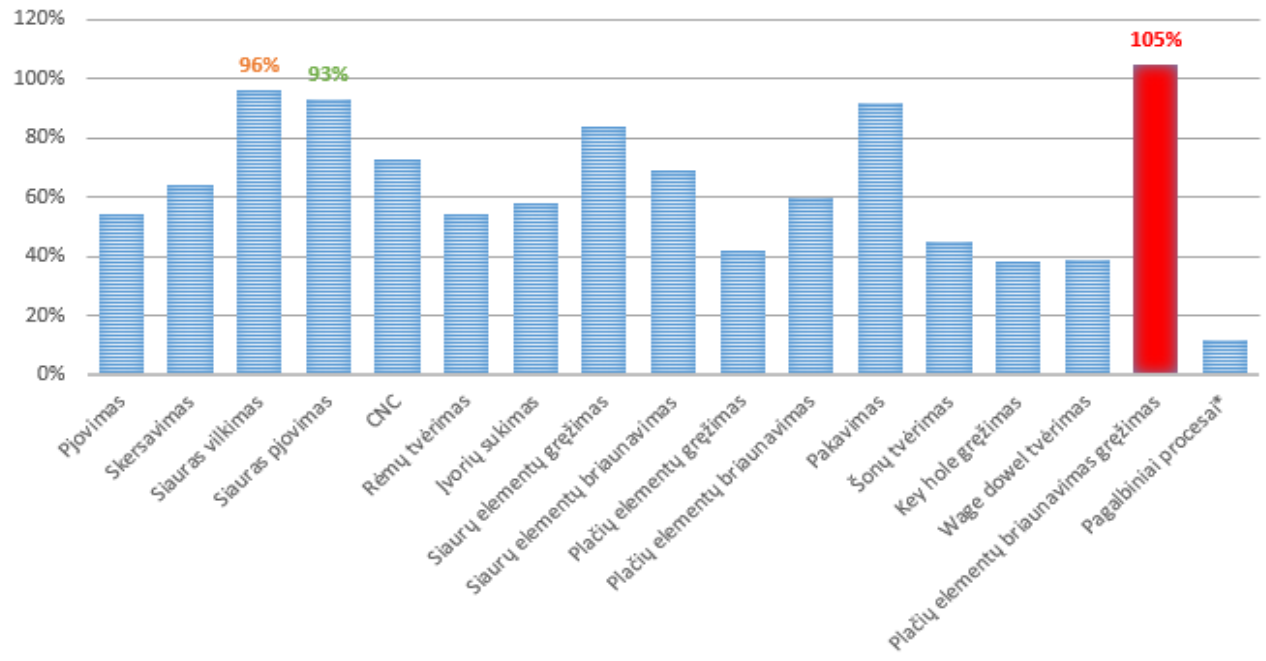
Įmonė yra orientuota į baldų gamybą iš kaširuotos plokštės. Siauras asortimentas, t.y. keturios gaminių kolekcijos, kurias sudaro įvairių tipų baldai: spintos, komodos, stalai, naktiniai staliukai ir spintelės. Gamyba yra suskaidyta į du lygiagrečiai veikiančius srautus: siaurų ir plačių elementų gamybą. Trečiasis gamybos srautas yra pirmuosiuose dviejuose srautuose pagamintos produkcijos komplektavimas ir pakavimas. Gamybos srautus sudaro nuosekliai veikiančios gamybos operacijos (pvz.: pjovimas, briaunavimas, frezavimas, gręžimas).

2.2 Pradiniai duomenys ir jų vertinimas

Keliama problema pagrįsta baldus gaminančios įmonės 2019 m. duomenimis. Pirmajame etape nustatomi kritiniai darbo centrai, ribojantys gamybos apimčių didinimą. Analizei naudojami duomenys:

- Mėnesiniai pagamintos produkcijos pardavimai FY 2019 m. žr. 1 priedą;
- Darbo dienų skaičius darbo centruose FY 2019 m. žr. 2 priedą;
- Darbo grafikai (pamainų apskaitos tipai) darbo centre FY 2019 m. žr. 3 priedą;
- Vidutinis darbo centrų efektyvumas FY 2019 m. žr. 4 priedą.

Visi darbo centrai sugrupuojami pagal jų tarpusavio pakeičiamumą. Tolimesni apkrovimo rodikliai yra pateikiami darbo centrų grupėms. Nustatyta, kad gamybos apimtis ribojantys darbo centrai priklauso grupei: plačių elementų briaunavimas ir gręžimas žr 4 pav. "Gamybos darbo centrų metinis apkrovimas". Įrenginių metinis apkrovimas 105%. Atotrūkis tarp artimiausios darbo centrų grupės ribojančios srautus siekia 9%. Tai suteikia galimybę susitelkti tik į vieną darbo centrų grupę. Ši darbo centrų grupė riboja visą gamybos srautą. Optimizavus srautą ir sumažinus bendrą apkrovimą konkrečiuose darbo centruose, galimas gamybos pajėgumo padidinimas iki 9%.



4 pav. Gamybos darbo centrų metinis apkrovimas

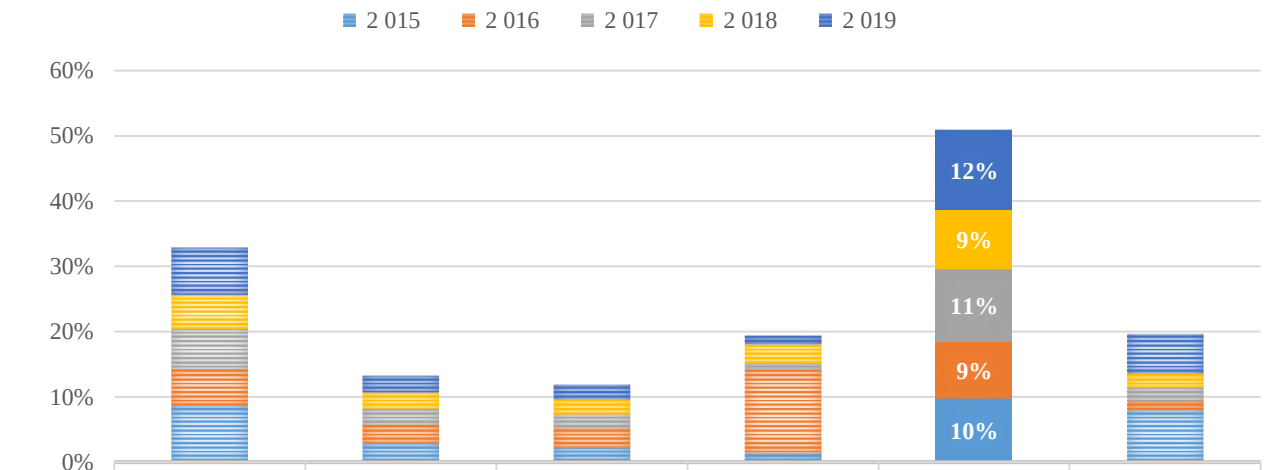
Peržvelgus mėnesinius apkrovimo rodiklius, matoma esant išskirčių siauro vilkimo grupėje. Tai pavieniai atvejai, kurie gali būti kompensuojami gamyboje išlyginant gamybos taktą, arba

perskirstant gamybos apkrovimą už įmonės ribų (subranga) žr. 5 pav. “Gamybos darbo centrų mėnesinis apkrovimas”. Tokiems darbo centrams taikyti optimizavimą nėra tikslinga.

Operacijų grupės	M09	M10	M11	M12	M01	M02	M03	M04	M05	M06	M07	M08	VISO
Pjovimas	61%	75%	70%	67%	69%	51%	51%	39%	36%	40%	43%	43%	54%
Skersavimas	34%	56%	90%	83%	64%	68%	76%	62%	51%	55%	69%	66%	64%
Siauras vilkimas	52%	62%	79%	72%	98%	108%	90%	101%	89%	113%	142%	140%	96%
Siauras pjovimas	57%	69%	94%	88%	92%	113%	89%	92%	68%	112%	117%	118%	93%
CNC	35%	38%	83%	94%	54%	77%	103%	72%	62%	71%	95%	90%	73%
Rėmų tvėrimas	23%	26%	63%	54%	46%	67%	48%	58%	54%	59%	76%	77%	54%
Įvorių sukimas	29%	32%	65%	55%	42%	58%	49%	60%	61%	64%	92%	92%	58%
Siaurų elementų gręžimas	56%	65%	99%	103%	89%	101%	84%	80%	72%	83%	85%	77%	84%
Siaurų elementų briaunavimas	23%	31%	64%	47%	56%	51%	69%	67%	99%	103%	115%	94%	69%
Plačių elementų gręžimas	12%	24%	64%	64%	41%	78%	56%	25%	43%	25%	34%	33%	42%
Plačių elementų briaunavimas	26%	30%	42%	47%	61%	56%	73%	67%	75%	79%	78%	84%	60%
Paletizavimas	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Pakavimas	62%	67%	99%	99%	99%	94%	106%	93%	91%	91%	99%	98%	92%
Šonų tvėrimas	0%	0%	0%	0%	0%	0%	80%	84%	99%	95%	93%	91%	45%
Key hole gręžimas	0%	0%	0%	0%	0%	0%	47%	58%	65%	87%	92%	108%	38%
Wage dowel tvėrimas	0%	0%	0%	0%	0%	0%	45%	72%	73%	83%	94%	104%	39%
Plačių elementų briaunavimas, gręžimas	**63 %**	**63 %**	**96 %**	**107 %**	**98 %**	**133 %**	**118 %**	**111 %**	**98 %**	**120 %**	**126 %**	**127 %**	**105 %**
Pagalbiniai procesai*	13%	19%	5%	8%	16%	9%	15%	13%	10%	12%	13%	12%	12%

5 pav. Gamybos darbo centrų mėnesinis apkrovimas

Remiantis ekspertine nuomone, išskiriamas gamybos srautą ribojantis darbo centras – HOMAG4X (gręžimo ir briaunavimo apdirbimo linija – kas atitinka gautus rezultatus dėl įrenginių užkrovimo). Toliau darbe naudojami duomenys yra siejami tik su šiuo įrenginiu HOMAG4X. Tarp gaminio efektyvumą apsprendžiančių veiksnių labiausiai išsiskiria įrenginio reguliavimo laikas, kai keičiamas gamybos srautas iš vienos detalės į kitą žr. 6 pav. Į”renginio HOMAG4X prastovos, procentinis pasiskirstymas nuo viso darbo laiko”. Šis segmentas yra potenciali galimybė pagerinti įrenginio efektyvumo rezultatus.



	Gedimai	Įreankių keitimas	Įrenginių valymas	Neapibrėžta	Perstatymai	Testai
2 019	7%	3%	2%	1%	12%	6%
2 018	5%	2%	2%	3%	9%	2%
2 017	6%	2%	2%	1%	11%	2%
2 016	6%	3%	3%	13%	9%	1%
2 015	9%	3%	2%	1%	10%	8%

6 pav. Įrenginio HOMAG4X prastovos, procentinis pasiskirstymas nuo viso darbo laiko

Gedimai, testai ir neapibrėžtos užduotys, pagal savo atsiradimo prigimtį, yra neplanuojamos prastovos, kurios yra neprognozuojamos. Įrankių keitimas ir įrenginių valymas yra periodiniai darbai, šių prastovų laikai priklauso nuo naudojamų įrankių savybių arba nuo darbo optimizavimo galimybių. Detalių perstatymo trukmė priklauso nuo veiksmų kiekio, kuriuos reikia atlikti reguliuojant skirtingus įrenginio mazgus pereinant iš vienos detalės į kitą. Veiksmų kiekis priklauso nuo detalių panašumo iš kurios į kurią pereinama. Detalių panašumas apsprendžiamas įvairiais technologiniais parametrais. Konkrečiu atveju galima išskirti tris grupes/ duomenų šaltinius:

- Gręžimo įranga. Pagrindiniai parametrai tiesiogiai nuskaitomi iš gręžimo įrangos duomenų bazės. Duomenų formatas *.b2;* žr. 5 priedą. Parametrai: gręžimo galvų modeliai ir gręžimo galvos surinkimas (t.y. gręžimo galvą sudaro nustatytas skaičius špindelių nustatytų žingsnių vienas nuo kito. Kiekvienai detalei individualiai priskiriama, kurie špindeliai montuojami ir kokio skersmens/ilgio grąžtai naudojami);
- Briaunavimo įranga. Pagrindiniai parametrai tiesiogiai nuskaitomi iš PLC duomenų bazės. Duomenų formatas *.accdb*. Duomenys: briaunų profilio tipai, briaunų pločiai;
- Pagalbinė įranga ir bazinis detalių aprašymas. Pagrindiniai parametrai tiesiogiai nuskaitomi iš technologinės gaminio informacinės duomenų bazės (duomenų formatas *.accdb*) ir esamos ERP sistemos:
    - Pakėlimo/ pasukimo liftai (įėjimo/išėjimo greičiai m/min);

- Detalių dalinimas (įėjimo/išėjimo greičiai m/min);
- Matmenys (ilgis, plotis, aukštis);
- Spalva.

Šiame darbe JSeP problema formuluojama kaip TSP, o atstumas tarp taškų (detalių) formuluojamas kaip detalės panašumo matas, išreikštas atsižvelgiant į detalės charakteristikas darbo centre (gręžimo galvos, nustatymų koordinatės, briaunavimo profiliai, pastūmos greičiai ir kt.) . Darbo metu tikrinami trys panašumo matai Euklido, Kosine ir Mahalanobio. Problemos sprendimas išskaidomas į dvi dedamąsias: gaminių klasterizavimas ir TSP sprendimas klasteryje.

2.3 Duomenų apdorojimas / transformacija

Duomenys, pagal gavimo vietą, suskirstyti į dvi grupes:

- Gręžimo / briaunavimo įrenginių duomenų bazė – duomenų failai .b2 formatu.
- Gaminio technologinės informacijos duomenų bazė .accdb formatu.

Duomenims išgauti ir transformuoti naudojami Python paketai: `pandas, pyodbc.` Pastarasis yra skirtas prisijungimo duomenis suformuoti su SQL tipo duomenų bazėmis. Duomenų nuskaitymui taikomas paketas `pandas`. Paketas skirtas duomenų nuskaitymui ir transformavimui į atitinkamą formatą tolimesniems darbo veiksmams. Yra taikomos ir įrenginio programinės įrangos eksporto funkcijos, kurios apdorojamos programine įranga MS Exel.

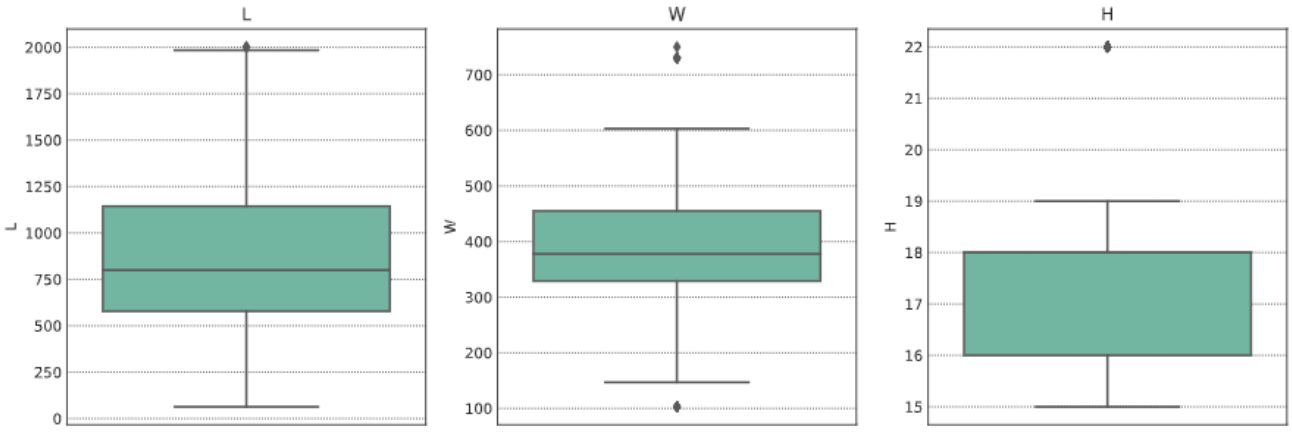
Duomenis sudaro trys grupės: binariniai kintamieji, kategoriniai kintamieji ir tolydieji kintamieji. Duomenims papildomai atliktas normalizavimas ir duomenų transformavimas, t.y. kategoriniai kintamieji paverčiami į binarinius. Pašalinami neaktualūs kintamieji, tokie, kaip detalių pavadinimai, darbo centrų koduotė (nagrinėjamos detalės apdirbamos viename darbo centre) ir agreguotos detalės kodas (darbui naudojamas tik išplėstinis). Patikrinami įrašai ar nėra trūkstamų reikšmių, nustatyta, kad tokių reikšmių nėra. Paruošimo veiksmai atliekami taikant python programavimo kalbą, naudojami paketai: `numpy` , `pandas`, `scipy` žr. 6 priedą Duomenų transformavimas ir apdorojimas.

2.4 Žvalgomoji duomenų analizė

Žvalgomoji analizė koncentruojama į technologinius duomenis aprašančius gaminio detales. Remiamasi tuo, kad ne visi parametrai gali būti reikšmingi esamoje problemoje. Jie netiesiogiai susiję su įrenginių perreguliavimu. Parametrai iš gręžimo įrenginių - yra tiesioginiai duomenys įrenginio perreguliavimui. Duomenų analizei taikomas python paketas `scipy,` o vizualizavimui `plotly` ir `matplotlib` paketai. Turimi duomenys:

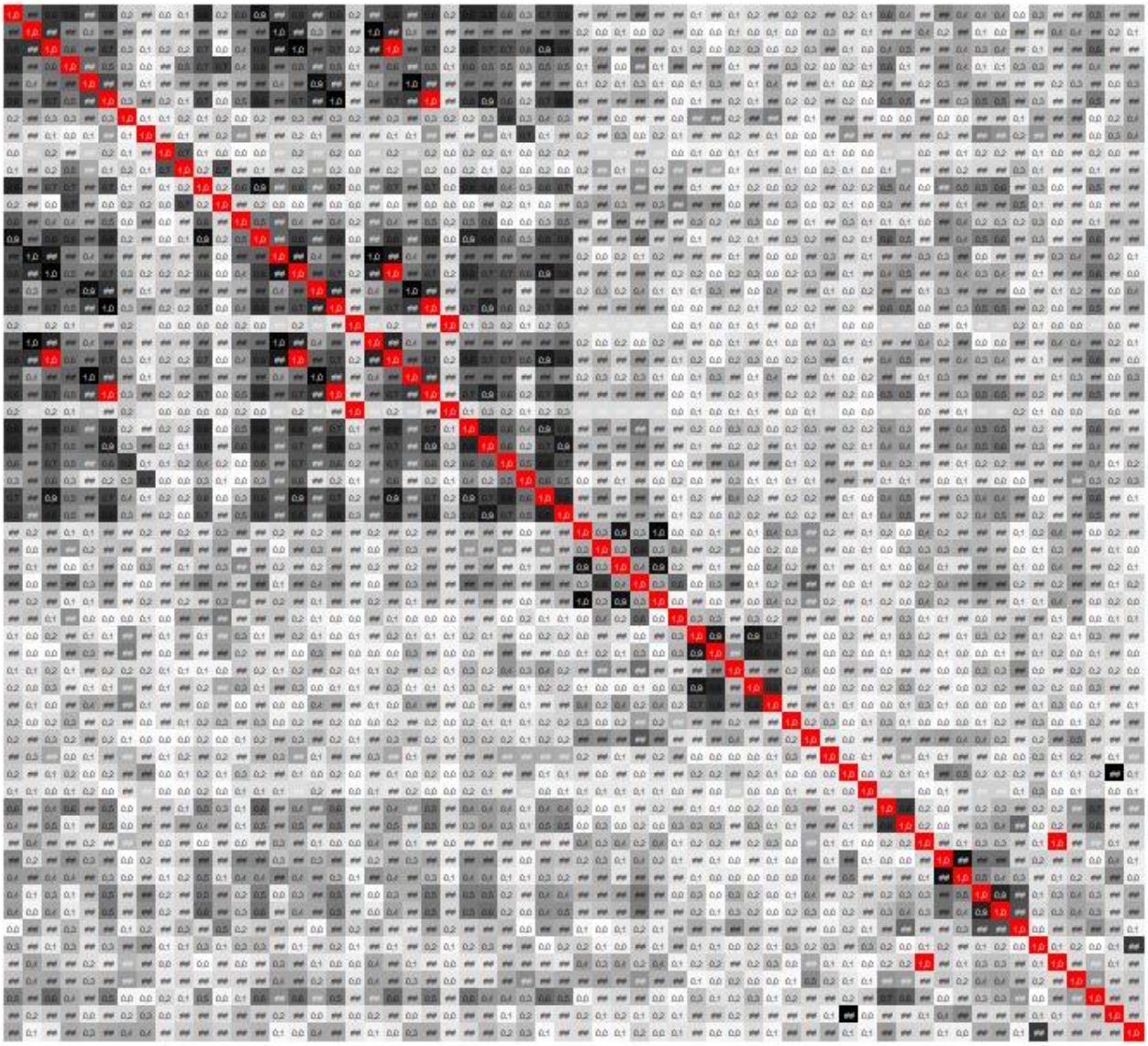
- Kategoriniai kintamieji (51);
- Binariniai kintamieji (19);
- Tolydieji kintamieji (3);

Atlikus pirminę duomenų peržiūrą, nustatėme, kad 51 kategorinį kintamąjį sudarė 428 kategorijos. Tarp visų binarinių ir kategorinių kintamųjų, rasta net 13, kurie yra nereikšmingi, t.y. juos sudarė tik viena kategorija. Šie kintamieji pašalinti ir toliau darbe buvo nebenaudojami. Iš kintamųjų histogramų sprendžiame, kad dauguma binarinių kintamųjų nėra pasiskirstę tolygiai ir jie dažniau parodo išimtis negu suskirsto duomenis į dvi grupes. Tai būdinga ir kai kuriems kategoriniams kintamiesiems žr. 7 priedą "Kategorinių ir tolydžių kintamųjų histogramos".



7 pav. Kategorinių kintamųjų Box-plot diagrama

Tolydžių kintamųjų mūsų duomenyse yra tik trys. Jie apibūdina detalių matmenis: ilgis, plotis, aukštis. Detalių aukščiai (storis) yra tik trijų rūšių, o 22 mm. storio detalės sudaro mažąja dalį viso asortimento. Detalių pločio pasiskirstymas yra artimas normaliajam skirstiniui su nedideliu viršūnės nuokrypiu, t.y mediana paslinkta, yra tik 3 išskirtys. Detalių ilgiuose išskirčių nėra, skirstinys nėra artimas normaliajam žr. 7 pav. "Kategorinių kintamųjų Box-plot diagrama". Toliau patikriname požymių koreliaciją tarpusavyje. Taikome Pirsono koreliacijos koeficientą. Koreliacijos taikymo tikslas: patikrinti ar duomenyse nėra reikšmių, kurių koreliacijos koeficientas yra =1. Tai reikštų, kad viena reikšmė pilnai apibūdina kitą. Nagrinėjant gautus duomenis, matome (neskaitant diagonalės), kad yra net 6 reikšmės tarpusavyje pilnai koreliuojančios žr. 8 pav. „Techninių charakteristikų tarpusavio koreliacija". Tokiu atveju, viena iš dviejų reikšmių yra pašalinama. Taip sumažinamas dimensijų skaičius neprarandant informacijos.



8 pav. Techninių charakteristikų tarpusavio koreliacija

2.5 Tyrimo metodai

2.5.1 Duomenų klasterizavimas

Šiam tyrimui buvo atrinkti aštuoni klasterizavimo metodai: K-Means, DBSCAN, BIRCH., Hrierarchical-ward, Hrierarchical-complete, Hrierarchical-average, Hrierarchical-single, HDBSCAN,. Klasterizavimo metodų realizavimui buvo taikomi `scikit-learn` ir `hdbscan` – Python programiniai paketai. Realizuotą kodą žr. 8 priedą “Klasterizavimo metodų realizavimas Python programavimo kalba”. toliau darbe pateiksime pilnus sudarytus modelius ir jiems priskirsime parametrus. Modeliams buvo koreguojami tik tam tikri parametrai, ši informacija pateikiama papildomai, visais kitais atvejais paliekami baziniai parametrų rinkiniai, atsižvelgiant į tai, kad jų įtaka galimai maža, arba jie yra tinkami esamai situacijai.

KMeans klasterizavimo modelis:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=30, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

- `n_clusters` – parametras, nurodantis klasterių skaičių. Tyrimo metu buvo tikrinamos visos reikšmės priklausančios intervalui [10;31]. Šis parametras taip pat yra naudojamas modeliuose: *Birch*, *Agglomerative Clustering*.
- `n_init` – Algoritmo pakartojimų skaičius skirtinguose centroiduose. Parametras padidintas iki 100.

DBSCAN klasterizavimo modelis:

```
DBSCAN(algorithm='auto', eps=6.9, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=5, n_jobs=-1, p=None)
```

- `eps` – Maximali distancija tar dviejų taškų klasteryje. Darbe priskiriama parametrų gardelė intervale [0.1:7] žingsniu =0.1.
- `min_samples` – mažiausias taškų kiekis klasteryje (apribojamas mažiausias klasterio dydis).
- `metric` – artumo matą apibrėžiantis atstumą tarp dviejų taškų (tam naudojamas Euklido algoritmas). Euklido artumo matas parinktas patikrinus koreleciją su experimentiniu būdu nustatytais perstatymo laikais. Taip pat šis parametras yra naudojamas modeliuose: AgglomerativeClustering, HDBSCAN.

BIRCH klasterizavimo modelis:

```
Birch(branching_factor=50, compute_labels=True, copy=True, n_clusters=30,
      threshold=0.5)
```

- `n_clusters` – parametras, nurodantis klasterių skaičių. Tyrimo metu buvo tikrinamos visos reikšmės priklausančios intervalui [10;31];
- `threshold`, slenkstis kuris apibrėžia kada klasteris gali skilti į du mažesnius klasterius. Darbe priskiriama parametrų gardelė intervale [0.1:1] žingsniu =0.1

Hierarchinio klasterzavimo modeliai (ward, complete, single, average):

```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='single', memory=None, n_clusters=30)
```

- `affinity` – naudojamas Euklido atstumas;
- `linkage` – parametras nurodo kokį hierarchinio klasterizavimo tipą naudosime. Darbe naudojami: *Ward*, *Complete*, *Average*, *Single*;
- `n_clusters` – parametras, nurodantis klasterių skaičių. Tyrimo metu buvo tikrinamos visos reikšmės priklausančios intervalui [10;31];

HDBSCAN klasterizavimo modelis:

```
HDBSCAN(algorithm='best', allow_single_cluster=False, alpha=1.0,
        approx_min_span_tree=True, cluster_selection_epsilon=0.0,
        cluster_selection_method='eom', core_dist_n_jobs=4,
        gen_min_span_tree=True, leaf_size=40,
        match_reference_implementation=False, memory=Memory(location=None),
        metric='euclidean', min_cluster_size=5, p=None,
        prediction_data=False)
```

- `metric` – naudojamas Euklido algoritmas;
- `min_cluster_size` – pagal savo prasmę yra identiškas jau nagrinėtam parametrui `min_samples` (DBSCAN klasterizavimo modelis)

2.5.2 Trumpiausio maršruto paieška

Šiam tyrimui buvo atrinkti aštuoni optimizavimo metodai: *Branch and Bound, Greedy, Randomized, Minimum spanning tree, 2-opt, Simulated annealing, Genetic, Ant colony*. Optimizavimo metodų realizavimui buvo taikomi Python programiniai paketai tokie kaip: `acopy`, `TSP_Solver`, `mlrose`. *Greedy* ir *Brabch and Bound* rializavimas atliktas tyrimo metu. Realizuotas kodą žr. 9 priede Optimizavimo metodų rializavimas python programavimo kalba. Toliau bus pateikti sudaryti modeliai ir jiems priskirti parametrai. Darbe pateiksime pilnus sudarytus modelius. Tokiu atveju, kai algoritmams buvo koreguojami tam tikri parametrai, ši informacija pateikiama papildomai, o visi kiti parametrai paliekami baziniai.

Optimizavimo modelis taikant *Genetic* algoritmą:

```
genetic_algorithm(TSPOpt, mutation_prob=0.1, max_attempts=10, pop_size=1000,
max_iters=np.inf, curve=False, random_state=None)
```

- `Problem` – optimizavimo objektas, šis paketas suteikia spręsti įvairęs optimizacimo problemas, todėl jam būtina nusakyti sprendžiamą problemą. Mūsų tyrimo atveju priskirta TSP, taip pat šis parametras nustatomas ir `simulated_annealing` modelyje.
- `pop_size` – populiacijos dydis algoritmo skaičiavimo metu, atsižvelgiant į klasterių dydžius (t.y. jie nėra dideli), šis parametras nustatytas 100.
- `mutation_prob` – mutacijos tikimybė reprodukcijos metu. Darbe priskiriama parametrų gardelė intervale [0.1:1] žingsniu =0.1

Optimizavimo modelis taikant *Simulated annealing* algoritmą:

```
simulated_annealing(TSPOpt, schedule=<mlrose.decay.GeomDecay
object>, max_attempts=10, max_iters=inf, init_state=None, curve=False, random_state=Non
e)
```

- `problem` – Mūsų tyrimo aveju priskirta TSP;
- `schedule` – Tvarkaraštis, naudojamas temperatūros parametro vertei nustatyti. Darbe naudojamas `<mlrose.decay.GeomDecay object>`;
- `max_attempts` (int, default: 10) – Maksimalus bandymų skaičius rasti geresnį kaimyną kiekviename žingsnyje.

Optimizavimo modelis taikant *Ant colony* algoritmą:

```
acopy.solvers.Solver(rho=0.03, q=1, top=None, plugins=None)
acopy.ant.Colony(alpha=1, beta=3)
```

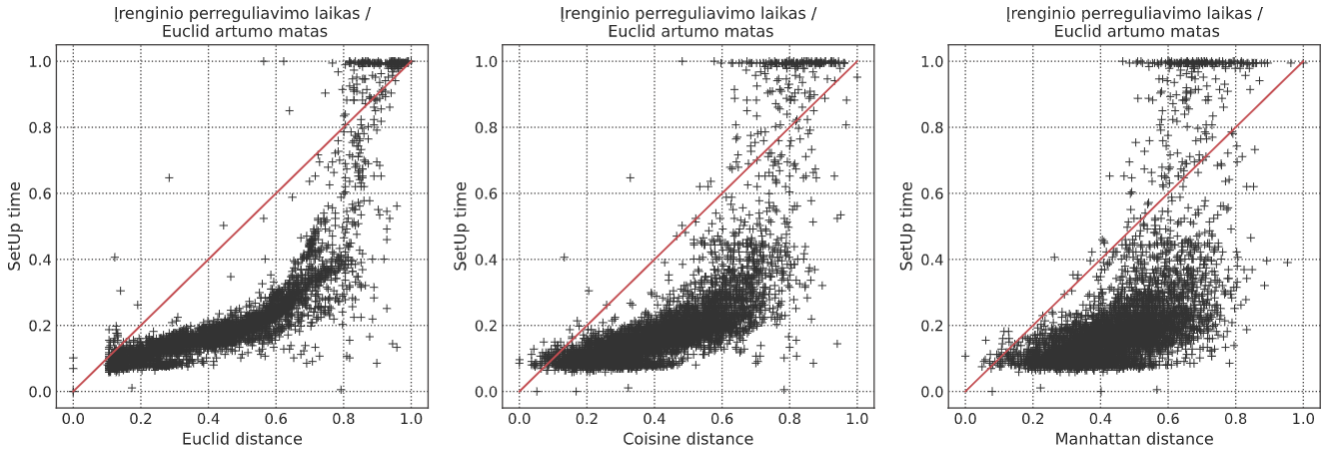
- `rho` – Feromono kiekis išgaruojantis kiekvieną iteracija. Darbe priskiriama parametrų gardelė intervale [0.1:1] žingsniu =0.1;
- `q` – feromono kiekis kurį gali sukaupti kiekvienas skruzdelynas, Darbe priskiriama parametrų gardelė intervale [0.1:1] žingsniu =0.1;
- `top` – Skruzdžių skaičius disponuojančių feromonu. Darbe priskiriama parametrų gardelė intervale [1:10] žingsniu =1

Likusiems optimizavimo metodams: *Brabch and Bound, Greedy, Randomized, Minimum spanning tree, 2-opt*, parametrų atrankos nebuvo.

3 Gamybos srautų optimizavimo vykdant optimalių gamybos sekų paiešką tyrimo rezultatai

3.1 Artumo mato parinkimas

Trumpiausio maršruto paieškos skaičiavimai yra paremti atstumu tarp taškų A ir B. Tyrimo atveju, taškai - tai detalės, o atstumas tarp taškų yra laikas, skirtas perreguliuoti gamybinę liniją keičiant gamybos partijas. Įmonėje yra ~300 gaminamų detalių, todėl iš viso yra (n^2)/2=45000 perstatymo laikų. Be to, nėra skaičiavimo metodikos, kuri nustatytų tikslų perreguliavimo laiką. Eksperimentiniu būdu patikrinti visas galimas kombinacijas perreguliuojant įrenginio gamybą iš vienos detalės į kitą yra labai imlu laikui ir finansiškai nuostolinga. Analizuojamoje įmonėje nėra nustatyti visi perstatymo laikai (nustatyta ~ 10% nuo visų galimų kombinacijų). Tyrimui atlikti pasirinkti artumo matai: Euklido, Kosino, Manhattan'o. Artumo matai nustatomi remiantis techninėmis kiekvienos detalės charakteristikomis (charakteristikos įtakojančios įrenginio perreguliavimo trukmę – tam tikri techniniai įrenginio reguliavimo parametrai). Nustatymui, kuris iš artumo matų yra tinkamiausias, tikrinamas jų ryšys su eksperimentiniu būdu nustatytais perstatymo laikais žr. 9 pav. "Artumo matų ir įrenginio perreguliavimo laiko ryšys".

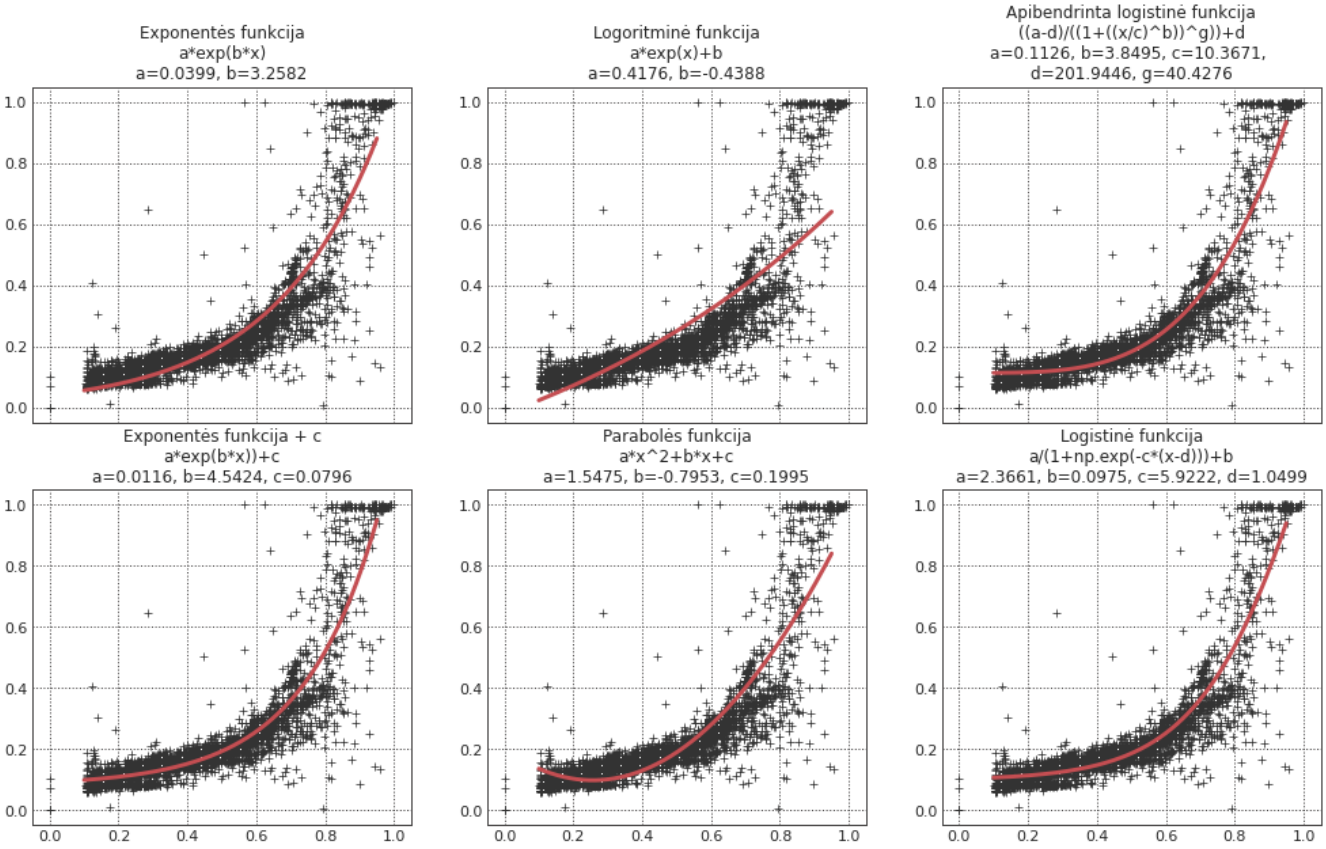


9 pav. Artumo matų ir įrenginio perreguliavimo laiko ryšys

Remiantis taškų sklaidos diagrama, galima teigti, kad, taikant visus artumo matus, įžvelgiamas ryšys tarp artumo mato ir įrenginio perreguliavimo laiko. Mažiausia duomenų sklaida matoma taikant Euklido artumo matą. Šis ryšys nėra tiesinis ir tam, kad pagrįsti priklausomybę Euklido artumo mato su įrenginio perreguliavimo laiku, atliekamas papildomas tyrimas, kuriame pritaikoma tinkama matematinė funkcija. Funkcijoje išreikštas kintamasis y (įrenginio perreguliavimo laikas) per kintamąjį x (Euklido atstumo matą). Tyrimo metu, atsižvelgiant į duomenų sklaidos pobūdį, pasirinktos 6 funkcijos (4 funkcijos su skirtingu skaičiumi koeficientų):

- Eksponentinė funkcija: $y = a * \exp(b * x)$;
- Eksponentinė funkcija su papildoma konstanta - $y = a * \exp(b * x) + c$;
- Logaritminė funkcija: $y = a * \exp(x) + c$;
- Parabolė: $y = a * x^2 + b * x + c$;
- Logistinė funkcija: $y = a/(1 + \exp(-c * (x - d))) + b$;
- Apibendrina logistinė funkcija $y = (a - d)/(1 + ((x/c)^b)^g) + d$.

Atlikti skaičiavimai turimai duomenų imčiai (t. y. turimiems eksperimentiniu būdu nustatytiems įrenginio perreguliavimo laikams ir juos atitinkantiems artumo matams) žr. 10 pav. "Įrenginio perstatymo laikų prognozavimo kreivės".



10 pav. Įrenginio perstatymo laikų prognozavimo kreivės

Pirminis vertinimo etapas vizualinis - galima nuspręsti, kad logaritminė funkcija visiškai blogai apibūdina sąryšį tarp taškų ir jos atitikimas yra prasčiausias. Parabolės ir eksponentės funkcijos prisitaiko prie duomenų geriau, tačiau vertinant su kitomis, jos atrodo kur kas prasčiau, kreivės nepasiskirsto tarp taškų ir nuokrypiai yra dideli. Vertinant vizualiai, eksponentės su papildoma konstanta, logistinės ir apibendrintos logistinės funkcijos yra tarpusavyje artimos, gerai pasiskirsto sklaidos diagramoje. Sekančiame etape įvertinimui pasitelkiamos papildomos įrenginio perstatymo laikų prognozavimo metrikos žr. 3 lentelę „Įrenginio perstatymo laikų prognozavimo rezultatai".

Šiame etape svarbiausia tinkamai parinkti vertinimo kriterijus ir juos interpretuoti. Taikomas *Python* paketas `StatsModels`, kuris pateikia įvairias statistikas suformuotiems modeliams, tačiau interpretavimas ir atrinkimas tinkamų ir yra mūsų tyrimo dalis. Aprašomi duomenys nėra laiko eilutė, stebėjimai nekinta laike, koreliacija tarpusavyje nėra galima - negalime taikyti Durbino-Vatsono statistikos. Ekscesas neapibūdina modelių tinkamumo, tačiau nusako dažnių kreivės aštrumą lyginant su normaliuoju skirstiniu. Mūsų skirstinys yra leptakurtinis t.y. skirstinio kreivės aštrumas yra didesnis negu normaliojo. AIC (Akaikės informacijos kriterijus) yra vienas iš pagrindinių vertinimo kriterijų modelio tinkamumui nustatyti, kadangi jį skaičiuojant, yra naudojamas baudos matas už papildomą koeficiento įtraukimą, kad išvengti modelio persimokymo. Jį tikslinga naudoti tada, kai prognozavimui yra naudojama daug dimensijų. Remiantis AIC ir BIC statistikomis, galima daryti išvadą, kad geriausiai duomenis aprašanti funkcija yra eksponentė. Visos statistikos apibūdina ir įvertina modelius, bet, remiantis turimų duomenų tipu, galima teikti, kad reikšmingiausia statistika yra determinacijos koeficientas R^2 . Aukščiausia R^2. apibūdina, kad modelis geriausiai aprašo turimus duomenis. Tyrimo metu pasirinktas geriausiai duomenis aprašantis modelis yra apibendrinta logistinė funkcija, kurios determinacijos koeficientas $R^2 = 0.956$. Stebėjimų skaičius yra ženkliai didesnis negu turimų regresorių todėl nevertinamas ir koreguotas determinacijos koeficientas.

3 lentelė. Įrenginio perstatymo laikų prognozavimo rezultatai

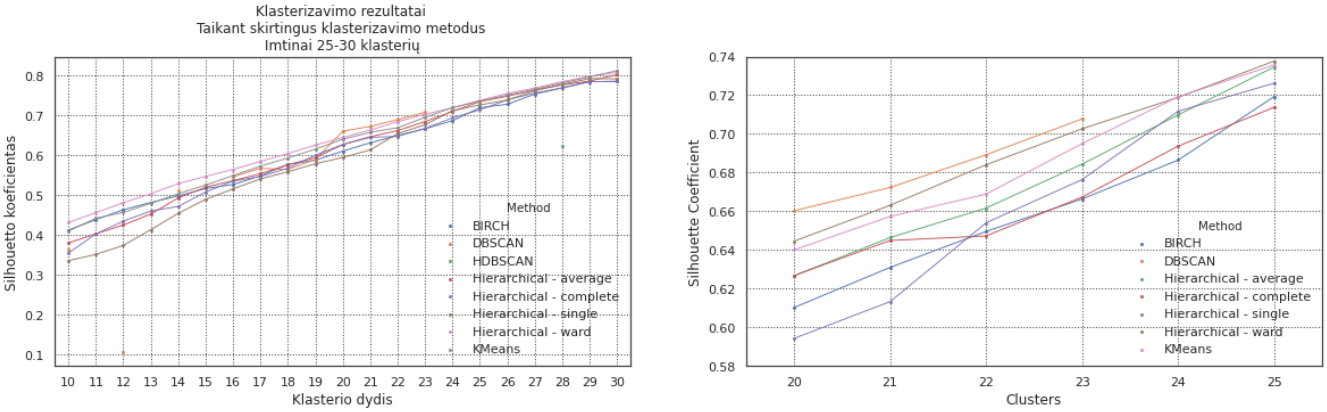
Funkcija	R^2	$Adj.R^2$	**AIC**	**BIC**	**Kurtosis**	**Durbin-Watson**
Eksponentinė funkcija	0.923	0.921	-8046	-8052	20.926	0.838
Eksponentinė funkcija su papildoma konstanta	0.941	0.939	-9115	-9092	21.334	0.855
Logaritminė funkcija	0.689	0.689	-9005	-8992	10.405	0.424
Parabolės funkcija	0.811	0.81	-10153	-10151	18.387	0.737
Logistinė funkcija	0.947	0.945	-11855	-11853	21.470	0.860
Apibendrina logistinė funkcija	0.956	0.954	-12968	-12901	21.674	0.868

3.2 Klasterizavimo metodų vertinimas, klasterių interpretavimas

Tyrimo tikslas suskaldyti duomenų imtį į mažesnes grupes, kurioms būtų galima taikyti optimizavimo metodus arba atlikti pilną duomenų perrinkimą. Tam tikslui yra pasirinkti 8 skirtingi klasterizavimo metodai (plačiau apie naudojamus klasterizavimo metodus pateikiama skyriuose Literatūros apžvalga ir Tyrimo metodai). Klasterizavimo eiga suskaldyta į du žingsnius:

1. taikant ribotą parametrų gardelę identifikuoti geriausią klasterizavimo metodą,
2. pritaikyti parametrų optimizavimo metodus klasterizavimo parametrams - vienam atrinktam metodui.

Pirmajame žingsnyje tikrinami visi klasterizavimo metodai ir atrenkami tie, kurie yra pranašesni pagal esamus duomenis. Visiems yra taikomas tas pats vertinimo kriterijus: silueto koeficientas. Klasterių skaičius yra vienas iš kintamųjų parametrų simuliuojamojo gardelėje ir yra vienas esminių kriterijų vykdomame tyrime. Nustatyta priklausomybė tarp silueto koeficiento ir klasterių skaičiaus, taikant skirtingus klasterizavimo metodus žr. 11 pav. „Klasterizavimo metodų silueto koeficiento rezultatai“.



11 pav. Klasterizavimo metodų silueto koeficiento rezultatai

Klasterių skaičiaus gardelė parenkama tarp [10:30] klasterių imtinai. Gardelės apimtys parenkamos atsižvelgiant į ekspertinę nuomonę optimalių partijų dydžiams ir jų kiekiui. Atsižvelgiant į tai, peržvelgiama ar nėra tyrimui naudingų šuolių visoje parinktoje imtyje. 11 pav. (kairė) matome, kad taikant visus klasterizavimo metodus, Silueto koeficiento kitimas yra tolygus. Augant klasterių skaičiui, tolygiai auga ir jų homogeniškumas. Atsižvelgiant į tai, apribojame savo rezultatų imtį Silueto koeficiento reikšmių intervale [0,55:0,75]. 11 pav. (dešinė) matome, kad iš visų taikomų klasterizavimo metodų, sąlygų netenkina tik 1 klasterizavimo metodas HDBSCAN. Tolimesniam klasterizavimo metodų vertinimui pasiteikiami papildomi kriterijai: minimalus ir maksimalus klasterio dydžiai žr 4 lentelėje „Geriausių klasterizavimo modelių rezultatai Top 10“ (pilnus rezultatus žr. 10 priede Klasterizavimo modelių rezultatai). Atsižvelgiant į šiuos duomenis, galime teigti, kad negalime vadovautis vien tik Silueto koeficientu, nes dauguma metodų sudaro klasterius, kurių dydžiai yra artimi 1 elementui visame klasteryje. Tokie klasteriai yra nenaudingi ir nepritaikomi tolimesniame tyrime. Todėl rezultatams įvedamas papildomas apribojimas ir atrenkami tik klasterizavimo modeliai, sudarančių klasterius, kurių narių kiekis yra nemažesnis nei 5 nariai. Remiantis gautais rezultatais, galima teigti, kad geriausias klasterizavimo metodas turimiems duomenims, yra tankiu grįstas metodas DBSCAN, jis tenkina išsikeltą sąlygą dėl minimalaus klasterio dydžio ir šioje kategorijoje yra vienintelis metodas tenkinantis kriterijų tarp 10 geriausių rezultatų.

4 lentelė. Geriausių klasterizavimo modelių rezultatai (Top 10)

Klasterizavimo metodas	Klasterizavimo modelis	Minimalus klasterio dydis	Maksimalus klasterio dydis	Klasterių skaičius	Silueto koeficientas
KMeans	{'n_clusters': 30}	4	51	30	0.812279
Hierarchical - ward	{'n_clusters': 30, 'linkage': 'ward'}	2	51	30	0.807682
Hierarchical - average	{'n_clusters': 30, 'linkage': 'average'}	2	51	30	0.802107
Hierarchical - ward	{'n_clusters': 29, 'linkage': 'ward'}	2	51	29	0.798327
KMeans	{'n_clusters': 29}	4	51	29	0.796306
Hierarchical - single	{'n_clusters': 29, 'linkage': 'single'}	2	51	29	0.792351
Hierarchical - single	{'n_clusters': 30, 'linkage': 'single'}	1	51	30	0.791512
Hierarchical - average	{'n_clusters': 29, 'linkage': 'average'}	2	51	29	0.786134
KMeans	{'n_clusters': 30}	4	51	30	0.812279
Hierarchical - ward	{'n_clusters': 30, 'linkage': 'ward'}	2	51	30	0.807682

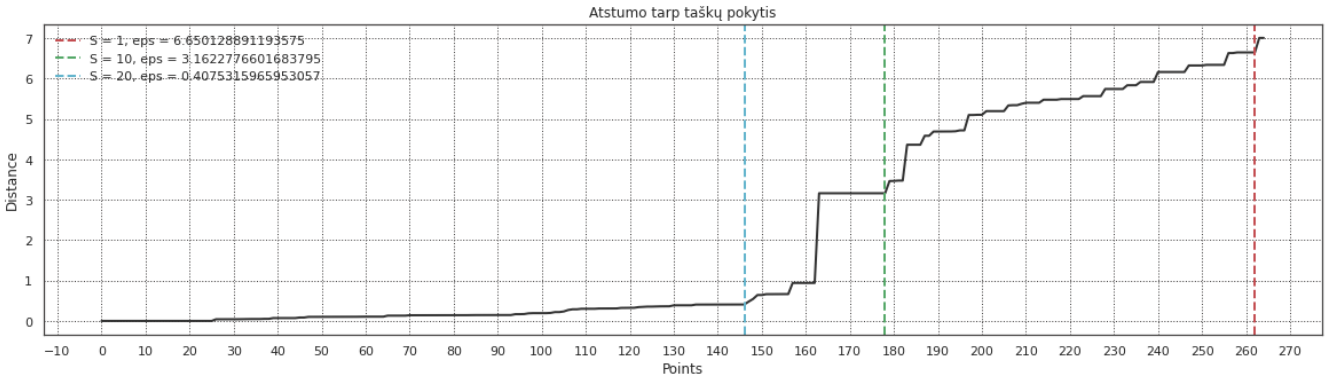
5 lentelė. Geriausių klasterizavimo modelių rezultatai esant klast. dydžių >=5 (Top 10)

Klasterizavimo metodas	Klasterizavimo modelis	Minimalus klasterio dydis	Maksimalus klasterio dydis	Klasterių skaičius	Silhouetto koeficientas
DBSCAN	{'eps': 4.8}	5	51	25	0.707756
DBSCAN	{'eps': 4.7}	5	51	25	0.707756
DBSCAN	{'eps': 5.1}	5	51	25	0.707756
DBSCAN	{'eps': 5.0}	5	51	25	0.707756
DBSCAN	{'eps': 4.9}	5	51	25	0.707756
DBSCAN	{'eps': 4.6}	5	52	25	0.704994
DBSCAN	{'eps': 4.4}	5	54	25	0.703912
DBSCAN	{'eps': 4.5}	5	54	25	0.703912
DBSCAN	{'eps': 4.0}	5	58	25	0.696738
DBSCAN	{'eps': 3.7}	5	58	25	0.696738

Atrinkome geriausią klasterizavimo metodą. Sekantis žingsnis - optimizuoti šio metodo parametrus, kad gautume geriausius rezultatus. Vertinimo kriterijus išlieka tas pats: Silueto koeficientas. Pagrindiniai užduodami parametrai DBSCAN metodui yra epsilon (eps) ir minimalus klasterių skaičius. Pastarojo reikšmė bus lygi penkiems. Tolimesnis parametrų optimizavimas apsiriboja tinkamos eps reikšmės parinkimu. Kadangi eps reikšmė yra proporcinga numatomam aptiktų kaimynų skaičiui, galime pasitelkti artimiausių kaimynų metodą, kad pasiektume teisingą eps įvertinimą. Atliekant artimiausio kaimyno skaičiavimus, taškas nuo kuriuo skaičiuojame atstumą skaičiuojamas kaip pirmasis taškas "Kaimynas". Tad imtis = n+1. Atrinkus ir surūšiavus atstūmus, jie atvaizduojami grafiškai žr. 12 pav. “Atstumo tarp klasterio taškų pokytis“. Aiškiai matomos bent dvi alkūnės maždaug [0,5:1] ir [3,1: 3,6] intervaluose. Tiksliam alkūnės taškui nustatyti taikomas *Kneed Locator* metodas (pristatytas autorės Ville Satopaa ir kt. 2011 m. darbe “Finding a “Kneedle” in a Haystack: Detecting Knee Points in System Behavior”). Skaičiavimai atliekami taikant kelis

skirtingus jautrumo lygius. Jautrumo parametras S leidžia mums koreguoti, kiek agresyvūs esame aptinkant lūžio tašką. Mažesnės S vertės aptinka lūžio taškus greičiau, tuo tarpu didesnės vertės yra konservatyvesnės. Paprasčiau tariant, S yra matas, kiek „plokščių“ taškų tikimės pamatyti nepakeistoje duomenų kreivėje prieš laikydami, jog radome lūžio tašką. Tyrimo metu gavome tris skirtingus lūžio taškus, kuriuose yra optimalios eps reikšmės. Sudarius tris klasterizavimo modelius su gautomis eps reikšmėmis, nustatome, kad geriausias klasterizavimo modelis yra esant eps=3.1623. žr. 6 lentelę “DBSCAN klasterizavimo modelių rezultatai“.

12 pav. Atstumo tarp klasterio taškų pokytis



6 lentelė. DBSCAN klasterizavimo modelių rezultatai

Klasterizavimo metodas	Klasterizavimo modelis	Minimalus klasterio dydis	Maksimalus klasterio dydis	Klasterių skaičius	Silueto koeficientas
DBSCAN	{'eps': 6.650128891193575}	2	189	4	0.216328
DBSCAN	{'eps': 3.1622776601683795}	5	51	25	0.714729
DBSCAN	{'eps': 0.4075315965953057}	5	97	18	0.506965

Lyginant modelius, gautus taikant parametrų gardelę su modeliais ir eps reikšmės paieškos algoritmą, galime teigti, kad gauti rezultatai yra geresni antruoju atveju (prieš silueto koef. = 0.707756, po silueto koeficientas = 0.714729). Kiti stebimi parametrai kaip minimalūs/maksimalūs klasterio dydžiai ir klasterių skaičius nepakito. Tokiu atveju turime tvirtesnius 25 klasterius (t.y. klasterio elementai viduje yra panašesni vieni į kitus). Klasteriai yra netolygaus dydžio, mažiausias klasteris yra vos penkių taškų, tuo tarpu didžiausias 51 taško žr. 7 lentelėje “Sudarytų klasterių charakteristikos“. Atsižvelgiant į šiuos duomenis, prieiname išvados, kad taikant tolimesnį tyrimą skirtingiems klasteriams, gali būti taikomi skirtingi optimizavimo metodai ir skirtingos tyrimo kryptys.

7 lentelė. Sudarytų klasterių charakteristikos

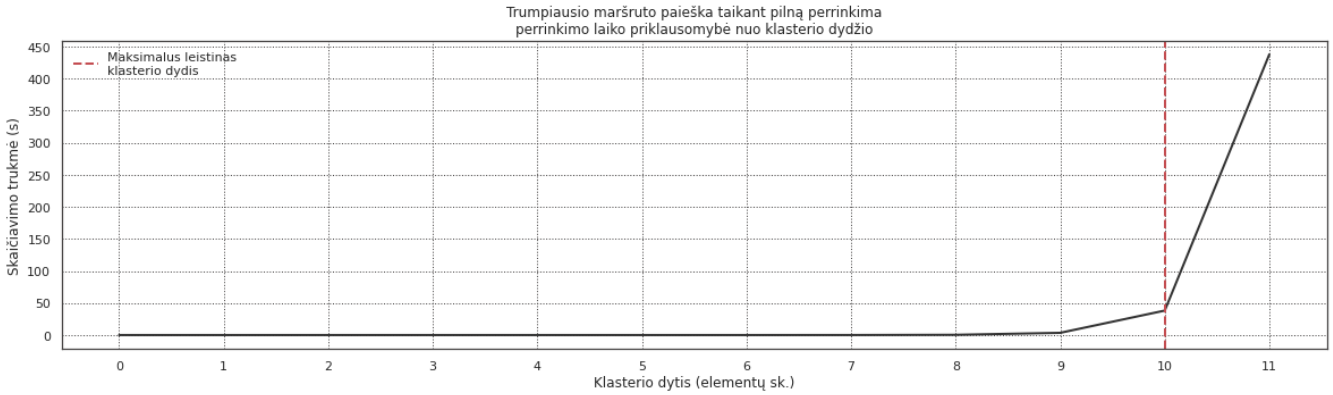
count	mean	std	min	25%	50%	75%	max
25.00	10.60	11.11	5.00	5.00	7.00	8.00	51.00

Tyrimo metu nustatyta, kad duomenys tarpusavyje yra artimi, didinant klasterių skaičių klasterių stiprumo augimas yra proporcingas. Duomenys neišsiskiria charakteringomis formomis, nes rezultatai, taikant labai skirtingus klasterizavimo metodus, buvo labai panašūs. Atsižvelgiant į metrikas, klasterių skaičius rekomenduojamas nemažesnis negu 20 klasterių. Tolimesni pasirinkimai yra labiau susiję su tuo kaip ir kur klasteriai naudojami, t.y. jų prasmė. Todėl, be tos srities ekspertinės nuomonės ir papildomų kriterijų įsivedimo, klasterizavimo tyrimo dalis gali tiek padėti tiek ir pakenkti tolimesniam tyrimui. Tyrimo metu kaip pranašiausias algoritmas nustatytas DBSCAN su Fuzzy branduoliu (ši modifikacija suteikė galimybę priskirti prie klasterių taškų, kurie buvo traktuojami kaip triukšmas). Darbe buvo naudojami tik 2 tankiu grįsti klasterizavimo algoritmai ir DBSCAN buvo vienas iš jų. Atsižvelgiant į šiuos rezultatus, rekomenduojama ateities tyrimo vystymui įtraukti daugiau klasterizavimo metodų paremtų tankiu, tokių kaip OPTICS, ar atlikti HDBSCAN modelio „hyper“ parametrų parinkimą, norint gauti geriausius rezultatus.

3.3 Trumpiausio maršruto paieškos metodų vertinimas

Atlikus duomenų klasterizavimą ir suskaldžius duomenų imtį į klasterius, kitas tyrimo žingsnis - atlikti optimalių sekų paiešką klasterių viduje. Tačiau, atliekant literatūros apžvalgą buvo, nustatyta, kad nėra garantuoto geriausio metodo keliaujančio pirklio uždaviniui spręsti. Visi taikomi metodai euristiniai, meta-euristiniai ir kt. turi savo privalumų ir trūkumų. Jie skirti rasti tam tikrą sprendinį, bet visada išlieka tikimybė, kad egzistuoja geresnis sprendinys. Vienintelis sprendimas garantuojantis optimalaus sprendinio radimą yra pilnas duomenų perrinkimas. Tačiau šio proceso trukmė didėja eksponentiškai, elementų skaičiui sekoje didėjant tiesiškai. Kadangi tyrimo metu taikoma *Python* programavimo kalba, kuri nepasižymi skaičiavimo sparta, šios tyrimo dalies tikslas nustatyti ribą - nuo kada taikyti pilną sekos perrinkimą geriausiam sprendiniui yra nebetikslinga. Tyrimo metu buvo patikrinta skaičiavimo trukmė klasteriams, kurių dydžiai yra intervale [0-15]. Darbo eiga: atliekamas pilnas duomenų perrinkimas fiksuojant skaičiavimo trukmę, testo rezultatai pateikiami 13. pav. „Trumpiausio maršruto paieškos taikant pilną perrinkimą skaičiavimo laiko priklausomybė nuo klasterio dydžio“. Rezultatai pateikiami tik pirmiesiems 11 skaičiavimų. Pasirinkta imtis buvo per didelė, skaičiavimo trukmė kito eksponentiškai ir sekos su >12 elementų nebepavyko realizuoti. Klasterio trumpiausio maršruto paieška, taikant 11 elementų perrinkimą, truko ~ 7.5 min. Skaičiavimai su 12 elementų buvo neįvykdyti ir buvo nutrauktas perrinkimo procesas kai

skaičiavimai truko ilgiau nei 2 val. Remiantis šiuo tyrimu, buvo nuspręsta, kad, pilną perrinkimą yra tikslinga taikyti klasteriams, kurių dydis yra intervale [0:10].

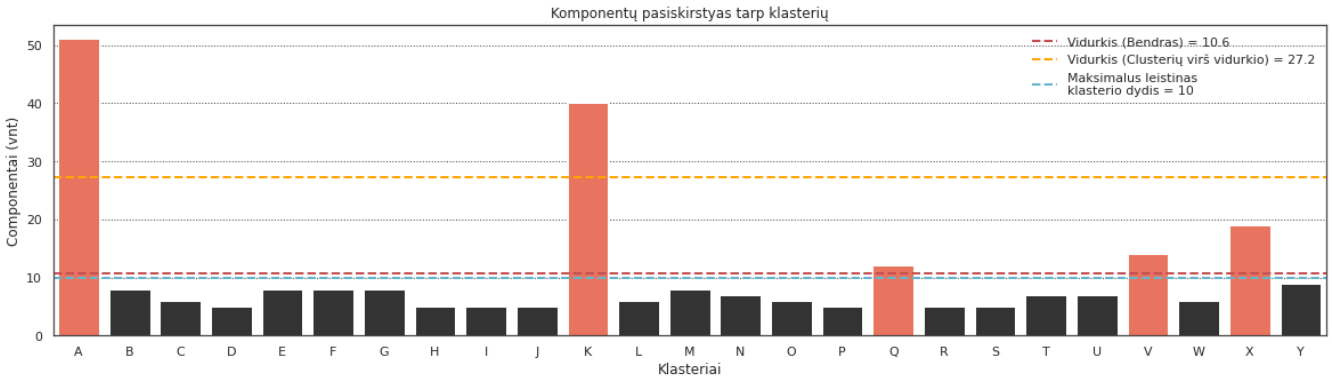


13 pav. Trumpiausio maršruto paieškos taikant pilną perrinkimą skaičiavimo laiko priklausomybė nuo klasterio dydžio

Nagrinėjamas duomenų imtis suskirstėme į klasterius, kurių dydžiai yra nevienodi. Atlikus skaičiavimus, įvertinus laiko sąnaudas ir programinės įrangos galimybes, nustatėme, kad savo imtį racionalu būtų suskaldyti į dvi grupes:

- klasteriai – kuriems taikomas pilnas sekų perrinkimas, norint nustatyti trumpiausią maršrutą (klasterių dydžiai <=10)
- klasteriai – kuriems taikomi optimizavimo metodai, norint nustatyti optimalų maršrutą (klasterių dydžiai >10)

Atsižvelgiant į šį suskirstymą, matome, kad net 20 klasterių taikomas pilnas perrinkimas, o likusiems penkiems klasteriams turės būtį parinkti ir taikomi optimizavimo metodai. Grafike matyti, kad pasirinkta riba, kada taikomas pilnas perrinkimas ir kada optimizavimo algoritmai, yra labai artima bendram klasterių dydžių vidurkiui kuris yra 10,6 žr. 14 pav. “Klasterių pasiskirstymas į grupes priklausomai nuo jų dydžio”. Atsižvelgiant į tai, kad tolimesnėje tyrimo eigoje mes dirbsime tik su 5 klasteriais (A, K, Q, V, X) kuriems negali būti taikomas pilnas perrinkimas, mes nustatome tolimesniems skaičiavimams jų vidurkį (27,2).



14 pav. Klasterių pasiskirstymas į grupes priklausomai nuo jų dydžio

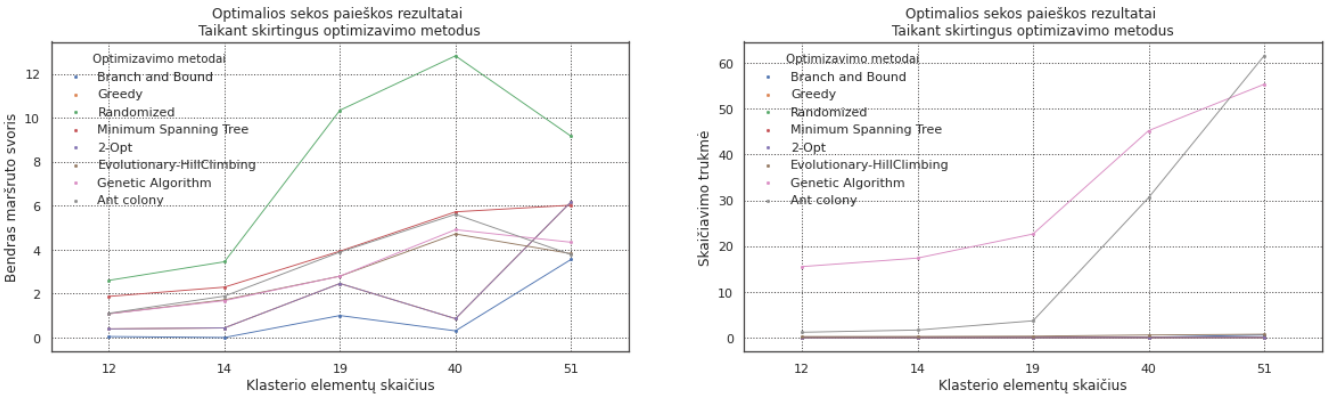
Sekantis tyrimo žingsnis pasirinktiems klasteriams - atrinkti ir pritaikyti geriausią optimizavimo metodą klasteriams, kuriems negalima taikyti pilnojo perrinkimo.

Tyrimo metu buvo pasirinkti 8 skirtingi optimizavimo metodai, taikomi TSP (keliaujančio pirklio) uždaviniui spręsti:

- Šakų ir rėžių algoritmas (*Brabch and Bound*);
- Godusis algoritmas (*Greedy*);
- Atsitiktinės sekos algoritmas (*Randomized*);
- Mažiausiai apibrėžiančio medžio algoritmas (*Minimum spanning tree*);
- 2-opt algoritmas;
- Imituojamo atkaitinimo algoritmas (*Simulated annealing*);
- Genetinis algoritmas (*Genetic*);
- Skruzdžių kolonijos algoritmas (*Ant colony*).

Visi šie metodai turi skirtingų savybių ir yra skirtingai vertinami: vieni pasižymi skaičiavimo greičiu, kiti randa optimalius sprendinius labai didelės apimties sekose. Pagrindiniai kriterijai, kuriais buvo vertinami algoritmai darbo metu, yra: skaičiavimo trukmė, bendras svoris (t.y suminė trukmė reikalinga perreguliuoti įrenginį iš vienos detalės į kitą sudarytoje sekoje). Optimizavimo metodai taikyti tik su sudarytais klasteriais, kuriems negalėjo būti taikomas pilnas perrinkimas dėl jų dydžio.

Atlikus tyrimą nustatyta, kad atsižvelgiant į skaičiavimo trukmę, išsiskyrė du metodai: Genetiniai ir Skruzdžių kolonijos algoritmai žr. 15 pav. “Optimizavimo metodų rezultatai atliekant optimalių sekų paiešką”. Kuo didesnis klasterių dydis, tuo didesnė atskirtis. Optimalios sekos paieškos trukmė tarp skirtingų metodų, klasteryje, kurio imtis 51 elementas, skiriasi net šimtus kartų. Daugumos metodų (išskyrus *Genetic* ir *Ant colony* algoritmus) skaičiavimo trukmė ieškant sprendinio didžiausiame klasteryje (51 taškas) tokia maža, kad daugumoje atveju nesudaro net 1 sekundės žr. 8 lentelėje “Optimizavimo metodų rezultatai atliekant optimalių sekų paiešką (Top 10)” (pilni rezultatai pateikiami 11 Priede „Optimizavimo metodų rezultatai atliekant optimalių sekų paiešką“), kas radikaliai skiriasi nuo pilno sekų perrinkimo, ką atlikome tyrimo pradžioje. Skaičiavimo trukmės standartinis nuokrypis kinta tolygiai su klasterių dydžiu, kuriems yra taikomi skaičiavimai žr. 9 lentelėje “ Optimalių sekų paieškos klasteriuose rezultatų apibendrinimas ”. Išvada, kad kuo didesnis klasteris tuo nuokrypiai tarp taikomų algoritmų spartos auga. Nuokrypių augimas nėra linijinis, yra labiau primenantis eksponentinį kitimą. Kas yra suprantama, nes šio uždavinio sudėtingumas ir susideda iš to, kad sprendinių skaičius auga eksponentiškai, kai auga taškų skaičius maršrute. Skaičiavimo trukmė nėra lemiantis faktorius mūsų uždavinyje (šie skaičiavimai turi būti atliekami tik pasikeitus gaminių asortimentui gamyboje). Įmonės atstovo teigimu, tai vyksta du kartus per metus. Atsižvelgiant į tai, kad atlikus tyrimą, visi metodai tenkina sąlygas dėl skaičiavimo trukmės, todėl lemiamą faktorių dėl optimizavimo metodo tinkamumo lems bendras sudaryto maršruto svoris.



15 pav. Optimizavimo metodų rezultatai atliekant optimalių sekų paiešką

Peržvelgus bendro maršruto svorio kitimo grafiką žr. 15 pav. “Optimizavimo metodų rezultatai atliekant optimalių sekų paiešką” matome, kad klasterių suminis svoris nėra tiesiogiai proporcingas klasterio dydžiui. Nėra nustatyta priklausomybė tarp klasterio dydžio ir maršruto svorio. Atsižvelgus į tai, galima teigti, kad atstumai tarp taškų klasteriuose yra labai skirtingi ir skiriasi net 2-3 kartus. Kai kurie klasteriai, nors ir mažesni pagal sudarančių elementų skaičių, tačiau yra sunkesni t.y. perstatymai tarp įeinančių detalių klasteriuose yra ilgesni. Atsižvelgiant į gautus rezultatus,

klasterius galima surūšiuoti pagal jų sunkumą/svorį – X, V, K, A, Q (atsižvelgiant į gautą vidutinį klasterio svorį ir jį sudarančių elementų skaičių). Iš rezultatų žr. 8 lentelėje " Optimizavimo metodų rezultatai atliekant optimalių sekų paiešką (Top 10)" matome, kad *Simulated annealing*, *Ant colony*, ir *Branch and Bound* metodai gavo labai artimus sprendinius didžiausiame klasteryje ir kai kuriuose kituose, bet, vertinant visus nagrinėjamus klasterius bendrai, geriausi rezultatai yra *Branch and Bound* metodo.

8 lentelė. Optimizavimo metodų rezultatai atliekant optimalių sekų paiešką (Top 10)

Optimizavimo metodai	**Bendras maršruto svoris**	**Skaičiavimo trukmė**	**Klasteris**	**Klasterio elementų skaičius**
Branch and Bound	3.5507	0.5605	A	51
Greedy	6.1697	0.0025	A	51
Randomized	9.1581	0.0004	A	51
Minimum Spanning Tree	6.0194	0.0066	A	51
2-Opt	6.1697	0.0054	A	51
Simulated annealing	3.8173	0.7157	A	51
Genetic Algorithm	4.3343	55.3332	A	51
Ant colony	3.7732	61.5547	A	51
Branch and Bound	0.3037	0.0591	K	40
Greedy	0.8575	0.0014	K	40

Standartinis nuokrypis svyruoja tarp 0,8-4,1 priklausomai nuo klasterio žr. 9 lentelėje "Optimalių sekų paieškos klasteriuose rezultatų apibendrinimas". Didžiausias standartinis nuokrypis matomas klasteryje K. Tai rodo, kad skirtingi optimizavimo metodai skirtingus klasterius veikia netolygiai, sklaida tarp jų svyruoja net 4 kartus tarp skirtingų optimizavimo algoritmų. Klasteryje K minimalus nustatytas svoris siekia 0,3, o tuo tarpu maksimalus svoris tame pačiame klasteryje siekia 12,8. Atsižvelgiant į tai, galima teigti, kad, esant tiksliam klasterio charakterizavimui, galima taikyti atitinkamą optimizavimo metodą kiekvienam klasteriui atskirai, kas suteiktų galimybę gauti optimalų sprendinį. Parinkus vieną bendrą optimizavimo metodą yra tam tikri praradimai.

9 lentelė. Optimalių sekų paieškos klasteriuose rezultatų apibendrinimas

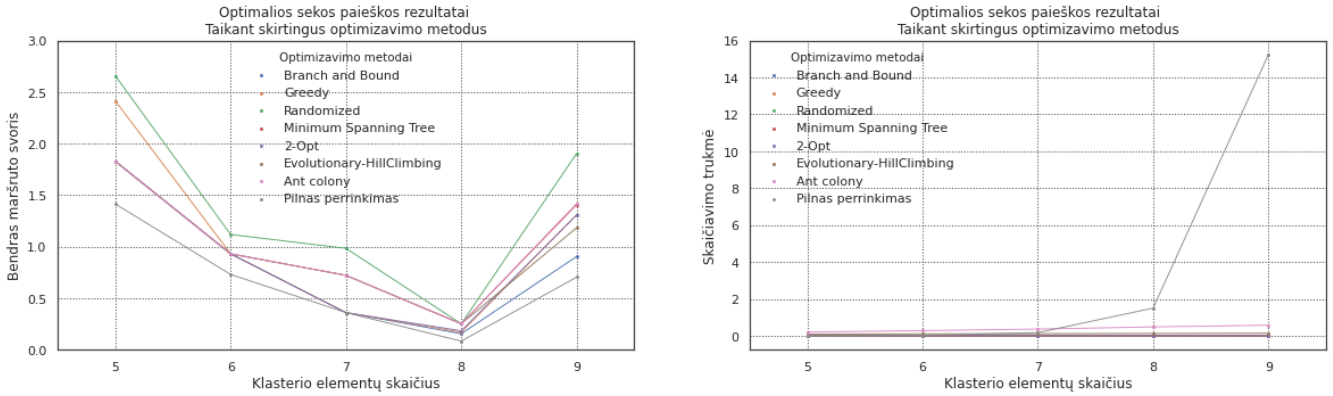
Klasteris	Klasterio dydis	Vidutinis maršruto svoris	Standartinis nuokrypis maršruto svorio	Min. maršruto svoris	Maksimalus maršruto svoris svoris	Vidutinė skaičiavimo trukmė	Standartini nuokrypis skaičiavimo trukmės	Minimali skaičiavimo trukmė	Maksimali skaičiavimo trukmė
A	50	5,3741	1,9046	3,5507	9,1581	14,7724	27,0074	0,0003747	61,5547
K	40	4,4714	4,0734	0,3038	12,8122	9,5485	17,9214	0,0002950	45,2008
Q	12	1,0711	0,8413	0,0445	2,6000	2,1079	5,4266	0,0000820	15,5018
V	14	1,4863	1,1431	0,0000	3,4461	2,4028	6,0703	0,0000671	17,3610
X	19	3,7064	2,8324	1,0006	10,3369	3,3219	7,9021	0,0001210	22,6268

Tyrimo metu nustatyta, kad geriausi rezultatai yra taikant *Branch and Bound* metodą žr. 8 lentelėje “Optimizavimo metodų rezultatai atliekant optimalių sekų paiešką (Top 10)”. Šis metodas nebuvo pranašiausias dėl skaičiavimo trukmės, bet jo sudaryto maršruto svoris visuose klasteriuose buvo geriausias. Nuokrypiai tarp metodų klasteriuose pasiskirsto nevienodai ir nėra susiję su taškų skaičiumi klasteryje. *Branch and Bound* metodo rastos optimalios sekos, lyginant su kitais metodais, buvo geresnės nuo 5% iki 80%. Vidutiniškai - tai 50% geresni sprendiniai (palyginimas atliktas su optimizavimo metodo sprendiniu kiekviename klasteryje, kurio rezultatas buvo antras lyginant maršruto svorį). Šie rezultatai derinasi su kitais tyrimais, kurie buvo nagrinėti literatūrose analizėje t.y dauguma tyrimų ir praktinių darbų teigia, kad Keliaujančio pirklio uždaviniui spręsti, iš šiuo metu esančių algoritmų, *Branch and Bound* metodas yra vienas iš geriausių. Pagrindžiama jo gera skaičiavimo sparta didelėms duomenų imtims ir optimaliu rezultato radimu. Atlikus palyginimą 8 optimizavimo algoritmų, nustatėme, kad esamiems duomenims geriausiai tiko *Branch and Bound* metodas. Darbe nenustatėme, kaip pagerinti maršruto svorio rezultatus taikant šį metodą, tačiau yra kelios galimybės, kaip pagerinti skaičiavimo trukmę taikant šį metodą:

1. visiems optimizavimo metodas tinkantis pasiūlymas yra jų realizavimas programavimo kalba, kurios kompiliavimas yra spartesnis nei Python, pvz. C++, JULIA ar kt.
2. *Branch and Bound* metodas išsiskiria tuo, kad jis yra tik dalinai nuoseklus t.y. jį galima išskaidyti į paralelines sekas. Tokiu atveju, galima taikyti paralelinius skaičiavimus skirtinguose kompiuterio branduoliuose, kas gali kelis kartus sutrumpinti skaičiavimo sąnaudas.

Kaip buvo minėta, skaičiavimo trukmė nėra pagrindinis rezultatus lemiantis faktorius, tad apsiribojama tik rekomendacijomis kaip būtų galima šiuos rezultatus pagerinti.

Tolimesnis tyrimo žingsnis - patikrinti ar yra tikslinga klasterius skaidyti į dvi grupes ir jiems taikyti skirtingus skaičiavimo metodus. Palyginimui atsitiktinai yra parenkami klasteriai, kuriems buvo taikomas pilnas perrinkimas (t.y. klasterių taškų skaičius <=10). Kad matytųsi kitimo tendencija, atrenkami tik skirtingo dydžio klasteriai.



16 pav. Optimizavimo metodų ir pilno perrinkimo rezultatai atliekant optimalių sekų paiešką

Atlikę tyrimą matome, kad skaičiavimo trukmė jau esant 5 elementų klasterio dydžiui, skiriasi 30 kartų žr. 10 lentelėje „Optimizavimo metodų ir pilno perrinkimo rezultatai skirtingiems klasteriams (Top 10)", pilni rezultatai pateikiami 12 Priede" Optimizavimo metodų ir pilno perrinkimo rezultatai skirtingiems klasteriams"). Kai klasterio dydis siekia 9, *Branch and Bound* metodo trukmė prailgėjo 6 kartus, o pilno perrinkimo trukmė prailgėjo 150 000 kartų. Žvelgiant į šiuos rezultatus matome, kodėl esant didesniems duomenų kiekiams yra taikomi optimizavimo algoritmai - jie nėra tokie jautrūs duomenų kiekio pokyčiui. Vertindami laiko sąnaudas galime teigti, kas visi optimizavimo algoritmai turi ženklų pranašumą lyginant su pilnu perrinkimu. Lyginant rezultatus tarpusavyje, matome, kad ilgiausia skaičiavimo trukmė yra *Genetic* algoritmo žr. 16 pav. "Optimizavimo metodų ir pilno perrinkimo rezultatai atliekant optimalių sekų paiešką" (antras pav.). Tokie pat rezultatai yra ir atlikus tyrimą su klasteriais, kurių dydžiai >10. Priešingai nei skaičiavimo trukmė, nustatyto optimalaus maršruto svorio skirtumas, taikant skirtingas skaičiavimo metodikas, nėra didelis. Matome, kad ir esant mažesniems klasteriams yra skirtumai tarp optimizavimo rezultatų. Pastebima, kad augimas ir kritimas yra būdinga visiems, t.y. optimizavimo metodai vienodai reaguoja į nagrinėjamus klasterius (žr. 16 pav. "Optimizavimo metodų ir pilno perrinkimo rezultatai atliekant optimalių sekų paiešką" (pirmas pav.)). Vidutinis maršruto svoris, taikant optimizavimo metodus, svyruoja nuo 0,2 iki 2,0 priklausomai nuo klasterio. Skirtumas tarp vidutinio maršruto svorio, taikant optimizavimo metodus ir pilną perrinkimą, svyruoja tarp 0,09 ir 0,45 (6.3.1.8 „Optimizavimo rezultatai tarp klasterių (klasterio dydis <= 10)") . Taip pat matoma, kad perrinkimo rezultatai yra geriausi (jie negali būti prastesni, nes taikant pilną perrinkimą gaunamas tikslus trumpiausias

maršrutas), bet skirtumas nuo optimizavimo metodų yra nedidelis. Tyrimo metu nustatome, kad *Branch and Bound* metodas yra geriausias optimizavimo metodas. Standartinis nuokrypis klasteriuose svyruoja tarp 0,0001 ir 0,29 lyginant šį metodą su pilnu perrinkimu. žr. 11 lentelė. „Optimalių sekų paieškos klasteriuose lyginant su pilnu perrinkimu rezultatų apibendrinimas“. Šie rezultatai, lyginant su kitais optimizavimo metodais, yra kelis kartus geresni. Kai kuriuose klasteriuose *Branch and Bound* metodo rasti sprendiniai yra beveik lygūs pilno perrinkimo gautiems rezultatams.

10 lentelė. Optimizavimo metodų ir pilno perrinkimo rezultatai skirtingiems klasteriams (Top 10)

Optimizavimo metodai	**Bendras maršruto svoris**	**Skaičiavimo trukmė**	**Klasteris**	**Klasterio elementų skaičius**
Branch and Bound	0.90467	0.00064	Y	9
Pilnas perrinkimas	0.70441	15.24181	Y	9
Branch and Bound	0.15588	0.00039	M	8
Pilnas perrinkimas	0.08458	1.49818	M	8
Branch and Bound	0.36110	0.00032	U	7
Pilnas perrinkimas	0.36110	0.16892	U	7
Branch and Bound	0.93093	0.00018	W	6
Pilnas perrinkimas	0.73103	0.02133	W	6
Branch and Bound	1.82842	0.00010	H	5
Pilnas perrinkimas	1.41421	0.00303	H	5

11 lentelė. Optimalių sekų paieškos klasteriuose lyginant su pilnu perrinkimu rezultatų apibendrinimas

Klasteris	**Klasterio dydis**	**Vidutinis maršruto svoris (Opt. Metodų)**	**Maršruto svoris (Pilnas perrinkimas)**	**Std. Nuokryis (tarp pilno perrinkimo ir Opt. Metodų**	**Std. Nuokryis (tarp pilno perrinkimo ir Opt. Metodų**
H	5	1,3474	0,7044	0,4547	0,1416
Y	9	0,2172	0,0845	0,0938	0,0504
M	8	0,6048	0,3611	0,1723	0,0001
U	7	0,9576	0,7310	0,1602	0,1413
W	6	2,0304	1,4142	0,4357	0,2928

Peržvelgę rezultatus darome išvadą, kad optimizavimo metodai randa pakankamai gerus sprendinius: laiko trukmė yra svariai geresnė lyginant su pilnu perrinkimu, nors rasti rezultatai nėra geriausi sprendiniai. Remiantis gautais rezultatais galime teigti, jog esant galimybei, naudoti pilną perrinkimą trumpiausio maršruto paieškai yra tikslinga.

3.4 Optimalių sekų vertinimas

Tyrimo metu sudarėme 25 klasterius, kuriems parinkome geriausią optimizavimo metodą ir radome optimalius maršrutus. Darbo tikslas - analizuojamoje gamybos įmonėje sutrumpinti suminį perstatymo laiką, kuris šiuo metu siekia apie 10% viso gamybos laiko. Kad įvertinti optimalių sekų naudą, atliekame papildomus palyginimus su istoriniais duomenimis. Šiam tikslui imama gamybos istorinė laiko eilutė su detaliu gamybos eiliškumu (1 metų duomenys). Joje išskiriame pasikartojančias sekas (pasirinktas sekos ilgis =10, kas yra lygu vidutiniam klasterio dydžiui). Nustatėme, kad pasikartojančių sekų metų bėgyje buvo tik 4, o pasikartojimo dažnis 2-3 kartai. Galima teigt, kad įmonė netaiko pasikartojančių sekų ir gamybos maršrutai nėra optimizuojami. Taigi palyginimui tęsti naudosime atsitiktiniu būdu istoriniuose duomenyse atrinktas sekas. Gautų optimizavimo rezultatų palyginimas atliekamas su 3 skirtingomis grupėmis. Grupės tarpusavyje skiriasi sekų ilgiais (visais atvejais generuoja 1000 atsitiktinių sekų):

1. atsitiktinės sekos, kurių ilgis =10 (kas yra lygu vidutiniam klasterio dydžiui);
2. atsitiktinės sekos, kurių ilgis = 27 (kas yra lygu vidutiniam klasterio dydžiui imant tik klasterius, kuriems taikomi optimizavimo metodai);
3. atsitiktinės sekos iš visų turimų detalių (265).

12 lentelė. Optimalių ir atsitiktinių sekų vertinimas

	Visi klasteriai	**Klasteriai kurių ilgis >10 narių**	**Klasteriai kurių ilgis <=10 narių**	**Atsitiktinės sekos kurių ilgis = 10 narių**	**Atsitiktinės sekos kurių ilgis = 27 nariai**	**Atsitiktinės sekos kurių ilgis = 265 nariai**
Vidurkis	0.905943	0.979920	0.887449	39.4141	200.38	1988.82
Standartinis nuokrypis	1.130395	1.491792	1.069457	9.28993	8.2373	19.8353
Suma	22.648573	4.899599	17.748974	-	-	-

Atsižvelgiant į gautus rezultatus žr. 12 lentelė. „Optimalių ir atsitiktinių sekų vertinimas“ galima teigti, kad optimizavimas gamybos maršrutų pasiteisina. Suminis viso maršruto svoris lygus 22 (gautas tyrimo eigoje). Viso maršruto svoris, generuojant atsitiktines sekas, yra 1988. Taikant optimalias sekas, maršruto svoris yra geresnis 98 %. Tačiau toks palyginimas nebūtų teisingas, racionalu būtų lyginti mažesnes sekas (10 ir 27 narių). Atlikus šį palyginimą, rezultatai svyruoja ~ 96 %. Tačiau rezultatai nėra galutiniai, darome prielaidą, kad įmonėje galimos sekos mažesnės nei iš 10 elementų. Atlikus sekų paiešką, kai imties dydis 3 – 5, nustatytos 46 istorinės sekos dažniu tarp 2-18. Matomas tam tikras dėsningumas ir pasikartojimas. Šių sekų svorius galima buvo palyginti tik su kelių klasterių rezultatais (klasteriai: P,R,H,I,D,S,J), kurių ilgis yra lygus rastoms sekoms.

Nagrinėjamu atveju gautas 68% trumpesnis perstatymo laikas. Kaip galima būtų tiksliai nusakyti, kiek taikymas optimalių sekų šioje įmonėje pagerintų rezultatus, nėra tikslaus kriterijaus. Todėl galutiniai rezultatai priimami kaip priklausantys intervalui tarp [68%-96%]. Skyriuje medžiagos ir tyrimo metodai nustatėme, kad perstatymo laikas sudaro ~10% nuo viso bendro laiko nagrinėjamame darbo centre. Remiantis tuo galima teigti, kad atliekant duomenų klasterizavimą ir taikant optimizavimo metodus, galima aptikti optimalias gaminių sekas ir bendrą gamybos trukmę sutrumpinti 6,8-9,6%.

Išvados

Darbe apžvelgti organizaciniai bei matematiniai gamybos pajėgumų ir srautų didinimo metodai. Analizuotas *Lean* ir *Agile* metodų vaidmuo gamybos srauto optimizavime. Analizuotas matematinių metodų taikymas optimizuojant gamybos srautus.

1. Atlikus analizę, galima daryti išvadą, kad organizaciniai ir matematiniai metodai papildo vienas kitą didinant gamybos pajėgumus.

Tyrimo metu atlika įmonės gamybos pajėgumus veikiančių veiksnių analizė. Įrenginio reguliavimas tarp skirtingų partijų išskirtas kaip kaip kritinis veiksnys ir galimybė taikyti matematinius metodus. Pagrįstas metodas sudaryti optimalias partijų sekas ribojančiuose gamybos srautą darbo centruose.

2. Pagrįstas metodas gali būti taikomas gamybos įmonėse vykdančiose masinę standartizuotų prekių gamybą ir pasižyminčiose gaminamų produktų įvairove ir reikšmingais įrenginių perderinimo laikais.
3. Pagrįstas metodas gali būti taikoma įvairiose prąmonės šakose, nepriklausomai nuo produktų rūšies.
4. Pagrįstas metodas leidžia koncentruotis į siaurą probleminę sritį, kas sumažina laiko sąnaudas analizei, modeliavimui ir diegimui. Pritaikymo sritis universali tiek nuosiakliems tiek paraleliniams srautams.

Darbe igyvendintas metodas kurio pagrindiai žingsniai gamybos asortimento klasterizavimas į smulkesnes grupes ir optimalių sekų paieška sudarytose klasteriuose. Klasterizuotas gamybos asortientas taikant, analizuojant ir tarpusavyje palyginant 8 klasterizavimo metodus. Nustatytos optimalios partijų sekos klasteriuose, atrenkant optimizavimo metodą iš aštuonių nagrinėtų. Sudarytų optimalių sekų suminis perstatymo laikas yra ~90% trumpesnis lyginant su vykdomą praktiką gamyboje. Tokiu būdu sutrumpinamas pridetinės vertės nekuriantis darbo laikas.

5. Pagrįstas metodas sudaro teorine galimybę padidinti bendrą gamybos pajėgumą net 9%. Tai sumažintu tiesioginius darbo kaštus ir netiesioginias sąnaudas tokias kaip elektra, įrenginių nusidevėjimas tenkančius vienam gaminiui. Pasekoje sumažėtu gaminio savikaina 2% kas suteikia konkurencinį pranašumą arba išlaikant tas pačias gaminio pardavimo kainas pelnas padidinimas 2%.
6. Pagrįstas metodas gali būti taikomas įvairiems įrenginiams didinant atskirų įrenginių pajėgumą, kas suteikia galimybęs sumažinti įrenginių darbo laiką.

Literatūros sąrašas

1. AMAYA, J.E. et al. A memetic algorithm for the tool switching problem. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* [interaktyvus]. [s.l.]: Springer, Berlin, Heidelberg, 2008. p. 190–202. [žiūrėta 2018-03-04]. Prieiga per internetą: <http://link.springer.com/10.1007/978-3-540-88439-2_14>.

2. BARD, J.F. A heuristic for minimizing the number of tool switches on a flexible machine. In *IIE Transactions* [interaktyvus]. 1988. Vol. 20, no. 4, p. 382–391. [žiūrėta 2018-03-05]. . Prieiga per internetą: <http://www.tandfonline.com/doi/abs/10.1080/07408178808966195>.

3. BRADLEY, P.S. et al. Constrained k-means clustering. In *Microsoft Research* [interaktyvus]. 2000. p. 1–8. [žiūrėta 2018-03-14]. . Prieiga per internetą: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.3257>.

4. CAMPELLO, R.J.G.B. et al. A framework for semi-supervised and unsupervised optimal extraction of clusters from hierarchies. In *Data Mining and Knowledge Discovery* [interaktyvus]. [s.l.]: Springer US, 2013. p. 344–371. [žiūrėta 2018-03-16]. Prieiga per internetą: <http://link.springer.com/10.1007/s10618-013-0311-4>.

5. CATANZARO, D. et al. Improved integer linear programming formulations for the job Sequencing and tool Switching Problem. In *European Journal of Operational Research* [interaktyvus]. 2015. Vol. 244, no. 3, p. 766–777. [žiūrėta 2018-03-05]. . Prieiga per internetą: <https://www.sciencedirect.com/science/article/pii/S0377221715001198>.

6. CHAKRABORTY, S. - NAGWANI, N.K. Analysis and Study of Incremental DBSCAN Clustering Algorithm. In *International Journal of Enterprise Computing and Business Systems* [interaktyvus]. 2011. Vol. 1, no. 2. [žiūrėta 2018-03-15]. . Prieiga per internetą: <http://arxiv.org/abs/1406.4754>.

7. CHAUDHURI, K. - DASGUPTA, S. Rates of convergence for the cluster tree. In *Advances in Neural Information Processing Systems 23* [interaktyvus]. 2010. p. 343–351. [žiūrėta 2018-03-16]. ISBN 9781617823800Prieiga per internetą: <http://cseweb.ucsd.edu/~dasgupta/papers/tree.pdf>.

8. DAVIDSON, I. - RAVI, S.S. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* [interaktyvus]. [s.l.]: Springer, Berlin, Heidelberg, 2005. p. 59–70. [žiūrėta 2018-03-15]. Prieiga per internetą: <http://link.springer.com/10.1007/11564126_11>.

9. DENIZEL, M. Minimization of the Number of Tool Magazine Setups on Automated Machines: A Lagrangean Decomposition Approach. In *Operations Research* [interaktyvus]. 2003.

Vol. 51, no. 2, p. 309–320. [žiūrėta 2018-03-05]. . Prieiga per internetą: <https://pubsonline.informs.org/doi/abs/10.1287/opre.51.2.309.12784>.

10. DORIGO, M. - CARO, G. DI Ant Colony Optimization: A new meta-heuristic. In *Congress on Evolutionary Computation* [interaktyvus]. 1999. p. 1470–1477. [žiūrėta 2018-03-09]. Prieiga per internetą: <http://ieeexplore.ieee.org/abstract/document/782657/>.

11. EBERHART, R. - KENNEDY, J. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science* [interaktyvus]. p. 39–43. [žiūrėta 2018-03-09]. Prieiga per internetą: <http://ieeexplore.ieee.org/abstract/document/494215/>.

12. ELDRIDGE, J. et al. Beyond Hartigan Consistency: Merge Distortion Metric for Hierarchical Clustering. In [interaktyvus]. 2015. [žiūrėta 2018-03-17]. . Prieiga per internetą: <https://arxiv.org/pdf/1506.06422v2.pdf>.

13. FAN, H. Discrete Particle Swarm Optimization for TSP based on Neighborhood. In *Journal Of Computational Information Systems* [interaktyvus]. 2010. Vol. 10, p. 3407–3414. [žiūrėta 2018-03-09]. . Prieiga per internetą: <https://pdfs.semanticscholar.org/63a4/1c7e9db00baacb249668ba112b7cf1830e87.pdf>.

14. FLEMING, D. *Lean Logic: A Dictionary for the Future and How to Survive It* [interaktyvus]. . 2016. 658 p. ISBN 978-1-60358-648-1.

15. FURRER, M. - MÜTZE, T. An algorithmic framework for tool switching problems with multiple objectives. In *European Journal of Operational Research* [interaktyvus]. 2017. Vol. 259, no. 3, p. 1003–1016. [žiūrėta 2018-03-05]. . Prieiga per internetą: <https://www.sciencedirect.com/science/article/pii/S0377221716309596>.

16. GHIANI, G. et al. An exact solution to the TLP problem in an NC machine. In *Robotics and Computer-Integrated Manufacturing* [interaktyvus]. 2007. Vol. 23, no. 6, p. 645–649. [žiūrėta 2018-03-04]. . Prieiga per internetą: <https://www.sciencedirect.com/science/article/pii/S0736584507000282>.

17. GREENBERG, H.J. Greedy Algorithms for Minimum Spanning Tree Proof of Optimality. In *Proceedings of the American Mathematical Society* [interaktyvus]. 1998. p. 2–4. [žiūrėta 2018-03-17]. . Prieiga per internetą: <http:::www.cudenver.eduu>.

18. GUTIN, G. et al. Traveling salesman should not be greedy: Domination analysis of greedy-type heuristics for the TSP. In *Discrete Applied Mathematics* [interaktyvus]. 2002. Vol. 117, no. 1–3, p. 81–86. [žiūrėta 2018-03-06]. . Prieiga per internetą: <https://www.sciencedirect.com/science/article/pii/S0166218X01001950>.

19. YVES CRAMA, ANTOON W. J. KOLEN, ALWIN G. OERLEMANS, F.C.R.S. Minimizing the Number of Tool Switches on a Flexible Machine. In [interaktyvus]. 1994. Vol. 6, p. 1987–1988. [žiūrėta 2018-03-04]. . Prieiga per internetą: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.867.7911&rep=rep1&type=pdf>.

20. KARAKAYALI, I. - AZIZOĞLU, M. Minimizing total flow time on a single flexible machine. In *International Journal of Flexible Manufacturing Systems* [interaktyvus]. 2006. Vol. 18, no. 1, p. 55–73. [žiūrėta 2018-03-04]. . Prieiga per internetą: <http://link.springer.com/10.1007/s10696-006-9000-6>.

21. KONAK, A. et al. Minimizing the number of tool switching instants in Flexible Manufacturing Systems. In *International Journal of Production Economics* [interaktyvus]. 2008. Vol. 116, no. 2, p. 298–307. [žiūrėta 2018-03-05]. . Prieiga per internetą: <https://www.sciencedirect.com/science/article/pii/S0925527308003034>.

22. LAPORTE, G. et al. Exact algorithms for the job sequencing and tool switching problem. In *IIE Transactions (Institute of Industrial Engineers)* [interaktyvus]. 2004. Vol. 36, no. 1, p. 37–45. [žiūrėta 2018-03-05]. . Prieiga per internetą: <http://www.tandfonline.com/doi/abs/10.1080/07408170490257871>.

23. MÜTZE, T. Scheduling with few changes. In *European Journal of Operational Research* [interaktyvus]. 2014. Vol. 236, no. 1, p. 37–50. [žiūrėta 2018-03-04]. . Prieiga per internetą: <http://page.math.tu-berlin.de/~muetze/papers/scheduling.pdf>.

24. PEI, J. et al. *Advances in Knowledge Discovery and Data Mining* [interaktyvus]. . 2014. ISBN 978-3-319-06607-3.

25. PRIVAULT, C. - FINKE, G. K-Server problems with bulk requests: An application to tool switching in manufacturing. In *Annals of Operations Research* [interaktyvus]. 2000. Vol. 96, no. 1/4, p. 255–269. [žiūrėta 2018-03-04]. . Prieiga per internetą: <http://link.springer.com/10.1023/A:1018939132489>.

26. TANG, C. - DENARDO, E. Models Arising fro a Flexible Manufacturing Machine, Part I: Minimization of the Number of Tool Switches. In *Operations Research* [interaktyvus]. 1988. Vol. 35, no. 5, p. 767–777. [žiūrėta 2018-03-05]. . Prieiga per internetą: <https://pubsonline.informs.org/doi/abs/10.1287/opre.36.5.778>.

27. WANG, K. et al. Adaptive Affinity Propagation Clustering. In *Automatica* [interaktyvus]. 2007. Vol. 33, no. 12, p. 1242–1246. [žiūrėta 2018-03-15]. . Prieiga per internetą: <http://arxiv.org/abs/0805.1096>.

28. ZHANG, T. et al. BIRCH: an efficient data clustering method for very large databases.

In *Proc. ACM-SIGMOD Int. Conf. Management of Data* [interaktyvus]. 1996. p. 103–114. [žiūrėta 2018-03-16]. Prieiga per internetą: <http://delivery.acm.org/10.1145/240000/233324/p103-zhang.pdf?ip=193.219.170.65&id=233324&acc=ACTIVE SERVICE&key=1FA3353941FE8055.4A4E353D8E57EBBB.4D4702B0C3E38B35.4D4702B0C3E38B35&__acm__=1521191017_aff6d364840f053eb21a719d05bdb293>.

29. *Graph Theory and Network Flows*. . 49-78 p. ISBN 2850476641.

30. Darrall Henderson et al. The Theory and Practice of Simulated Annealing. In *Handbook of Metaheuristics* [interaktyvus]. 2003. Vol. 57, pp 287-319. [žiūrėta 2018-03-06]. . Prieiga per internetą: < https://link.springer.com/chapter/10.1007/0-306-48056-5_10>.

32. Aravind Srinivasan Approximation algorithms via randomized rounding: asurvey. In *Discrete Applied Mathematics* [interaktyvus]. [žiūrėta 2018-03-06]. . Prieiga per internetą: <http://homepage.divms.uiowa.edu/~sriram/196/fall08/rr-final.pdf>.

Priedai

1 priedas. Pagamintos produkcijos pardavimai FY 2019 m.

Id	19M09	19M10	19M11	19M12	20M01	20M02	20M03	20M04	20M05	20M06	20M07	20M08
407274			-	-	-	-						
40456903			-	-	-	-						
40407291			-	-	-	-						
50402641	-	-	-	-	-	-	5 556	30 644	-	-	8 172	-
30407296	-	-	-	-	-	-	-	-	-	-	-	-
50407295	-	-	-	-	-	-	1 379	-	-	-	-	-
40456922	-	-	-	-	-	-	1 111	-	-	9 792	-	3 302
50456926	-	-	-	-	-	-	-	-	-	-	-	-
20456923	-	-	-	-	-	-	977	-	-	-	-	-
407269	-	-	-	-	-	-	1 667	-	3 222	-	9 822	4 906
407306	-	-	-	-	-	-	-	-	-	-	-	-
80407307	-	-	-	-	-	-	706	-	-	-	-	-
415052	-	-	-	-	-	-	1 667	-	11 111	16 405	-	5 598
60415073	-	-	-	-	-	-	-	-	-	-	-	-
20415046	-	-	-	-	-	-	1 356	-	-	-	-	-
50459779	-	-	-	-	-	-	1 667	-	4 444	10 213	-	10 813
50415059	-	-	-	-	-	-	1 756	-	-	-	-	-
30415055	-	-	-	-	-	-	1 111	-	10 778	-	11 582	-
70459778	-	-	-	-	-	-	11 111	-	4 667	-	-	-
10415080	-	-	-	-	-	-	1 518	-	-	-	-	-
60407266	-	-	-	-	-	-	-	1 667	-	5 556	-	2 030
407293	-	-	-	-	-	-	1 449	-	-	-	-	-
80407294	-	-	-	-	-	-	-	1 503	-	-	-	-
415047	-	-	-	-	-	-	-	1 667	-	10 000	-	6 642
60415068	-	-	-	-	-	-	-	-	-	-	-	-
60456916	-	-	-	-	-	-	-	1 404	-	-	-	-
20456918	-	-	-	-	-	-	1 667	-	10 000	-	6 222	9 770
456919	-	-	-	-	-	-	1 899	-	-	-	-	-
20377681	-	-	-	-	-	-	-	1 667	-	-	15 556	13 997
80313554	-	-	-	-	-	-	-	-	-	-	-	-
30311307	-	-	-	-	-	-	-	2 415	-	-	-	-
20357895	-	3 111	-	3 333	6 111	2 222	2 222	333	-	-	-	-
10385950	3 327	-	-	2 222	5 000	4 444	-	1 187	533	80	-	-
330519	5 483	4 254	2 778	4 444	8 889	2 222	4 444	4 247	2 924	3 427	3 713	3 617
50322847	1 456	1 056	-	2 222	2 222	2 222	-	1 618	1 344	1 578	1 511	1 511
70373261	17 933	19 000	11 667	22 778	28 889	6 667	20 000	16 151	14 044	17 318	22 693	18 567
90387969	22	1 111	-	-	1 111	-	-	133	-	-	11	11
10311308	1 780	13	-	-	-	1 111	-	601	560	840	1 113	1 127
70387970	-	1 667	-	1 111	1 111	-	2 222	1 036	967	996	984	987
30302299	-	960	1 111	2 222	-	-	1 111	695	260	475	412	473

30379670	-	3	-	-	2 222	1 111	2 222	803	902	1 361	1 330	1 050
50304990	-	-	-	-	-	1 111	2 222	1 817	2 341	2 383	2 167	1 750
70379668	67	7 733	2 222	2 222	3 333	2 778	6 667	3 700	2 911	3 449	3 662	3 478
70302301	-	-	-	-	-	-	-	200	167	240	338	533
80354182	-	1 544	1 667	-	1 111	-	2 222	1 349	978	1 124	1 053	967
10379671	-	-	-	-	-	-	-	62	56	84	171	189
10304992	5 411	5 567	3 889	20 000	3 896	29 716	-	5 551	4 756	6 318	10 215	9 233
20354180	-	-	5 000	-	8 720	3 333	9 444	307	278	302	487	456
50379669	-	1 300	1 111	-	1 953	1 111	2 222	-	-	-	-	-
70367444	1 656	11	2 222	5 264	4 680	6 667	6 667	1 731	1 567	1 892	3 107	2 751
10367442	-	-	-	3 543	-	556	2 222	40	22	53	69	67
372552	-	-	-	-	-	556	-	-	-	-	-	-
90372519	-	1 791	1 667	-	-	2 222	-	459	444	1 707	2 222	1 760
50372521	1 831	462	556	-	-	1 111	-	345	338	405	412	391
10372523	-	-	-	-	-	1 111	-	397	396	498	475	282
60372525	338	1 049	-	-	556	-	-	231	231	302	338	249
80372534	1 067	2 400	-	-	-	-	-	270	276	302	347	329
20372546	-	98	-	556	1 111	-	-	311	311	332	370	276
60375133	1 093	1 227	-	-	-	-	556	313	249	286	345	320
10375135	1 069	433	-	-	444	-	1 111	248	207	288	310	264
90366764	780	1 676	-	-	667	-	-	253	158	247	293	328
10366838	-	-	-	-	556	-	-	84	53	181	223	147
60366789	-	276	-	-	556	-	-	150	129	154	171	160
30366776	191	204	-	-	-	-	-	85	89	102	84	36
10366796	160	-	-	-	-	-	-	68	76	87	95	84
30366795	311	622	1 111	-	-	-	-	233	173	214	212	151
70366784	-	4	-	556	-	-	-	117	102	132	140	124
366787	-	-	-	-	-	-	-	92	80	260	372	236
30367441	253	-	-	-	-	556	-	148	124	172	183	142
30372503	-	-	-	-	-	-	-	93	86	70	90	44
60372506	-	4	-	-	-	1 111	-	114	76	112	137	129
50372516	5 031	-	-	5 000	-	5 556	-	3 367	2 916	3 339	4 004	3 342
30372517	-	44	-	-	1 111	-	1 111	651	631	764	1 049	871
30372536	1 236	-	-	1 111	556	1 111	-	567	524	528	690	533
10372542	1 636	4 436	-	-	2 222	1 111	2 222	1 628	1 333	1 582	1 831	1 671
90347351	640	1 111	-	-	1 111	1 111	-	553	524	544	576	471
90366839	2 918	87	2 222	-	-	2 778	-	1 725	1 416	1 778	1 985	1 602
50366761	2 398	2 196	1 111	-	-	4 444	1 111	1 404	1 271	1 138	1 404	1 127
10366777	5 516	9 347	7 222	11 667	13 333	15 555	15 555	11 577	10 058	11 699	13 706	12 253
20366791	2 182	1 991	-	2 222	1 333	2 222	1 111	1 562	1 338	1 501	2 045	1 453
70366798	-	111	-	-	-	-	-	136	98	137	156	142
90366797	2 609	2 396	-	2 222	1 333	3 333	1 111	2 017	1 711	1 900	2 375	1 853
90366783	476	1 231	-	-	-	-	-	247	227	289	311	249
40366785	-	2 596	-	-	2 222	-	1 667	1 014	884	1 138	1 387	1 089

2 priedas. Darbo dienų skaičius darbo centruose FY 2019m.

Id	19M9	19M10	19M11	19M12	20M1	20M2	20M3	20M4	20M5	20M6	20M7	20M8
00001	30	31	13	19	22	20	21	-	-	-	-	-
00002	-	-	13	19	22	20	21	21	20	21	22	21
00003	30	31	13	19	22	20	21	21	20	21	22	21
00004	-	-	13	19	22	20	21	21	20	21	22	21
11000	21	21	13	19	22	20	21	21	20	21	22	21
11010	30	31	13	26	30	28	30	21	20	21	22	21
11030	30	31	13	26	30	28	30	21	20	21	22	21
11070	30	31	13	26	30	28	30	21	20	21	22	21
11090	-	-	13	19	22	20	21	21	20	21	22	21
11100	30	31	13	26	30	28	30	21	20	21	22	21
11110	20	31	13	26	30	28	30	21	20	21	22	21
11120	30	31	13	26	30	28	30	23	21	21	22	21
11130	30	31	13	26	30	28	30	21	30	21	22	21
11150	30	31	13	26	30	28	30	21	20	21	22	21
12010	30	31	13	26	30	28	30	21	20	21	22	21
12020	30	31	13	26	30	28	30	21	20	21	22	21
12030	30	31	13	26	30	28	30	21	20	21	22	21
12050	30	31	13	26	30	28	30	21	20	21	22	21
12060	30	31	13	26	30	28	30	21	21	22	23	21
12070	30	31	13	19	22	20	21	21	20	21	22	21
12080	-	-	13	19	22	20	21	21	20	21	22	21
12090	-	-	13	26	30	28	30	21	20	21	22	21
13010	30	31	13	26	30	28	30	21	20	21	22	21
13050	30	31	13	26	30	28	30	21	20	21	22	21
13070	30	31	13	26	30	28	30	21	20	21	22	21
13080	30	31	13	26	30	28	30	21	20	21	22	21
13090	30	31	13	26	30	28	30	21	20	21	22	21
13091	30	31	13	26	30	28	30	21	20	21	22	21
13092	-	-	13	26	30	28	30	21	20	21	22	21
14000	30	31	13	26	30	28	30	21	20	21	22	21
14010	-	-	13	19	22	20	21	21	20	21	22	21
14020	-	-	13	19	22	20	21	21	20	21	22	21
15000	30	31	13	26	30	28	30	21	20	21	22	21
15050	30	31	13	26	30	28	30	21	20	21	22	21
15060	30	31	13	26	30	28	30	21	20	21	22	21
16020	30	31	13	26	30	28	30	21	20	21	22	21
16030	30	31	13	26	30	28	30	21	20	21	22	21
16050	30	31	13	26	30	28	30	21	20	21	22	21
17010	30	31	13	26	30	28	30	21	20	21	22	21
17020	30	31	13	26	30	28	30	21	20	21	22	21
17030	-	-	13	19	22	20	21	21	20	21	22	21
18010	30	31	13	26	30	28	30	21	20	21	22	21

18020	30	31	13	26	30	28	30	21	20	21	22	21
18030	30	31	13	26	30	28	30	21	20	21	22	21
18040	30	31	13	26	30	28	30	21	20	21	22	21
18050	30	31	13	26	30	28	30	21	20	21	22	21

3 priedas. Darbo grafikai (pamainų apskaitos tipai) darbo centre 2019 m.

Id	**19M9**	**19M10**	**19M11**	**19M12**	**20M1**	**20M2**	**20M3**	**20M4**	**20M5**	**20M6**	**20M7**	**20M8**
00001	480	480	480	480	480	480	480	-	-	-	-	-
00002	-	-	480	480	480	480	480	480	480	480	480	480
00003	480	480	480	480	480	480	480	480	480	480	480	480
00004	-	-	480	480	480	480	480	480	480	480	480	480
11000	-	-	480	480	480	480	480	480	480	480	480	480
11010	360	360	360	360	360	360	360	480	480	480	480	480
11030	360	360	360	360	360	360	360	480	480	480	480	480
11070	360	360	360	360	360	360	360	480	480	480	480	480
11090	-	-	-	-	-	480	480	480	480	480	480	480
11100	360	360	360	360	360	360	360	480	480	480	480	480
11110	360	360	360	360	360	360	360	480	480	480	480	480
11120	360	360	360	360	360	360	360	480	480	480	480	480
11130	360	360	360	360	360	360	360	480	480	480	480	480
11150	360	360	360	360	360	360	360	480	480	480	480	480
12010	360	360	360	360	360	360	360	480	480	480	480	480
12020	360	360	360	360	360	360	360	480	480	480	480	480
12030	360	360	360	360	360	360	360	480	480	480	480	480
12050	360	360	360	360	360	360	360	480	480	480	480	480
12060	360	360	360	360	360	360	360	480	480	480	480	480
12070	480	480	480	480	480	480	480	480	480	480	480	480
12080	-	-	-	-	-	-	-	-	-	480	480	480
12090	-	-	-	-	-	-	-	-	-	480	480	480
13010	360	360	360	360	360	360	360	480	480	480	480	480
13050	360	360	360	360	360	360	360	480	480	480	480	480
13070	360	360	360	360	360	360	360	480	480	480	480	480
13080	360	360	360	360	360	360	360	480	480	480	480	480
13090	360	360	360	360	360	360	360	480	480	480	480	480
13091	360	360	360	360	360	360	360	480	480	480	480	480
13092	-	-	-	-	-	-	-	-	-	-	480	480
14000	360	360	360	360	360	360	360	480	480	480	480	480
14010	-	-	-	-	-	480	480	480	480	480	480	480
14020	-	-	-	-	-	-	-	-	480	480	480	480
15000	360	360	360	360	360	360	360	480	480	480	480	480
15050	360	360	360	360	360	360	360	480	480	480	480	480
15060	360	360	360	360	360	360	360	480	480	480	480	480
16020	360	360	360	360	360	360	360	480	480	480	480	480
16030	360	360	360	360	360	360	360	480	480	480	480	480

16050	360	360	360	360	360	360	360	480	480	480	480	480
17010	360	360	360	360	360	360	360	480	480	480	480	480
17020	360	360	360	360	360	360	360	480	480	480	480	480
17030	-	-	-	-	480	480	480	480	480	480	480	480
18010	360	360	360	360	360	360	360	480	480	480	480	480
18020	-	-	360	360	360	360	360	480	480	480	480	480
18030	360	360	360	360	360	360	360	480	480	480	480	480
18040	360	360	360	360	360	360	360	480	480	480	480	480
18050	360	360	360	360	360	360	360	480	480	480	480	480

4 priedas. Vidutinis darbo centrų efektyvumas FY 2019m.

Id	**19M9**	**19M10**	**19M11**	**19M12**	**20M1**	**20M2**	**20M3**	**20M4**	**20M5**	**20M6**	**20M7**	**20M8**
00001	30%	35%	40%	40%	40%	40%	40%					
00002					10%	15%	25%	30%	35%	45%	45%	50%
00003					10%	15%	25%	30%	35%	35%	35%	35%
00004					10%	15%	30%	30%	35%	35%	35%	35%
11000	30%	35%	20%	30%	35%	40%	45%	50%	50%	55%	60%	60%
11010	65%	65%	65%	65%	60%	65%	65%	65%	65%	65%	65%	65%
11030	55%	60%	55%	55%	55%	55%	55%	55%	55%	55%	55%	55%
11070	55%	55%	55%	55%	55%	55%	55%	55%	55%	60%	60%	60%
11090	0%	0%	1%	5%	10%	20%	25%	30%	35%	40%	40%	40%
11100	50%	50%	50%	50%	50%	55%	55%	55%	55%	55%	55%	55%
11110	30%	30%	0%	20%	30%	35%	40%	50%	50%	50%	50%	50%
11120	63%	64%	65%	65%	50%	55%	55%	60%	65%	65%	65%	65%
11130	40%	50%	50%	50%	45%	45%	50%	55%	55%	55%	55%	55%
11150	41%	41%	41%	41%	41%	41%	41%	41%	41%	41%	41%	41%
12010	55%	60%	60%	60%	60%	60%	60%	65%	65%	65%	65%	65%
12020	55%	60%	60%	60%	60%	60%	60%	65%	65%	65%	65%	65%
12030			30%	30%	30%	30%	30%	30%	30%	30%	30%	30%
12050	55%	55%	55%	55%	55%	55%	55%	55%	55%	55%	55%	55%
12060	50%	50%	45%	50%	55%	55%	60%	55%	60%	60%	60%	60%
12070	40%	45%	45%	50%	50%	40%	45%	55%	55%	55%	60%	60%
12080					0%	0%	0%	0%	0%	10%	20%	30%
12090	0%	0%	0%	0%	0%	0%	0%	0%	0%	10%	30%	35%
13010	42%	42%	42%	42%	42%	42%	42%	42%	42%	42%	42%	42%
13050	40%	40%	40%	40%	40%	40%	40%	40%	40%	40%	40%	40%
13070	30%	30%	50%	50%	50%	50%	30%	30%	30%	30%	30%	30%
13080	40%	40%	45%	45%	50%	50%	50%	40%	40%	40%	40%	40%
13090	40%	40%	45%	45%	50%	50%	50%	40%	40%	40%	40%	40%
13091	45%	45%	40%	40%	45%	45%	45%	45%	45%	45%	45%	45%
13092	0%	0%	0%	0%	0%	0%	0%	0%	0%	10%	20%	30%
14000	73%	73%	73%	73%	73%	73%	73%	73%	73%	73%	73%	73%
14010	0%	0%			5%	7%	10%	13%	15%	20%	23%	23%
14020					0%	0%	5%	10%	15%	20%	25%	30%

15000	45%	50%	100%	100%	100%	100%	100%	60%	60%	60%	60%	60%
15050	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%
15060			30%	30%	30%	30%	30%	30%	30%	30%	30%	30%
16020			30%	30%	30%	30%	30%	30%	30%	30%	30%	30%
16030	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%	50%
16050	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%	70%
17010	55%	55%	55%	55%	50%	55%	55%	55%	55%	55%	55%	55%
17020	55%	55%	55%	55%	50%	55%	55%	55%	55%	55%	55%	55%
17030					30%	40%	45%	50%	55%	55%	55%	55%
18010	45%	50%	46%	46%	48%	50%	51%	50%	51%	52%	54%	54%
18020	45%	50%	46%	46%	48%	50%	51%	50%	51%	52%	54%	54%
18030	45%	50%	46%	46%	48%	50%	51%	50%	51%	52%	54%	54%
18040	52%	52%	52%	52%	52%	52%	52%	52%	52%	52%	52%	52%
18050	52%	52%	52%	52%	52%	52%	52%	52%	52%	52%	52%	52%

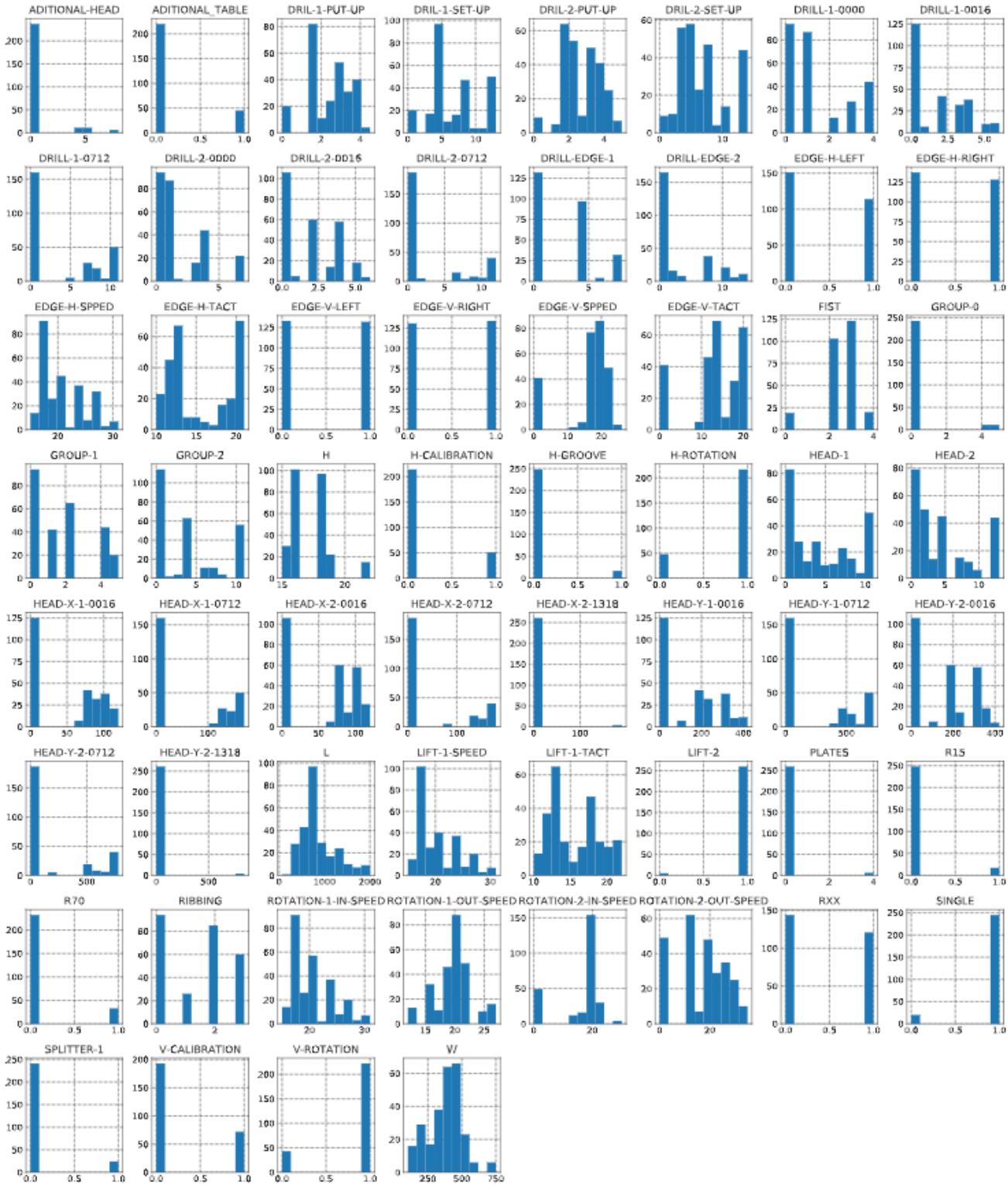
5 priedas. Darbo centro HOMAG4X įrangos pozicionavimo schema

```
_INHALTSVERZEICHNIS : ><           KUNDE     : >XXXXXXXXXXXXXXXXXX<
_MASCHINENNUMMER    : >0-251-11-0950/51<     SACHBEARB.: >BSTWIN32<
_PRODUKTSCHLUESSEL  : >BST 500<           DATUM     : >03-02-2017<
-------------------------------------------------------------------
 *BOHRKOPF !SPINDEL!WK!DELTA-X   !DELTA-Y   !DURCHMESSER! KOMMENTAR
----------+----------+----------+----------+-----------+-----------
                                 [V11]
   A11       !      2!62!     0.000!  -128.000!      5.000!  70.0
   A11       !    -11!62!     0.000!   160.000!      5.000!  70.0
                                 [V11]
   A12       !     -5!62!     0.000!   -32.000!     15.000!  70.0
                                 [V11]
   A22       !     -1!62!     0.000!  -160.000!     15.000!  70.0
                                 [V11]
   A13       !      2!62!     0.000!  -128.000!      5.000!  70.0
   A13       !     -3!62!     0.000!   -96.000!      8.000!  70.0
                                 [V11]
   A14       !      2!62!     0.000!  -128.000!      5.000!  70.0
   A14       !    -11!62!     0.000!   160.000!      5.000!  70.0
                                 [V11]
   A15       !      2!62!     0.000!  -128.000!      5.000!  70.0
   A15       !    -11!62!     0.000!   160.000!      5.000!  70.0
-------------------------------------------------------------------
 *BOHRKOPF !SPINDEL!WK!DELTA-X   !DELTA-Y   !DURCHMESSER! KOMMENTAR
----------+----------+----------+----------+-----------+-----------
                              [NoValidForUse]
     O11       !      1!62!     0.000!     0.000!      0.000!
                              [NoValidForUse]
     O21       !      1!62!     0.000!     0.000!      0.000!
                              [NoValidForUse]
     O12       !      1!62!     0.000!     0.000!      0.000!
                              [NoValidForUse]
     O22       !      1!62!     0.000!     0.000!      0.000!
                              [NoValidForUse]
     O13       !      1!62!     0.000!     0.000!      0.000!
                              [NoValidForUse]
     O23       !      1!62!     0.000!     0.000!      0.000!
-------------------------------------------------------------------
 *BOHRKOPF !SPINDEL!WK!DELTA-X   !DELTA-Y   !DURCHMESSER! KOMMENTAR
----------+----------+----------+----------+-----------+-----------
                                 [h11]
   H11       !      6!62!     0.000!   160.000!      8.000!  70.0
   H11       !      8!62!     0.000!   224.000!      8.000!  60.0
                                 [h11]
   H21       !     -1!62!     0.000!     0.000!      8.000!  60.0
   H21       !     -3!62!     0.000!    64.000!      8.000!  70.0
```

6 priedas. Duomenų transformavimas ir paruošimas

```
# Check data sets for missing values

print('Bad rows (with missing values) caunt: ', data.shape[0]-data.dropna().shape[0],
     '\nRaw data shape: ', data.shape,
     '\nClean data shape: ', data.dropna().shape,);

>>> Bad rows (with missing values) caunt:  0
>>> Raw data shape:  (795, 21)
>>> Clean data shape:  (795, 21)

# Seperate data to differents categorys by type
continuous = ['NPC', 'Width', 'Lenght', 'LiftPushSpeed', 'LiftRegularSpeed',
              'RT1EntranceSpeed', 'RT1ExitSpeed', 'RT2EntranceSpeed', 'RT2ExitSpeed',
              'EM1PushSpeed', 'EM1PushCycle', 'EM2PushSpeed', 'EM2PushCycle', 'Parts']
id = ['Product']
categorical = ['Position', 'BR', 'Color']
binary = ['Path', 'Chipboard']
drop = ['ProductAGR', 'ProductName', 'WorckCenter']

# functions to menage original data
def set_chipboard(row):
    if row['Product'][-1] == 'A': return 1
    else: return 0

def set_color(row):
    if row['Product'][6] == 'W': return 'white'
    elif row['Product'][6] == 'B': return 'black'
    elif row['Product'][6] == 'O': return 'oack'
    elif row['Product'][6] == 'C': return 'chary'
    elif row['Product'][6] == 'X': return 'none'
    else: return ''

def set_path(row):
    if row['Path'] == 1: return 1
    else: return 0

def set_dummies(data, dummies):
    df = data
    for dummy in dummies:
        df_dummies = pd.get_dummies(data[dummy], prefix=dummy)
        df = pd.concat([df, df_dummies], axis=1)
    return df

# Data normalization
df[continuous] = df[continuous].apply(lambda x: (x - np.mean(x)) / (np.max(x) -
np.min(x)))
```

7 priedas. Kategorinių ir tolydžių kintamųjų histogramos



8 priedas. Klasterizavimo metodų rializavimas python programavimo kalba

```
#Parameter grid preparation
def split_grid(parameter_grid):
    """Input: {x:[y]}, Output: [{x:y}]"""
    return [list_of_toople_to_dic([*zip(parameter_grid.keys(), values)])
            for values
            in [*itertools.product(*parameter_grid.values())]]

def list_of_toople_to_dic(values):
    """Input: [(x,y)], Output: {x:y}"""
    return {key:value for key, value in values}


#K-Means
from sklearn.cluster import KMeans

parameter_grid = dict(
    n_clusters = np.round(np.arange(10, 31, 1), decimals=0),
    n_init = 100
)

options_grid = split_grid(parameter_grid)

for options in options_grid:
    model = KMeans(random_state = 0, **options).fit(df.values)
    labels = model.labels_
    clust_results['Method'].append('KMeans')
    clust_results['Model'].append(options)
    clust_results = add_metrics(model, df.values, clust_results)


# Hierarchical Clustering
from sklearn.cluster import AgglomerativeClustering

parameter_grid = dict(
    n_clusters = np.round(np.arange(10, 31, 1), decimals=0),
    linkage=['ward','complete','average','single']
)

options_grid = split_grid(parameter_grid)

for options in options_grid:
    model = AgglomerativeClustering(**options).fit(df.values)
    labels = model.labels_
    clust_results['Method'].append('Hierarchical - {}'.format(options.get('linkage')))
    clust_results['Model'].append(options)
    clust_results = add_metrics(model, df.values, clust_results)









# DBSCAN
from sklearn.cluster            import DBSCAN

parameter_grid = dict(
```

```
    eps=np.round(np.arange(0.1, 7, 0.1), decimals=1),
    min_samples=5,
    metric=`euclidian`
)

options_grid = split_grid(parameter_grid)

for options in options_grid:
    model = DBSCAN(n_jobs=-1, **options, min_samples = 5).fit(df.values)
    labels = model.labels_
    clust_results['Method'].append('DBSCAN')
    clust_results['Model'].append(options)
    clust_results = add_metrics(model, df.values, clust_results)


# HDBSCAN
import hdbscan

parameter_grid = dict(
    min_cluster_size = 5,
    metric=`euclidian`
)

options_grid = split_grid(parameter_grid)

for options in options_grid:
    model = hdbscan.HDBSCAN(gen_min_span_tree=True, **options, min_cluster_size=5).fit(
df.values)
    clust_results['Method'].append('HDBSCAN')
    clust_results['Model'].append(options)
    clust_results = add_metrics(model, df.values, clust_results)


# Birch
from sklearn.cluster            import Birch

parameter_grid = dict(
    n_clusters = getattr(np.round(np.arange(10, 31, 1, dtype=int), decimals=0), "tolist
", lambda: value)(),
threshold= getattr(np.round(np.arange(0.1, 1, 0.1, dtype=int), decimals=0.1), "tolist",
 lambda: value)()
)

options_grid = split_grid(parameter_grid)

for options in options_grid:
    model = Birch(**options).fit(df.values)
    clust_results['Method'].append('BIRCH')
    clust_results['Model'].append(options)
    clust_results = add_metrics(model, df.values, clust_results)
```

9 priedas. Optimizavimo metodų rializavimas python programavimo kalba

```
# Branch and bound
import networkx as nx

# This function computes a lower bound on the length of Hamiltonian cycles starting wit
h vertices in the list sub_cycle.
# I would recommend to first see the branch_and_bound function below, and then return t
o lower_bound.
def lower_bound(g, sub_cycle):
    # The weight of the current path.
    current_weight = sum([g[sub_cycle[i]][sub_cycle[i + 1]]['weight'] for i in range(le
n(sub_cycle) - 1)])

    # For convenience we create a new graph which only contains vertices not used by g.
    unused = [v for v in g.nodes() if v not in sub_cycle]
    h = g.subgraph(unused)


    # Compute the weight of a minimum spanning tree.
    t = list(nx.minimum_spanning_edges(h))
    mst_weight = sum([h.get_edge_data(e[0], e[1])['weight'] for e in t])

    # If the current sub_cycle is "trivial" (i.e., it contains no vertices or all verti
ces), then our lower bound is
    # just the sum of the weight of a minimum spanning tree and the current weight.
    if len(sub_cycle) == 0 or len(sub_cycle) == g.number_of_nodes():
        return mst_weight + current_weight

    # If the current sub_cycle is not trivial, then we can also add the weight of two e
dges connecting the vertices
    # from sub_cycle and the remaining part of the graph.
    # s is the first vertex of the sub_cycle
    s = sub_cycle[0]
    # t is the last vertex of the sub_cycle
    t = sub_cycle[-1]
    # The minimum weight of an edge connecting a vertex from outside of sub_sycle to s.
    min_to_s_weight = min([g[v][s]['weight'] for v in g.nodes() if v not in sub_cycle])
    # The minimum weight of an edge connecting the vertex t to a vertex from outside of
 sub_cycle.
    min_from_t_weight = min([g[t][v]['weight'] for v in g.nodes() if v not in sub_cycle
])

    # Any cycle which starts with sub_cycle must be of length:
    # the weight of the edges from sub_cycle +
    # the minimum weight of an edge connecting sub_cycle and the remaining vertices +
    # the minimum weight of a spanning tree on the remaining vertices +
    # the minimum weight of an edge connecting the remaining vertices to sub_cycle.
    return current_weight + min_from_t_weight + mst_weight + min_to_s_weight

# The branch and bound procedure takes
# 1. a graph g;
# 2. the current sub_cycle, i.e. several first vertices of cycle under consideration.
# Initially sub_cycle is empty;
# 3. currently best solution current_min, so that we don't even consider paths of great
er weight.
# Initially the min weight is infinite
def branch_and_bound(g, sub_cycle=None, current_min=float("inf"), path=[]):
```

```
    # If the current path is empty, then we can safely assume that it starts with the v
ertex 0.
    if sub_cycle is None:
        sub_cycle = [0]

    # If we already have all vertices in the cycle, then we just compute the weight of
this cycle and return it.
    if len(sub_cycle) == g.number_of_nodes():
        weight = sum([g[sub_cycle[i]][sub_cycle[i + 1]]['weight'] for i in range(len(su
b_cycle) - 1)])
        weight = weight + g[sub_cycle[-1]][sub_cycle[0]]['weight']
        return weight, path

    # Now we look at all nodes which aren't yet used in sub_cycle.
    unused_nodes = list()
    for v in g.nodes():
        if v not in sub_cycle:
            unused_nodes.append((g[sub_cycle[-1]][v]['weight'], v))

    # We sort them by the distance from the "current node" -
- the last node in sub_cycle.
    unused_nodes = sorted(unused_nodes)

    for (d, v) in unused_nodes:
        assert v not in sub_cycle
        extended_subcycle = list(sub_cycle)
        extended_subcycle.append(v)
        # For each unused vertex, we check if there is any chance to find a shorter cyc
le if we add it now.
        if lower_bound(g, extended_subcycle) < current_min:
            # If there is such a chance, we add the vertex to the current cycle, and pr
oceed recursively.
            # If we found a short cycle, then we update the current_min value.
            branch_min, path= branch_and_bound(g, extended_subcycle, current_min, path)
            if current_min > branch_min:
                current_min=branch_min
                if len(extended_subcycle) == g.number_of_nodes():
                    path=extended_subcycle

    # The procedure returns the shortest cycle length.
    return current_min, path


def greedy(cities, start_city=0):
    start = timeit.default_timer()
    #Path of the tour
    path = [start_city]
    #Cost of the tour
    cost = 0
    #Assume the first city as starting point, create a temporary value
    temp = start_city
    #Hold the position in the list
    position = start_city
    #The number of the unvisited cities
    flag = len(cities) - 1
    #Current cost
    current_cost = 0
    while flag > 0:
        #Find the nearest city
```

```
        for x in range(0, len(cities)):
            #x not in path, mean we don't care about the cities that we visited
            if (cities[temp][x] != 0) and (x not in path):
                if current_cost == 0:
                    current_cost = cities[temp][x]
                    position = x
                if current_cost > cities[temp][x]:
                    current_cost = cities[temp][x]
                    position = x
        cost += int(current_cost)
        #Reset current cost for next calculating
        current_cost = 0
        temp = position
        path.append(position)
        if flag == 1:
            #Add the connected path from last city to the start city
            current_cost = cities[position][start_city]
            cost += current_cost
            path.append(start_city)
        flag -= 1
    stop = timeit.default_timer()
    time_finish = stop - start
    algorithm = "Greedy"
    result = [algorithm, cost, path, time_finish]
    # print "The cost of the tour is:"+str(result[1])
    # print "The path of the tour is:"+str(result[2])
    # print "The time to finish is:"+str(result[3])+" in second"
    return result


for c in opt_clusters:
    list(df_labels[df_labels['Label']==c]['Component'])
    cluster = list(df_labels[df_labels['Label']==c]['Component'])

    tmp_df_euclidean_distances= df_euclidean_distances\
        .filter(cluster)\
        .filter(cluster, axis=0)

    # Branch and Bound algorithm
    opt_results = branch_and_bound(tmp_df_euclidean_distances.values)

    dic_opt_results['Optimizavimo metodai'].append(opt_results[0])
    dic_opt_results['Bendras maršruto svoris'].append(opt_results[1])
    dic_opt_results['Skaičiavimo trukmė'].append(opt_results[3])
    dic_opt_results['Klasteris'].append(c)
    dic_opt_results['Klasterio elementų skaičius'].append(len(cluster))

    # Greedy algorithm
    opt_results = greedy(tmp_df_euclidean_distances.values)

    dic_opt_results['Optimizavimo metodai'].append(opt_results[0])
    dic_opt_results['Bendras maršruto svoris'].append(opt_results[1])
    dic_opt_results['Skaičiavimo trukmė'].append(opt_results[3])
    dic_opt_results['Klasteris'].append(c)
    dic_opt_results['Klasterio elementų skaičius'].append(len(cluster))

    # Randomized algorithm
    opt_results = randomized(tmp_df_euclidean_distances.values)
```

```
    dic_opt_results['Optimizavimo metodai'].append(opt_results[0])
    dic_opt_results['Bendras maršruto svoris'].append(opt_results[1])
    dic_opt_results['Skaičiavimo trukmė'].append(opt_results[3])
    dic_opt_results['Klasteris'].append(c)
    dic_opt_results['Klasterio elementų skaičius'].append(len(cluster))

    # Minimum span tree algorithm
    opt_results = MinSpanTree(tmp_df_euclidean_distances.values)

    dic_opt_results['Optimizavimo metodai'].append(opt_results[0])
    dic_opt_results['Bendras maršruto svoris'].append(opt_results[1])
    dic_opt_results['Skaičiavimo trukmė'].append(opt_results[3])
    dic_opt_results['Klasteris'].append(c)
    dic_opt_results['Klasterio elementų skaičius'].append(len(cluster))

    # 2-Opt algorithm
    opt_results = two_opt(tmp_df_euclidean_distances.values)

    dic_opt_results['Optimizavimo metodai'].append(opt_results[0])
    dic_opt_results['Bendras maršruto svoris'].append(opt_results[1])
    dic_opt_results['Skaičiavimo trukmė'].append(opt_results[3])
    dic_opt_results['Klasteris'].append(c)
    dic_opt_results['Klasterio elementų skaičius'].append(len(cluster))

    # Simulated annealing
    opt_results = hillclimbing_algorithm(tmp_df_euclidean_distances.values, len(cluster
))

    dic_opt_results['Optimizavimo metodai'].append('Simulated annealing')
    dic_opt_results['Bendras maršruto svoris'].append(opt_results[1])
    dic_opt_results['Skaičiavimo trukmė'].append(opt_results[3])
    dic_opt_results['Klasteris'].append(c)
    dic_opt_results['Klasterio elementų skaičius'].append(len(cluster))

    # Genetic algorithm
    opt_results = genetic_algorithm(tmp_df_euclidean_distances.values, len(cluster))

    dic_opt_results['Optimizavimo metodai'].append(opt_results[0])
    dic_opt_results['Bendras maršruto svoris'].append(opt_results[1])
    dic_opt_results['Skaičiavimo trukmė'].append(opt_results[3])
    dic_opt_results['Klasteris'].append(c)
    dic_opt_results['Klasterio elementų skaičius'].append(len(cluster))

    # Ant colony
    G= g.subgraph(cluster)

    start_time = datetime.datetime.now()

    solver = acopy.Solver(rho=.03, q=1)
    colony = acopy.Colony(alpha=1, beta=3)
    tour = solver.solve(G, colony, limit=100)

    duration = datetime.datetime.now() - start_time

    dic_opt_results['Optimizavimo metodai'].append('Ant colony')
    dic_opt_results['Bendras maršruto svoris'].append(tour.cost)
    dic_opt_results['Skaičiavimo trukmė'].append(duration.total_seconds())
    dic_opt_results['Klasteris'].append(c)
    dic_opt_results['Klasterio elementų skaičius'].append(len(cluster))
```

10 priedas. Klasterizavimo modelių rezultatai

Method	Model	Min. cluster size	Max. cluster size	Clusters	Silhouette Coefficient
KMeans	{'n_clusters': 10}	8	66	10	0.4098150183805426
KMeans	{'n_clusters': 11}	10	66	11	0.44186686506007256
KMeans	{'n_clusters': 12}	11	66	12	0.4565125743911773
KMeans	{'n_clusters': 13}	10	61	13	0.479211575034006
KMeans	{'n_clusters': 14}	8	66	14	0.5034710445945942
KMeans	{'n_clusters': 15}	6	60	15	0.5251000620660606
KMeans	{'n_clusters': 16}	8	66	16	0.5488149855791139
KMeans	{'n_clusters': 17}	6	60	17	0.5724544613733544
KMeans	{'n_clusters': 18}	5	60	18	0.5931859295225232
KMeans	{'n_clusters': 19}	5	60	19	0.6150608761643286
KMeans	{'n_clusters': 20}	5	55	20	0.6400523259751949
KMeans	{'n_clusters': 21}	4	55	21	0.6573188592936017
KMeans	{'n_clusters': 22}	4	55	22	0.6688704140460378
KMeans	{'n_clusters': 23}	4	51	23	0.6950008466553169
KMeans	{'n_clusters': 24}	4	51	24	0.7191939316973505
KMeans	{'n_clusters': 25}	4	51	25	0.7356369998933476
KMeans	{'n_clusters': 26}	4	51	26	0.7507903789923075
KMeans	{'n_clusters': 27}	4	51	27	0.7653511179853559
KMeans	{'n_clusters': 28}	4	51	28	0.7815914372989664
KMeans	{'n_clusters': 29}	4	51	29	0.7963058942957979
KMeans	{'n_clusters': 30}	4	51	30	0.8122788769165918
Hierarchical - ward	{'n_clusters': 10, 'linkage': 'ward'}	11	56	10	0.43100267516066176
Hierarchical - complete	{'n_clusters': 10, 'linkage': 'complete'}	4	81	10	0.3547342037541641
Hierarchical - average	{'n_clusters': 10, 'linkage': 'average'}	2	85	10	0.37977809656981815
Hierarchical - single	{'n_clusters': 10, 'linkage': 'single'}	2	151	10	0.33549363818916306
Hierarchical - ward	{'n_clusters': 11, 'linkage': 'ward'}	11	53	11	0.4561271152817022
Hierarchical - complete	{'n_clusters': 11, 'linkage': 'complete'}	4	81	11	0.4024724790006419
Hierarchical - average	{'n_clusters': 11, 'linkage': 'average'}	2	85	11	0.40257637738260665
Hierarchical - single	{'n_clusters': 11, 'linkage': 'single'}	2	147	11	0.3510500793717889
Hierarchical - ward	{'n_clusters': 12, 'linkage': 'ward'}	11	51	12	0.4807966546980807
Hierarchical - complete	{'n_clusters': 12, 'linkage': 'complete'}	4	61	12	0.4345590288070794
Hierarchical - average	{'n_clusters': 12, 'linkage': 'average'}	2	70	12	0.4248672247427056

Hierarchical - single	{'n_clusters': 12, 'linkage': 'single'}	2	147	12	0.37351915946343295
Hierarchical - ward	{'n_clusters': 13, 'linkage': 'ward'}	11	51	13	0.5032762571838821
Hierarchical - complete	{'n_clusters': 13, 'linkage': 'complete'}	4	61	13	0.4599634911449599
Hierarchical - average	{'n_clusters': 13, 'linkage': 'average'}	2	70	13	0.4523632725199087
Hierarchical - single	{'n_clusters': 13, 'linkage': 'single'}	2	128	13	0.4130215030233271
Hierarchical - ward	{'n_clusters': 14, 'linkage': 'ward'}	8	51	14	0.5287318187618757
Hierarchical - complete	{'n_clusters': 14, 'linkage': 'complete'}	2	61	14	0.471533445921666
Hierarchical - average	{'n_clusters': 14, 'linkage': 'average'}	2	66	14	0.4926767771078465
Hierarchical - single	{'n_clusters': 14, 'linkage': 'single'}	2	83	14	0.4541168886936578
Hierarchical - ward	{'n_clusters': 15, 'linkage': 'ward'}	8	51	15	0.5467665117285792
Hierarchical - complete	{'n_clusters': 15, 'linkage': 'complete'}	2	61	15	0.5074778305654686
Hierarchical - average	{'n_clusters': 15, 'linkage': 'average'}	2	66	15	0.5197056785070175
Hierarchical - single	{'n_clusters': 15, 'linkage': 'single'}	2	60	15	0.4892632566785908
Hierarchical - ward	{'n_clusters': 16, 'linkage': 'ward'}	8	51	16	0.5644689424366375
Hierarchical - complete	{'n_clusters': 16, 'linkage': 'complete'}	2	61	16	0.5348900611434173
Hierarchical - average	{'n_clusters': 16, 'linkage': 'average'}	2	66	16	0.5361661295292935
Hierarchical - single	{'n_clusters': 16, 'linkage': 'single'}	2	56	16	0.515412274723993
Hierarchical - ward	{'n_clusters': 17, 'linkage': 'ward'}	8	51	17	0.5843096178186983
Hierarchical - complete	{'n_clusters': 17, 'linkage': 'complete'}	2	61	17	0.5468120358363824
Hierarchical - average	{'n_clusters': 17, 'linkage': 'average'}	2	61	17	0.5534866167704274
Hierarchical - single	{'n_clusters': 17, 'linkage': 'single'}	2	56	17	0.5406442556736966
Hierarchical - ward	{'n_clusters': 18, 'linkage': 'ward'}	8	51	18	0.6041639162152624
Hierarchical - complete	{'n_clusters': 18, 'linkage': 'complete'}	2	61	18	0.5672985086978729
Hierarchical - average	{'n_clusters': 18, 'linkage': 'average'}	2	61	18	0.5760348267348772
Hierarchical - single	{'n_clusters': 18, 'linkage': 'single'}	2	56	18	0.5585176022204859
Hierarchical - ward	{'n_clusters': 19, 'linkage': 'ward'}	5	51	19	0.6253048526777111
Hierarchical - complete	{'n_clusters': 19, 'linkage': 'complete'}	2	55	19	0.6001268161488568
Hierarchical - average	{'n_clusters': 19, 'linkage': 'average'}	2	61	19	0.5938662717307729
Hierarchical - single	{'n_clusters': 19, 'linkage': 'single'}	2	56	19	0.5785324962279157

Hierarchical - ward	{'n_clusters': 20, 'linkage': 'ward'}	5	51	20	0.6445563991787872
Hierarchical - complete	{'n_clusters': 20, 'linkage': 'complete'}	2	55	20	0.6267642838002502
Hierarchical - average	{'n_clusters': 20, 'linkage': 'average'}	2	55	20	0.6266945791817566
Hierarchical - single	{'n_clusters': 20, 'linkage': 'single'}	2	52	20	0.5943061532617557
Hierarchical - ward	{'n_clusters': 21, 'linkage': 'ward'}	5	51	21	0.6631007247654905
Hierarchical - complete	{'n_clusters': 21, 'linkage': 'complete'}	2	55	21	0.6449051873242284
Hierarchical - average	{'n_clusters': 21, 'linkage': 'average'}	2	55	21	0.64640924282315
Hierarchical - single	{'n_clusters': 21, 'linkage': 'single'}	2	52	21	0.6132970994061695
Hierarchical - ward	{'n_clusters': 22, 'linkage': 'ward'}	4	51	22	0.6840625409599052
Hierarchical - complete	{'n_clusters': 22, 'linkage': 'complete'}	2	55	22	0.6472851112319801
Hierarchical - average	{'n_clusters': 22, 'linkage': 'average'}	2	55	22	0.6616783682187978
Hierarchical - single	{'n_clusters': 22, 'linkage': 'single'}	2	51	22	0.6540427421891083
Hierarchical - ward	{'n_clusters': 23, 'linkage': 'ward'}	4	51	23	0.7026553216374675
Hierarchical - complete	{'n_clusters': 23, 'linkage': 'complete'}	2	55	23	0.6673556660672312
Hierarchical - average	{'n_clusters': 23, 'linkage': 'average'}	2	55	23	0.6844050683363723
Hierarchical - single	{'n_clusters': 23, 'linkage': 'single'}	2	51	23	0.676389532188853
Hierarchical - ward	{'n_clusters': 24, 'linkage': 'ward'}	4	51	24	0.7188817366067026
Hierarchical - complete	{'n_clusters': 24, 'linkage': 'complete'}	2	55	24	0.6936821290169386
Hierarchical - average	{'n_clusters': 24, 'linkage': 'average'}	2	51	24	0.7098258027468639
Hierarchical - single	{'n_clusters': 24, 'linkage': 'single'}	2	51	24	0.71173913620031
Hierarchical - ward	{'n_clusters': 25, 'linkage': 'ward'}	4	51	25	0.7376397470595051
Hierarchical - complete	{'n_clusters': 25, 'linkage': 'complete'}	2	55	25	0.7137519181058813
Hierarchical - average	{'n_clusters': 25, 'linkage': 'average'}	2	51	25	0.7345763744327153
Hierarchical - single	{'n_clusters': 25, 'linkage': 'single'}	2	51	25	0.7262010457482184
Hierarchical - ward	{'n_clusters': 26, 'linkage': 'ward'}	4	51	26	0.7548481103371588
Hierarchical - complete	{'n_clusters': 26, 'linkage': 'complete'}	2	51	26	0.7391726525163731
Hierarchical - average	{'n_clusters': 26, 'linkage': 'average'}	2	51	26	0.7480466153081382
Hierarchical - single	{'n_clusters': 26, 'linkage': 'single'}	2	51	26	0.7382107441136946
Hierarchical - ward	{'n_clusters': 27, 'linkage': 'ward'}	4	51	27	0.7686294077577924

Hierarchical - complete	{'n_clusters': 27, 'linkage': 'complete'}	2	51	27	0.755964370945682
Hierarchical - average	{'n_clusters': 27, 'linkage': 'average'}	2	51	27	0.7634832178695822
Hierarchical - single	{'n_clusters': 27, 'linkage': 'single'}	2	51	27	0.7618444507113401
Hierarchical - ward	{'n_clusters': 28, 'linkage': 'ward'}	4	51	28	0.7846023903785864
Hierarchical - complete	{'n_clusters': 28, 'linkage': 'complete'}	2	51	28	0.7688594957166455
Hierarchical - average	{'n_clusters': 28, 'linkage': 'average'}	2	51	28	0.7763783426405456
Hierarchical - single	{'n_clusters': 28, 'linkage': 'single'}	2	51	28	0.7778174333321338
Hierarchical - ward	{'n_clusters': 29, 'linkage': 'ward'}	2	51	29	0.7983274070217544
Hierarchical - complete	{'n_clusters': 29, 'linkage': 'complete'}	2	51	29	0.7845566168384475
Hierarchical - average	{'n_clusters': 29, 'linkage': 'average'}	2	51	29	0.7861343613492351
Hierarchical - single	{'n_clusters': 29, 'linkage': 'single'}	2	51	29	0.7923513252613396
Hierarchical - ward	{'n_clusters': 30, 'linkage': 'ward'}	2	51	30	0.8076824597291054
Hierarchical - complete	{'n_clusters': 30, 'linkage': 'complete'}	1	51	30	0.7852951888643004
Hierarchical - average	{'n_clusters': 30, 'linkage': 'average'}	2	51	30	0.8021073439700288
Hierarchical - single	{'n_clusters': 30, 'linkage': 'single'}	1	51	30	0.7915121527764049
DBSCAN	{'eps': 0.1}	5	194	12	0.1062118601458152
DBSCAN	{'eps': 0.2}	5	145	18	0.28803355614896564
DBSCAN	{'eps': 0.3}	5	123	17	0.4257004957501208
DBSCAN	{'eps': 0.4}	5	118	17	0.4611377289129841
DBSCAN	{'eps': 0.5}	5	107	18	0.5138169070182574
DBSCAN	{'eps': 0.6}	5	105	18	0.5268689651306311
DBSCAN	{'eps': 0.7}	5	99	19	0.5573330747692327
DBSCAN	{'eps': 0.8}	5	99	19	0.5573330747692327
DBSCAN	{'eps': 0.9}	5	99	18	0.5645127897518883
DBSCAN	{'eps': 1.0}	5	93	19	0.5893812647572341
DBSCAN	{'eps': 1.1}	5	93	19	0.5893812647572341
DBSCAN	{'eps': 1.2}	5	93	19	0.5893812647572341
DBSCAN	{'eps': 1.3}	5	93	19	0.5893812647572341
DBSCAN	{'eps': 1.4}	5	93	19	0.5893812647572341
DBSCAN	{'eps': 1.5}	5	88	20	0.6060043949336565
DBSCAN	{'eps': 1.6}	5	88	20	0.6060043949336565
DBSCAN	{'eps': 1.7}	5	88	20	0.6060043949336565
DBSCAN	{'eps': 1.8}	5	88	20	0.6060043949336565
DBSCAN	{'eps': 1.9}	5	88	20	0.6060043949336565
DBSCAN	{'eps': 2.0}	5	88	20	0.6060043949336565

DBSCAN	{'eps': 2.1}	5	88	20	0.6060043949336565
DBSCAN	{'eps': 2.2}	5	88	20	0.6060043949336565
DBSCAN	{'eps': 2.3}	5	88	20	0.6060043949336565
DBSCAN	{'eps': 2.4}	5	88	20	0.6060043949336565
DBSCAN	{'eps': 2.5}	5	78	22	0.6429571937036649
DBSCAN	{'eps': 2.6}	5	78	22	0.6429571937036649
DBSCAN	{'eps': 2.7}	5	78	22	0.6429571937036649
DBSCAN	{'eps': 2.8}	5	78	22	0.6429571937036649
DBSCAN	{'eps': 2.9}	5	78	22	0.6429571937036649
DBSCAN	{'eps': 3.0}	5	78	22	0.6429571937036649
DBSCAN	{'eps': 3.1}	5	78	22	0.6429571937036649
DBSCAN	{'eps': 3.2}	5	62	23	0.6842570906428985
DBSCAN	{'eps': 3.3}	5	62	23	0.6842570906428985
DBSCAN	{'eps': 3.4}	5	62	23	0.6842570906428985
DBSCAN	{'eps': 3.5}	5	58	23	0.6967376559089434
DBSCAN	{'eps': 3.6}	5	58	23	0.6967376559089434
DBSCAN	{'eps': 3.7}	5	58	23	0.6967376559089434
DBSCAN	{'eps': 3.8}	5	58	23	0.6967376559089434
DBSCAN	{'eps': 3.9}	5	58	23	0.6967376559089434
DBSCAN	{'eps': 4.0}	5	58	23	0.6967376559089434
DBSCAN	{'eps': 4.1}	5	58	23	0.6967376559089434
DBSCAN	{'eps': 4.2}	5	58	23	0.6967376559089434
DBSCAN	{'eps': 4.3}	5	58	23	0.6967376559089434
DBSCAN	{'eps': 4.4}	5	54	23	0.7039116443073766
DBSCAN	{'eps': 4.5}	5	54	23	0.7039116443073766
DBSCAN	{'eps': 4.6}	5	52	23	0.7049942827343102
DBSCAN	{'eps': 4.7}	5	51	23	0.7077556624613545
DBSCAN	{'eps': 4.8}	5	51	23	0.7077556624613545
DBSCAN	{'eps': 4.9}	5	51	23	0.7077556624613545
DBSCAN	{'eps': 5.0}	5	51	23	0.7077556624613545
DBSCAN	{'eps': 5.1}	5	51	23	0.7077556624613545
DBSCAN	{'eps': 5.2}	5	51	22	0.6890253117115381
DBSCAN	{'eps': 5.3}	5	51	22	0.6890253117115381
DBSCAN	{'eps': 5.4}	5	51	22	0.6892328521979331
DBSCAN	{'eps': 5.5}	5	51	21	0.6723216584321708
DBSCAN	{'eps': 5.6}	5	51	20	0.6603119600666948
DBSCAN	{'eps': 5.7}	5	51	20	0.6603119600666948
DBSCAN	{'eps': 5.8}	5	51	20	0.6603119600666948
DBSCAN	{'eps': 5.9}	5	52	17	0.5669523635722621
DBSCAN	{'eps': 6.0}	5	56	16	0.5458594600040819

DBSCAN	{'eps': 6.1}	5	56	14	0.5107507412482406
DBSCAN	{'eps': 6.2}	5	147	10	0.3637496039260799
DBSCAN	{'eps': 6.3}	5	147	10	0.3637496039260799
DBSCAN	{'eps': 6.4}	2	167	7	0.2947545933262453
DBSCAN	{'eps': 6.5}	2	167	7	0.2947545933262453
DBSCAN	{'eps': 6.6}	2	187	6	0.2453144948998443
DBSCAN	{'eps': 6.7}	2	200	4	0.21632756645198506
DBSCAN	{'eps': 6.8}	2	266	2	0.04556158193821139
DBSCAN	{'eps': 6.9}	2	266	2	0.04556158193821139
HDBSCAN	{}	5	44	28	0.6229396126237883
BIRCH	{'n_clusters': 10}	8	66	10	0.4123349052603325
BIRCH	{'n_clusters': 11}	8	66	11	0.43822756197928026
BIRCH	{'n_clusters': 12}	8	66	12	0.46316305025029103
BIRCH	{'n_clusters': 13}	8	66	13	0.4814662031056738
BIRCH	{'n_clusters': 14}	5	61	14	0.4976837276260621
BIRCH	{'n_clusters': 15}	5	61	15	0.516529515183802
BIRCH	{'n_clusters': 16}	5	61	16	0.5263111116079167
BIRCH	{'n_clusters': 17}	5	61	17	0.548314588459082
BIRCH	{'n_clusters': 18}	5	61	18	0.5768059194075347
BIRCH	{'n_clusters': 19}	2	61	19	0.5874512616878569
BIRCH	{'n_clusters': 20}	2	61	20	0.6102520955256405
BIRCH	{'n_clusters': 21}	2	61	21	0.6309966194187757
BIRCH	{'n_clusters': 22}	2	61	22	0.6495894000963381
BIRCH	{'n_clusters': 23}	2	61	23	0.6663677057678966
BIRCH	{'n_clusters': 24}	2	61	24	0.6864556077612889
BIRCH	{'n_clusters': 25}	2	55	25	0.7192839152122728
BIRCH	{'n_clusters': 26}	2	55	26	0.7278575307855777
BIRCH	{'n_clusters': 27}	2	51	27	0.7532782651960694
BIRCH	{'n_clusters': 28}	2	51	28	0.7688594957166455
BIRCH	{'n_clusters': 29}	2	51	29	0.7845566168384475
BIRCH	{'n_clusters': 30}	1	51	30	0.7852951888643004

11 priedas. Optimizavimo metodų rezultatai atliekant optimalių sekų paiešką

Optimizavimo metodai	**Bendras maršruto svoris**	**Skaičiavimo trukmė**	**Klasteris**	**Klasterio elementų skaičius**
Branch and Bound	3.55069819428482	0.56054129999363795	A	51
Greedy	6.16974581333243	0.00253950001206249	A	51
Randomized	9.15806445219072	0.000374700000975281	A	51
Minimum Spanning Tree	6.01936301527527	0.006604000000038468	A	51
2-Opt	6.16974581333243	0.005411999999978434	A	51
Evolutionary-HillClimbing	3.8173155177676	0.7156750000000329	A	51
Genetic Algorithm	4.3343878721014	55.33323199999995	A	51
Ant colony	3.77320264306418	61.554735	A	51
Branch and Bound	0.3037977670309478	0.05911179998656735	K	40
Greedy	0.8575168844456862	0.0014146000030450523	K	40
Randomized	12.8122393053002	0.000295000005280599	K	40
Minimum Spanning Tree	5.717754492204559	0.003818000000023858	K	40
2-Opt	0.8575168844456862	0.0032110000000784567	K	40
Evolutionary-HillClimbing	4.709929857953088	0.554663000000005	K	40
Genetic Algorithm	4.905669785284157	45.20082100000002	K	40
Ant colony	5.606600463264302	30.565024	K	40
Branch and Bound	0.0444717620531176	0.001559700001962483	Q	12
Greedy	0.3923788448607614	0.00011900000390596688	Q	12
Randomized	2.59995998236317	8.200001320801675e-05	Q	12
Minimum Spanning Tree	1.8712963997492773	0.0002789999998640269	Q	12
2-Opt	0.3923788448607614	0.0002509999999347201	Q	12
Evolutionary-HillClimbing	1.0829060393450414	0.20202900000003865	Q	12
Genetic Algorithm	1.0829060393450414	15.501755000000003	Q	12
Ant colony	1.1025785032132558	1.157381	Q	12
Branch and Bound	0.0	0.002535200008423999	V	14
Greedy	0.43755677182498426	0.00016870000399649143	V	14
Randomized	3.4460736003912946	6.710001616738737e-05	V	14
Minimum Spanning Tree	2.2923221259264444	0.00040500000000065484	V	14
2-Opt	0.43755677182498426	0.00048199999991993536	V	14
Evolutionary-HillClimbing	1.7143695286120926	0.22195899999996982	V	14
Genetic Algorithm	1.6785446100635841	17.361012999999957	V	14
Ant colony	1.8836048412068385	1.635909	V	14
Branch and Bound	1.0006313876170831	0.0058723000111058354	X	19
Greedy	2.458228573186076	0.0003519000019878149	X	19
Randomized	10.336898595438356	0.00012100001913495362	X	19
Minimum Spanning Tree	3.9274287061098176	0.0008769999999458378	X	19
2-Opt	2.458228573186076	0.0006419999999707215	X	19

Evolutionary-HillClimbing	2.7908643455119075	0.2900250000000142	X	19
Genetic Algorithm	2.790864345511907	22.626845000000003	X	19
Ant colony	3.8882828883868408	3.650517	X	19

12 priedas. Optimizavimo metodų ir pilno perrinkimo rezultatai skirtingiems klasteriams

Optimizavimo metodai	**Bendras maršruto svoris**	**Skaičiavimo trukmė**	**Klasteris**	**Klasterio elementų skaičius**
Branch and Bound	0.9046776767842331	0.0006426999998438987	Y	9
Greedy	1.30759404215173725	4.910000006930204e-05	Y	9
Randomized	1.9032366756302528	4.500000000007276e-05	Y	9
Minimum Spanning Tree	1.408269468897529	0.0002480000000559812	Y	9
2-Opt	1.30759404215173725	0.00011000000006333721	Y	9
Evolutionary-HillClimbing	1.1834807485988916	0.13914599999998245	Y	9
Ant colony	1.4175300454661666	0.572913	Y	9
Pilnas perrinkimas	0.7044190192669215	15.241816	Y	9
Branch and Bound	0.15588145423261035	0.0003908000001047185	M	8
Greedy	0.17130139560821855	5.920000012338278e-05	M	8
Randomized	0.2522865829471608	3.249999986110197e-05	M	8
Minimum Spanning Tree	0.2522865829471608	0.00017099999990932702	M	8
2-Opt	0.18458005862439179	9.299999999257125e-05	M	8
Evolutionary-HillClimbing	0.2522865829471608	0.12778300000002218	M	8
Ant colony	0.2522865829471608	0.480563	M	8
Pilnas perrinkimas	0.08458005862439179	1.498188	M	8
Branch and Bound	0.3611039463502749	0.000326500000028318	U	7
Greedy	0.3611039463502749	6.529999996018887e-05	U	7
Randomized	0.9842661851947367	3.2400000009147334e-05	U	7
Minimum Spanning Tree	0.7222078927005172	0.0001199999999244028	U	7
2-Opt	0.3611039463502749	7.299999992937956e-05	U	7
Evolutionary-HillClimbing	0.7222078927004267	0.11481500000002143	U	7
Ant colony	0.7222078927004559	0.362929	U	7
Pilnas perrinkimas	0.36110394635021337	0.168925	U	7
Branch and Bound	0.930933797615573	0.000183999999990192	W	6
Greedy	0.930933797615573	2.5300000061179162e-05	W	6
Randomized	1.11808519322492	5.160000000614673e-05	W	6
Minimum Spanning Tree	0.930933797615573	0.0001089999999995217513	W	6
2-Opt	0.930933797615573	5.29999999798747e-05	W	6
Evolutionary-HillClimbing	0.930933797615573	0.10330999999996493	W	6
Ant colony	0.930933797615573	0.277909	W	6
Pilnas perrinkimas	0.7310371497163564	0.021332	W	6
Branch and Bound	1.8284271247461903	0.00010040000006483751	H	5

Greedy	2.414213562373095	1.5899999880275573e-05	H	5
Randomized	2.656854249492381	3.700000002027082e-05	H	5
Minimum Spanning Tree	1.8284271247461903	7.39999999268548e-05	H	5
2-Opt	1.8284271247461903	3.300000003036985e-05	H	5
Evolutionary-HillClimbing	1.8284271247461903	0.08704799999998158	H	5
Ant colony	1.8284271247461903	0.203923	H	5
Pilnas perrinkimas	1.4142135623730951	0.003025	H	5