Kauno technologijos universitetas

Informatikos fakultetas

2D konvoliucija paremtų metodų, skirtų apytikslio trumpiausio maršruto radimui, tyrimas

Baigiamasis magistro studijų projektas

Marius Teleiša

Projekto autorius

doc. dr. Dalia Čalnerytė

Vadovė

Kaunas, 2023

Kauno technologijos universitetas

Informatikos fakultetas

2D konvoliucija paremtų metodų, skirtų apytikslio trumpiausio maršruto radimui, tyrimas

Baigiamasis magistro studijų projektas

Dirbtinio Intelekto Informatika (6211BX007)

Marius Teleiša

Projekto autorius

doc. dr. Dalia Čalnerytė

Vadovė

prof. dr. Rytis Maskeliūnas

Recenzentas

Kaunas, 2023

Kauno technologijos universitetas

Informatikos fakultetas

Marius Teleiša

2D konvoliucija paremtų metodų, skirtų apytikslio trumpiausio maršruto radimui, tyrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdama(s) kitų asmenų autoriaus ar kitų teisių, laikydamasi(s) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs (-usi);

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalinta(s) iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Marius Teleiša

Patvirtinta elektroniniu būdu

Teleiša, Marius. 2D konvoliucija paremtų metodų, skirtų apytikslio trumpiausio maršruto radimui, tyrimas. Magistro baigiamasis projektas / vadovė doc. dr. Dalia Čalnerytė; Kauno technologijos universitetas, informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): informatikos mokslai, informatika (B01).

Reikšminiai žodžiai: kelio paieška, hierarchinė kelio paieška, 2D konvoliucija.

Kaunas, 2023. 69 p.

Santrauka

Darbe apžvelgiami įvairūs hierarchinės kelio paieškos ir randamo kelio ilgio gerinimo metodai. Analizės metu įgytos žinios pritaikytos projektuojant patobulinimus konvoliucinės hierarchinės paieškos *A** (*CHPA**) kelio paieškos algoritmui. Pirmas pakeitimas – pataisomos abstrakcijos sluoksnio klaidos panaikinant kraštines vedančias iš segmento, kur algoritmas mato kelią iš viršaus ir kairės į segmentą, tačiau įstrižai segmento kelio nėra. Toliau pateikiamas naujas optimalesnis koordinačių perskaičiavimo metodas, kuris nutraukia tarpinių taškų kelio paiešką pasiekus bent vieną segmento viršūnę ir geba rasti trumpiausią kelią tarp segmentų. Trečias pakeitimas – algoritmo pritaikymas 8-ių krypčių jungimo kvadratinio tinklo grafui.

Eksperimentinių tyrimu metu lyginami *A**, *Jump Point Search* (*JPS*) ir *CHPA** rezultatai atsitiktinai generuotuose, labirintų, kvadratinių kambarių bei strateginio žaidimo žemėlapiuose. Nustatyta, kad *CHPA** algoritmo patobulinimai gerina randamo kelio ilgį ir reikšmingai nepaveikia greitaveikos.

Teleiša, Marius. Analysis of 2D Convolution-based Methods for Finding Approximate Shortest Path. Master's Final Degree Project / supervisor prof. Dalia Čalnerytė; Faculty of Informatics, Kaunas University of Technology.

Study field and area (study field group): computer science, informatics (B01).

Keywords: pathfinding, hierarchical pathfinding, 2D convolution.

Kaunas, 2023. 69 p.

Summary

The paper reviews various methods for hierarchical pathfinding and path optimization. The knowledge gained from the analysis is applied to the design of improvements to the Convolutional Hierarchical Path *A** (*CHPA**) algorithm. The first change corrects abstraction layer errors by removing edges leading out of a segment, where the algorithm sees a path from the top and left segments, but there is no diagonal path across the segment. Second, a new more optimal method of coordinate translation is presented, which stops the search for intermediate paths when at least one segment vertex is reached and is able to find the shortest path between segments. Third, the algorithm is adapted to work on octile grids.

Experimental studies compare the results of *A**, Jump Point Search (*JPS*) and *CHPA** on randomly generated, maze, square room and strategy game maps. Results show, that the enhancements to the *CHPA** algorithm improve the path length and do not significantly affect the search time.

Turinys

Paveikslų sąrašas

Lentelių sąrašas

1. Įvadas

1.1. Problemos aktualumas

Žmonės, naudodamiesi automobiliais, juda erdvėje kiekvieną dieną ir kartais šiam procesui įvykdyti pasitelkia navigacijos sistemas. Navigacijos sistemos, naudodamos kelio paieškos algoritmus, suranda maršrutą, kurį nukeliauti užtruks mažiausiai laiko, ir gali būti taikomos žmonėms padėti judėti miesto gatvėse, robotams efektyviai keliauti po sandėlius, sudaryti maršrutus lėktuvams, išvengiant susidūrimų ir daugelyje kitų sričių. Realiame pasaulyje objektai dažnai keičia poziciją, todėl reikia, kad algoritmai sugebėtų sparčiai surasti maršrutą atnaujintoje erdvėje. Kartais norint pasiekti didesnį maršruto paieškos greitį galima atsisakyti optimalaus maršruto ieškojimo ir pasitenkinti maršrutu, artimu optimaliam.

Vienas plačiausiai žinomų kelio paieškos algoritmų yra *A**, kuris yra nesudėtingas ir jo veikimą galima keisti naudojant įvairias euristines funkcijas. Šios euristinės funkcijos skirtos nukreipti algoritmą į tikslo pusę, kad būtų sumažinta paieškos erdvė, tačiau kai kurios funkcijos gali sumažinti paieškos erdvę ir nebegarantuoti optimalaus kelio. Kompiuteriniuose žaidimuose ypač svarbi greita maršruto paieška, todėl neretai yra naudojami spartesni metodai, kurie negarantuoja surasti pačio optimaliausio kelio.

Sparti kelio paieška yra vis dar aktualus uždavinys, kuriam nuolat stengiamasi sukurti spartesnį sprendimą. Sukurti visiškai naują algoritmą dažnai yra sudėtinga ir pirminiai jų įgyvendinimai nėra tobuli, todėl kartais geriau yra pasirinkti esamą algoritmą ir mėginti jį tobulinti. Šiame darbe pasirinkta analizuoti ir patobulinti *CHPA** algoritmą, kuris šio darbo rašymo metu buvo neseniai sukurtas.

1.2. Projekto tikslas

Šio darbo tikslas - nustatyti *CHPA** algoritmo trūkumus ir atlikti patobulinimus gerinančius šio algoritmo stabilumą, randamo kelio ilgį bei greitaveiką. Šiam tikslui įgyvendinti iškelti šie uždaviniai:

1. atlikti panašių kelio paieškos algoritmų, jų metodikų ir patobulinimų analizę;
2. apibrėžti *CHPA** algoritmo trūkumus ir, remiantis analizės dalyje surinkta informacija, aprašyti siūlomus algoritmo trūkumų sprendimus;
3. realizuoti pasiūlytus sprendimus ir reikiamus kelio paieškos algoritmus;
4. atlikti originalios ir patobulintos *CHPA** algoritmo versijos bei įgyvendintų kelio paieškos algoritmų eksperimentinius tyrimus, apibrėžti jų rezultatus.

2. Kelio paieškos algoritmų analizė

Analizės skyriuje apibrėžiama pasirinktos temos problematika, analizuojami kelio paieškos algoritmai ir *CHPA** algoritmo trūkumai, aprašomi siūlomi sprendimai, jų pagrįstumas ir realizavimo galimybės.

2.1. Kelio paieškos sprendimų apžvalga

2.1.1. Kelio paieškos uždavinys

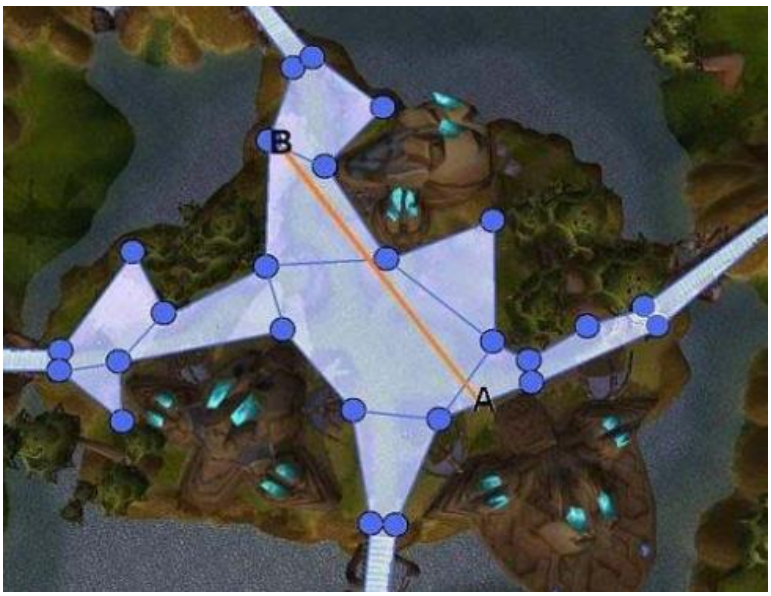
Kelio paieškos algoritmai skirti spręsti kelio radimo tarp dviejų taškų ir optimalaus kelio radimo uždavinius. Kelio optimalumas gali būti apibrėžtas pagal įvairius kriterijus: keliaujamas atstumas, laikas, greitis, kelio tiesumas ir t. t. Automobilių navigacijos sistemos ieško maršruto, kurio kelionė trunka mažiausiai laiko, o toks maršrutas ne visada sutampa su mažiausio atstumo maršrutu, kadangi gatvėse susidariusios spūstys gali ilginti kelionės laiką.

Kelio radimas tarp dviejų taškų yra trivialus uždavinys ir tai galima atlikti pasinaudojus paprastu algoritmu, tokiu kaip paieškos į plotį (angl. *breadth-first search*) (toliau *BFS*) metodas, kurio veikimo greitis yra tiesinis [1]. Šio uždavinio sprendimas užtikrina, kad kelias egzistuoja tarp nurodytų taškų, tačiau negarantuoja, kad rastas kelias bus optimalus.

Optimalaus kelio radimo uždavinys yra daug sudėtingesnis ir ypač aktualus norint efektyviai judėti erdvėje. Šio uždavinio sprendimui naudojamo *Dijkstra* algoritmo veikimo greitis yra logaritminis [2], todėl platesnės ieškojimo erdvės sparčiai didina optimalaus kelio paieškos laiką. Optimalaus maršruto radimui sukurta daug įvairių algoritmų ir renkantis algoritmą reikia atsižvelgti į uždaviniui taikomas sąlygas:

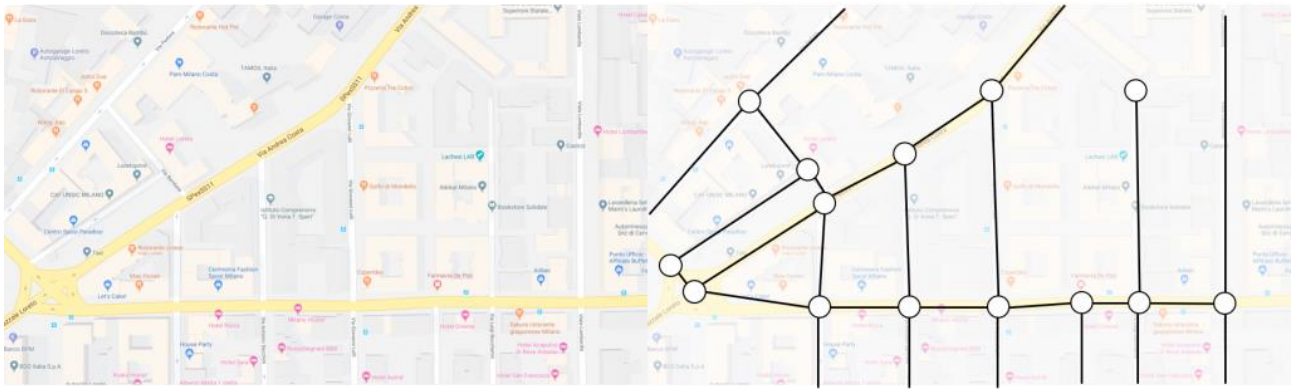
- paieškos erdvės struktūra;
- erdvės navigacijos taisyklės;
- ieškomo kelio optimalumo kriterijai.

Kelio paieškos erdvės tipas yra vienas svarbiausių bruožų apibrėžiančių skirtingas kelio radimo problemas ir galimus sprendimus. Moderniuose vaizdo žaidimuose veiksmas dažniausiai vyksta 3D tęstinėse erdvėse ir kelio paieškos radimui suformuojamas navigacijos tinklelis (angl. *NavMesh* arba *Navigation mesh*) [3,4,5]. Navigacijos tinklelis yra sudarytas iš išgaubtų poligonų [6], kurie suformuojami pagal kliūčių nelygumus, ir nusako atviras erdvės sekcijas (žr. **1 pav.**). Šis metodas paremtas idėja, kad trumpiausias kelias aplink kliūtį bus tarp kliūties išsikišimų, todėl galima judėti tarp šių tarpinių taškų tol, kol bus tiesiogiai matomas tikslas ir taip bus smarkiai sumažinama paieškos erdvė bei sparčiai randamas kelias, kurio ilgis artimas optimaliam.



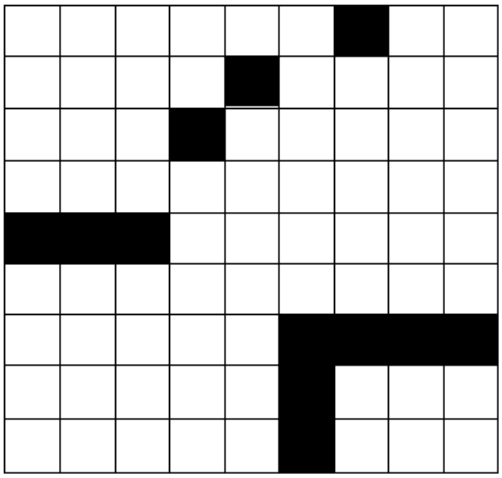
1 pav. Kelio radimas navigacijos tinklelyje tarp dviejų tikro žemėlapio taškų, kur A - pradžios taškas, B - pabaigos [6]

Diskrečios paieškos erdvės gali būti atvaizduojamos grafais [7,8,9], kurie yra viršūnių ir jas jungiančių kraštinių rinkinys [10]. Vienas realaus pasaulio grafų taikymo pavyzdys yra GPS navigacijos sistemos. **2 pav.** pavaizduotas grafas sudarytas naudojant sankryžas kaip viršūnes ir kelius kaip kraštines. Deja, tokia paieškos erdvė neturi kliūčių poligonų ir turi imtis kitokių būdų sumažinti informacijos kiekį.



2 pav. GPS navigacijos sistemos žemėlapio atvaizdavimas grafu [11]

Vaizdo žaidimų diskrečioms erdvėms taip pat naudojami grafai ir jie dažniausiai turi geometrinės struktūros bruožų [6,12]. Klasikinis pavyzdys yra kvadratinio tinklelio grafas, kuris pavaizduotas **3 pav.**

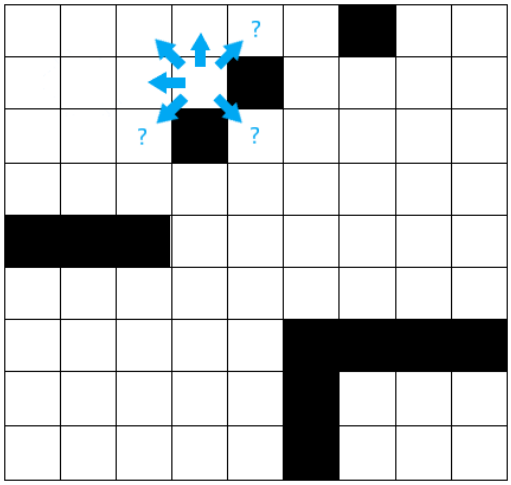


3 pav. 9x9 kvadratinio tinklo struktūros žemėlapis

Šioje erdvėje balti langeliai yra laisvi langeliai, o juodi - kliūtys. 4-ių krypčių jungimo kvadratinio tinklo grafo kraštinių kiekį galima paskaičiuoti pasinaudojus formule:

$$E = (m-1)n + (n-1)m; \tag{2.1}$$

čia *m* ir *n* yra atitinkamai eilučių ir stulpelių kiekis. Pagal šią formulę 9x9 žemėlapyje gautume grafą su 144 kraštinėmis. **2 pav.** pavaizduotame miesto dalies grafe tik 5 iš 14 viršūnių turi 4 kraštines, todėl kvadratinio tinklo grafas dažniausiai turės didesnę paieškos erdvę. Kompiuteriniuose žaidimuose neretai taikomas 8 krypčių jungimas, todėl kraštinių skaičius dar smarkiau išauga. Kai kurie kelio paieškos algoritmai veikia būtent tik su tam tikru grafo jungimu, todėl šiame darbe bus daugiausiai kalbama apie 8 krypčių jungimą.



4 pav. 8-ių krypčių grafų jungimo dilemos

Naudojant šį jungimo būdą taip pat reikia apibrėžti kaip kliūčių langeliai paveikia gretimų laisvų langelių jungimą ir **4 pav.** pavaizduota būtent ši dilema. Viena iš galimų taisyklių nusprendžia ar galima sujungti įstrižai esančius langelius jei abu langeliai turi vieną bendrą kliūtį kardinalioje kryptyje (šiaurė, pietūs, rytai, vakarai). Jungimo taisyklės smarkiai nekeičia grafo paieškos erdvės, tačiau kartais jų negalima pasirinkti laisva valia, kadangi naudojamas paieškos algoritmas gali būti sukurtas pagal griežtas autorių apibrėžtas taisykles.

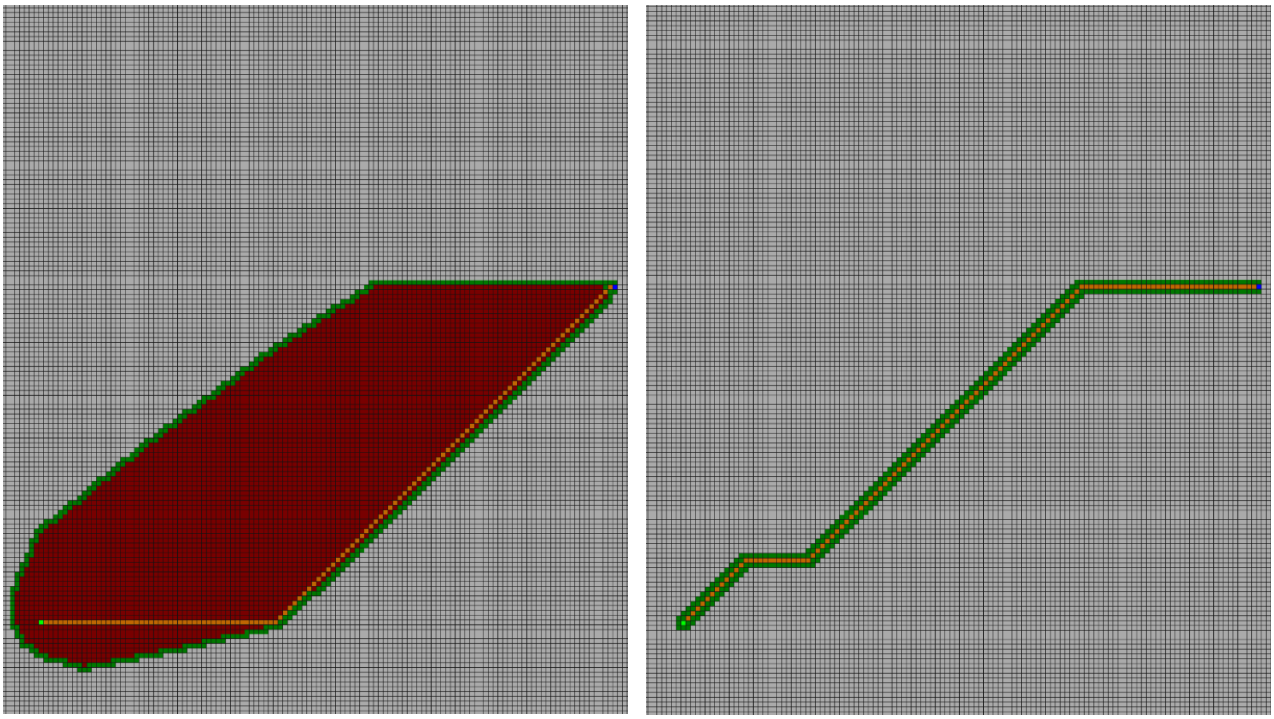
2.1.2. Euristinės funkcijos

Tam tikri kelio paieškos algoritmai, tokie kaip *JPS*, *A** ir juo paremti metodai, naudoja euristines funkcijas atliekant informuotą paiešką [13]. Naudojant priimtiną euristinę funkciją (angl. *admissible heuristic function*) kelio paieškos algoritmai gali informuotai nukreipti kelio paiešką, sumažinti apieškomą erdvę ir garantuoti rasti optimalų kelią.

Euristinės funkcijos dažniausiai naudoja išskirtinę informaciją apie aplinką, kad būtų pagerintas skaičiavimų efektyvumas [14], todėl skirtingo tipo žemėlapiams reikia įvairių euristinių funkcijų. Euklidinis atstumas nusako tiesės tarp dviejų taškų atstumą, gali būti naudojama kaip euristinė funkcija. Euklidinis atstumas tarp taškų $A = (x_A; y_A)$ ir $B = (x_B; y_B)$ gali būti apskaičiuotas pagal (2.2) formulę [15]:

$$d(\boldsymbol{A}, \boldsymbol{B}) = C\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}; \tag{2.2}$$

čia *C* yra atstumas tarp gretimų kardinalios krypties viršūnių. Ši Euklidinio atstumo euristinė funkcija garantuos optimalų kelią, tačiau patikrins didesnę erdvę nei būtina [16], kadangi euristinė funkcija dažniausiai suskaičiuos trumpesnį atstumą nei įmanoma kvadratinio tinklo grafe. Euklidinio atstumo tyrinėjamos erdvės problema 8-ių krypčių žemėlapyje pavaizduota **5 pav.**



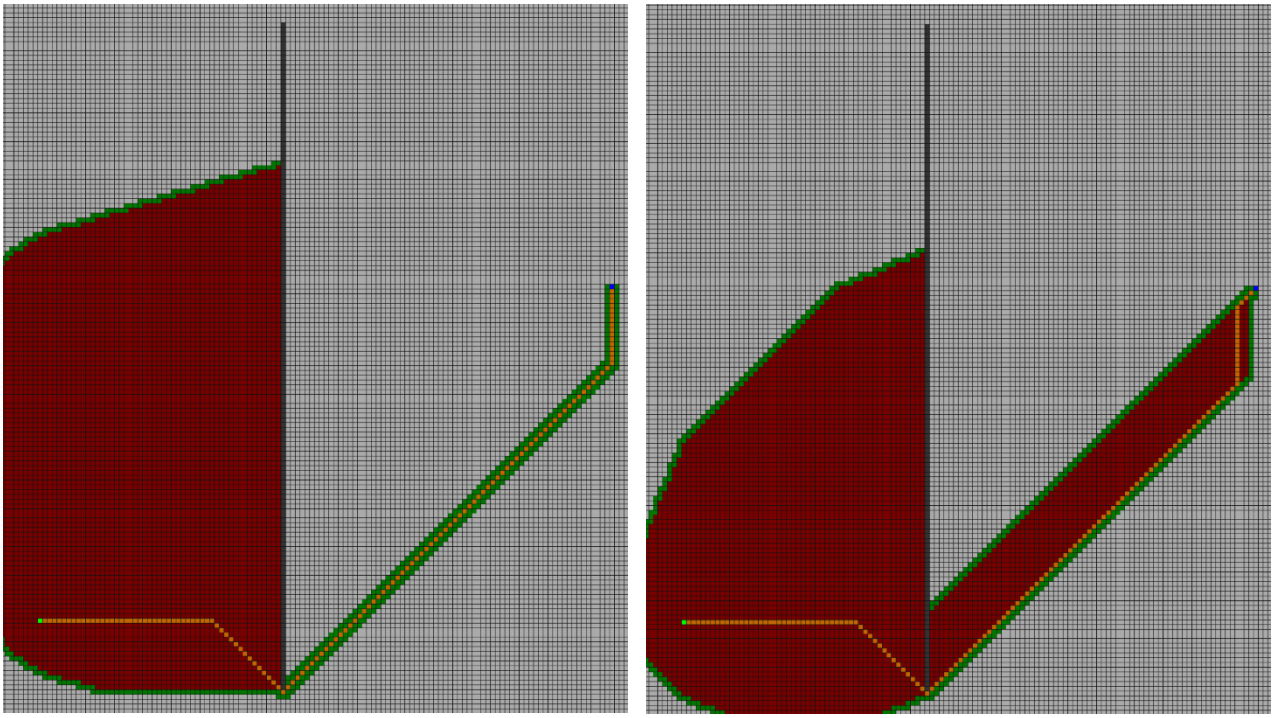
a) b)

5 pav. *A** algoritmo tyrinėjama erdvė 8-ių krypčių jungimo žemėlapyje, naudojant a) Euklidinio atstumo ir b) įstrižo atstumo euristines funkcijas , kur pilka spalva žymima laisva erdvė, šviesiai žalia – pradžios taškas, mėlyna – pabaigos taškas, oranžinė – rastas kelias, raudona – apžiūrėta viršūnė, tamsiai žalia – potenciali viršūnė tikrinimui

4-ių krypčių jungimo kvadratinio tinklo grafui tinkamesnė Manheteno atstumo euristinė funkcija, kuri skaičiuojama pagal (2.3) formulę [17]:

$$d(A, \boldsymbol{B}) = C(|x_A - x_B| + |y_A - y_B|). \quad (2.3)$$

Ši euristinė funkcija yra priimtina 4-ių krypčių jungimo grafe t. y. visada suskaičiuos optimistišką trumpiausią kelią. 8-ių krypčių jungimo grafas leidžia keliauti įstrižai trumpesniu atstumu nei būtų galima kardinaliomis kryptis 4-ių krypčių jungimo grafe, todėl Manheteno atstumo funkcija pervertins atstumą ir padidins tyrinėjamą erdvę, kai algoritmui reikės apeiti kliūtį, kaip pavaizduota **6 pav.** Šio jungimo paieškos erdvei reikia taikyti skirtingą euristinę funkciją.



a) b)

6 pav. *A** algoritmo tyrinėjama erdvė 8-ių krypčių jungimo žemėlapyje, naudojant a) Manheteno atstumo ir b) įstrižo atstumo euristines funkcijas , kur pilka spalva žymima laisva erdvė, šviesiai žalia – pradžios taškas, mėlyna – pabaigos taškas, oranžinė – rastas kelias, raudona – apžiūrėta viršūnė, tamsiai žalia – potenciali viršūnė tikrinimui

Įstrižo atstumo (angl. *octile* arba *diagonal distance*) euristinė funkcija yra panaši į Manheteno atstumo funkciją, pritaikyta 8-ių krypčių jungimo grafui ir skaičiuojama pagal (2.4) formulę [17]:

$$d(\boldsymbol{A}, \boldsymbol{B}) = C(|x_A - x_B| + |y_A - y_B|) + (D - 2C)Min(|x_A - x_B|, |y_A - y_B|); \quad (2.4)$$

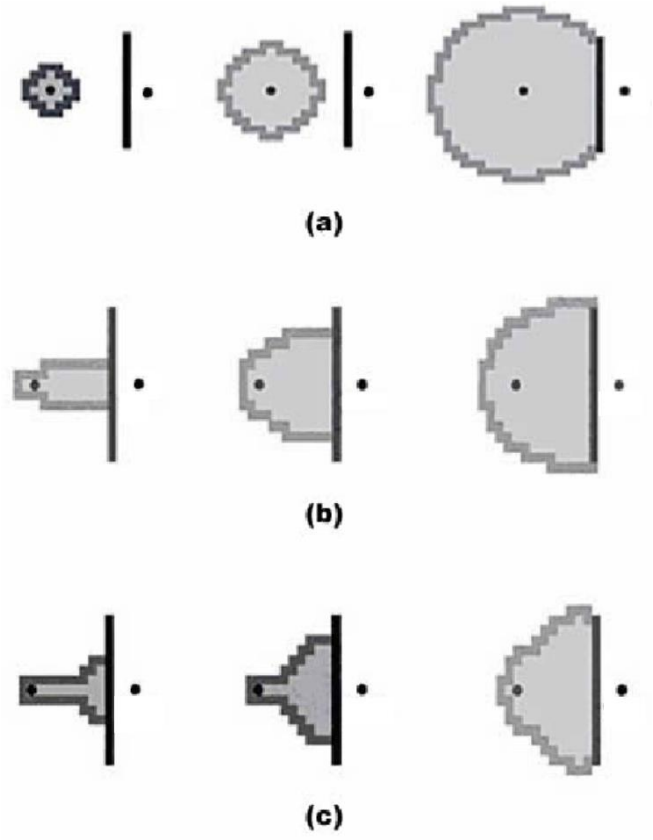
čia *D* yra atstumas tarp gretimų įstrižos krypties viršūnių.

Šios trys euristinės funkcijos yra paprastos ir dažniausiai naudojamos kvadratinio tinklo grafų kelio paieškos uždavinių sprendime [18,19].

2.1.3. *A** algoritmas

*A**, sukurtas 1968 m. [20], yra geriausias-pirmiausia (angl. *best-first*) kelio paieškos algoritmas galintis garantuoti randamo kelio optimalumą [14] bei vienas žinomiausių ir efektyviausių kelio paieškos algoritmų [21]. Šis algoritmas naudoja euristines funkcijas apibrėžti tyrinėjamų viršūnių optimalumą, kad algoritmas galėtų atlikti informuotus sprendimus ir minimizuoti tyrinėjamą erdvę.

Būtent euristinės funkcijos nusako ar *A** randamas kelias bus optimalus ir kaip atrodys algoritmo tyrinėta erdvė. **7 pav.** pavaizduota, kaip *A** kelio paieškos algoritmui, naudojant įvairias euristines funkcijas, susidūrus su kliūtimi smarkiai didėja paieškos erdvė.



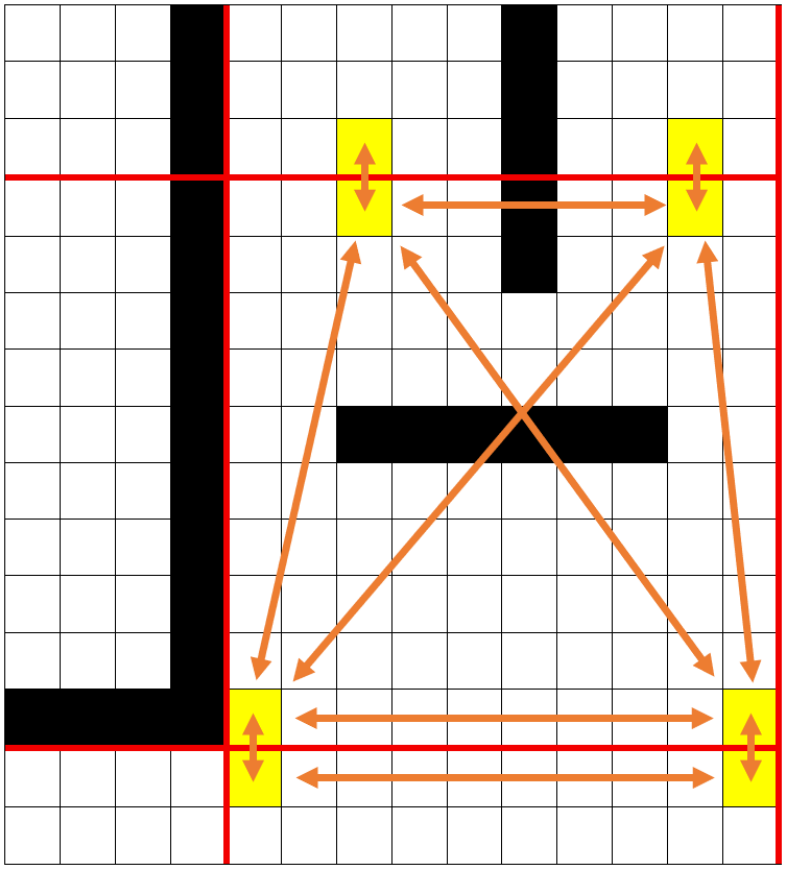
7 pav. Euristinių funkcijų palyginimas: a) nulinė euristinė funkcija,
b) Euklidinis atstumas, c) pervertinanti euristinė funkcija [6]

Kiekviena iteracija *A** prideda viršūnę, kuri yra parenkama pagal euristinę funkciją, į esamų kelių sąrašą. Šis sąrašas turi būti surūšiuotas, o tai yra brangi operacija bei viena didžiausių *A** algoritmo greitaveikos problemų.

2.1.4. *HPA** algoritmas

*Hierarchical Pathfinding A** (toliau *HPA**) hierarchinis kelio paieškos algoritmas ir tai buvo pirmoji hierarchinio sprendimo taikymo kelio paieškai vaizdo žaidimuose panaudojimas [22]. Šis algoritmas naudoja kvadratinio tinklo geometrinę struktūrą ir apdorojimo metu suskaido tikrą žemėlapį į fiksuoto dydžio klasterius (angl. *clusters*). Hierarchinio sprendimo tikslas yra sumažinti sprendimo erdvę ir taip paspartinti kelio radimą.

*HPA** algoritmo kontekste įėjimas apibrėžiamas kaip bendros klasterių kraštinės segmentas, neturintis kliūčių ir yra sudarytas iš dviejų grafo viršūnių, po vieną kiekviename iš klasterių. Jeigu įėjimo segmentas trumpesnis nei nurodyta konstanta (šiuo atveju 6) – segmento centre sukuriamas vienas perėjimas tarp klasterių, o jeigu ilgesnis – du perėjimai segmento kraštuose. Apdorojimo metu klasteryje tarp visų nustatytų perėjimų viršūnių atliekama paieška ir atitinkamai sukuriamos kraštinės abstrakcijos sluoksnyje. *HPA** abstrakcijos sluoksnio formavimo pavyzdys pavaizduotas **8 pav.**

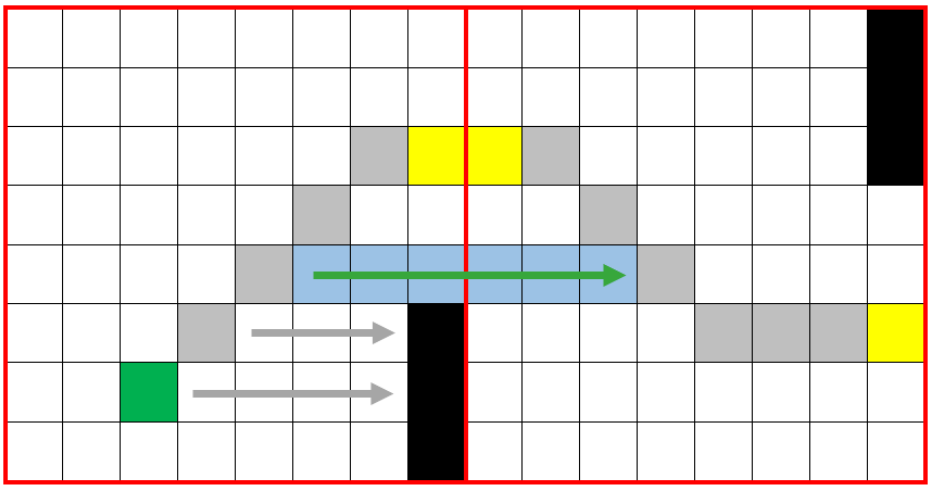


8 pav. *HPA** įėjimų tarp klasterių formavimas, kur raudoni segmentai – klasteriai, geltoni langeliai – perėjimo viršūnės, oranžinės rodyklės – perėjimo viršūnių kraštinės

*HPA** kelio paieškos pradžioje turi atlikti paiešką tarp pradžios taško ir visų tame klasteryje esančių perėjimų. Toliau sukuriamas pradžios taškas abstrakcijos sluoksnyje ir sujungiamas su greičiausiai pasiekiamu klasterio perėjimo tašku. Atitinkamas procesas kartojamas pabaigos taškui. Didinant klasterius mažinamas skaičiavimo laikas atliekant paiešką abstrakčiame sluoksnyje, tačiau didinamas skaičiavimo laikas įterpiant pradžios ir galo taškus į abstrakcijos sluoksnį.

Kintančioje aplinkoje abstrakcijos sluoksnis taip pat sukuriamas prieš paiešką, tačiau pasikeitus aplinkai reikia perskaičiuoti paveiktus klasterius ir atnaujinti abstrakcijos sluoksnį. Klasterių perskaičiavimas yra skaičiavimo laiko prasme brangi operacija, o didėjantys klasteriai didina ir skaičiavimo laiką, todėl kintančioje aplinkoje reikia pasverti kelio paieškos greitaveikos spartinimą ir perskaičiavimo lėtėjimą.

*HPA** algoritmas vidutiniškai randa iki 10 % ilgesnį kelią nei optimalus. Klasterio perėjimai nėra visada optimaliausioje pozicijoje ir būtent dėl to didėja randamo kelio ilgis, todėl *HPA** algoritmui pridėta kelio lyginimo (angl. *Path smoothing*) fazė kaip paskutinis žingsnis kelio paieškos procese. Pritaikius kelio lyginimą, *HPA** algoritmas vidutiniškai randa iki 1 % ilgesnį kelią nei optimalus. Autorių kelio lyginimo metodas yra gana paprastas – pradedant nuo pradinės viršūnės, kardinaliomis kryptimis tikrinama ar keliaujant tiesiai galima pasiekti kitą abstraktaus kelio viršūnę. Pasiekus kitą kelio viršūnę, neoptimalus kelias tarp šių viršūnių pakeičiamas tiesaus kelio viršūnėmis. Šio kelio lyginimo pavyzdys pateiktas **9 pav.**

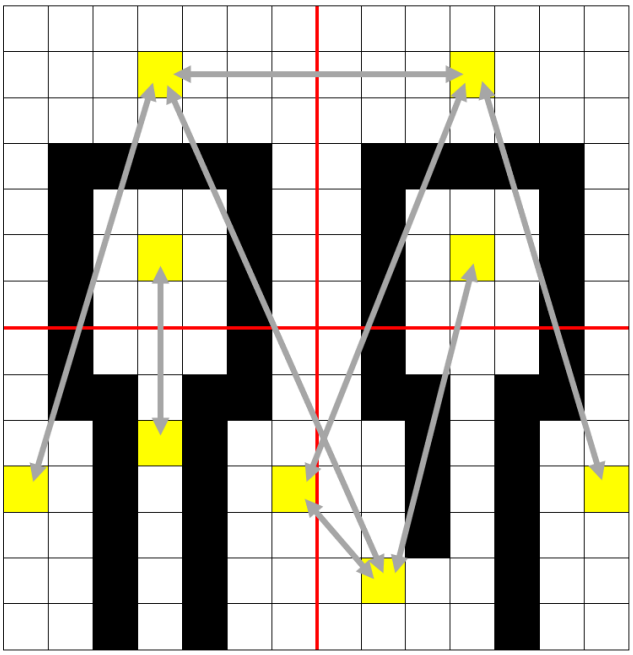


9 pav. *HPA** rasto kelio tiesinimo proceso pavyzdys, kur žalias langelis yra pradžios taškas, geltonas – klasterio perėjimas, mėlynas – lyginimo proceso rastas trumpesnis kelias

2.1.5. Minimalios atminties abstrakcijos metodas

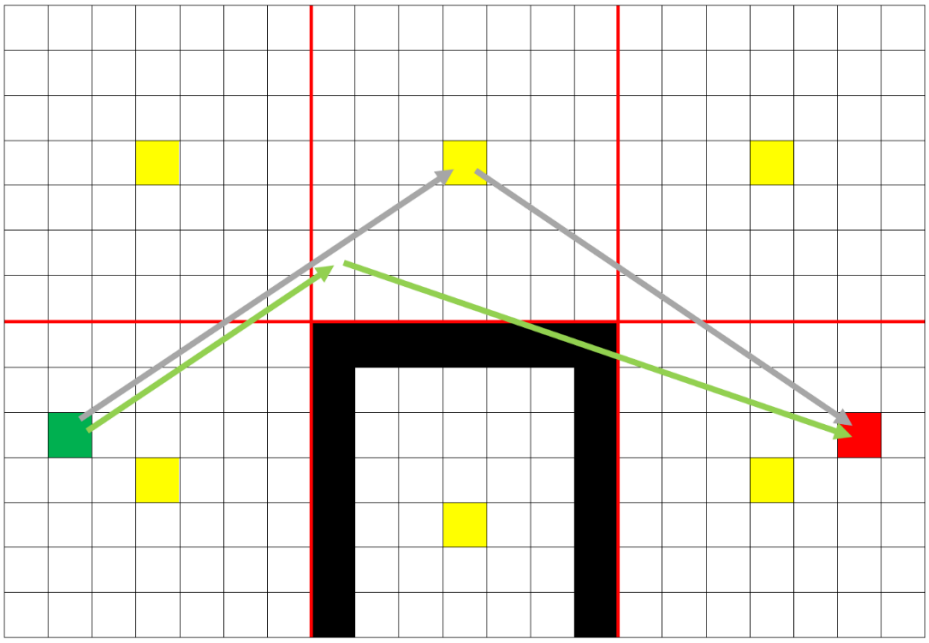
Minimalios atminties (angl. *minimal-memory*) (toliau *MM*) abstrakcijos metodas pristatytas 2007 m. ir sukurtas siekiant paspartinti kelio paiešką atminties apribotose aplinkose. Šis metodas demonstravo apie 3 % didesnį atminties naudojimą ir iki 100 kartų greitesnę kelio paiešką nei *A** algoritmas [23]. *MM* metodas yra išbaigtas, gali veikti ir 4-ių, ir 8-ių krypčių jungimo grafuose, tačiau negarantuoja optimalaus kelio radimo.

MM metodas abstrakcijos sluoksnio kūrimui, panašiai kaip *HPA**, pirma žemėlapį suskaido į klasterius, kurių dydis yra iš anksto apibrėžtas ir nekinta. Toliau, šie klasteriai yra paskirstomi į unikalius regionus naudojant *BFS* ir kiekvienam iš regionų priskiriamas centrinis taškas, kuris bus naudojamas kaip viršūnė abstrakcijos sluoksnyje. Galiausiai gretimų regionų centrinės viršūnės yra sujungiamos ir abstrakcijos sluoksnio formavimo procesas yra baigiamas. Pavyzdinis *MM* metodo sukurtas abstrakcijos sluoksnis pavaizduotas **10 pav.**



10 pav. *MM* metodo sukuriamo abstrakcijos sluoksnio pavyzdys, kur klasteriai atskirti raudonomis linijomis, juodi langeliai yra kliūtys, geltoni – abstrakcijos sluoksnio viršūnės, pilkos rodyklės – abstrakcijos sluoksnio kraštinės

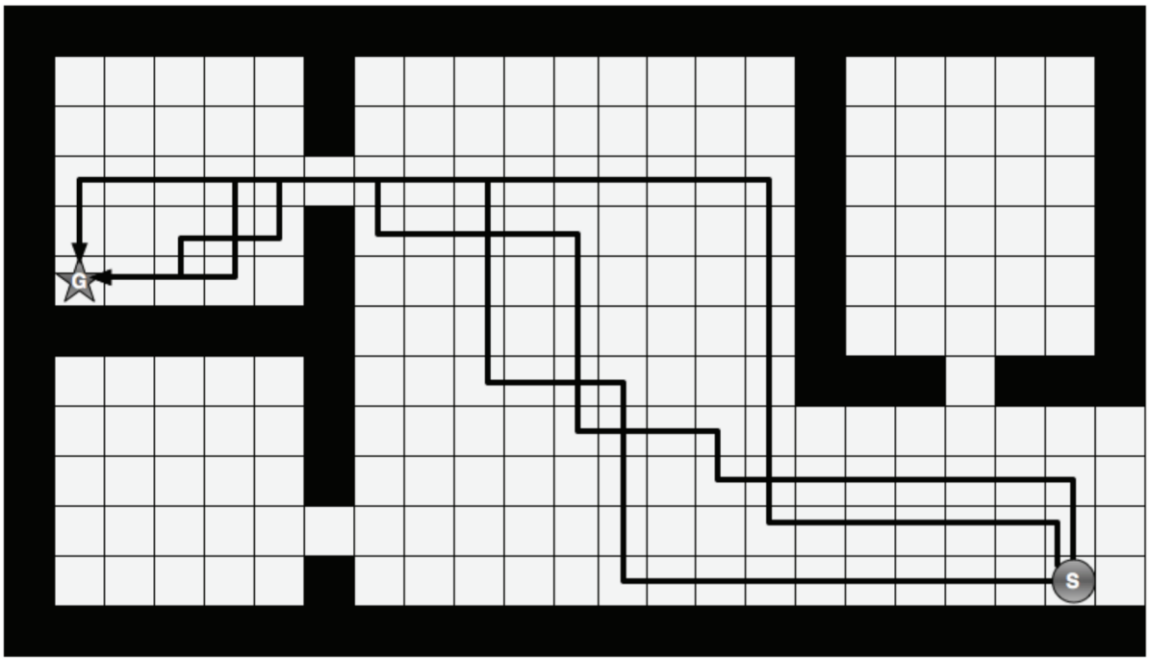
Kelio paieška pirma atliekama abstrakcijos sluoksnyje ir vėliau tarp abstrakcijos sluoksnyje rasto kelio taškų. Kelias tarp regionų centrinių taškų nebūtinai bus optimalus visos kelio paieškos kontekste, todėl papildomai gali būti taikomas kelio lyginimo procesas. *MM* metodo autorių siūlomas kelio lyginimo metodas nuo galo panaikina dalį rasto kelio tarp tarpinių taškų. Panaikinus 10 % rasto kelio tarp tarpinių taškų, galutinių kelių vidutinė optimalumo paklaida svyravo tarp 4-18 %. Šio kelio lyginimo pavyzdys pateiktas **11 pav.**



11 pav. *MM* metodo kelio lyginimo metodo pavyzdys , kur klasteriai atskirti raudonomis linijomis, juodi langeliai yra kliūtys, geltoni – abstrakcijos sluoksnio viršūnės, pilkos rodyklės – abstrakcijos sluoksnio kraštinės, žalios rodyklės – trumpesnis kelias po lyginimo taikymo

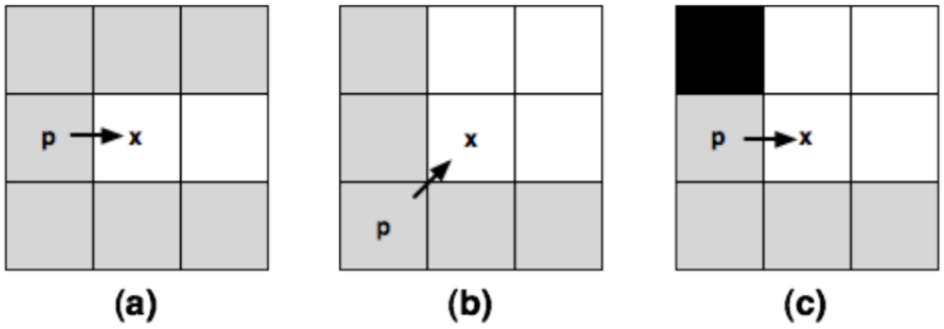
2.1.6. *JPS* algoritmas

Jump Point Search (toliau *JPS*) algoritmas buvo sukurtas 2011 m. *Daniel Harabor* ir *Alban Grastien* [24]. Šis algoritmas atlieka simetrijos naikinimą (angl. *Symmetry breaking*), kad būtų sumažintas tikrinamų viršūnių skaičius ir paspartinta paieška kvadratinio tinklo grafuose. Kelių simetrija yra problema kvadratinio tinklo grafuose, kadangi tokioje erdvėje gali egzistuoti daug trumpiausių kelių tarp dviejų taškų, o kelio paieškos algoritmas norėdamas rasti optimalų kelią turi įvertinti visus lygiaverčius kelius. Simetrinių kelių kiekis taip pat sparčiai didėja su didelėmis paieškos erdvėmis. Kelių simetrija pavaizduota **12 pav.**, kur galima pamatyti kelis iš daugelio skirtingų trumpiausių kelių, kurių sudaryto kelio ilgis yra vienodas.



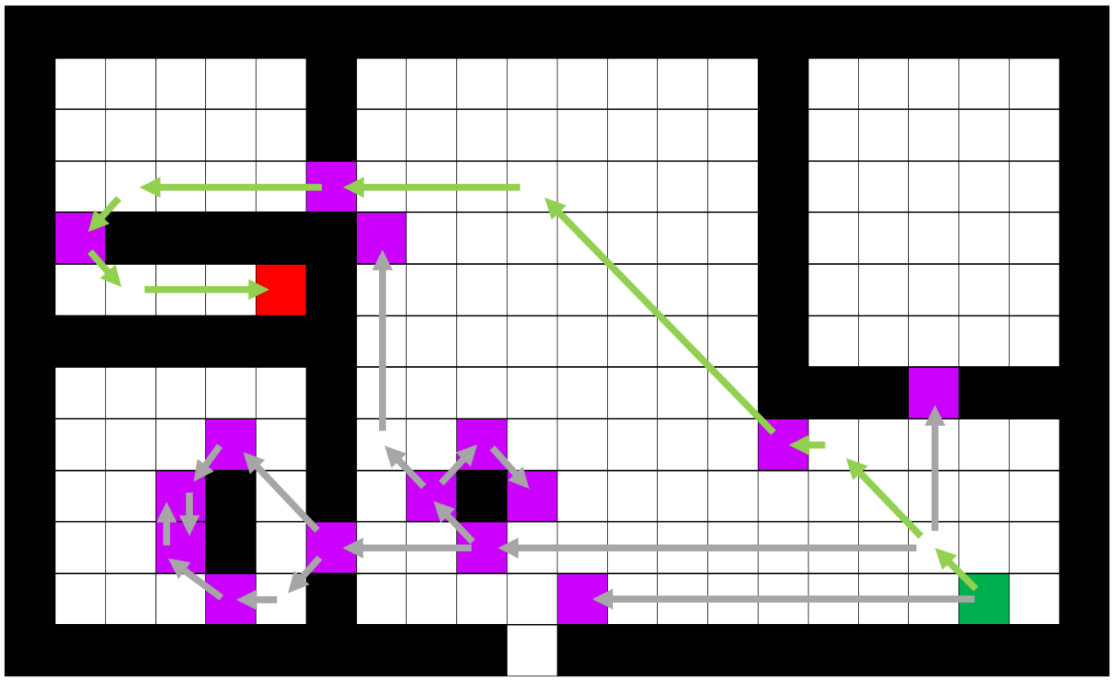
12 pav. Keletas vienodo ilgio kelių žemėlapyje [25]

Originali *JPS* algoritmo versija tiesiog pritaiko dvi paprastas taisykles viršūnių skaičiaus mažinimui: viena skirta tiesiems žingsniams, kita – skersiniams. Šių taisyklių idėja yra atmesti artimiausių kaimynų (viršūnių) tikrinimą įrodant, kad egzistuoja optimalus kelias tarp tėvinės viršūnės ir visų dabartinės viršūnės kaimynų, kuriam nepriklauso dabartinė viršūnė [26]. *JPS* taisyklių taikymo pavyzdys pavaizduotas **13 pav.**, kur keliaujant iš taško *p* į tašką *x* visi pilki langeliai yra atmetami, kadangi juos optimaliai galima pasiekti neaplankius taško *x*. Trečiame pavyzdyje viršutiniai taškai nėra atmetami, nes juodas kliūties taškas verčia optimaliam keliui įtraukti tašką *x*, todėl taškas *p* yra įdomus ir įtraukiamas į šuolio taškų sąrašą.



13 pav. *JPS* pilkų kaimyninių viršūnių naikinimas judant iš *p* į *x*, kur juodas langelis yra kliūtis, baltas - laisvas: (a) judėjimas kardinalia kryptimi, (b) judėjimas įstrižai, (c) judėjimas kardinalia kryptimi šalia kliūties [26]

Šios dvi *JPS* algoritmo taisyklės sumažina viršūnių skaičių atvirajame sąraše (angl. *open list*), o tai smarkiai didina algoritmo veikimo greitį, kadangi elemento paieška sąraše yra sąlyginai lėta operacija. Pritaikius šias taisykles sukurtas *JPS* veikimo pavyzdys, kuris pavaizduotas **14 pav.**



14 pav. *JPS* kelio paieškos pavyzdys, kur žalias langelis yra kelio paieškos pradžia, raudonas – pabaiga, violetinis – šuolio taškas, pilkos rodyklės – kelias tarp šuolio taškų, žalios rodyklės – rastas optimalus kelias

JPS algoritmo autoriai 2014 m. pasiūlė kelis patobulinimus algoritmui:

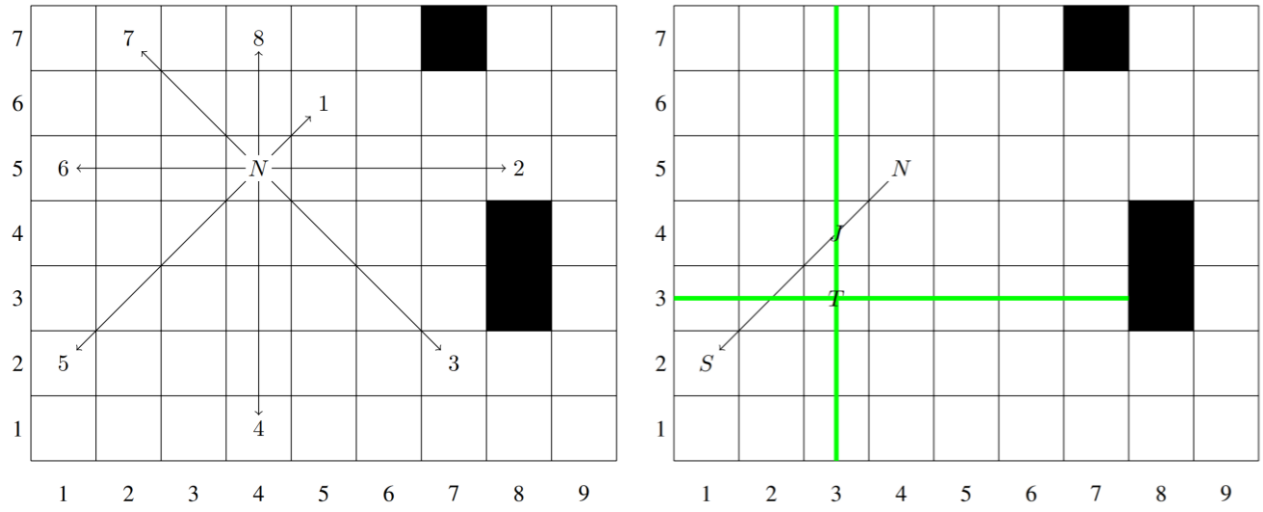
1. blokais paremtą simetrijos naikinimą (angl. *block-based simmetry breaking*);
2. žemėlapio apdorojimą prieš paiešką (angl. *offline pre-processing*);
3. šuolio taškų naikinimą (angl. *jump point pruning*).

Pirmosios *JPS* versijos simetrijos naikinimo metodas rekursyviai apžiūrėdavo po vieną viršūnę ir jos kaimynus, tačiau taip yra kelis kartus patikrinami tie patys kaimynai. **15 pav.** pavaizduotas patobulinto metodo veikimas, kuris tikrina iš anksto nustatytą skaičių viršūnių. Šiame pavyzdyje *x* yra dabartinė viršūnė, kurios atžvilgiu surenkama trijų eilių žemėlapio informacija į tris dvejetainio tipo reprezentacijas. Toliau įgyvendinamos tos pačios simetrijos naikinimo taisyklės naudojant

specialias *CPU* operacijas pirmojo vieneto pozicijos suradimui. Žinoma, šį simetrijos naikinimo metodą galima paspartinti tikrinant ilgesnius žemėlapio ruožus, tačiau reikia atsižvelgti į *CPU* galimybes.

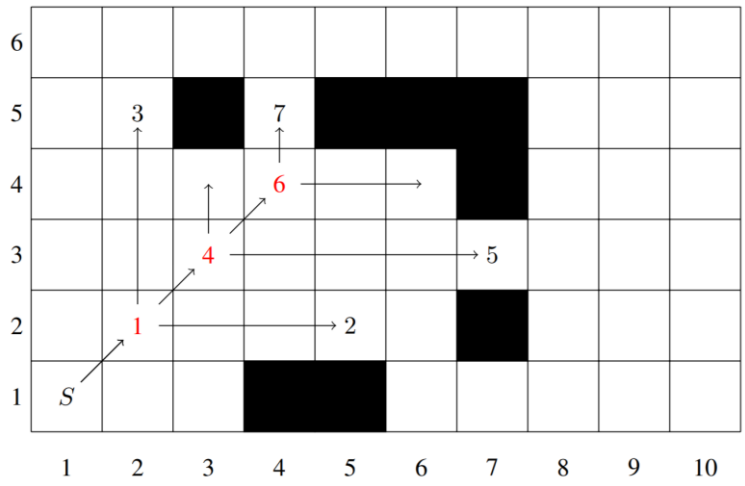
15 pav. Patobulintas *JPS+* simetrijos naikinimas, kai tikrinamas iš anksto apibrėžtas viršūnių skaičius, kur *x* – pradžios taškas, *y* – šuolio taškas, juodas langelis – kliūtis, mėlynas segmentas – šuolio kryptis, geltoni segmentai – tikrinama erdvė šuolio sudarymui

Antras patobulinimas atlieka skaičiavimus prieš paiešką. Kiekvienai laisvai viršūnei yra iš anksto apskaičiuojami pirmieji šuolio taškai ir aklavietės visomis kryptimis. **16 pav.** pavyzdyje a) pavaizduoti viršūnei *N* apskaičiuoti pirmieji šuolio taškai, kurie yra pažymėti skaičiais 1-3 ir aklavietės 4-8. Paieškos metu galutinis tikslas gali būti peršoktas naudojant šiuos šuolio taškus, todėl pavyzdyje b) yra sukuriamas tarpinis šuolio taškas *J* galutinės viršūnės vertikalios tiesės ir šuolio *N* → *S* sankirtoje. Šis metodas naudoja daugiau atminties, kad paspartintų paiešką, tačiau jį pritaikyti kintančiai aplinkai yra sudėtinga.



16 pav. *JPS+* algoritmo šuolio taškų apskaičiavimo prieš paiešką pavyzdys [27]

Trečiasis algoritmo patobulinimas spartina paiešką naikinant nereikalingus šuolio taškus. Skersai ieškant šuolio taško gali būti sukurtas šuolio taškas, kuris neturi gretimų kliūčių ir yra skirtas sujungti kitus du šuolio taškus. **17 pav.** raudona spalva pavaizduotos viršūnės, kurias panaikina *JPS+* algoritmas.



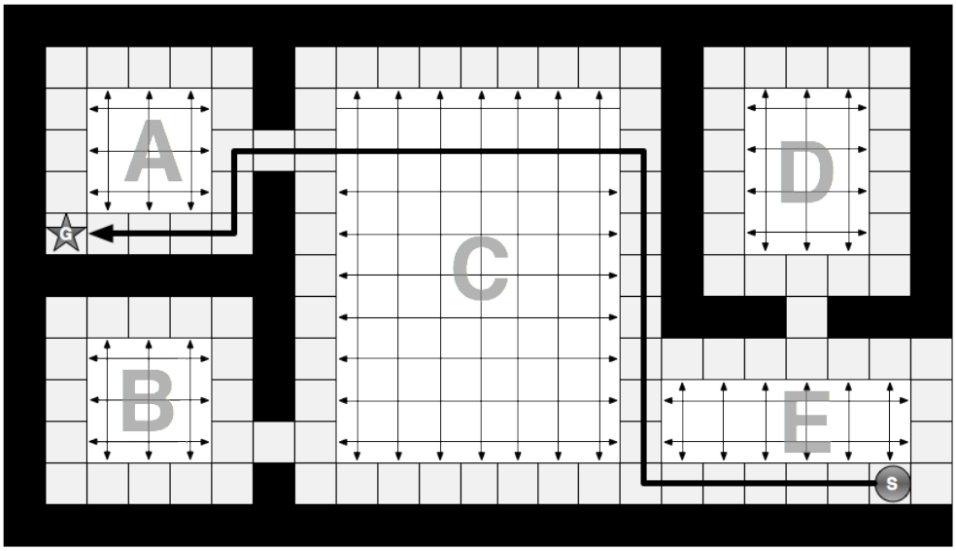
17 pav. *JPS+* šuolio taškų naikinimo pavyzdys, kai raudona spalva pažymėti šuolio taškai yra panaikinami [26]

Norint pasiekti šias atskirtas viršūnes užtenka nueiti kelią tarp jų teikiant prioritetą skersiniams žingsniams. Straipsnyje autoriai įrodė, kad šių šuolio taškų panaikinimas nedaro įtakos randamo kelio ilgiui ir paspartina kelio paiešką.

2.1.7. Stačiakampių kambarių dekompozicija

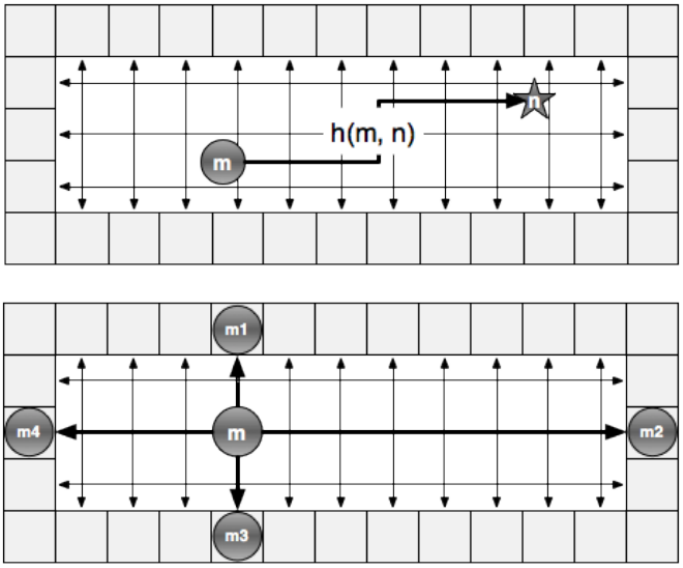
Stačiakampių kambarių apdorojimo būdas pirmą kartą aprašytas 2010 m. ir yra taikomas 4-ių krypčių jungimo kvadratinio tinklo grafuose [25]. Šis metodas mažina paieškos erdvę ir naikina kelių simetriją, o tai spartina kelio paieškos algoritmo greitaveiką.

Kambarių apdorojimo metodas paremtas idėja, kad stačiakampį kambarį, kuris neturi papildomų kliūčių, visada galima praeiti naudojant tik perimetro langelius. Žinant tai, šis metodas panaikina visus kambaryje esančius langelius, išskyrus perimetro, o tai panaikina didžiąją dalį simetrinių kelių. Priešingose kambario pusėse esantys langeliai yra sujungiami makro-kraštinėmis (angl. *macro edges*) naudojant kainą, nustatytą pagal Manheteno atstumą. **18 pav.** pateiktas kambarių dekompozicijos metodo grafo ir kelio paieškos pavyzdys.



18 pav. Kelio paieškos rezultatas stačiakampių kambarių dekompozicijos metodo apdorotame žemėlapyje, kur *S* yra pradžios taškas ir *G* yra pabaigos taškas [25]

Atliekant kelio paiešką, pradžios ir pabaigos viršūnės gali būti panaikintose žemėlapio vietose. Jei abi viršūnės yra tame pačiame kambaryje, užtenka atlikti paprastą kelio paiešką tarp jų, kadangi kambaryje garantuojama, kad nebus kliūčių. Jei viršūnės yra skirtinguose kambariuose, jos yra laikinai prijungiamos prie artimiausių kiekvienos kambario sienos viršūnių, o toliau kelio paieška veikia įprastai. Šios prijungimo logikos pavyzdys pateiktas **19 pav.**



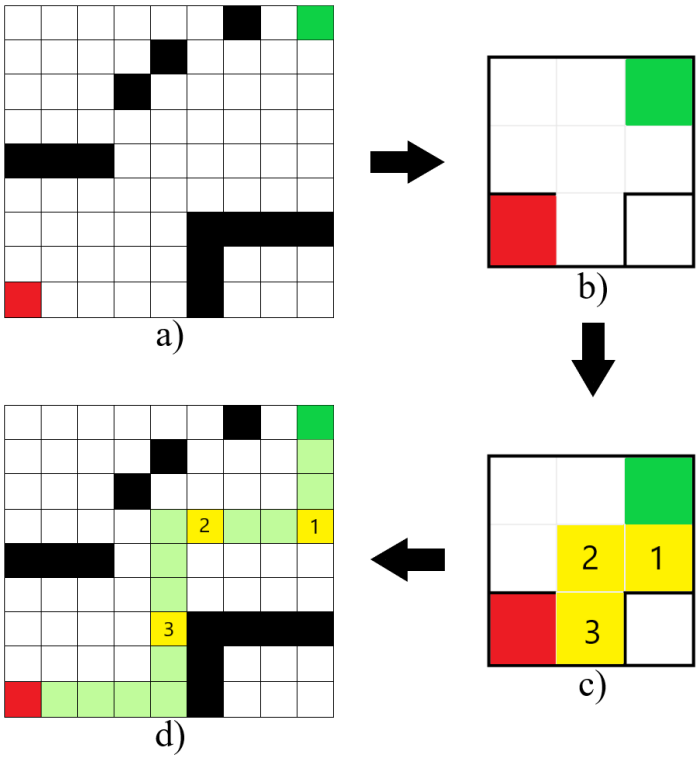
19 pav. Kambarių dekompozicijos metodo pradžios ir pabaigos taško prijungimas, kur *m* yra pradžios taškas ir *n* – pabaigos [25]

Stačiakampių kambarių dekompozicijos metodo privalumas yra tai, kad jį galima naudoti kartu su kitais hierarchinės abstrakcijos metodais ar kelio paieškos algoritmais.

2.1.8. *CHPA** algoritmas

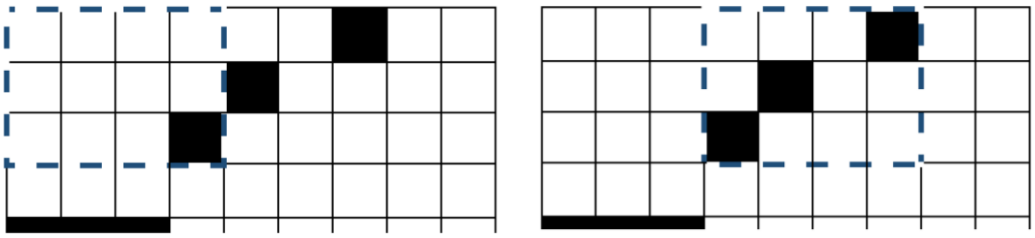
*Convolutional Hierarchical Pathfinding A** (toliau *CHPA**) yra hierarchinis kelio paieškos algoritmas, gebantis sparčiai surasti kelią, artimą trumpiausiam ir jo kūrimas buvo įkvėptas paveikslų apdorojimo filtravimo procesu. Paveikslų filtravimas dažniausiai atliekamas kaip grupė vietinių operacijų, naudojant judančio lango principą [28], o *CHPA** algoritmas pritaiko būtent šį principą žemėlapio apdorojimui. *CHPA** algoritmas buvo pirmą kartą viešai pristatytas tarptautinėje konferencijoje *Informacinės Visuomenės ir Universitetinės Studijos* (angl. *Information Society and University Studies* , *IVUS*) (žr, nuorodą **1 priede**).

Šis algoritmas prieš paiešką apdoroja žemėlapį, suformuoja hierarchinį abstrakcijos sluoksnį, ir tikros paieškos metu suranda tarpinius taškus abstrakcijos sluoksnyje, kurie bus naudojami tikrojo žemėlapio paieškos žingsnyje. Bendro *CHPA** algoritmo veikimo pavyzdys pateiktas **20 pav.Error! Reference source not found.**



20 pav. *CHPA** veikimo pavyzdys. a) Tikrasis žemėlapis apdorojamas ir sudaromas b) abstrakcijos sluoksnis. Pradinė paieška atliekama c) abstrakcijos sluoksnyje ir tarpiniai taškai naudojami paieškos algoritmo nukreipimui d) tikrame žemėlapyje

*CHPA** algoritmas žemėlapio apdorojimui naudoja panašaus į filtrą stačiakampės formos langus, kurių tikslas – apibrėžti galimybę praeiti pro nurodytą žemėlapio sekciją ir suformuoti abstrakcijos sluoksnį. Horizontalaus filtro naudojimas pavaizduotas **21 pav.Error! Reference source not found.**, kur galima pamatyti kaip antrame žingsnyje filtras perkeliamas per tris langelius horizontaliai ir persidengia su buvusia filtro pozicija, kad būtų užtikrinta, jog galima praeiti tarp filtrų apimtų sekcijų nors vienu keliu. Apdorojimui taip pat naudojami tokios pačios formos vertikalūs filtrai.

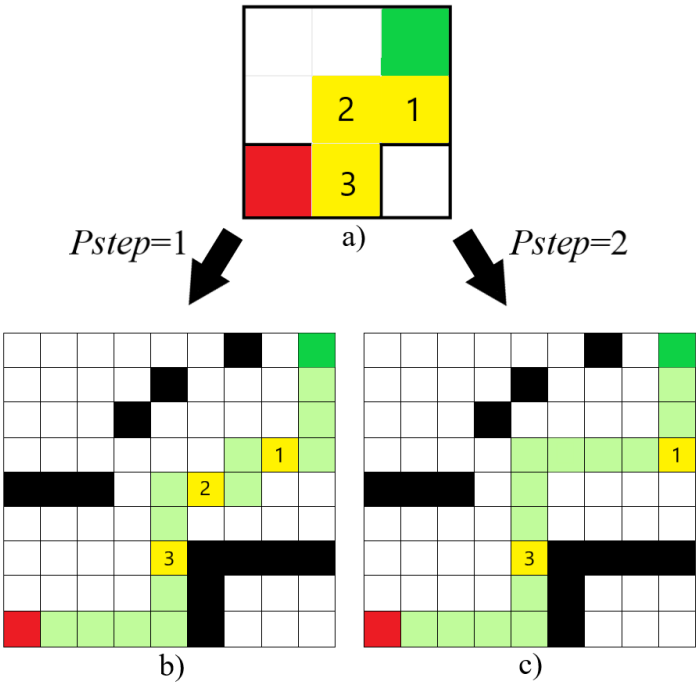


21 pav. *CHPA** abstrakcijos sluoksnio formavimas: a) pirmas horizontalus žingsnis, b) antras horizontalus žingsnis

Abstrakcijos sluoksnis yra sudarytas iš apdorotos informacijos apie tikrąjį žemėlapį, yra mažesnis nei tikrasis žemėlapis ir naudoja grafo struktūrą, todėl jame galima atlikti aukštesnio lygio kelio paiešką mažesnėje paieškos erdvėje naudojant klasikinius kelio paieškos algoritmus, tokius kaip *A**.

Norint surasti kelią tarp dviejų taškų, pirma atliekama paieška abstrakcijos sluoksnyje, tada gautų kelio taškų koordinatės perskaičiuojamos į tikrojo žemėlapio koordinates. Šie perskaičiuoti taškai gali būti imami tam tikru intervalu *Pstep* ir toliau yra naudojami kaip tarpiniai taškai kelio paieškai

tikrame žemėlapyje, kaip pavaizduota **22 pav.Error! Reference source not found.** geltonais langeliais.



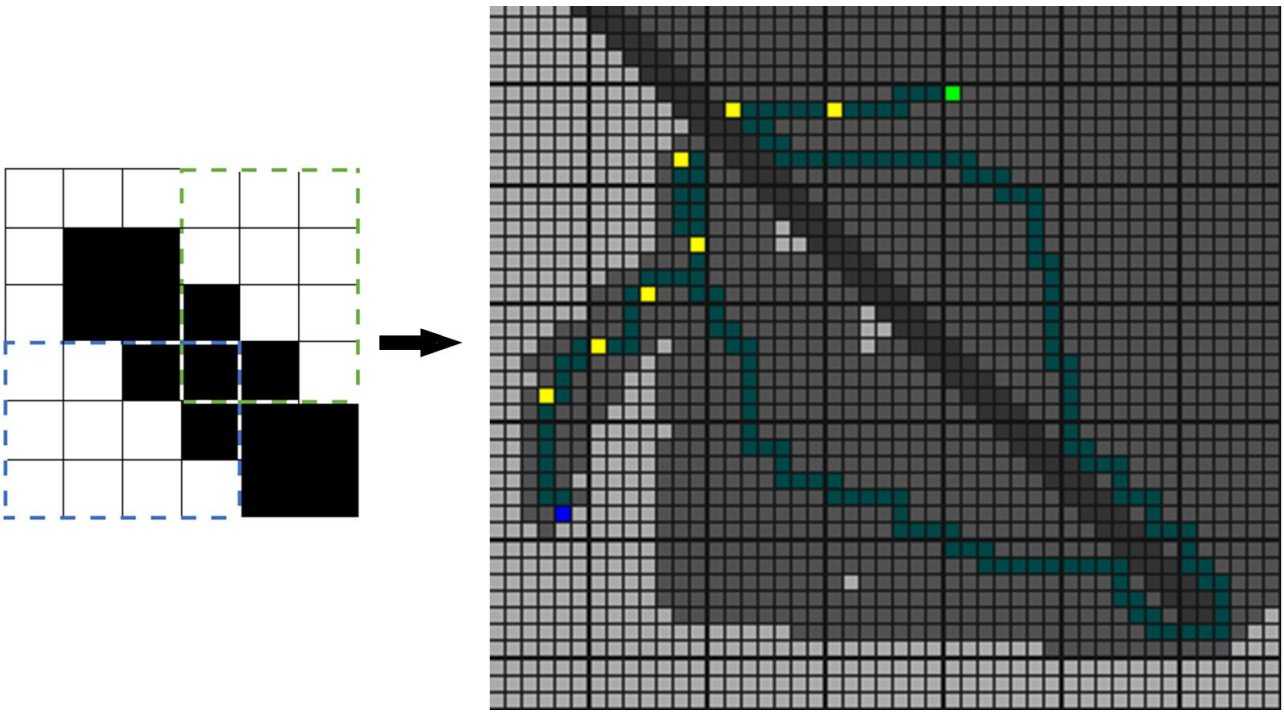
22 pav. *CHPA** kelio radimas nuo žalio iki raudono langelio: (a) kelias abstrakcijos sluoksnyje, (b) kelias tikrame žemėlapyje, kai *Pstep* = 1, (c) tikro žemėlapio kelias, kai *Pstep* = 2

Didesnės *Pstep* reikšmės sumažina tarpinių taškų skaičių ir sumažina koordinačių perskaičiavimo į žemesnį hierarchijos sluoksnį paklaidą, tačiau tada tikrame žemėlapyje atliekama paieška tarp vis didesnių atstumų ir pasitaikius kliūtims sulėtėja paieška.

2.2. *CHPA** problemos ir siūlomi sprendimai

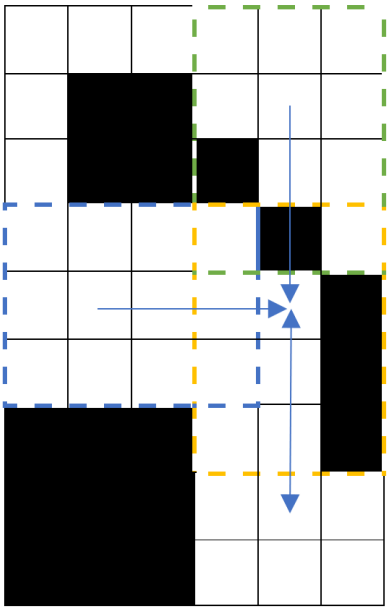
2.2.1. Klaidinga abstrakcijos sluoksnio formavimo prielaida

Viena iš *CHPA** algoritmo problemų yra galimos abstrakcijos sluoksnio formavimo klaidos. Žemiau esančiame **23 pav.** pavaizduota, kaip horizontalaus ir vertikalaus lango erdvėje galimas praėjimas į tą patį segmentą, tačiau tai nereiškia, kad yra kelias tarp skersinių segmento taškų. Šiai abstrakcijos sluoksnio formavimo problemai spręsti reikėtų keisti abstrakcijos sluoksnio formavimo logiką, arba atlikti pataisymus po formavimo.



23 pav. Abstrakcijos proceso problema, kai vertikalus ir horizontalus langas mato kelią į tą patį segmentą, todėl sudaroma prielaida, kad galima keliauti skersai segmento, kur mėlynas langelis - pradinė pozicija, šviesiai žalias – galutinė pozicija, žalias – rastas kelias, geltonas – tarpinis taškas

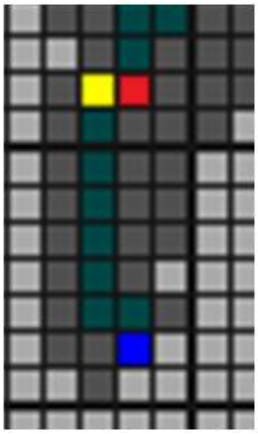
Identifikuoti klaidingai sujungtas sekcijas galima apdorojimo fazėje patikrinus ar sekcijoje galimi keliai tarp skersinių sekcijos taškų. Šios klaidingai sujungtos sekcijos visiškai atjungti negalima, kadangi joje esančio galutinio taško pasiekti būtų neįmanoma, tačiau galima palikti kraštinę vedančią į šią sekciją ir dalinai išspręsti šią problemą. Šis sprendimas taip pat nėra tobulas, kadangi gali būti situacijų, kai trumpiausias kelias privalo naudoti šį segmentą praėjimui, tačiau tokios situacijos turėtų susidaryti tik kai kliūtys yra siauros ir išdėstytos nedideliais atstumais, o *CHPA** algoritmas tam yra nepritaikytas. Viena tokių situacijų pavaizduota **24 pav.**, kur abstrakcijos sluoksnio pataisymas panaikina vienos krypties kelią iš probleminio segmento, tačiau viršutinė pavyzdinio žemėlapio dalis tampa nepasiekiama iš apatinės dalies.



24 pav. Siūlomo *CHPA** abstrakcijos sluoksnio pataisymo problema, kur vienintelis kelias į viršutinę žemėlapio dalį yra panaikinamas ir ji tampa nepasiekiama

2.2.2. Neoptimali kelio paieška tarp tarpinių taškų

Kita klaida yra koordinačių perskaičiavime į tikrąjį žemėlapį, kadangi nėra žinoma, kuris iš abstraktaus sluoksnio sekcijos aprėpiamų taškų yra tinkamas optimaliam keliui. Esamas tarpinių taškų nustatymo sprendimas parenka pirmą laisvą poziciją pradedant nuo vidurinės sekcijos pozicijos ir vėliau tikrina kraštines pozicijas. Žemiau esančiame **25 pav.** mėlyna spalva pavaizduota pradinė paieškos pozicija nėra centriniame segmento stulpelyje, todėl parenkamas neoptimalus geltona spalva pavaizduotas tarpinis taškas.



25 pav. Tarpinio taško parinkimas. Mėlynas langelis - pradinė pozicija, raudona – galutinė pozicija, žalias – rastas kelias, geltonas – tarpinis taškas

Vienas iš sprendimų būtų tikrinti pradedant nuo tos pačios eilės, tačiau tai prideda sudėtingesnę logiką ir atsiradus kliūčių tarp šių taškų šis sprendimas nebegarantuoja optimalaus kelio.

Paprastesnis sprendimas būtų neapibrėžti tikslaus tarpinio taško, tikrinti ar kelio paieškos algoritmo tikrinamas taškas yra reikiamos abstrakcijos sluoksnio segmento ribose ir nutraukti paiešką kai įžengiama į ją. Šis sprendimas nesudėtingas, turėtų tikti įvairioms situacijoms ir sutrumpinti randamo kelio ilgį.

2.2.3. Pritaikymas 8-ių krypčių judėjimui

Pirmosios *CHPA** algoritmo versijos tikslas buvo idėjos patikrinimas, todėl algoritmas buvo lyginamas tik su *A** algoritmu ir pritaikytas judėjimui 4-iomis kryptimis. Šio projekto metu *CHPA** algoritmas bus pritaikomas 8-ių kryčių jungimo grafui ir tai suteiks kelias naujas galimybes:

1. tiesiogiai palyginti algoritmo veikimą su *JPS* algoritmu;
2. panaudoti pateiktus duomenų rinkinio kelio paieškos scenarijų rezultatus;
3. praplėsti algoritmui tinkamas duomenų struktūras.

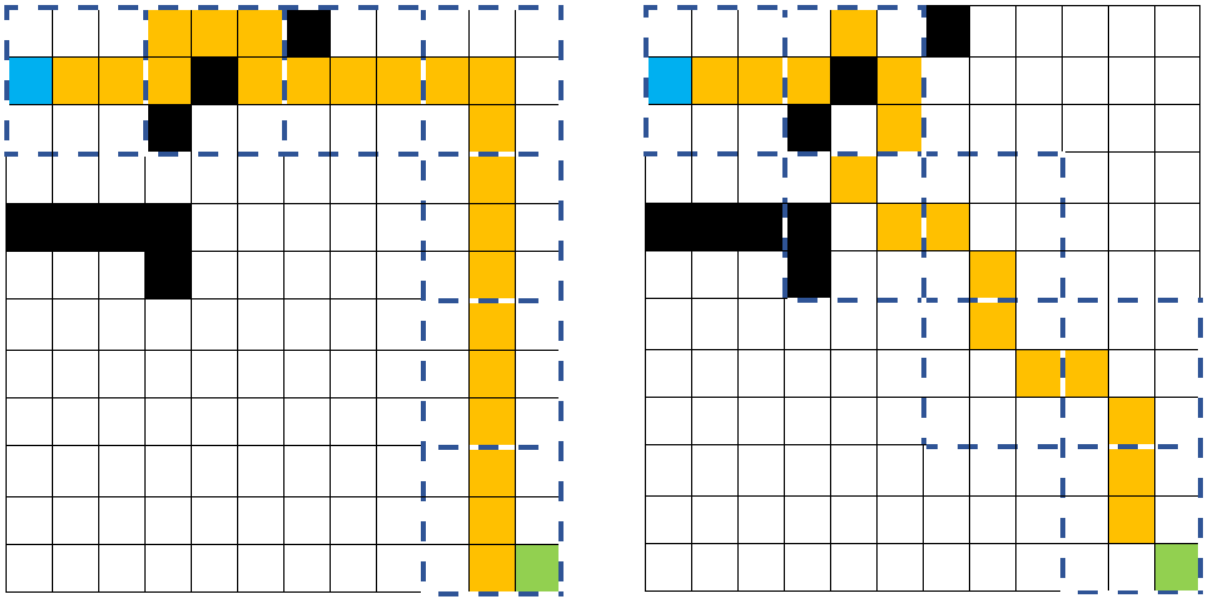
*CHPA** algoritmas naudoja abstrakcijos sluoksnius, todėl juos taip pat galima pritaikyti 8-ių krypčių judėjimui ir sutrumpinti randamo kelio ilgį, tačiau tai gali sukelti naujų problemų

Tikro žemėlapio lygyje jungimo krypčių skaičiui nėra apribojimų, kadangi jame kelio paiešką atlieka *A** algoritmas. Tikro žemėlapio segmentai taip pat yra apdorojami *A** algoritmo, tačiau jų apjungimo į abstrakcijos sluoksnį logika buvo pritaikyta tik 4-ioms kryptims.

8-ių krypčių jungimas abstrakcijos sluoksnyje nėra būtinas norint pritaikyti algoritmą 8-ių krypčių veikimui, kadangi tikrame žemėlapyje *A** algoritmas turi palaikymą. *CHPA** randamas kelias yra panašus į *A** kai visuose hierarchijos lygiuose naudojamos vienodos taisyklės, tačiau skirtingos taisyklės gali smarkiai padidinti randamo kelio ilgį. **26 pav.** pavaizduotas scenarijus kur tam tikros realizacijos *A** algoritmas abstrakcijos sluoksnyje randa kelią sudarytą iš nedidelio kiekio posūkių. Tai reiškia, kad tikrame žemėlapyje naudojami tarpiniai taškai nesuteiks galimybės *A** algoritmui keliauti įstrižai ir padidins kelio ilgį. Šiame scenarijuje nemodifikuotos *A** versijos rasto kelio ilgis yra 23, algoritmo versijos su krypčių keitimo optimizacija – 20,2, o trumpiausias kelias – 15.

26 pav. paveiksle dešinėje pusėje taip pat pavaizduotas vienas galimų sprendimų. Šio sprendimo principas yra modifikuoti abstrakcijos sluoksnyje naudojamą *A** algoritmą, kad jo randamo kelio forma turėtų daugiau krypčių pokyčių, taip suteiktų *A** algoritmui daugiau galimybių keliauti įstrižai ir sumažinti randamo kelio ilgį. 4-ių krypčių abstrakcijos sluoksnio jungimas taip pat gali turėti geresnę greitaveiką, kadangi 4-ių krypčių jungimas turės mažiau kraštinių grafe.

Kitas sprendimas yra žemėlapio apdorojimo proceso pritaikymas abstrakcijos sluoksnio jungimui 8-iomis kryptimis naudojant turimą apdorojimo fazės informaciją. Šio sprendimo privalumas yra tikro žemėlapio ir abstrakcijos sluoksnio jungimo suvienodinimas, dėl kurio bus vidutiniškai randamas trumpesnis kelias, nei 4-ių krypčių jungimo sprendimas. Šį sprendimą taip pat paprasčiau įgyvendinti, kadangi nereikės turėti papildomos *A** algoritmo versijos.



a) b)

26 pav. Pavyzdinis *CHPA** 8-ių krypčių tikro žemėlapio ir 4-ių abstrakcijos sluoksnio jungimo veikimas, naudojant *Pstep=1, kur* Mėlynas langelis – pradžia, žalias – pabaiga, geltonas – kelias: a) Nemodifikuotas *A**, b) *A** su krypčių keitimo optimizacija

Viena šio projekto idėjų yra pasiekti kuo trumpesnį *CHPA** randamo kelio ilgį smarkiai nepakenkiant greitaveikai, todėl pasirinkta 8-ių krypčių pritaikymą spręsti naudojant turimą abstrakcijos sluoksnio informaciją išplėsti 4-ių krypčių jungimui į 8-ių krypčių.

2.3. Darbo priemonės ir technologijos

Vienas svarbiausių pasirinkimų programinės įrangos projektams yra programavimo kalba, kadangi tai nusako kuriamos programinės įrangos greitaveikos galimybes, realizavimo greitį, palaikymo sudėtingumą ir t. t. Algoritmų, tokių kaip kelio paieškos algoritmai, paskirtis dažniausiai yra greičiau gauti sprendimą ir žemo lygio kalbos, tokios kaip *C/C++*, neretai naudojamos jų realizacijai, kadangi jos leidžia efektyviau valdyti techninę įrangą ir maksimizuoti greitaveiką. Šio projekto tikslas yra realizuoti *CHPA** algoritmo prototipo patobulinimus, todėl svarbiau greitai realizuoti turimas idėjas, o aukšto lygio kalbos suteikia šią galimybę. Taip pat tam, kad būtų galima pasinaudoti jau projekte esančių algoritmų realizacija, atvaizdavimo posisteme ir teisingai įvertinti algoritmų greitaveiką laiko atžvilgiu, reikėtų juos visus įgyvendinti naudojant tą pačią programavimo kalbą ir bibliotekas. Esama realizacija parašyta *Python* programavimo kalba, dėl šių priežasčių šis projektas bus realizuojamas naudojant *Python* programavimo kalba.

Python programavimo kalba yra plačiai palaikoma įvairiose operacinėse sistemose, turi įvairių įskiepių, bibliotekų ir detalizuotą dokumentaciją. Šios programavimo kalbos aukštas lygis ir turima patirtis dirbant su ja smarkiai sumažins realizavimo laiką. Dar vienas svarbus šios programavimo kalbos aspektas yra galimybė lengvai perkelti projektą į kitą įrenginį arba debesų serverį ir šiuo privalumu buvo pasinaudota šio projekto metu.

Projekte naudojamos kelios bibliotekos:

- *Pygame* – *Python* programavimo kalbos modulių rinkinys skirtas kompiuterinių žaidimų kūrimui [29];

- *Tqdm* – greita progreso juosta *Python* programavimo kalbai [30];
- *Pillow – PIL, Python* programavimo bibliotekos vaizdavimui, atšaka sukurta Alex Clark ir bendraautorių [31];
- *Numpy* – fundamentalus *Python* paketas moksliniams skaičiavimams [32];

Tqdm ir *Pillow* bibliotekos darbo eigos per daug nekeičia, todėl jų pasirinkimui nebuvo atlikta plati analizė.

Algoritmų darbo eigos atvaizdavimui nutarta pasinaudoti kompiuterinių žaidimų kūrimui skirta biblioteka, kadangi jos privalo turėti efektyvų ir dažniausiai nesudėtingą naudoti atvaizdavimo posistemę. Labiausiai išvystyti įrankiai šiai užduočiai yra *pygame* ir *pyglet*. *Pyglet* sukurtas vėliau ir gali efektyviau panaudoti techninę įrangą, o *pygame* yra populiaresnis, turi daugiau priedų ir pasirodė intuityvesnis, todėl atsižvelgus į greito iteravimo poreikį ir laiko trūkumus nusprendėme naudoti *pygame*.

Numpy paketas įtraukia patobulintų duomenų struktūrų ir operacijų, kurios palengvina ir pagreitina darbą su didesniais duomenų kiekiais. Vietoje šio paketo būtų galima naudoti *Scipy*, *SymPy* arba *tinynumpy*, tam kad sumažinti projekto atminties reikalavimus arba apeiti įdiegimo problemas, tačiau *Numpy* pasižymi greičiu bei yra labai plačiai naudojamas moksliniuose *Python* projektuose ir net paketuose, todėl jis buvo pasirinktas.

Pirmos *CHPA** algoritmo versijos projekto įgyvendinimui buvo naudojamas asmeninis stacionarus kompiuteris, kurio specifikacijos buvo tokios:

- procesorius (angl. *CPU*) – *Ryzen 5 3600*;
- operatyvinės atminties kiekis (angl. *RAM*) – 16 GB.

Realizacijai šios techninės įrangos užteko, tačiau pradėjus eksperimentinius tyrimus pastebėta, kad testavimo ciklas vienam žemėlapyje galėjo užtrukti daugiau nei 5 valandas, o tai smarkiai sulėtino klaidų aptikimą ir taisymą. Testavimas asmeniniame kompiuteryje taip pat apriboja naudojimosi galimybes testavimo metu ir foniniai procesai gali paveikti rezultatus, todėl šio projekto metu nutarta perkelti testavimą į tyrimams skirtą aplinką, taip pagreitinant testavimo procesą ir padidinant rezultatų patikimumą. Šiai problemai spręsti buvo pasinaudota, KTU universiteto suteiktu, didelio našumo skaičiavimo klasteriu.

Projekto versijavimui ir talpinimui nuotoliniam pasiekimui pasirinkta naudoti *github*, kadangi turima patirtis, paskyra ir įrankiai leis sutaupyti laiko. Versijavimas yra itin svarbus, kadangi projekto talpinimas serveryje turi mažiau rizikos būti prarastas, gali būti pasiekiamas daugelio iš skirtingų prietaisų, nesudėtinga laikyti kelias skirtingas programos versijas ir atstatyti į buvusią versiją.

Paprastų UML diagramų sudarymui galima būtų naudoti nemokamas internetines paslaugas, kaip *lucidchart.com, creately.com, draw.io*, tačiau studijų metu mes buvome apmokyti naudotis įrankiu *MagicDraw*. Šio įrankio naudojimosi licencija nemokamai suteikiama studentams ir jis yra plačiai naudojamas KTU dėstytojų, todėl pažįstama aplinka palengvins pateikiamų diagramų supratimą. Taip pat turima naudojimosi patirtis, įgyta studijų metu, sumažins diagramų kūrimui skiriamą laiką.

3. Siūlomų *CHPA** algoritmo patobulinimų ir *JPS* algoritmo specifikacija

Šiame skyriuje apibrėžiamas bendras sistemos projekto planas, funkciniai ir nefunkciniai reikalavimai, kokybės kriterijai ir pasirinktos realizacijos priemonės. Taip pat pateikiamos sistemos ir algoritmo veikimo principus specifikuojančios diagramos.

3.1. Projekto planas

Šio darbo metu kuriama sistema bus paremta jau sukurta eksperimentine *CHPA** algoritmo testavimo sistema, kuri buvo sukurta 2020 m. [33]. Pasinaudojus esama ir žinoma sistema galima bus pridėti daugiau algoritmų palyginimui ir skirti daugiau laiko kokybiškam sprendimų kūrimui bei testavimui. Šio tyrimo metu be patobulinimų sprendimų taip pat bus realizuojamas ankščiau analizuotas *JPS* algoritmas.

Sistema turi kelias posistemes:
1. atvaizdavimo;
2. regresinio testavimo;
3. eksperimentinio testavimo.

Atvaizdavimo posistemės veikimo pradžioje užkraunami pasirinktų žemėlapių duomenys, jie apdorojami ir suskaičiuojamas vieno langelio dydis pikseliais. Langelių dydis skaičiuojamas tam, kad būtų galima patikrinti ar yra prasmė bandyti atvaizduoti žemėlapį, kadangi smulkius langelius bus sunku pamatyti ir didelių žemėlapių atvaizdavimas gali smarkiai sulėtinti sistemos darbą. Po šio uždavinio taip pat atliekamas žemėlapio apdorojimas, jeigu to reikalauja algoritmas. Galiausiai atvaizduojamas pilnas žemėlapis programos lange.

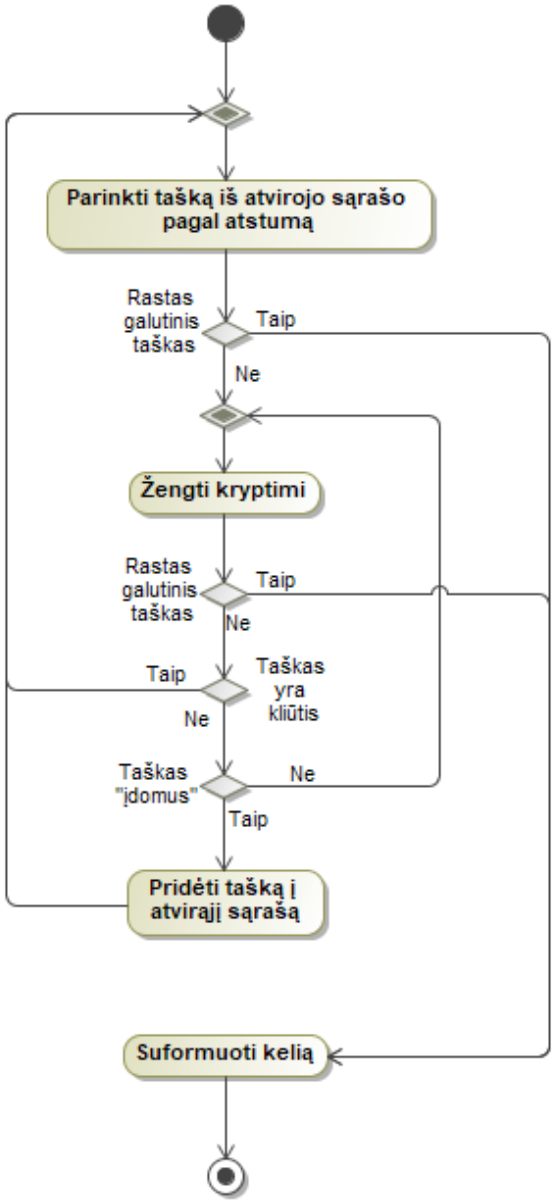
Toliau sistema atsitiktinai pasirenka du žemėlapio taškus, tarp kurių bus atliekama kelio paieška naudojant pasirinktą algoritmą ir atvaizduojama jo veikimo eiga. Šie veiksmai kartojami tol, kol aptinkamas lango uždarymo mygtuko paspaudimas. Šioje sistemoje kelio paieška yra nuoseklus ir galimai ilgai trunkantis procesas, todėl atvaizduoti algoritmo veikimo eigą tik po to, kai pilnai atliekama kelio paieška nebūtų protinga, kadangi kelio paiešką gali trukti net kelias sekundes, naudojant pakankamai didelius žemėlapius. Atskyrus kelio paiešką ir eigos atvaizdavimą į atskirus lygiagrečiai veikiančius procesus sistemai nebereikės laukti kelio paieškos pabaigos, bus dažniau atvaizduojama algoritmo eiga ir bus galima matyti tarpinius algoritmo eigos rezultatus, o ne tik galutinį.

Sistemos dalys yra stipriai susietos, todėl smulkūs pakeitimai gali pakeisti posistemės ar algoritmo veikimą. Šios klaidos dažniausiai pastebimos vėlai ir prailgina projekto realizaciją bei tyrimus. Taip pat algoritmai nėra nuolat tikrinami įvairiuose scenarijuose, todėl norint užtikrinti projekto kokybę ir efektyviau atlikti tyrimus buvo sukurta regresinio testavimo posistemė. Čia aprašyti įvairūs testavimo scenarijai naudojamiems algoritmams, kurie leidžia pamatyti rezultatų pokyčius ir įvertinti algoritmų realizacijų veikimą.

Eksperimentinio testavimo posistemė skirta atlikti eksperimentinius tyrimus, todėl joje kelio paieškos algoritmų eiga neatvaizduojama ir visi rezultatai yra pateikiami *CSV* formato failuose.

3.2. *JPS* algoritmo specifikacija

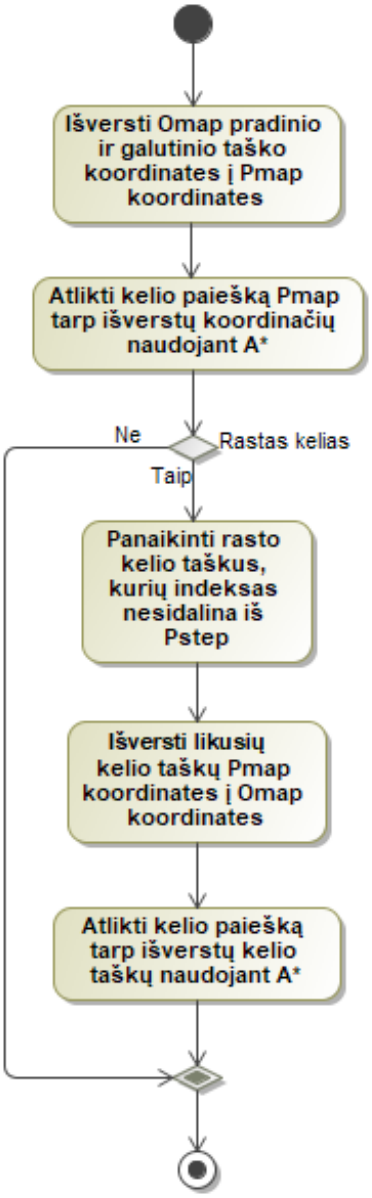
JPS algoritmo veiklos diagrama pavaizduota **27 pav.** Galima pastebėti, kad šis algoritmas prieš pridėdamas tašką į atvirąjį sąrašą atlieka daug patikrinimų, o taip yra sumažinamas atvirojo sąrašo dydis ir spartinama paieška.



27 pav. *JPS* algoritmo veiklos diagrama

3.3. *CHPA** algoritmo specifikacija

*CHPA** algoritmo veikimo principas išlieka toks pat ir yra pavaizduotas **28 pav.** Algoritmas perskaičiuoja duotas koordinates į abstraktaus žemėlapio koordinates, jame atlieka paiešką, atmeta dalį gautų tarpinių taškų, išverčia jų koordinates į tikro žemėlapio koordinates ir šiuos taškus naudoja paieškos tikrame žemėlapyje vedimui.

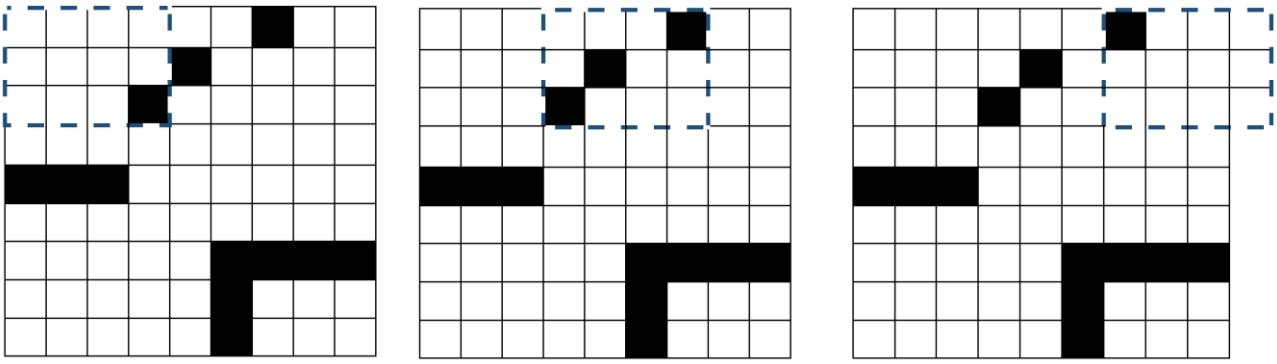


28 pav. *CHPA** algoritmo veiklos diagrama

3.4. *CHPA** žemėlapio apdorojimas

*CHPA** algoritmas pasinaudoja išankstiniu žemėlapio apdirbimu, kad suspaustų informaciją, sudarytų abstrakcijos sluoksnį ir pagreitintų realaus laiko kelio paiešką, tačiau ši algoritmo versija taip pat įgauna apribojimą draudžiantį pokyčiams tikrame žemėlapyje po apdorojimo. Manome, kad ateityje galima būtų įgyvendinti abstraktaus žemėlapio atnaujinimo funkciją, tačiau tai nėra šio projekto darbo dalis.

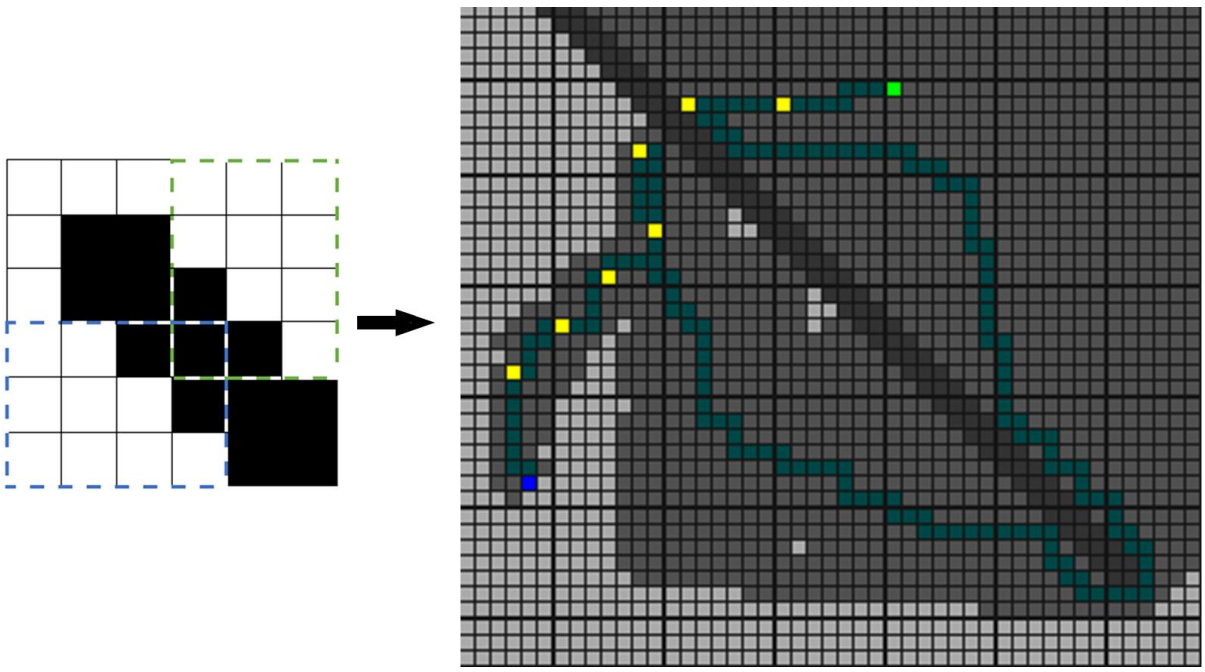
Šio projekto metu kuriamo algoritmo idėja pagrįsta paveikslų apdorojimo konvoliucijos proceso veikimu, kuriame naudojami įvairių formų filtrai bruožų aptikimui paveiksluose. 2D kvadratinio tinklo išdėstymo žemėlapiai taip pat gali būti traktuojami kaip paveikslai, todėl kilo idėja pamėginti pritaikyti filtrus žemėlapio apdorojimui.



29 pav. Apdorojimo filtro naudojimo pavyzdys

Apdorojimo fazėje filtrų darbas yra nusakyti ar egzistuoja nors vienas kelias tarp priešingų pusių langelių ir **29 pav.** pavaizduotas tokio filtro panaudojimas. Abstrakcijos žemėlapis sudarytas iš tikro žemėlapio 3x3 segmentų praėjimo informacijos, tačiau apdorojimui naudojami 4x3 formos filtrai, kadangi tokiu būdu filtrai persidengia viena eile langelių ir taip bus apibrėžiama galimybė pereiti iš vieno segmento į kitą, o ne iš vienos segmento pusės į kitą. Suformuotas abstrakcijos žemėlapis taip pat yra grafas ir galima būtų sukurti daugiau hierarchinių abstrakcijos sluoksnių, tačiau taip bus prarandama vis daugiau informacijos, randamo kelio ilgis didėtų ir abstrakcijos problemų skaičius didėtų.

Filtro dydis taip pat keičia galimų apdorojimo klaidų kiekį, todėl esamas 4x3 dydis pasirinktas norint turėti nedidelį segmentą, kad klaidas būtų paprasčiau analizuoti. Bakalaurinio projekto metu buvo atrasta abstrakcijos problema, kuri pavaizduota **30 pav.**, kai horizontalus ir vertikalus filtras mato kelią į tą patį segmentą ir priimama išvada, kad galima keliauti ir skersai segmentu.

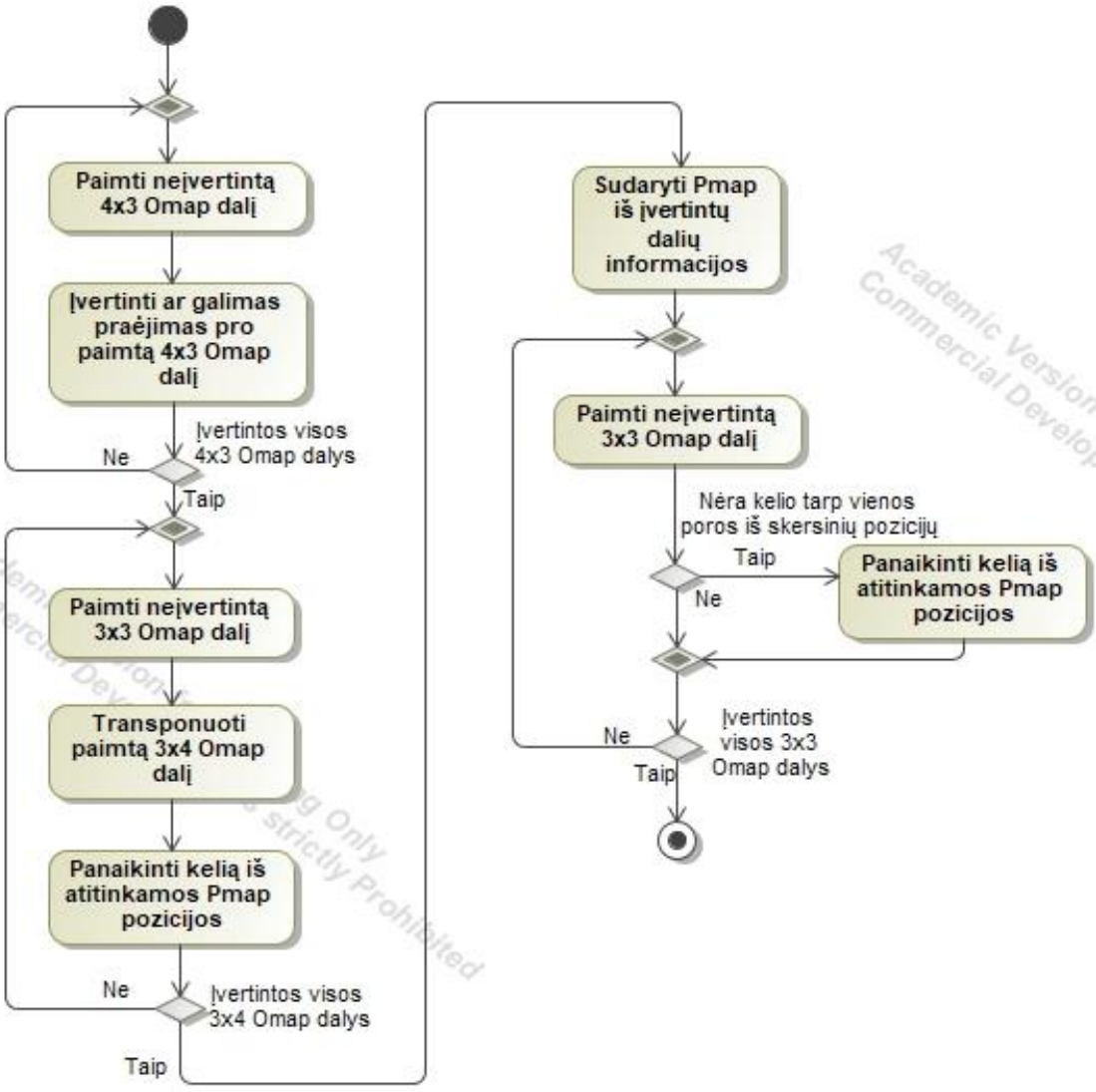


30 pav. Apdorojimo proceso problema

Šiai problemai spręsti bus papildomai patikrinama ar yra praėjimas tarp skersinių segmento taškų ir jeigu jo nėra, šio segmento grafo viršūnė bus sujungiama viena kryptimi. Ši modifikacija užtikrins, kad galutinis langelis, esantis šiame segmente, išliks pasiekiamas, o kitu atveju nebus naudojamas

kaip tarpinis segmentas. Taip pat atsiras nauja problema, kai pradinis langelis yra šiame segmente, kadangi išėjimo iš jo nėra.

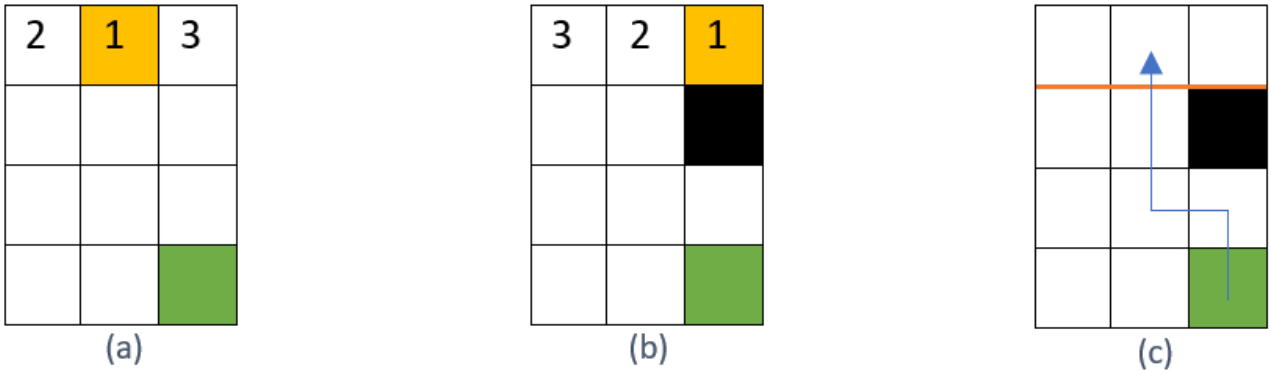
Abstrakcijos sluoksnio kūrimo proceso veiklos diagrama pavaizduota **31 pav.** Čia *Omap* – tikras žemėlapis, *Pmap* – abstraktus žemėlapis. Žemėlapio apdorojimo procesas yra gana paprastas – apeiti visą žemėlapį horizontaliais ir vertikaliais filtrais, išplėsti žemėlapį jei nėra galimybės jo pilnai padalinti 3x3 segmentais ir pasinaudojus šia informacija sudaryti abstrakcijos sluoksnį.



31 pav. Abstraktaus žemėlapio sudarymo veiklos diagrama. *Omap* – tikras žemėlapis, *Pmap* – abstrakcijos sluoksnis.

3.5. *CHPA** koordinačių perskaičiavimas tarpiniams taškams

Atliekant paiešką *CHPA** algoritmas pirma perskaičiuoja pradinio ir galutinio taško koordinates į abstraktaus sluoksnio koordinates. Toliau atliekama įprasta *A** algoritmo kelio paieška abstrakčiame sluoksnyje ir gauto kelio taškų koordinatės yra perskaičiuojamos į tikro žemėlapio koordinates. Šis antrasis koordinačių perskaičiavimas yra sudėtingesnis nei pirmasis, kadangi netinkamo tarpinio taško parinkimas didins randamo kelio ilgį. Pirmojoje *CHPA** algoritmo versijoje buvo atsižvelgiama į kelionės kryptį ir iš anksto aprašyta tvarka ieškoma laisvo langelio, kaip pavaizduota **32 pav.**



32 pav. Koordinačių perskaičiavimo į tikrą žemėlapį pavyzdžiai:
(a) iš anksto aprašyta (pirmosios versijos metodas), (b) vienodas stulpelis, (c) tinka visi

Variantas (a) nurodo pirmosios *CHPA** versijos veikimą, kur skaičiai nurodo kokia tvarka bus tikrinama ir pasirinktas pirmasis laisvas langelis. Varianto (b) atveju atsižvelgiama į pradinį langelį ir taip pat iš anksto nustatyta tvarka ieškoma laisvo langelio, tačiau tai taip pat gali parinkti neoptimalų kelią. Variantas (c) yra šio projekto metu įgyvendinamas patobulinimas, kuris reikalaus modifikuoti *A** algoritmą, kad jis galėtų ieškoti kelio iki vieno iš kelių nurodytų taškų.

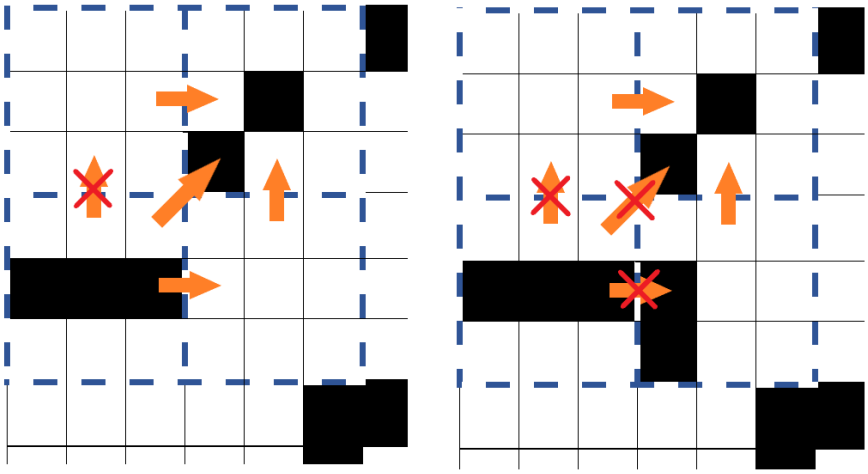
Realizacijai bus sukuriama papildoma *A** algoritmo versija. Ši *A** versija turės vienodą kelio paieškos logiką, tačiau bus nutraukiama kai bus randamas pirmas langelis, esantis galutiniame segmente. Galutinis langelis, skirtas euristinių funkcijų atstumų skaičiavimams, bus centrinis galutinio segmento langelis.

3.6. *CHPA** algoritmo pritaikymas paieškai 8-ių krypčių grafo jungimui

Šio projekto metu siūlomas sprendimas panaudoti sukurto 4-ių krypčių abstrakcijos sluoksnio duomenis ir pridėti įstrižus langelių jungimus. Šio metodo trūkumas yra kraštinių skaičiaus didėjimas abstrakcijos sluoksnyje, kuris galimai sulėtins algoritmo greitaveiką, tačiau pagerins randamo kelio ilgį.

Jungimo metu patikrinama ar abstrakcijos sluoksnyje tarp svarstomų įstrižų langelių galimas praėjimas kardinaliomis kryptimis, naudojant greta esančius langelius. Šios logikos pavyzdys pateiktas **33 pav.**

Dar viena problema yra žemėlapio apdorojimo proceso ilgėjimas, kadangi 8-ių krypčių pritaikymą bus galima vykdyti tik kai bus sukurtas 4-ių krypčių abstrakcijos sluoksnis. Žemėlapio apdorojimo procesą galima lygiagretinti, tačiau taikant *CHPA** algoritmą kintančiam grafui tai gali kelti greitaveikos problemų perskaičiuojant segmentus.



33 pav. *CHPA** abstrakcijos sluoksnio 8-ių krypčių jungimas

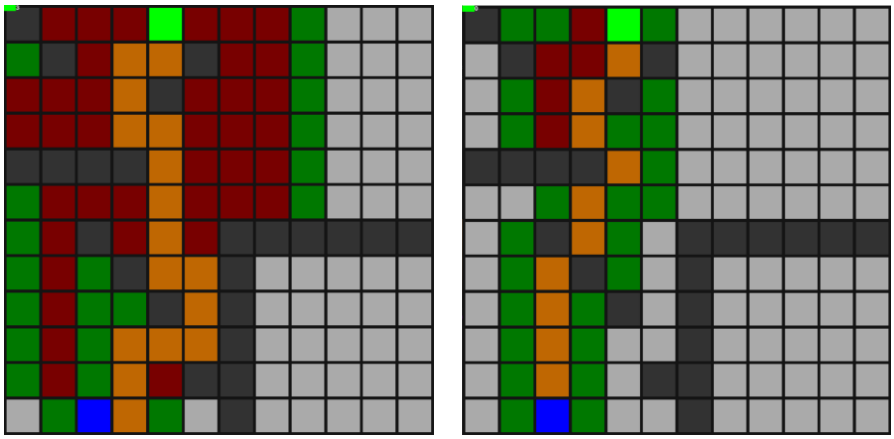
4. Algoritmų testavimas ir eksperimentiniai tyrimai

Šiame skyriuje aprašytas projekto metu atliktas algoritmų testavimas, eksperimentiniai tyrimai, jų naudos, bei analizuojami jų rezultatai.

4.1. Regresinis testavimas

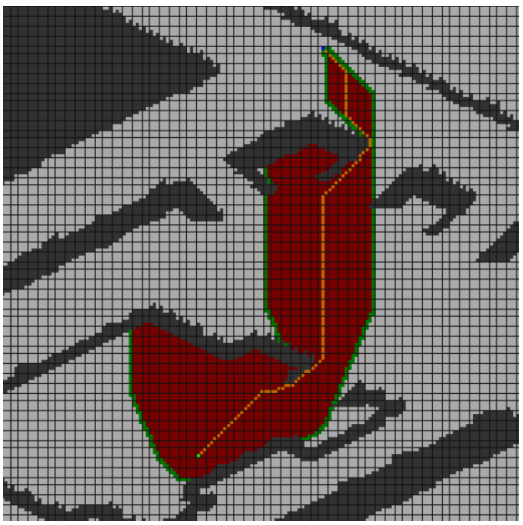
Pradinio projekto realizavimo metu viena didžiausių problemų buvo įvairios programavimo klaidos, kurios buvo pastebėtos tik eksperimentinio testavimo metu ir jų taisymas truko daug laiko. Norint išvengti šios problemos buvo sukurti įvairūs testavimo scenarijai regresiniam testavimui, kad būtų užtikrinta, jog algoritmai veiks teisingai ir tam tikri pakeitimai nepakeis jų rezultatų. Dalis testų pavyzdžių pavaizduoti žemiau:

1. rastas kelias atitinka patikrintą rezultatą, tikrinamas 4-ių / 8-ių krypčių jungimas, įvairūs kelio ilgiai, skirtingi algoritmai (žr. **34 pav.**);
2. *A** rasto kelio ilgis atitinka naudojamo duomenų rinkinio scenarijaus optimaliausio kelio rezultatą (žr. **35 pav.**);
3. *JPS* ir *A** randamų kelių ilgiai vienodi (žr. **36 pav.**);
4. nerandama kelio kai kelias tarp taškų neegzistuoja (žr. **37 pav.**);
5. randamas teisingas kelias taikant skirtingas taisykles kelio jungimui (žr. **38 pav.**).

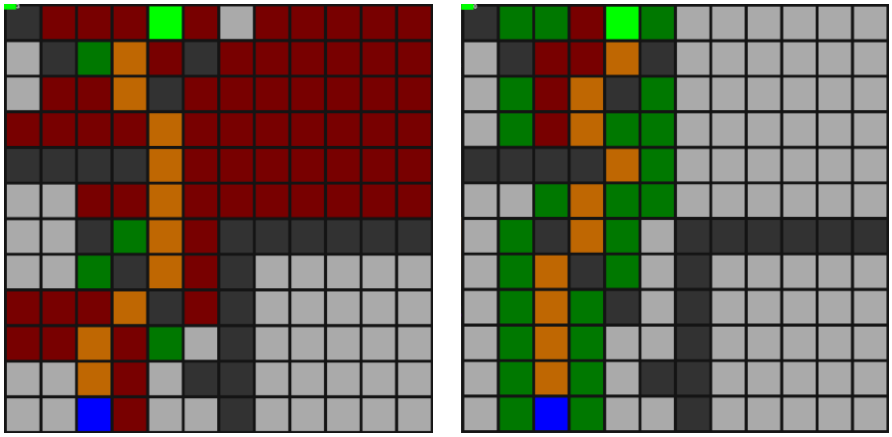


a) b)

34 pav. Randamo kelio patikrinimo testo scenarijus: a) *A** algoritmas 4 krypčių jungimu, b) *A** algoritmas 8 krypčių jungimu, kur pilki kvadratai – laisvi, juodi – kliūtys, šviesiai žalia – pradžia, mėlynas – kelio pabaiga, raudoni – algoritmo apžiūrėti, tamsiai žali – kandidatai patikrai, oranžinė – rastas kelias

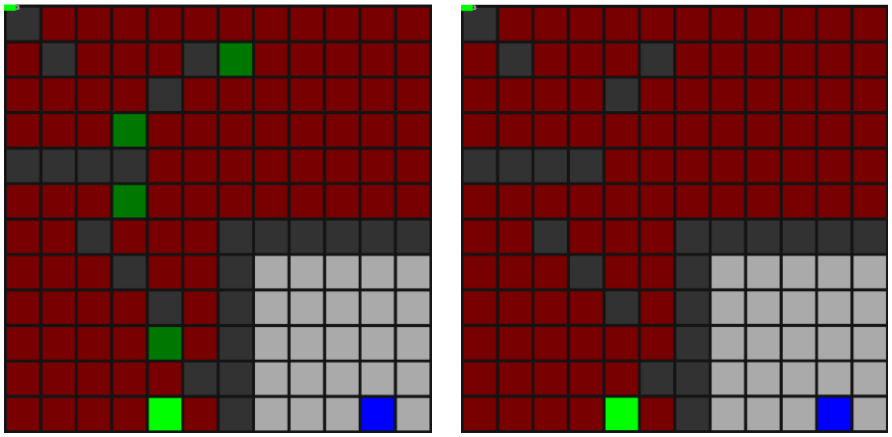


35 pav. *A** algoritmo rastas kelias pagal *Aurora* žemėlapio scenarijų



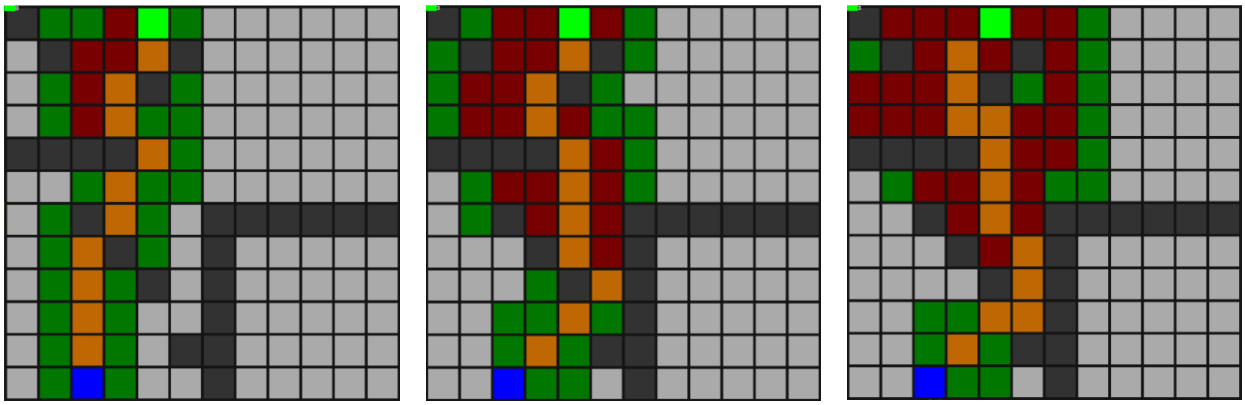
a) b)

36 pav. Algoritmų rasti keliai vienodam scenarijui: a) *JPS*, b) *A**



a) b)

37 pav. Algoritmų patikrinimas neegzistuojančio kelio scenarijui: a) *JPS*, b) *A**



a) b) c)

38 pav. Teisingo kelio radimo patikrinimas naudojant skirtingas žemėlapio taisykles: a) įstrižas judėjimas neribojamas, b) negalimas praėjimas tarp greta esančių įstrižų kliūčių, c) negalimas įstrižas judėjimas greta kliūčių

Iš viso sukurti 125 regresiniai testai ir prieš pradedant eksperimentinius tyrimus visada įsitikinama, kad visi testai gražina teisingus rezultatus.

4.2. Eksperimentinių tyrimų planas

*CHPA** algoritmo rezultatai lyginami su *A** ir *JPS* algoritmais įvairių tipų žemėlapiuose bei naudojant 4-ių / 8-ių krypčių jungimą. Visi kelio paieškos algoritmai naudojami šiame projekte buvo įgyvendinti šio projekto autoriaus. *CHPA** algoritmas naudoja tą pačią algoritmo realizaciją kaip *A**, taip išvengiama greitaveikos skirtumo tarp įvairiai optimizuotų algoritmų realizacijų.

Eksperimentiniai tyrimai atlikti naudojant universiteto suteiktą didelio našumo skaičiavimo klasterį, kurio techninės įrangos specifikacijos buvo tokios:

- procesorius – 2x *AMD EPYC 7452 32-Core*;
- operatyvinės atminties kiekis – 2x 256GB (512 GB).

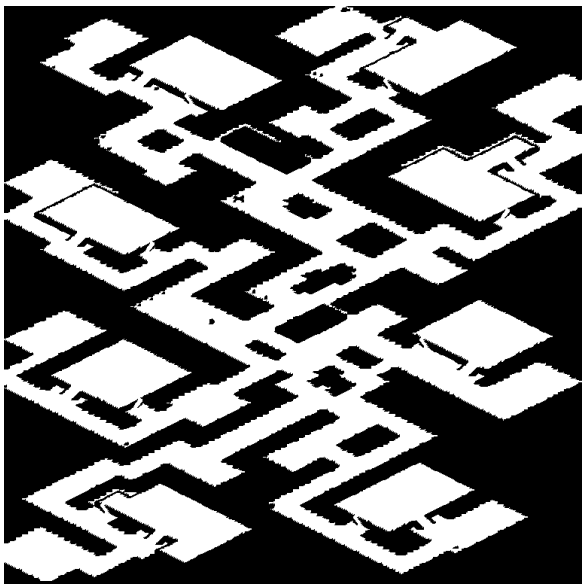
Siekiant įvertinti *CHPA** algoritmo pakeitimų efektyvumą apibrėžtas eksperimentinio tyrimo planas:

1. abstrakcijos sluoksnio formavimo pataisymo efektyvumas – analizuoti kokia dalis rastų kelių buvo paveikta abstrakcijos sluoksnio formavimo pataisymu bei jo įtaka randamo kelio ilgiui;
2. abstrakcijos sluoksnio jungimo 8-iomis kryptimis poveikis randamam keliui – palyginti skirtingų abstrakcijos sluoksnio jungimo metodų poveikį kelio paieškos greitaveikai, randamo kelio ilgiui;
3. *Pstep* poveikis randamo kelio ilgiui naudojant naują koordinačių perskaičiavimo metodą – analizuoti *Pstep* parametro poveikį randamo kelio ilgiui, naudojant tikro žemėlapio ir abstrakcijos sluoksnio įvairių krypčių jungimus bei naują koordinačių perskaičiavimo metodą;
4. naujo koordinačių perskaičiavimo metodo efektyvumas – palyginti naujo koordinačių perskaičiavimo metodo įtaką randamo kelio ilgiui su senu, iš anksto aprašyto tikrinimo, metodu;
5. įgyvendintų algoritmų lyginimas – palyginti *A**, *JPS*, *CHPA** algoritmų greitaveiką, randamo kelio ilgį ir tyrinėjamą paieškos erdvę.

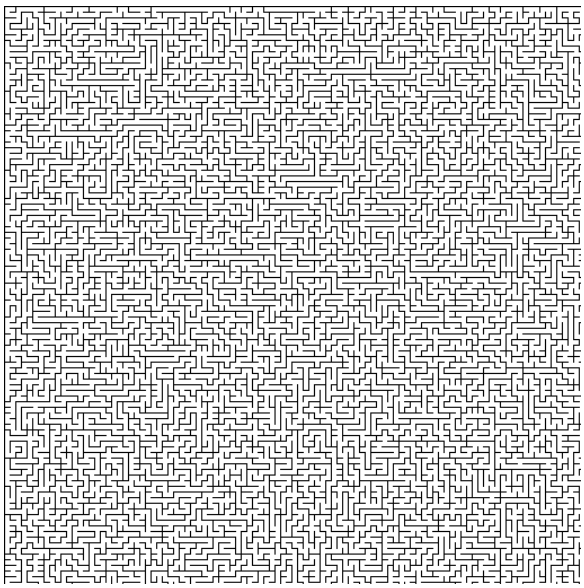
Eksperimentiniam tyrimui naudoti kelio paieškos algoritmų testavimui skirti žemėlapiai ir scenarijai [34]. Norint įvertinti algoritmų efektyvumą įvairiuose scenarijuose parinkti žemėlapiai su įvairiais kliūčių dėstymais:

1. strateginio žaidimo žemėlapiai:
    a. Enigma – 768x768 dydis, 3370 uždavinių, didelis Starcraft žemėlapis, mažo ploto segmentai (žr. **2 priedą**);
    b. Brokensteppes – 768x768 dydis, 3000 uždavinių, didelis Starcraft žemėlapis, didelio ploto segmentai (žr. **3 priedą**);
2. labirintai:
    a. maze512-4-0 – 512x512 dydis, 10510 uždavinių, 1 langelio pločio sienos, 4 langelių pločio praėjimai (žr. **4 priedą**);
    b. maze512-32-0 – 512x512 dydis, 6170 uždavinių, 1 langelio pločio sienos, 32 langelių pločio praėjimai (žr. **5 priedą**);
3. kambariai:
    a. 8room_000 – 512x512 dydis, 2140 uždavinių, atsitiktiniai sujungimai, 1 langelio pločio sienos, 8 langelių pločio kvadratiniai kambariai (žr. **6 priedą**);
    b. 64room_003 – 512x512 dydis, 2160 uždavinių, atsitiktiniai sujungimai, 1 langelio pločio sienos, 8 langelių pločio kvadratiniai kambariai (žr. **7 priedą**);
4. atsitiktinai generuoti žemėlapiai:
    a. random512-35-1 – 512x512 dydis, 2310 uždavinių, tankios atsitiktinės kliūtys (žr. **8 priedą**);
    b. random512-10-1 – 512x512 dydis, 1810 uždavinių, retos atsitiktinės kliūtys (žr. **9 priedą**).

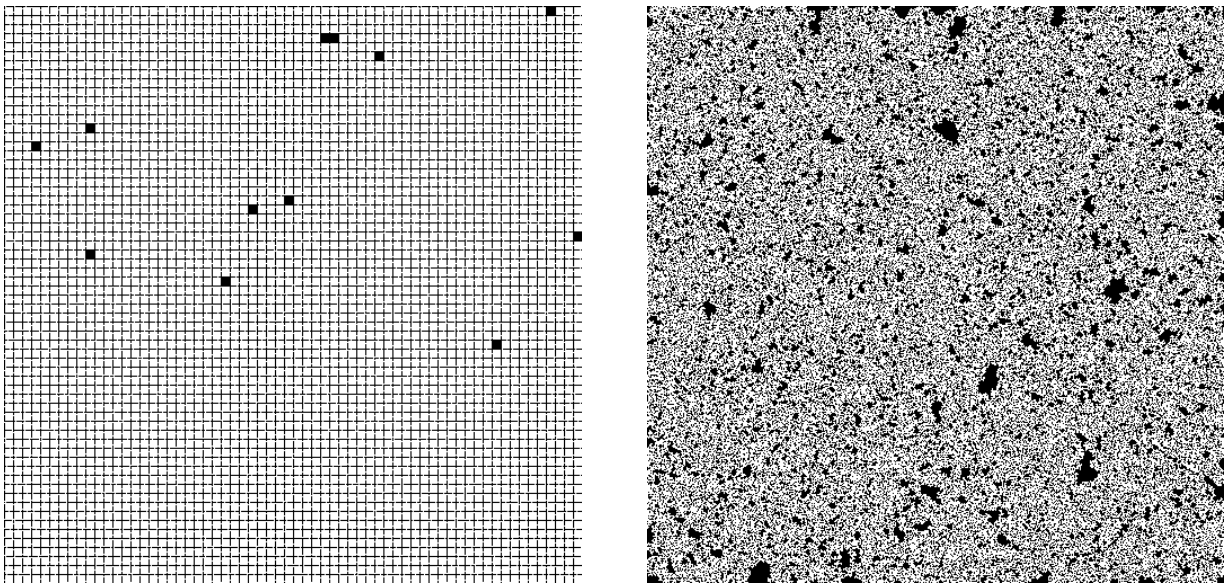
Dalis žemėlapių pavaizduoti **39 pav.** Strateginio žaidimo žemėlapiai turėtų būti palankesni *CHPA** ir *JPS* algoritmui, dėl didesnių atvirų plotų, bei geriau pavaizduoti algoritmų veikimą vaizdo žaidimo aplinkoje. Atsitiktinai generuoti žemėlapiai turi daug įvairių kliūčių išdėstymų ir parodys vidutinį *CHPA** algoritmo veikimą. Kambarių bei labirintų žemėlapiai parodys algoritmų gebėjimą atlikti kelio paiešką sudėtingose aplinkose.



a)



b)



c) d)

39 pav. Testavimui naudoti žemėlapiai: a) *Enigma*, b) *maze512-4-0*, c) *8room_000*, d) *random512-35-1*

Algoritmų konfigūracijos parinktos norint patikrinti specifinius scenarijus ir spėti atlikti eksperimentinius tyrimus per priimtiną darbo laiką. Naudotos konfigūracijos aprašomos šiais sutrumpinimais:

1. 4W, 8+4W, 8+8W – nusako naudotą žemėlapio jungimą. Skaičius nurodo krypčių kiekį, skaičiaus eiliškumas – atitinkamai tikro žemėlapio ir abstraktaus sluoksnio jungimas;
2. PT / AT – koordinačių perskaičiavimo metodas, iš anksto aprašytas perskaičiavimas (angl. *predetermined translation*) ir *A** perskaičiavimas (angl. *A* translation*) atitinkamai;
3. PS1, PS2, ... – *Pstep* parametras, skaičius nurodo *Pstep* reikšmę;
4. DF – abstrakcijos sluoksnio formavimo taisymo taikymas (angl. *diagonal fix*), minusas arba žodis „be“ nusako šio pataisymo nebuvimą.

Šiame projekte naudotos konfigūracijos:

1. A* 4W;
2. CHPA* 4W + PT + PS1;
3. CHPA* 4W + PT + PS2;
4. CHPA* 4W + PT + PS3;
5. CHPA* 4W + PT + PS4;
6. CHPA* 4W + AT + PS1;
7. CHPA* 4W + AT + PS2;
8. CHPA* 4W + AT + PS3;
9. CHPA* 4W + AT + PS4;
10. CHPA* 4W + PT + PS1 – DF;
11. CHPA* 4W + AT + PS1 – DF;
12. A* 8W;
13. JPS 8W;
14. CHPA* 8+4W + PT + PS2;
15. CHPA* 8+8W + PT + PS2;
16. CHPA* 8+4W + AT + PS2;

17. CHPA* 8+8W + AT + PS1;
18. CHPA* 8+8W + AT + PS2;
19. CHPA* 8+8W + AT + PS3;
20. CHPA* 8+8W + AT + PS4;
21. CHPA* 8+4W + PT + PS1 – DF;
22. CHPA* 8+8W + PT + PS1 – DF;
23. CHPA* 8+4W + PT + PS1;
24. CHPA* 8+8W + PT + PS1.

Paieškai 4-ių krypčių jungimo žemėlapiuose ir abstrakcijos sluoksniuose buvo naudojama Manheteno atstumo funkcija kaip euristinė funkcija, o 8-ių krypčių jungimui – įstrižo atstumo.

Rezultatuose pateikta kelio ilgio santykinė paklaida skaičiuota pagal (4) formulę:

$$e = \frac{hl - ol}{ol} * 100\%; \tag{4.1}$$

čia *hl* yra kelio paieškos algoritmo rasto kelio atstumas ir *ol* yra optimalaus kelio, rasto su *A** algoritmu, ilgis.

4.3. *CHPA** abstrakcijos sluoksnio formavimo pataisymo efektyvumas

Šio tyrimo metu siekiama nustatyti abstrakcijos sluoksnio formavimo klaidos bei pataisymo poveikį kelio ilgiui bei šios problemos dydį. **1 lentelėje** pavaizduoti nepavykusių *CHPA** kelio paieškų žemėlapyje rezultatai procentais.

Šiuose rezultatuose galima pastebėti, kad *CHPA** algoritmas su abstrakcijos sluoksnio pataisymu visuose žemėlapiuose rado vienodą arba mažesnį kiekį kelių nei algoritmas be šio pataisymo. Tokio rezultato ir tikėtasi, kadangi pataisymas naikina kraštinių skaičių, o tos panaikintos kraštinės gali būti vienintelis kelias į žemėlapio regionus, todėl algoritmas praranda gebėjimą į juos patekti.

Didžiausias šio pakeitimo poveikis matomas *random512-35-1* žemėlapyje 4-ių krypčių jungime, kur nerastų kelių kiekis padidėjo net 25 % ir sudaro 21,3 % šio žemėlapio scenarijaus uždavinių. 8-ių krypčių jungime nerastų kelių kiekis padidėjo 14% ir sudaro tik 3,72 %. 8-ių krypčių jungime yra daugiau būdų apeiti kliūtis, todėl šiame atsitiktinai generuotame žemėlapyje padidėja randamų kelių kiekis. *CHPA** algoritmas su abstrakcijos sluoksnio pataisymu žemėlapyje *random512-10-1* nerado tik 0,06 % kelių, kadangi šiame žemėlapyje daug rečiau pasirodo nepalankios algoritmui situacijos.

Žemėlapiuose *8room_000*, *64room_003*, *maze512-32-0* ir *Brokensteppes* rezultatas nepakito tarp skirtingų konfigūracijų bei žemėlapio jungimų. Šių žemėlapių bendras bruožas yra 1 langelio pločio kardinalių krypčių arba platesnės įvairių krypčių sienos, todėl jie nebuvo paveikti abstrakcijos sluoksnio formavimo pataisymu. Vienintelė išimtis buvo *maze512-4-0* žemėlapis, kuriame tik vienas uždavinys tapo neįmanomas po pataisymo taikymo.

Bendrai visuose žemėlapiuose *CHPA** nerado 6,82 % kelių po pataisymo taikymo 4-ių jungime (4,92 % daugiau nei be pataisymo) bei 5,52 % 8-ių krypčių jungime (0,07 % daugiau nei po pataisymo).

1 lentelė. Nepavykusių *CHPA** kelio paieškų kiekio rezultatai

Algoritmo konfigūracija		**Nepavykusių kelio paieškų kiekis, %**								
		Enigma	**Brokensteppes**	**maze512-4-0**	**maze512-32-0**	**8room_000**	**64room_003**	**random512-35-1**	**random512-10-1**	**Visi**
CHPA* 4W + PT + PS1	DF	0,89	0,77	9,50	5,79	10,19	1,25	21,34	0,06	6,82
	Be DF	0,86	0,77	9,49	5,79	10,19	1,25	17,06	0,00	6,50
CHPA* 4W + AT + PS1	DF	0,89	0,77	9,50	5,79	10,19	1,25	21,34	0,06	6,82
	Be DF	0,86	0,77	9,49	5,79	10,19	1,25	17,06	0,00	6,50
CHPA* 8+4W + PT + PS1	DF	0,86	0,77	9,50	5,79	10,19	1,25	3,72	0,00	5,52
	Be DF	0,83	0,77	9,49	5,79	10,19	1,25	3,25	0,00	5,48
CHPA* 8+8W + PT + PS1	DF	0,86	0,77	9,50	5,79	10,19	1,25	3,72	0,00	5,52
	Be DF	0,83	0,77	9,49	5,79	10,19	1,25	3,25	0,00	5,48

Abstrakcijos sluoksnio pataisymas taip pat paveikė ir randamų kelių ilgį, todėl toliau apžvelgsime jų rezultatus, kurie pavaizduoti **2 lentelėje**.

Rezultatai žaidimo, labirintų ir kambarių žemėlapiuose buvo paveikti ganėtinai menkai – mažiau nei 1 % kelių pagerėjo arba pablogėjo, išskyrus „Enigma“ žemėlapį, kuriame teigiamai paveikti buvo 6,7 % kelių. Kaip ir rezultatai **1 lentelėje**, atsitiktinai generuoti žemėlapiai buvo labiausiai paveikti, kur *random512-35-1* žemėlapyje buvo teigiamai paveikti net 53,94 % kelių ir 26,2 % - neigiamai.

2 lentelė. Pagerėjusių *CHPA** kelio paieškų kiekiai lyginant su algoritmu be abstrakcijos sluoksnio pataisymo

Algoritmo konfigūracija		**Rasto kelio ilgių skirtumai, %**								
		Enigma	**Brokensteppes**	**maze512-4-0**	**maze512-32-0**	**8room_000**	**64room_003**	**random512-35-1**	**random512-10-1**	**Visi**
CHPA* 4W + PT + PS1	Trumpesni	6,77	0,00	0,67	0,00	0,00	0,00	53,94	3,54	4,55
	Ilgesni	0,00	0,00	0,42	0,00	0,00	0,00	26,20	0,61	1,80
CHPA* 4W + AT + PS1	Trumpesni	6,77	0,00	0,67	0,00	0,00	0,00	53,94	3,54	4,55
	Ilgesni	0,00	0,00	0,42	0,00	0,00	0,00	26,20	0,61	1,80
CHPA* 8+4W + PT + PS1	Trumpesni	6,76	0,00	0,67	0,00	0,00	0,00	48,11	3,59	4,79
	Ilgesni	0,00	0,00	0,42	0,00	0,00	0,00	21,31	0,61	1,77
CHPA* 8+8W + PT + PS1	Trumpesni	6,76	0,00	0,67	0,00	0,00	0,00	48,11	3,59	4,79
	Ilgesni	0,00	0,00	0,42	0,00	0,00	0,00	21,31	0,61	1,77

Bendrai, iš visuose žemėlapiuose *CHPA** algoritmo rastų kelių, daugiau nei 4,5 % pagerėjo ir mažiau nei 1,9 % pablogėjo, tačiau šis rezultatas daugiausiai atspindi poveikį atsitiktinai generuotuose

žemėlapiuose. Neįtraukus atsitiktinių žemėlapių rezultatų į skaičiavimus teigiamai paveikti buvo apie 0,98 % kelių ir 0,13 % neigiamai.

Abstrakcijos sluoksnio formavimo taisymas daugiausiai teigiamai paveikė algoritmo randamo kelio ilgį, todėl tolimesniuose tyrimuose visur bus naudojamas šis abstrakcijos sluoksnio formavimo pataisymas.

4.4. Abstrakcijos sluoksnio jungimo 8-iomis kryptimis poveikis randamam keliui

4-ių krypčių jungimo abstrakcijos sluoksnio taikymo 8-ių krypčių jungimo žemėlapiui poveikis randamo kelio ilgiui ir algoritmo greitaveikai nežinomas, todėl buvo atlikti tyrimai norint tai įvertinti.

Paklaidų nuo optimalaus sprendimo rezultatai 8-ių krypčių jungimo tyrimui pavaizduoti **3 lentelėje**. Kaip ir nepavykusių paieškų rezultatuose, *maze512-4-0* žemėlapio rezultatai yra labai prasti. Antra blogiausia kelio ilgio paklaida gauta *random512-35-1* žemėlapyje (51,57 %), tačiau *maze512-4-0* žemėlapio rezultatai yra net penkis kartus prastesni (260,16 %). Šis rezultatas rodo *maze512-4-0* žemėlapio netikimą *CHPA** algoritmo veikimui.

*CHPA** algoritmas, naudodamas 8+4W konfigūraciją, visuose žemėlapiuose gavo prastesnius kelio ilgio paklaidos rezultatus nei 4W konfigūracija, išskyrus atsitiktinai generuotuose žemėlapiuose.

3 lentelė. Vidutiniai randamo kelio ilgio paklaidos rezultatai 8-ių krypčių jungimo analizei

Algoritmo konfigūracija	**Kelio ilgio santykinė paklaida, %**								
	Enigma	**Brokensteppes**	**maze512-4-0**	**maze512-32-0**	**8room_000**	**64room_003**	**random512-35-1**	**random512-10-1**	**Visi**
CHPA* 4W + PT + PS2	1,21	0,93	260,16	1,66	15,72	17,19	51,57	9,66	91,03
CHPA* 8+4W + PT + PS2	10,14	11,24	261,16	13,00	19,85	30,03	20,53	20,04	94,55
CHPA* 8+8W + PT + PS2	0,83	0,60	246,42	1,45	13,06	16,58	9,72	4,01	82,28

Žvelgiant į 8+8W konfigūracijos rezultatus galima pastebėti, kad visuose žemėlapiuose kelio ilgio paklaidos yra geresnės, negu 8+4W ir net 4W konfigūracijos rezultatuose. 8+8W konfigūracijos atsitiktinių žemėlapių rezultatai nebe prasčiausi, todėl, kad 8-ių krypčių jungime, *CHPA** algoritmas abstrakcijos segmente turi daug mažesnę galimybę sudaryti kritinę situaciją ir abstrakcijos sluoksnio formavimo klaidą, kurios smarkiai didina randamo kelio ilgį.

Kaip ir tikėtasi, 8+4W konfigūracijos rezultatai yra prastesni (94,55 %), nei 8+8W konfigūracijos (82,28 %), kadangi paieška 4-ių krypčių jungimo abstrakcijos sluoksnyje randa kelius su daug tiesių atkarpų, todėl tikrame žemėlapyje algoritmas neturi daug galimybių keliauti įstrižai tarp tarpinių taškų.

8+8W konfigūracijos blogiausi rezultatai gauti kambarių žemėlapiuose, tačiau jų santykinis skirtumas nuo 4-ių krypčių grafo rezultatų yra panašus į kitų žemėlapių, kadangi jų pagrindinė problema liko neišspręsta. Ši problema yra siauros ir ilgos kliūtys, kurios yra problematiškos *CHPA** algoritmui ir didina randamo kelio ilgį.

Bendrai pažvelgus, *CHPA** algoritmas, naudodamas 8+8W konfigūraciją, vidutiniškai randa 10,6 % trumpesnį kelią, nei naudodamas 4W konfigūraciją.

Žinoma, vidutinis randamo kelio ilgis yra tik viena rezultatų dalis, todėl reikia apžvelgti ir greitaveikos rezultatus, kurie yra pavaizduoti **4 lentelėje**.

4 lentelė. Vidutiniai greitaveikos rezultatai 8-ių krypčių jungimo taikymui 8-ių krypčių jungimo analizei

Algoritmo konfigūracija		**Vidutinis kelio paieškos laikas, s**									
		Enigma	**Brokensteppes**	**maze512-4-0**	**maze512-32-0**	**8room_000**	**64room_003**	**random512-35-1**	**random512-10-1**	**Visi**	**Visi***
CHPA* 4W + PT + PS2	Bendras	0,19	0,60	5,41	0,33	0,20	0,48	0,18	0,01	1,96	0,31
	Abs. sl.	0,18	0,59	0,11	0,31	0,19	0,22	0,17	0,01	0,22	0,22
CHPA* 8+4W + PT + PS2	Bendras	0,20	0,61	10,19	0,34	0,21	0,83	0,10	0,01	3,49	0,34
	Abs. sl.	0,18	0,60	0,12	0,31	0,20	0,23	0,09	0,00	0,21	0,21
CHPA* 8+8W + PT + PS2	Bendras	0,34	0,84	10,15	0,58	0,29	1,04	0,15	0,15	3,60	0,52
	Abs. sl.	0,33	0,83	0,20	0,56	0,28	0,36	0,15	0,15	0,36	0,36

* Išskyrus *maze512-4-0*

Atvirkščiai nei kelio ilgio paklaidos rezultatai, greitaveikos rezultatai vidutiniškai visuose žemėlapiuose yra 80% prastesni 8-ių krypčių jungimo grafe ir prasčiausi 8+8W konfigūracijoje. Žinoma, išimtis yra atsitiktinai generuoti žemėlapiai, kadangi 8-ių krypčių jungimas padidina praeinamų segmentų skaičių, todėl keliai žemėlapyje trumpėja ir paieškos laikas mažėja. Mažesnio tankumo žemėlapyje *random512-10-1* algoritmas vidutiniškai kelią rado taip pat greitai 4W konfigūracijoje, kaip ir 8+4W (0,01s), todėl, kad 8-ių krypčių jungimas turėjo mažesnį poveikį kelio ilgiui.

Tarp kambarių žemėlapių rezultatų galima pastebėti greitaveikos skirtumą, kur žemėlapio *8room_000* bendri greitaveikos rezultatai yra 0,19 - 0,29 s tarpe, o *64room_003* žemėlapyje – 0,48 - 1,04 s. Žemėlapyje *8room_000* bendro paieškos laiko ir abstrakcijos sluoksnyje skirtumas atskleidžia, kad žemėlapyje *64room_003* algoritmas praleidžia 2 - 3 kartus daugiau laiko atlikdamas kelio paiešką tikrame žemėlapyje. Taip yra dėl to, kad algoritmo randamas kelias yra užstojamas sienų, dėl abstrakcijos sluoksnyje susidariusių klaidų. Tankesniame kambarių žemėlapyje *8room_000* ši klaida taip pat egzistuoja, tačiau algoritmui tenka apeiti trumpesnius kliūčių segmentus, todėl žemėlapyje randamo kelio ilgis yra geresnis ir kelio paieška atliekama greičiau.

Labirinto žemėlapyje *maze512-4-0* kelio paieška vidutiniškai truko net 5,4 - 10,1 s, o kituose žemėlapiuose – mažiau nei 1 s. Šie lėtos greitaveikos rezultatai atitinka didelę kelio ilgio santykinę paklaidą.

Rezultatai parodė, kad šio darbo metu įgyvendinta 8+4W algoritmo konfigūracija vidutiniškai atlieka kelio paiešką 65% greičiau, nei 8+8W konfigūracija. Žemėlapiai *maze512-4-0* ir *64room_003* yra netinkami *CHPA** algoritmo naudojimui, todėl tolimesniuose rezultatuose jie nebus įtraukiami.

4.5. *Pstep* poveikis randamo kelio ilgiui naudojant naują koordinačių perskaičiavimo metodą

Pirmosios *CHPA** versijos tyrime vidutinis randamo kelio ilgis 4-ių krypčių jungimo žemėlapiuose pagerėjo 8 % pakeitus *Pstep* reikšmę iš 1 į 2, tačiau 8-ių krypčių žemėlapiuose *Pstep* poveikis gali būti skirtingas. Šio eksperimentinio tyrimo testavimo apimtys ir žemėlapiai skiriasi nuo pirmosios *CHPA** algoritmo versijos testavimo, todėl 4-ių krypčių testavimas buvo pakartotas.

*CHPA** algoritmo randamo kelio ilgio rezultatai naudojant įvairias *Pstep* reikšmes pavaizduoti **5 lentelėje**. Čia galima pastebėti, kad visose 4W konfigūracijose algoritmas, naudojant naują koordinačių perskaičiavimo metodą ir *Pstep=1*, vidutiniškai gavo 2,22 % mažesnę santykinę kelio ilgio paklaidą. Didžiausias pagerėjimas matomas atsitiktinai generuotuose žemėlapiuose, kur vidutiniškai paklaida yra iki ketvirčio mažesnė.

5 lentelė. Kelio paieškos, naudojant įvairias *Pstep* reikšmes, ilgio paklaidos rezultatai

Algoritmo konfigūracija		**Kelio ilgio santykinė paklaida, %**						
		Enigma	**Brokensteppes**	**maze512-32-0**	**8room_000**	**random512-35-1**	**random512-10-1**	**Visi***
CHPA* 4W + PT	PS1	1,41	1,05	1,72	19,47	64,99	13,11	11,15
	PS2	1,21	0,93	1,66	15,72	51,57	9,66	8,93
	PS3	1,07	0,83	1,61	12,94	44,72	7,72	7,66
	PS4	0,98	0,77	1,52	11,25	40,63	6,49	6,88
CHPA* 4W + AT	PS1	1,20	0,99	1,67	17,07	50,25	9,92	8,97
	PS2	1,03	0,87	1,59	13,72	43,46	7,62	7,60
	PS3	0,95	0,78	1,54	11,19	38,77	6,27	6,66
	PS4	0,88	0,72	1,46	9,85	35,90	5,38	6,08
CHPA* 8+8W + AT	PS1	0,71	0,49	1,29	13,88	10,24	3,77	3,74
	PS2	0,56	0,42	1,22	10,92	7,63	3,35	3,00
	PS3	0,60	0,49	1,33	9,12	6,25	3,21	2,68
	PS4	0,55	0,44	1,29	8,01	5,34	2,83	2,38

* Išskyrus *maze512-4-0* ir *64room_003*

4W konfigūracijose didinant *Pstep* reikšmę, sumažėja ir vidutinė paklaida, tačiau šis pokytis nėra tiesinis. PT konfigūracijos bendra paklaida sumažėjo 19,9 % tarp PS1 ir PS2, o tarp PS3 ir PS4 – 10,2 %. AT konfigūracijos bendra paklaida tiems patiems *Pstep* sumažėjo 15,2 % ir 8,7 %. Žvelgiant į 8W konfigūraciją, strateginio žaidimo žemėlapiuose *Pstep* poveikis pasikeitė, kur paklaida tarp PS2 ir PS3 padidėja, o tarp PS3 ir PS4 – vėl sumažėja. Ši anomalija pakankamai bendros paklaidos rezultatų neiškreipia ir matomas randamo kelio ilgio mažėjimas su didėjančia *Pstep* reikšme.

Pstep poveikis vidutiniam paieškos laikui tikrame žemėlapyje pavaizduotas **6 lentelėje**. 4W konfigūracijose vidutinis paieškos laikas mažėja <1 ms didėjant *Pstep* reikšmei, tačiau vėl pakyla PS4 konfigūracijoje 0,2-0,3 ms. Ši PS4 konfigūracijos rezultatų anomalija matoma beveik visų

žemėlapių rezultatuose. AT metodas vidutiniškai atliko paiešką 0,3-1,2 ms greičiau, tačiau tai yra smulkus skirtumas ir sudaro tik apie 4,1 % bendros kelio paieškos trukmės.

8W konfigūracijoje kelio paieškos laikas didėja su *Pstep* reikšme. Taip pat rezultatų skirtumas tarp PS1-PS4 yra netiesinis – 5,7 %, 19,8 % ir 26,7 %.

6 lentelė. Įvairių *Pstep* reiškmių greitaveikos rezultatai tikrame žemėlapyje

Algoritmo konfigūracija		**Vidutinis kelio paieškos laikas, ms**						
		Enigma	**Brokensteppes**	**maze512-32-0**	**8room_000**	**random512-35-1**	**random512-10-1**	**Visi***
CHPA* 4W + PT	PS1	10,80	9,70	19,93	8,42	15,65	6,09	13,38
	PS2	9,96	9,12	18,27	7,79	13,22	5,50	12,20
	PS3	9,81	8,95	17,10	7,85	13,20	5,48	11,75
	PS4	9,93	8,74	17,36	8,97	12,87	5,53	11,92
CHPA* 4W + AT	PS1	10,31	9,48	18,03	7,28	13,33	5,55	12,20
	PS2	9,73	8,68	16,95	7,03	12,33	5,19	11,44
	PS3	9,85	8,60	16,42	7,59	12,07	5,24	11,31
	PS4	9,89	8,73	16,91	8,35	12,22	5,53	11,63
CHPA* 8+8W + AT	PS1	13,33	12,40	24,31	10,97	6,54	6,74	14,96
	PS2	13,99	12,90	25,28	13,13	7,27	6,91	15,81
	PS3	16,76	14,48	29,76	16,58	9,52	9,71	18,95
	PS4	23,84	18,90	37,06	19,65	11,18	11,32	24,02

Didesnė *Pstep* reikšmė mažina ir randamo kelio ilgį, tačiau 8W konfigūracijoje taip pat auga kelio paieškos laikas tikrame žemėlapyje. Kelio paieška tikrame žemėlapyje sudaro mažą bendros kelio paieško trukmės dalį, todėl *Pstep* parinkimas turėtų būti atliekamas pagal kelio optimalumo poreikį. Tolimesniuose rezultatuose bus naudojama tik *Pstep=2*.

4.6. Naujo koordinačių perskaičiavimo metodo efektyvumas

Naujas koordinačių perskaičiavimo metodas sukurtas sumažinti *CHPA** algoritmo randamo kelio ilgį, tačiau šis pakeitimas galimai paveiks tyrinėjamą paieškos erdvę bei greitaveiką.

Aplankytų viršūnių rezultatai pateikti **7 lentelėje**. Čia galima pastebėti, kad naujas koordinačių perskaičiavimo metodas visuose žemėlapiuose patikrino mažiau viršūnių nei senas metodas, išskyrus žemėlapį *random512-35-1* su *CHPA** 8+4W + AT + PS2 konfigūracija. Didžiausias poveikis patikrinamų viršūnių kiekiui matomas 8+4W konfigūracijose, kur vidutiniškai patikrinamų viršūnių kiekis sumažėjo 0,07 % tarp PT ir AT metodų. 4W ir 8+8W konfigūracijose patikrinamų viršūnių pokytis tik 0,04 % ir 0,02 % atitinkamai, todėl galima teigti, kad AT metodas vidutiniškai sumažina patikrinamų viršūnių kiekį.

7 lentelė. Abiejų *CHPA** koordinačių perskaičiavimo metodų patikrinamų viršūnių rezultatai lyginant su *A**

Algoritmo konfigūracija	Vidutinis patikrinamų viršūnių kiekis lyginant su *A**, %						
	Enigma	Brokensteppes	maze512-32-0	8room_000	random512-35-1	random512-10-1	Visi*
CHPA* 4W + PT + PS2	16,82	15,92	14,43	22,53	25,69	48,23	16,07
CHPA* 4W + AT + PS2	16,82	15,91	14,43	22,37	25,17	47,76	16,03
CHPA* 8+4W + PT + PS2	17,79	15,99	14,89	17,91	21,14	15,94	15,67
CHPA* 8+4W + AT + PS2	17,67	15,90	14,83	17,77	21,16	15,76	15,60
CHPA* 8+8W + PT + PS2	17,13	15,97	14,57	19,38	21,25	20,20	15,53
CHPA* 8+8W + AT + PS2	17,12	15,97	14,57	19,10	21,11	20,07	15,51

* Išskyrus *maze512-4-0* ir *64room_003*

Toliau apžvelgsime greitaveikos rezultatus, kurie gauti naudojant skirtingus koordinačių perskaičiavimo metodus ir pateikti **8 lentelėje**. AT metodo greitaveikos rezultatai visuose žemėlapiuose yra geresni nei PT metodo, išskyrus žemėlapyje *random512-35-1* tarp 8+4W konfigūracijų (0,08 ms) ir žemėlapyje *Brokensteppes* tarp 8+8W konfigūracijų (0,07 ms). Mažiausias 1,88 % vidutinio kelio paieškos trukmės pokytis matomas tarp 8+8W konfigūracijų, o didžiausias 6,23 % pokytis tarp 4W konfigūracijų. Šie rezultatai atitinka tikrinamų viršūnių rezultatus, kur AT metodas vidutiniškai patikrino mažiau viršūnių negu PT metodas, o mažesnį kiekį viršūnių patikrinti trunka mažiau laiko.

8 lentelė. Abiejų *CHPA** koordinačių perskaičiavimo metodų greitaveikos rezultatai tikrame žemėlapyje

Algoritmo konfigūracija	Vidutinis kelio paieškos laikas, ms						
	Enigma	Brokensteppes	maze512-32-0	8room_000	random512-35-1	random512-10-1	Visi*
CHPA* 4W + PT + PS2	9,96	9,12	18,27	7,79	13,22	5,50	12,20
CHPA* 4W + AT + PS2	9,73	8,68	16,95	7,03	12,33	5,19	11,44
CHPA* 8+4W + PT + PS2	15,99	15,76	31,09	12,33	7,41	8,77	18,64
CHPA* 8+4W + AT + PS2	15,42	14,61	29,77	11,82	7,42	8,65	17,86
CHPA* 8+8W + PT + PS2	14,11	12,82	25,77	14,11	7,49	6,94	16,12
CHPA* 8+8W + AT + PS2	13,99	12,90	25,28	13,13	7,27	6,91	15,81

* Išskyrus *maze512-4-0* ir *64room_003*

Naujas koordinačių perskaičiavimo metodas buvo sukurtas pagerinti randamo kelio ilgį, todėl galiausiai apžvelgsime *CHPA** randamo kelio ilgio santykinės paklaidos rezultatus, kurie pavaizduoti **9 lentelėje**. Šiuose rezultatuose taip pat pavaizduoti *CHPA** algoritmui netinkamų žemėlapių

rezultatai, kad būtų galima įvertinti AT metodo efektyvumą atliekant kelio paiešką tarp klaidingai suformuotų abstrakcijos segmentų.

9 lentelė. Abiejų *CHPA** koordinačių perskaičiavimo metodų randamo kelio ilgio santykinės paklaidos rezultatai

Algoritmo konfigūracija	**Kelio ilgio santykinė paklaida, %**									
	Enigma	**Brokensteppes**	**maze512-4-0**	**maze512-32-0**	**8room_000**	**64room_003**	**random512-35-1**	**random512-10-1**	**Visi**	**Visi***
CHPA* 4W + PT + PS2	1,21	0,93	260,16	1,66	15,72	17,19	51,57	9,66	91,03	8,93
CHPA* 4W + AT + PS2	1,03	0,87	200,37	1,59	13,72	15,83	43,46	7,62	70,73	7,60
CHPA* 8+4W + PT + PS2	10,14	11,24	261,16	13,00	19,85	30,03	20,53	20,04	94,55	14,54
CHPA* 8+4W + AT + PS2	8,62	9,75	201,02	11,76	18,37	27,09	18,34	18,36	74,17	13,02
CHPA* 8+8W + PT + PS2	0,83	0,60	246,42	1,45	13,06	16,58	9,72	4,01	82,28	3,70
CHPA* 8+8W + AT + PS2	0,56	0,42	193,00	1,22	10,92	13,84	7,63	3,35	64,56	3,00

* Išskyrus *maze512-4-0* ir *64room_003*

Žemėlapio *maze512-4-0* rezultatuose mažiausia PT metodo paklaida yra 246,42 %, tačiau AT metodas visose konfigūracijose pagerino paklaidą daugiau nei 50 % lyginant su PT metodu. Žemėlapis *64room_003* yra taip pat netinkamas *CHPA** algoritmui, kadangi didžiausia vidutinė kelio ilgio paklaida yra 30 %. Šiame žemėlapyje didžiausias paklaidos pokytis yra 2,9 %, naudojant 8+4W konfigūraciją.

Santykinė kelio ilgio paklaida, naudojant AT metodą, yra geresnė nei PT metodo visuose žemėlapiuose, be jokių išimčių. Neatsižvelgiant į *CHPA** netinkamus žemėlapius, didžiausias bendras vidutinės paklaidos pokytis matomas naudojant 8+4W konfigūraciją (1,51 %), tačiau didžiausias santykinis pokytis – naudojant 8+8W konfigūraciją (19,06 %).

4.7. Įgyvendintų algoritmų palyginimas

Paskutiniam palyginimui parinktos *CHPA** konfigūracijos su nauju koordinačių perskaičiavimu, *Pstep=2* ir abiem abstrakcijos sluoksnio jungimo būdais. Pirma bus apžvelgiami vidutiniai greitaveikos ir tikrinamos paieškos erdvės rezultatai, kad būtų galima įžvelgti kiekvienam kelio paieškos algoritmui tinkamus žemėlapius.

10 lentelėje galima pastebėti, kad abi *CHPA** konfigūracijos vidutiniškai kelią randa greičiau nei *JPS* algoritmas daugiau nei pusėje žemėlapių, tačiau tai nusako tik skirtumą tarp šio darbo algoritmų realizacijų. Atsitiktinai generuotuose žemėlapiuose *JPS* algoritmas rodė prasčiausius rezultatus, kur žemėlapyje *random512-35-1 JPS* algoritmas vidutiniškai kelią rado daugiau nei 5 kartus lėčiau nei

*A**. Šio tipo žemėlapiai yra *JPS* algoritmo silpnybė, kadangi yra sukuriama daug šuolio taškų, kuriuos reikia rūšiuoti pagal kelio ilgį. *JPS* algoritmas rodė geriausius rezultatus žemėlapiuose, kurie turi daug tiesių ir ilgų kliūčių, tokie kaip labirintai ir kambariai. Šiuose žemėlapiuose *JPS* kelio paieška vidutiniškai truko mažiau nei 1,2 s. Vidutinė šio algoritmo kelio paieška strateginio žaidimo žemėlapiuose padidėja daugiau nei 2 kartus, lyginant su labirintais bei kambariais.

10 lentelė. Kelio paieškos algoritmų greitaveikos rezultatai

Algoritmo konfigūracija	**Vidutinis kelio paieškos laikas, s**								
	Enigma	**Brokensteppes**	**maze512-4-0**	**maze512-32-0**	**8room_000**	**64room_003**	**random512-35-1**	**random512-10-1**	**Visi**
A* 8W	7,80	23,79	4,82	14,35	7,93	14,05	2,22	4,70	9,46
JPS 8W	2,64	3,63	1,16	0,51	0,80	0,30	11,31	4,48	2,28
CHPA* 8+4W + AT + PS2	0,20	0,60	8,86	0,34	0,21	0,66	0,10	0,01	3,05
CHPA* 8+8W + AT + PS2	0,35	0,85	8,88	0,58	0,30	0,88	0,15	0,15	3,18

*A** algoritmas blogiausius rezultatus rodė *Brokensteppes, maze512-32-0* bei *64room_003* žemėlapiuose, kadangi šiuose žemėlapiuose kliūtys yra ilgos ir algoritmui reikia apžiūrėti didelius plotus, kad jas būtų galima apeiti optimaliai. *CHPA** algoritmas rodo panašią greitaveikos priklausomybę nuo kliūčių ilgio, kadangi jis naudoja tą patį *A** algoritmą kelio paieškai, tačiau su netinkamų žemėlapių išimtimi, tokių kaip *maze512-4-0*.

Kelio paieškos algoritmų tiriamo ploto rezultatai pavaizduoti **11 lentelėje**. Žemėlapio *Brokensteppes* rezultatai aiškiai parodo *JPS* algoritmo greito viršūnių tikrino logikos pranašumą, kur šis algoritmas vidutiniškai patikrino beveik 5,5 mln. viršūnių, o tai yra daugiau nei 90 kartų daugiau patikrintų viršūnių nei *A**. Tai reiškia, kad *JPS* algoritmas šiame žemėlapyje vidutiniškai patikrino net 1,5 mln. viršūnių per sekundę. Deja, bet šie rezultatai įtraukia ir pakartotinai *JPS* algoritmo patikrintas viršūnes, todėl patikrintų viršūnių kiekio tiesiogiai lyginti su kitais algoritmais nederėtų.

Strateginio žaidimo žemėlapiai taip pat pavaizduoja *JPS* algoritmo trūkumą, kuris yra pernelyg dideli atviri žemėlapio plotai. Šis algoritmas atlieka šuolį iki kliūties arba įdomaus taško visomis 8-iomis kryptimis, o šuolis į didelį atvirą plotą užtrunka daug laiko, todėl šiuose žemėlapiuose *JPS* algoritmas apžiūri daug viršūnių ir atlieką sąlyginai lėtą kelio paiešką. Atsitiktinai generuotuose žemėlapiuose *JPS* algoritmas apžiūrėjo mažiausiai viršūnių - tik po 99 tūkst., o vidutinė viršūnių tikrinimo sparta buvo tik 8,81 tūkst. per sekundę žemėlapyje *random512-35-1* ir 22,26 tūkst. per sekundę žemėlapyje *random512-10-1*.

Lyginant su *A**, *CHPA** algoritmas prasčiausius rezultatus gavo *random512-35-1* žemėlapyje, kur *CHPA** patikrino 4,7 karto mažiau viršūnių nei *A**, o kituose žemėlapiuose – 5-6 kartus mažiau nei *A** atitinkamai. Bendrai visuose žemėlapiuose, išskyrus *maze512-4-0* ir *64room_003*, vidutiniškai *CHPA** kiekvieną kelio paiešką patikrino 10,2 tūkst. viršūnių, o tai yra daugiau nei 6 kartus mažesnis kiekis nei *A**.

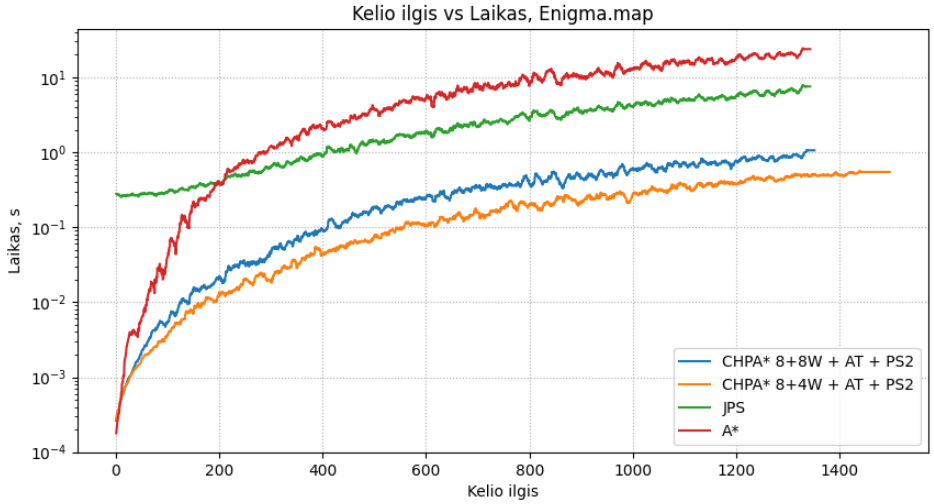
11 lentelė. Kelio paieškos algoritmų apžiūrimo ploto rezultatai

Algoritmo konfigūracija	**Vidutinis patikrinamų viršūnių kiekis, tūkst.**						
	Enigma	**Brokensteppes**	**maze512-32-0**	**8room_000**	**random512-35-1**	**random512-10-1**	**Visi***
A* 8W	51,27	57,43	126,55	28,01	13,29	12,46	65,91
JPS 8W	3239,14	5477,40	631,98	141,30	99,70	99,79	1700,04
CHPA* 8+4W + AT + PS2	9,06	9,13	18,77	4,98	2,81	1,96	10,28
CHPA* 8+8W + AT + PS2	8,78	9,17	18,44	5,35	2,81	2,50	10,22

* Išskyrus *maze512-4-0* ir *64room_003*

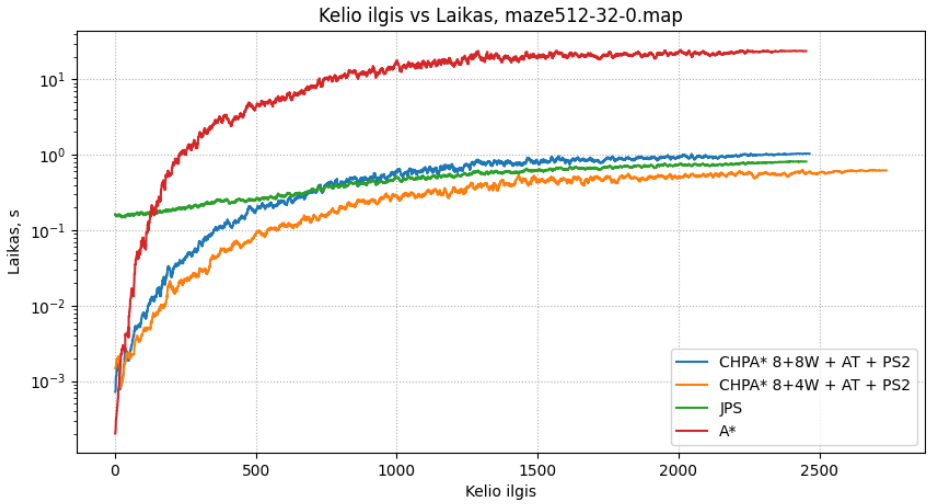
Galiausiai bus apžvelgiama algoritmų randamo kelio ilgio ir paieškos laiko priklausomybė. Visi algoritmai parodė sąlyginai gerus greitaveikos rezultatus žemėlapyje *Enigma*, todėl pirma apžvelgsime rezultatus šiame žemėlapyje. **40 pav.** pavaizduoti šio žemėlapio rezultatai, kur duomenys gauti naudojant slenkantį vidurkį su 25-ių narių imtimi.

Šiame žemėlapyje, *CHPA** algoritmas kelią randa daugiau nei 10 kartų greičiau lyginant su *A** algoritmu, kai kelio ilgis yra didesnis nei 100. *CHPA** 8+4W konfigūracija kelią randa iki 2-ių kartų greičiau nei 8+8W konfigūracija, tačiau randamas kelias yra ilgesnis.



40 pav. Kelio ilgio ir paieškos laiko rezultatai žemėlapyje *Enigma*

Žemėlapio *maze512-32-0* rezultatai pavaizduoti **41 pav.** Šiame žemėlapyje *CHPA** algoritmas taip pat vidutiniškai daugiau nei 10 kartų greičiau rado kelią nei *A**. *JPS* algoritmo vidutinė kelio paieška truko trumpiau nei *Enigma* žemėlapyje ir paieškos laikas didėja mažesne sparta. Taip pat šio algoritmo kelio paieška buvo atliekama sparčiau nei *CHPA** kai kelio ilgis didesnis nei 750.



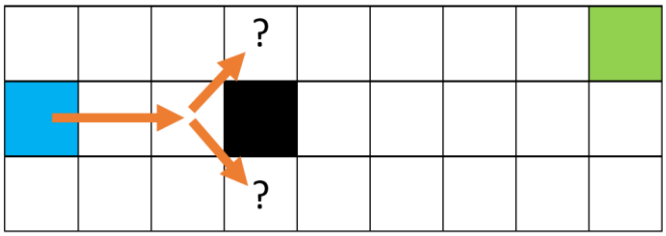
41 pav. Kelio ilgio ir paieškos laiko rezultatai žemėlapyje *maze512-32-0*

5. *CHPA** algoritmo problemos

Šiame skyriuje analizuojami *CHPA** algoritmo bruožai ir jų poveikis.

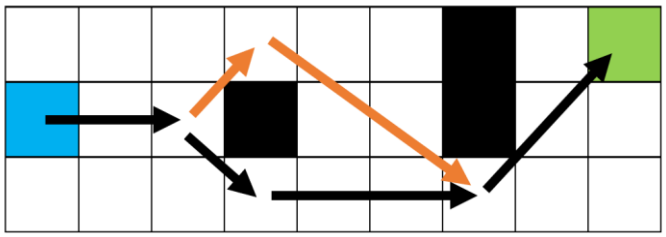
5.1. Naujas koordinačių perskaičiavimas

Naujasis koordinačių perskaičiavimo metodas leidžia *CHPA** algoritmui rasti trumpesnį kelią nei iš anksto aprašyto tikrinimo metodas. Mažiau palankiuose žemėlapiuose naujasis metodas geba pagerinti ir greitaveiką, tačiau jis nėra tobulas ir vis dar turi problemų didinančių randamo kelio ilgį. Pagrindinė problema pavaizduota **42 pav.**



42 pav. Naujo koordinačių perskaičiavimo metodo kliūties apėjimo dilema

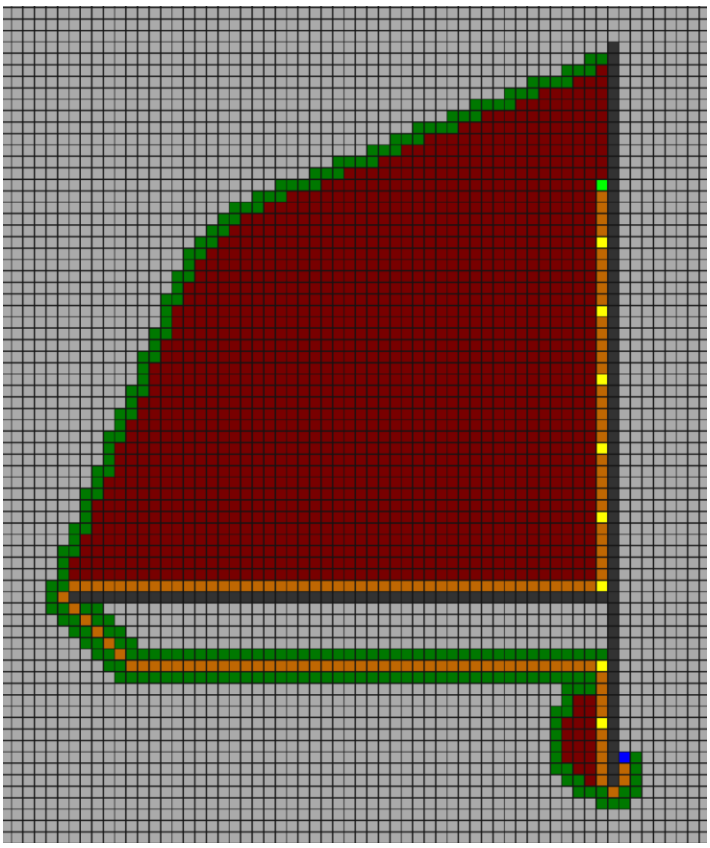
Naujasis koordinačių perskaičiavimo metodas sustabdo paiešką atradus pirmąjį tarpinio segmento tašką, tačiau susidūrus su kliūtimi šis metodas nežino kuria kryptimi optimaliai apeiti kliūtį, kadangi jis neturi informacijos apie tolimesnius žingsnius. Šio projekto *A** realizacija lygiąsias sprendžia iš anksto numatyta tvarka, panašiai kaip pirmosios versijos koordinačių perskaičiavimo metodas (žr. **32 pav.**), tačiau suteikus daugiau informacijos galima protingiau spręsti šią problemą. Deja, bet net ir suteikus informaciją apie kito eilėje segmento kryptį metodas nebūtų globaliai optimalus, kaip pavaizduota **43 pav.** oranžinėmis rodyklėmis.



43 pav. Naujo koordinačių perskaičiavimo metodo kliūties apėjimo dilema su informacija apie kitą segmentą, kai algoritmas galimai pasirinks neoptimalų oranžinį kelią

5.2. *CHPA** algoritmo abstrakcijos problemos

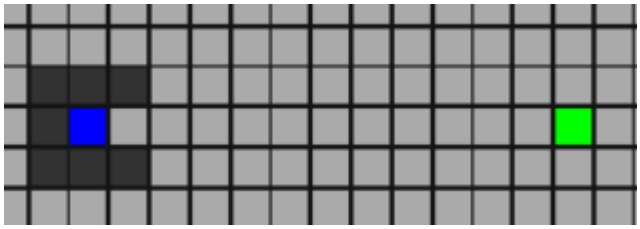
*CHPA** silpnybė yra kliūtys, kurios yra siauresnės ir ilgesnės nei algoritmo naudojamas segmentas. Sutikus tokias kliūtis *CHPA** algoritmo naudojamos prielaidos klaidingai sujungia abstrakcijos sluoksnio viršūnes ir mato kelią kur jo nėra tikrame žemėlapyje, kaip pavaizduota **44 pav.**



44 pav. Neoptimalus *CHPA** kelias dėl abstrakcijos sluoksnio klaidos

5.3. Nepasiekiamos žemėlapio dalys

Šiame darbe realizuotas algoritmas nėra išbaigtas (angl. *incomplete*), kadangi jis negarantuoja, kad suras kelią jeigu toks egzistuoja tikrame žemėlapyje. *CHPA** algoritmas apdorodamas segmentus atlieka paiešką iš kairės arba viršutinės segmento kraštinės, todėl segmentai, kuriuose šios kraštinės yra sudarytos iš kliūčių, tampa nepasiekiami abstrakcijos sluoksnyje, kaip pavaizduota **45 pav.**



45 pav. *CHPA** algoritmui nepasiekiamas kelias

6. Išvados

Atlikus šį kelio paieškos algoritmų, skirtų apytikslio maršruto radimui, tyrimą, priimtos šios išvados:

1. Projekto temos analizės metu apžvelgti kelio paieškos erdvių perteikimo būdai, jų problemos ir nustatyta, kad viena pagrindinių kvadratinio tinklo struktūros grafų problemų yra kelių simetrija. Simetrinių kelių naikinimu paremti metodai parodė, kad sumažinus simetrinių kelių kiekį, sumažėja paieškos erdvė ir spartėja kelio paieška. Analizės metu taip pat tyrinėti hierarchinės kelio paieškos metodai, kurie naudojo įvairius kelio lyginimo metodus, kad sumažinti randamo kelio ilgį.
2. *CHPA** kelio paieškos algoritmo analizės metu nustatytos klaida abstrakcijos sluoksnio formavimo procese, dėl kurios algoritmas mato praėjimą kur neturėtų, ir koordinačių perskaičiavimo metode, kur parenkamas neoptimalus tarpinis taškas ir randas ilgesnis kelias. Pasitelkus projekto temos analizės metu sukauptą informaciją, apibrėžtas *CHPA** algoritmo abstrakcijos sluoksnio klaidų taisymo metodas ir naujas, vidutiniškai sutrumpinantis randamo kelio ilgį, koordinačių perskaičiavimo metodas. Taip pat aprašytas patobulinimas, skirtas pritaikyti algoritmą 8-ių krypčių jungimo kvadratinio tinklo grafams.
3. Esamos sistemos ir aukšto lygio *Python* programavimo kalbos naudojimas leido realizuoti projekto sistemą per priimtiną laiką. Realizuoti regresiniai testai užtikrino teisingą kelio paieškos algoritmų bei sistemos veikimą, o taip buvo sumažintas laikas kartojant eksperimentinius tyrimus dėl netinkamo sistemos veikimo.
4. Atlikti eksperimentiniai tyrimai parodė, kad *CHPA** algoritmas randa iki 5 % trumpesnį kelią 8-ių krypčių žemėlapyje nei 4-ių krypčių. Naujo koordinačių perskaičiavimo metodo rezultatai parodė vidutiniškai iki 0,07 % mažiau patikrinamų viršūnių, 6,23 % trumpesnę kelio paiešką tikrame žemėlapyje ir 1,51 % trumpesnį kelio ilgį. Kelio paieškos algoritmų palyginimo metu pastebėta, kad *CHPA** algoritmas kelio paiešką atlieka daugiau nei 10 kartų greičiau nei *A** ir tam tikrais atvejais greičiau nei *JPS*.

7. Literatūros sąrašas

1. CORMEN, Thomas, Charles LEISERSON, Ronald RIVEST ir Clifford STEIN. Introduction to Algorithms, Second Edition. In . 2001. pp. 451–507. ISBN 0-262-03293-7.
2. GOLDBERG, Andrew V. ir Robert E. TARJAN. Expected Performance of Dijkstra's Shortest Path. In [interaktyvus]. Princeton, 1996. [žiūrėta 2023-05-20]. Prieiga per: http://norbertwiener.umd.edu/Education/IMA2015/t6/DijkstraAlgorithm_Golberg_Tarjan.pdf.
3. GRAHAM, Ross, Hugh MCCABE ir Stephen SHERIDAN. Pathfinding in Computer Games. In *The ITB Journal* [interaktyvus]. 2015. Vol. 4, no. 2. [žiūrėta 2023-04-19]. . Prieiga per: https://arrow.tudublin.ie/itbj/vol4/iss2/6.
4. JOHNSON, G. Smoothing a Navigation Mesh Path. In RABIN, SteveSud. *AI Game Programming Wisdom 3* . USA: Charles River Media, Inc., 2006. pp. 129–140. ISBN 1584504579.
5. DEMYEN, Douglas ir Michael BURO. Efficient Triangulation-Based Pathfinding. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1* [interaktyvus]. [s.l.]: AAAI Press, 2006. pp. 942–947. [žiūrėta 2023-05-20]. Prieiga per: https://cdn.aaai.org/AAAI/2006/AAAI06-148.pdf.
6. CUI, Xiao ir Hao SHI. A*-based Pathfinding in Modern Computer Games. In *International Journal of Computer Science and Network Security* [interaktyvus]. 2011. pp. 125–130. [žiūrėta 2021-01-02]. Prieiga per: http://paper.ijcsns.org/07_book/201101/20110119.pdf.
7. MILLINGTON, Ian ir John FUNGE. Pathfinding. In MILLINGTON, Ian ir John FUNGESud. *Artificial Intelligence for Games* [interaktyvus]. Second Edition. Ed.Boston: Elsevier, 2009. pp. 197–291. Prieiga per: https://linkinghub.elsevier.com/retrieve/pii/B9780123747310000049.
8. MESDAGHI, Syrus. Artificial Intelligence: Pathfinding Overview. In MESDAGHI, SyrusSud. *Introduction to game development* . 2nd ed. Ed.Boston, Mass.: Charles River Media, Course Technology, 2010. ISBN 1584507098; 9781584507093; 1584507055; 9781584507055.
9. YAP, Peter. Grid-Based Path-Finding. In [interaktyvus]. 2002. pp. 44–55. Prieiga per: http://link.springer.com/10.1007/3-540-47922-8_4.
10. EPP, Susanna S. Discrete Mathematics with Applications. In . [s.l.]: Cengage Learning, 2010. pp. 625–716. ISBN 9781133168669.
11. CROVARI, PIETRO. Google Maps and graph theory. In *Impactscool Magazine* . 2019. .
12. BOTEA, Adi, Bruno BOUZY, Michael BURO, Christian BAUCKHAGE ir Dana NAU. Pathfinding in Games. In LUCAS, Simon M, Michael MATEAS, Mike PREUSS, Pieter SPRONCK ir Julian TOGELIUSSud. *Artificial and Computational Intelligence in Games* . Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013. pp. 21–31. ISBN 978-3-939897-62-0.
13. PETRIK, Marek ir Shlomo ZILBERSTEIN. [interaktyvus]. .2008. [žiūrėta 2020-05-11]. Prieiga per: http://cs.umass.edu/~petrik/pub/Petrik2008b.pdf.
14. HART, Peter, Nils NILSSON ir Bertram RAPHAEL. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In *IEEE Transactions on Systems Science and Cybernetics* [interaktyvus]. 1968. Vol. 4, no. 2, pp. 100–107. [žiūrėta 2023-03-04]. . Prieiga per: http://ieeexplore.ieee.org/document/4082128/.
15. DEZA, Michel Marie ir Elena DEZA. Metrics on Normed Structures. In *Encyclopedia of Distances* [interaktyvus]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016. pp. 97–108. Prieiga per: http://link.springer.com/10.1007/978-3-662-52844-0_5.
16. HU, Cong, Quanjun YIN, Yue HU, Junjie ZENG ir Long QIN. Speeding up FastMap for Pathfinding on Grid Maps. In *2019 IEEE International Conference on Mechatronics and Automation (ICMA)* [interaktyvus]. [s.l.]: IEEE, 2019. pp. 2501–2506. Prieiga per: https://ieeexplore.ieee.org/document/8816354/.

17. DEZA, Michel Marie ir Elena DEZA. Distances on Real and Digital Planes. In *Encyclopedia of Distances* [interaktyvus]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016. pp. 365–381. Prieiga per: http://link.springer.com/10.1007/978-3-662-52844-0_19.
18. MATHEW, Geethu Elizebeth ir G. MALATHY. Direction based heuristic for pathfinding in video games. In *2015 2nd International Conference on Electronics and Communication Systems (ICECS)* [interaktyvus]. [s.l.]: IEEE, 2015. pp. 1651–1657. Prieiga per: http://ieeexplore.ieee.org/document/7124867/.
19. KUMAR, Neerendra, Zoltan VAMOSSY ir Zsolt Miklos SZABO-RESCH. Heuristic approaches in robot navigation. In *2016 IEEE 20th Jubilee International Conference on Intelligent Engineering Systems (INES)* [interaktyvus]. [s.l.]: IEEE, 2016. pp. 219–222. Prieiga per: http://ieeexplore.ieee.org/document/7555123/.
20. HART, Peter, Nils NILSSON ir Bertram RAPHAEL. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In *IEEE Transactions on Systems Science and Cybernetics* . 1968. Vol. 4, no. 2, pp. 100–107. .
21. BJÖRNSSON, Yngvi, Markus ENZENBERGER, Robert C. HOLTE ir Jonathan SCHAEFFER. Fringe search: Beating A* at pathfinding on game maps. In *IEEE 2005 Symposium on Computational Intelligence and Games, CIG'05* [interaktyvus]. 2005. no. January 2005, pp. 125–132. [žiūrėta 2023-05-20]. . Prieiga per: https://www.researchgate.net/publication/221157471_Fringe_Search_Beating_A_at_Pathfinding_on_Game_Maps.
22. BOTEA, Adi, Martin MÜLLER ir Jonathan SCHAEFFER. Near optimal hierarchical path-finding (HPA*). In *Journal of Game Development* [interaktyvus]. 2004. Vol. 1. [žiūrėta 2023-05-15]. . Prieiga per: https://webdocs.cs.ualberta.ca/~mmueller/ps/hpastar.pdf.
23. STURTEVANT, Nathan R. Memory-Efficient Abstractions for Pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment Conference* [interaktyvus]. 2007. [žiūrėta 2023-05-20]. Prieiga per: https://webdocs.cs.ualberta.ca/~nathanst/papers/mmabstraction.pdf.
24. HARABOR, Daniel ir Alban GRASTIEN. Online Graph Pruning for Pathfinding on Grid Maps. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence* [interaktyvus]. [s.l.]: AAAI Press, 2011. pp. 1114–1119. [žiūrėta 2023-05-15]. Prieiga per: https://users.cecs.anu.edu.au/~dharabor/data/papers/harabor-grastien-aaai11.pdf.
25. HARABOR, Daniel ir Adi BOTEA. Breaking Path Symmetries on 4-Connected Grid Maps. In *Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010* [interaktyvus]. Stanford: Association for the Advancement of Artificial Intelligence (AAAI), 2010. pp. 33–38. [žiūrėta 2023-05-15]. Prieiga per: https://ojs.aaai.org/index.php/AIIDE/article/view/12393/12252.
26. HARABOR, Daniel ir Alban GRASTIEN. The JPS Pathfinding System. In *Proceedings of the International Symposium on Combinatorial Search* [interaktyvus]. 2021. Vol. 3, no. 1, pp. 207–208. [žiūrėta 2021-01-10]. . Prieiga per: https://ojs.aaai.org/index.php/SOCS/article/view/18254.
27. HARABOR, Daniel ir Alban GRASTIEN. Improving Jump Point Search. In *Proceedings of the International Conference on Automated Planning and Scheduling* [interaktyvus]. 2014. Vol. 24, no. January, pp. 128–135. [žiūrėta 2021-01-10]. . Prieiga per: https://ojs.aaai.org/index.php/ICAPS/article/view/13633.
28. SOLOMON, Chris ir Toby BRECKON. Fundamentals of Digital Image Processing. In . [s.l.]: Wiley, 2010. pp. 87. ISBN 9780470844724.
29. PYGAME COMMUNITY. About - pygame wiki. In [interaktyvus]. 2018. [žiūrėta 2020-05-15]. Prieiga per: https://www.pygame.org/wiki/about.

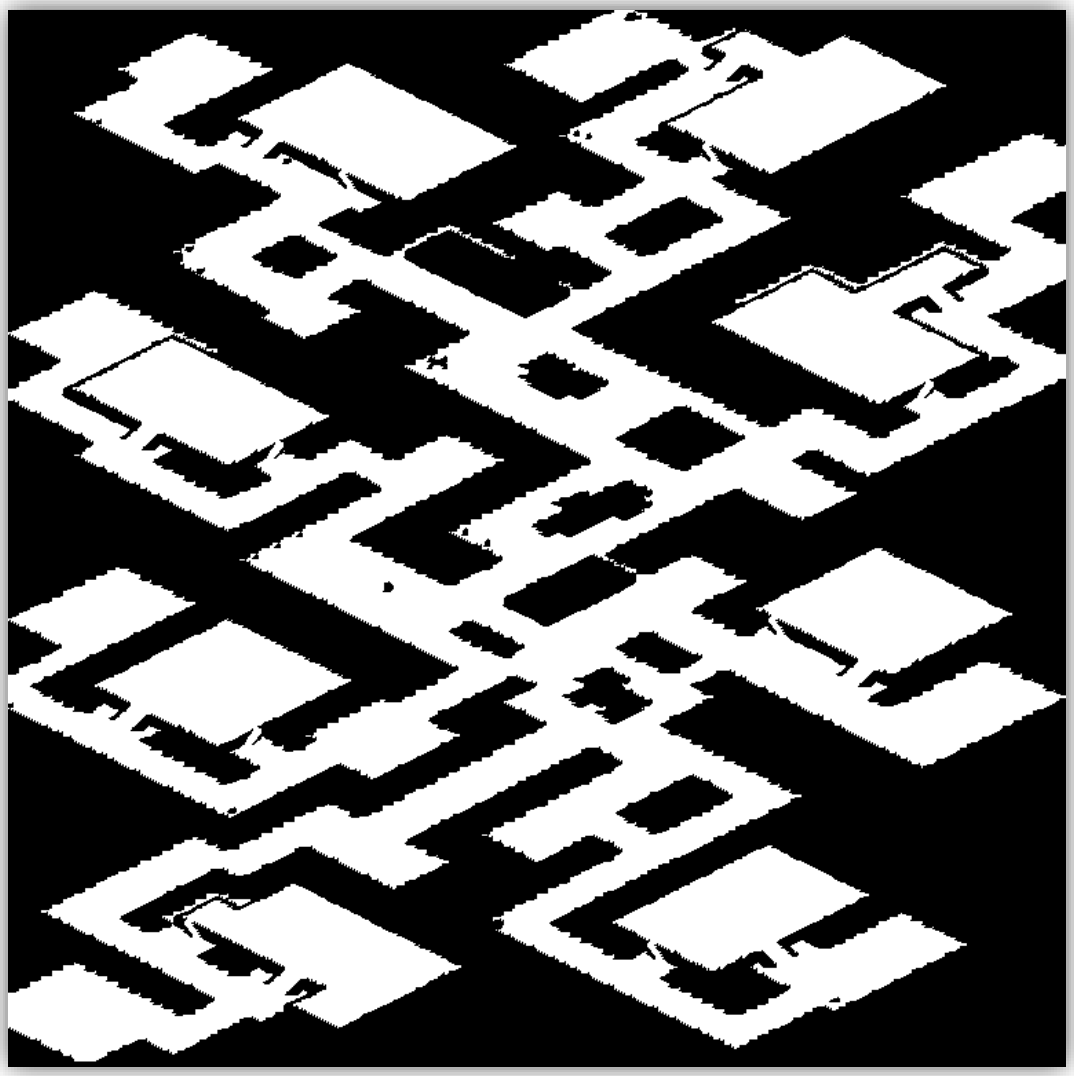
30. tqdm/tqdm: A Fast, Extensible Progress Bar for Python and CLI. In [interaktyvus]. [žiūrėta 2020-05-15]. Prieiga per: https://github.com/tqdm/tqdm.
31. Pillow — Pillow (PIL Fork) 7.1.2 documentation. In [interaktyvus]. [žiūrėta 2020-05-15]. Prieiga per: https://pillow.readthedocs.io/en/stable/index.html.
32. NUMPY DEVELOPERS. NumPy — NumPy. In *NumPy Website* [interaktyvus]. 2017. [žiūrėta 2020-05-15]. Prieiga per: http://www.numpy.org/.
33. TELEIŠA, Marius. *Neuroninių tinklų pritaikymas sparčiam apytikslio trumpiausio maršruto radimui*. Kaunas: Kauno Technologijos Universitetas, 2020. .
34. STURTEVANT, N. R. Benchmarks for Grid-Based Pathfinding. In *IEEE Transactions on Computational Intelligence and AI in Games* [interaktyvus]. 2012. Vol. 4, no. 2, pp. 144–148. Prieiga per: http://ieeexplore.ieee.org/document/6194296/.

Priedai

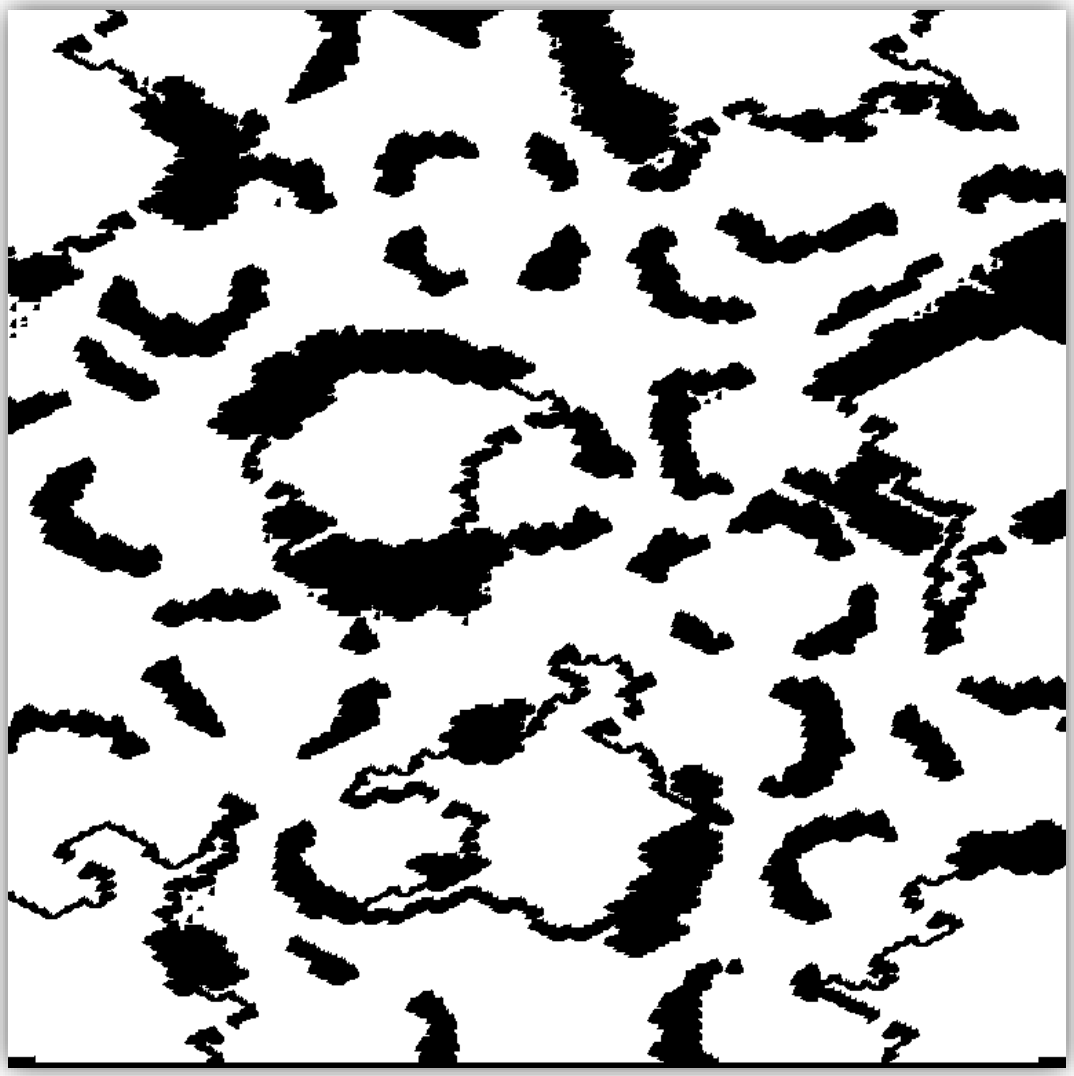
1 Priedas. *IVUS* 2023 konferencijos publikacija

Pristatytas pranešimas 28-ojoje tarptautinėje informacinių technologijų konferencijoje *IVUS 2023* Teleiša M., Čalnerytė D. (2023). *Fast Approximate Pathfinding Based on 2D Convolution*.

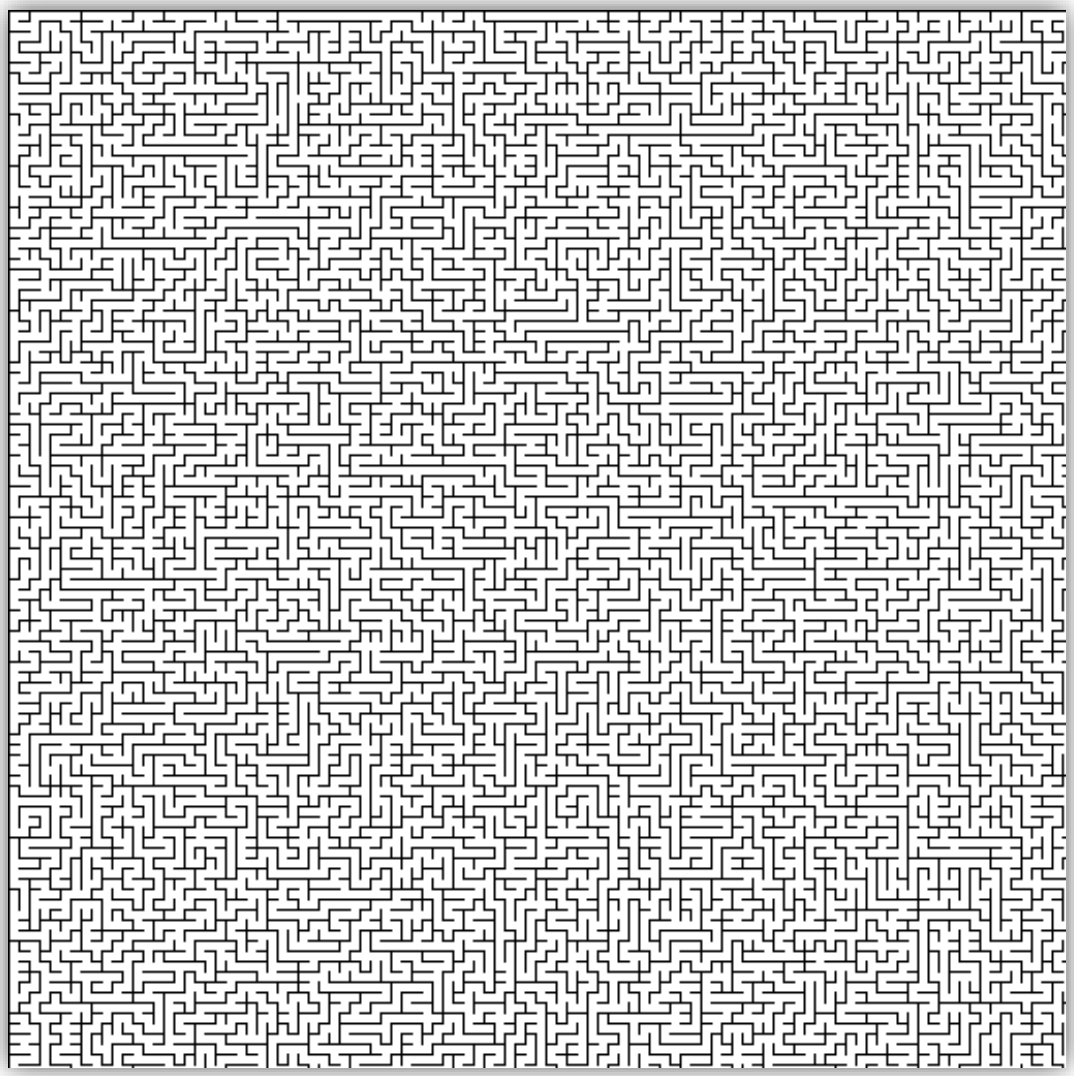
2 Priedas. Žemėlapis *Enigma*



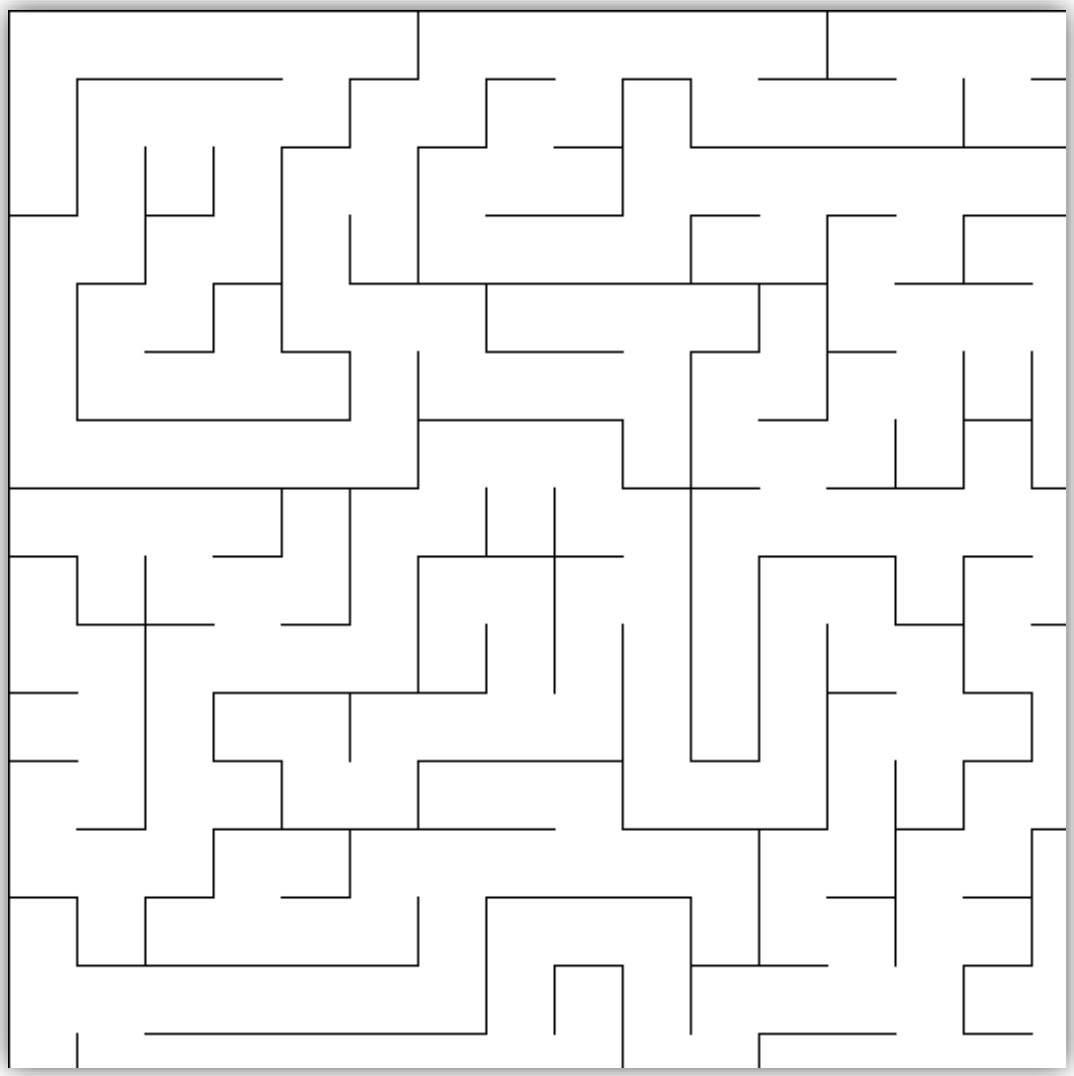
3 Priedas. Žemėlapis *Brokensteppes*



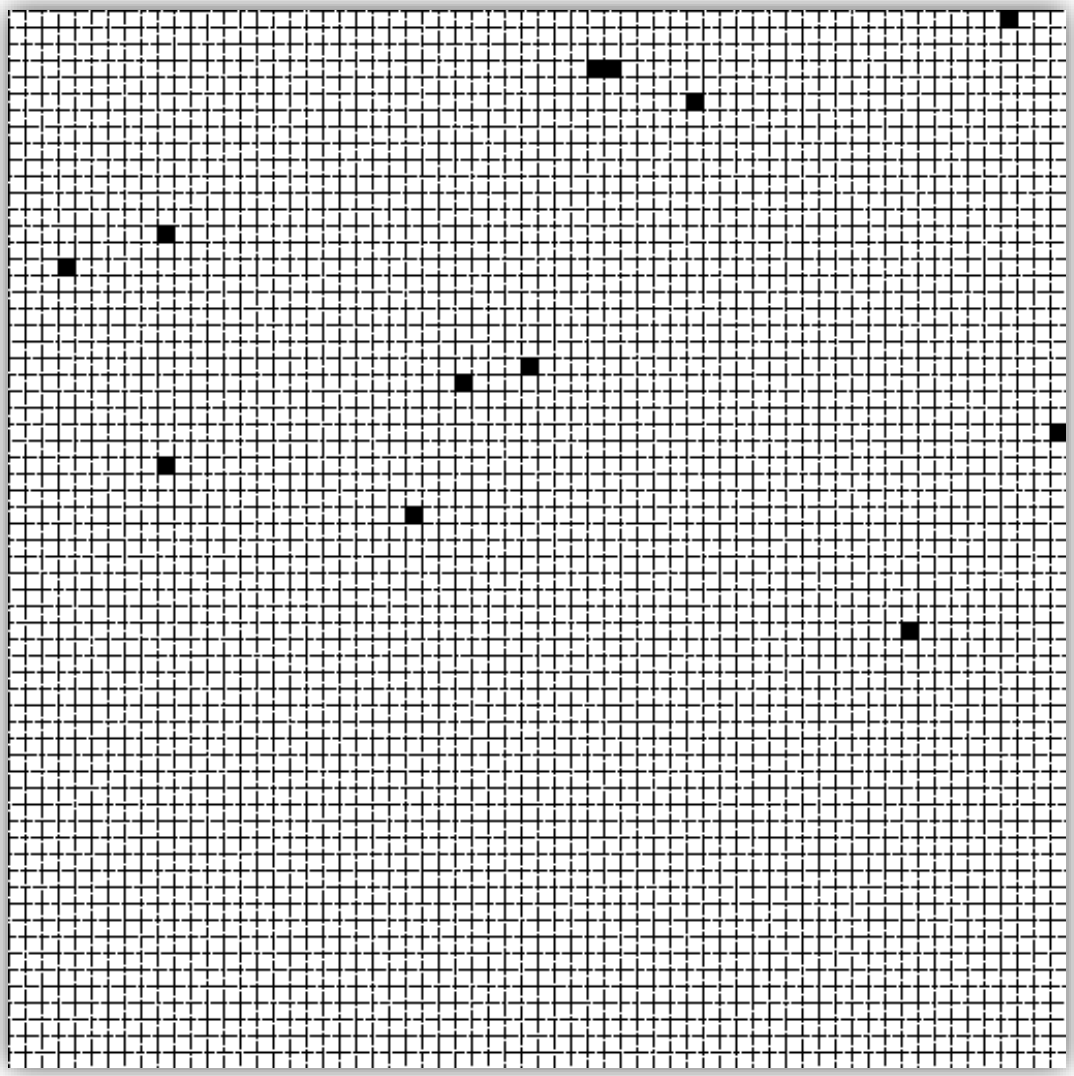
4 Priedas. Žemėlapis *maze512-4-0*



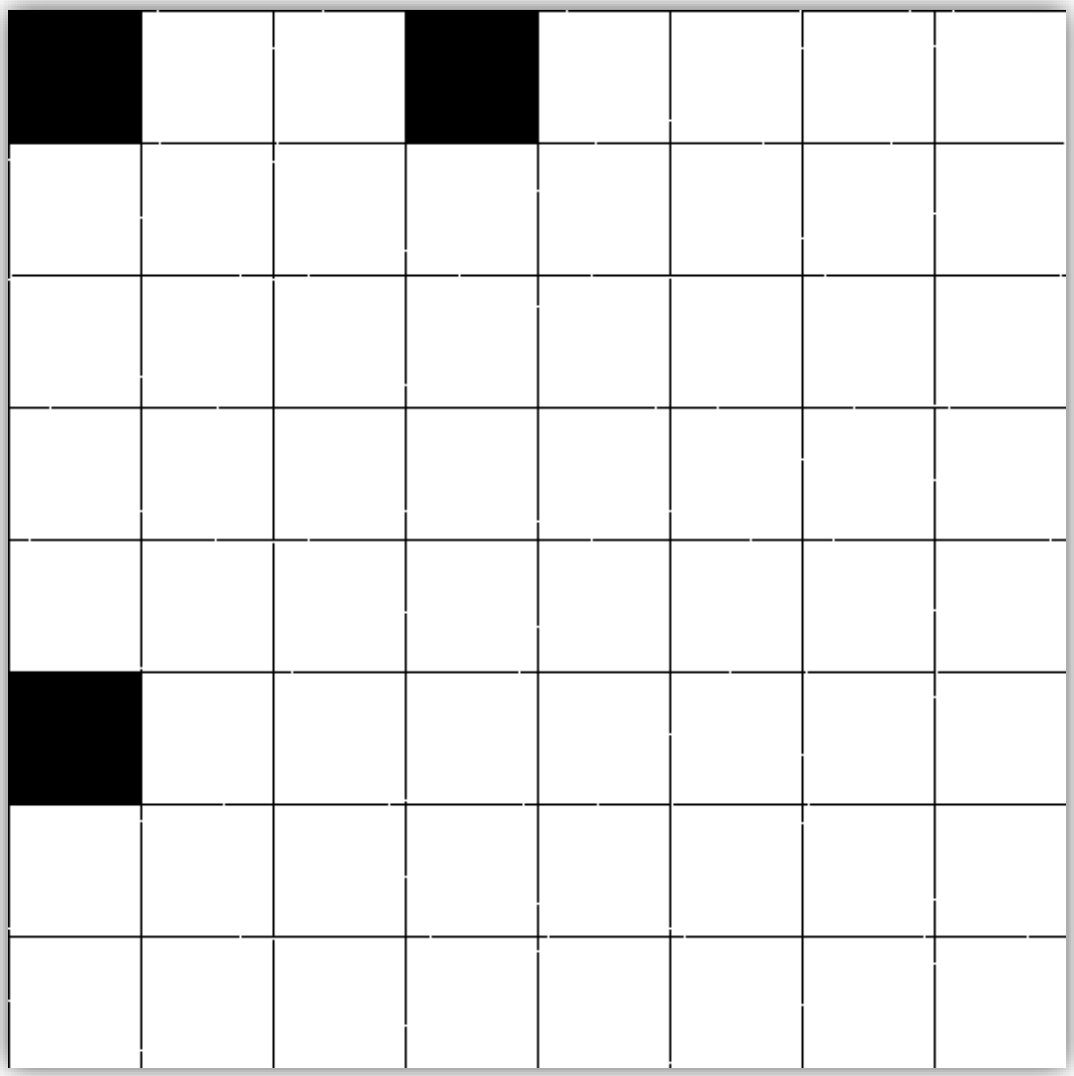
5 Priedas. Žemėlapis *maze512-32-0*



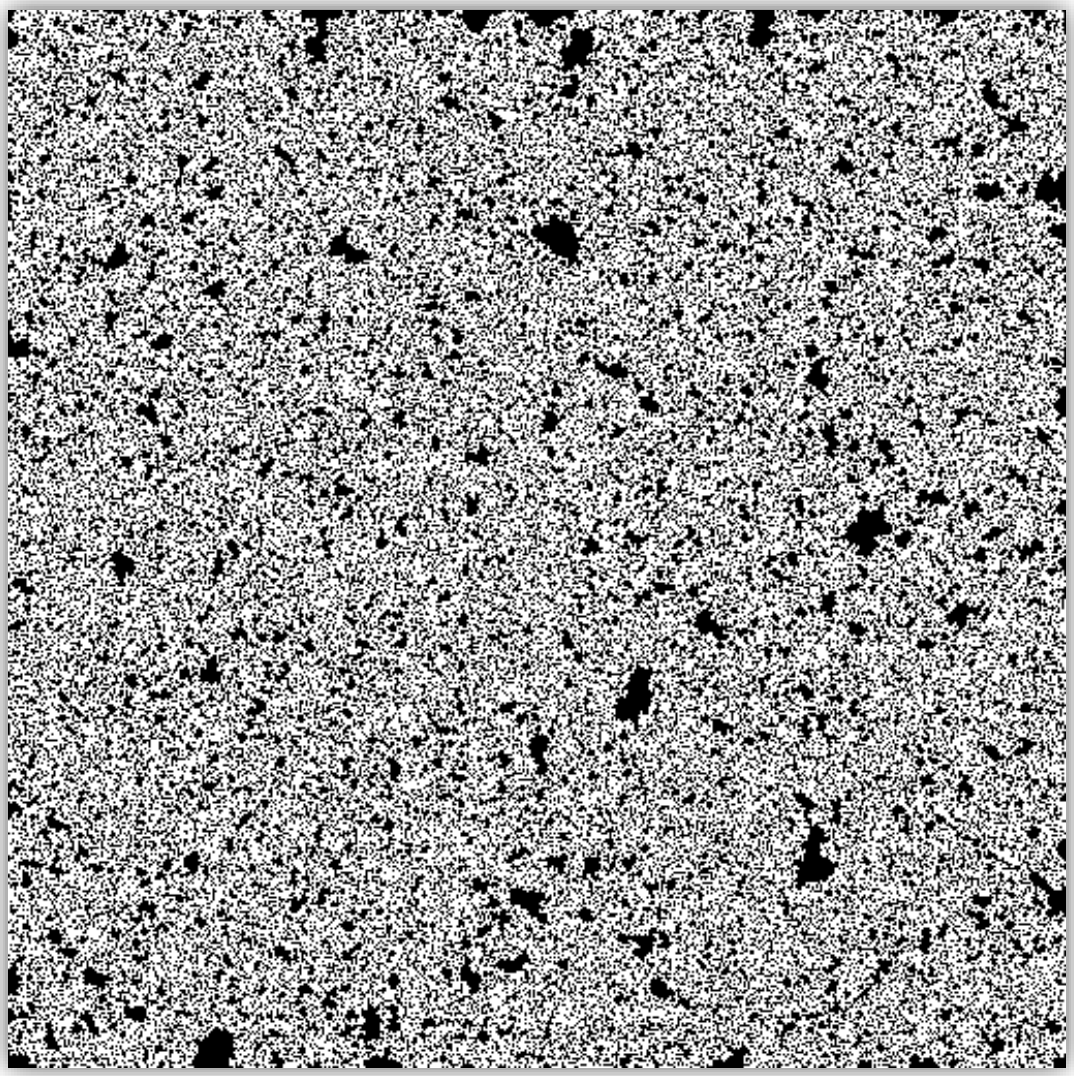
6 Priedas. Žemėlapis *8room_000*



7 Priedas. Žemėlapis *64room_000*



8 Priedas. Žemėlapis *random512-35-1*



9 Priedas. Žemėlapis *random512-10-1*