Nadgradnja večagentnega planiranja poti z varnimi intervali in prioritetami

Matic Sedej
Mentor: izr. prof. dr. Gregor Klančar
Univerza v Ljubljani, Fakulteta za elektrotehniko
Tržaška c. 25, 1000 Ljubljana, Slovenija
ms2986@student.uni-lj.si, gregor.klancar@fe.uni-lj.si

Upgrade of multi-agent path planning with safe intervals and priorities

The paper describes the operation of the SIPP algorithm for the needs of multi-agent path planning using safe intervals and priorities for the purpose of path planning of automated guided vehicles in surroundings such as production halls and warehouses. To understand the upgrade of the algorithm, the following is a brief description of the simulation environment used at the Faculty of Electrical Engineering in Ljubljana. Further, the upgrade of the basic algorithm with two different approaches, which enables a more optimal path, is described. The first approach is based on adding intermediate nodes to an existing road, while the second one uses waiting on the road before the node. Finally, a comparison of the results between the existing algorithm and its upgrades is presented on the example.

Kratek pregled prispevka

Prispevek opisuje delovanje algoritma SIPP za potrebe večagentnega planiranja poti z uporabo varnih intervalov in prioritet za namene planirana poti avtomatskih vodenih vozil v okoljih, kot so proizvodne hale in skladišča. Za razumevanje nadgradenj algoritma sledi kratek opis simulacijskega okolja, ki ga uporabljajo na Fakulteti za elektrotehniko v Ljubljani. Nadaljnje je opisana nadgradnja osnovnega algoritma z dvema različnima pristopoma, ki omogočata optimalnejšo pot. Prvi pristop temelji na dodajanju vmesnih vozlišč na obstoječe ceste, drugi pa uporablja čakanje na cesti pred vozliščem. Na koncu je na primeru predstavljena primerjava rezultatov med obstoječim algoritmom in njegovima nadgradnjama.

1 Uvod

Avtomatska vodena vozila (AGV) v skladiščih in tovarnah morajo znati določiti varno pot od startnega do ciljnega postajališča. Zaradi poenostavitve naloge in predvsem varnosti je njihovo gibanje v prostoru omejeno. To pomeni, da je njihovo dovoljeno gibanje predpisano z zemljevidom, ki vsebuje ceste, križišča in postaje. Tak zemljevid lahko predstavimo z grafom prehajanja stanj, kateri vsebuje povezave in vozlišča.

Problem iskanja poti z več agenti (MAPF) je posplošitev problema iskanja poti iz enega agenta na k>1 agentov [1]. Za vsakega agenta je podana unikatna startna in ciljna postaja, naloga pa je najti optimalne poti za vse agente od startnih do ciljnih postaj, z omejitvijo, da agenti med svojimi premiki ne smejo trčiti med seboj.

2 Planiranje poti z varnimi intervali

Algoritem SIPP (Safe Interval Path Planing) je nadgradnja algoritma A* z dodatnim upoštevanjem varnih intervalov vozlišč. Algoritem določi optimalno pot glede na znano gibanje ostalih vozil, ki predstavljajo dinamične ovire.

*2.1 Algoritem A**

A* je iskalni algoritem, ki se uporablja za iskanje najcenejše poti med začetno in končno točko. Uporablja hevristiko, ki je ocena cene poti od trenutnega vozlišča do cilja (cena-do-cilja), zaradi česar lahko algoritem razlikuje med bolj ali manj obetavnimi vozlišči in je učinkovitejši pri iskanju rešitve [2].

Algoritem A* je popoln, saj vedno najde pot, če le-ta obstaja, pri uporabi optimistične hevristike (cena do cilja za vsako vozlišče manjša ali enaka pravi ceni do cilja) pa tudi optimalen [2].

2.2 Varni in zasedeni intervali vozlišč

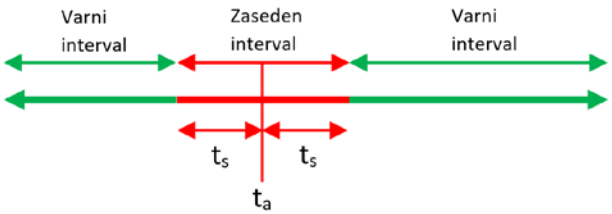
Varni interval je časovno okno, v katerem vozilu dovolimo prihod ali čakanje v vozlišču. Zasedeni interval je časovno okno v katerem je vozilu prepovedan prihod ali čakanje v vozlišču. To pomeni, da je v tem časovnem oknu vozlišče že zasedlo drugo vozilo.

Vsako vozilo ima neko geometrijsko obliko (ni neskončno majhna točka). Ko se vozilo pelje skozi vozlišče to pomeni, da le tega ne zasede samo za trenutek prispetja v vozlišče *t*, ampak ga zasede za nek časovni interval [t_1, t_2], ki je odvisen od geometrije vozila. Na tem mestu vpeljemo pojem varnosti čas t_s, ki predstavlja čas vožnje varnostnega radija R_s vozila skozi vozlišče. Izračunamo ga s pomočjo varnostnega radija in hitrosti *v* po enačbi (1). V primeru pravokotnega vozila z dolžino *l* in širino *w*, varnostni radij izračunamo po enačbi (2), kjer je *M* varnostni faktor s katerim pomnožimo radij vozila.

$$t_s = \frac{R_s}{v} \tag{1}$$

$$R_s = M\sqrt{\left(\frac{l}{2}\right)^2 + \left(\frac{w}{2}\right)^2}\ ;\ M > 1 \tag{2}$$

Ko se vozilo pelje skozi vozlišče, le tega zasede za časovni interval [t_1, t_2]. t_1 je čas začetka zasedenega intervala in predstavlja čas prispetja vozila v vozlišče t_a zmanjšan za varnosti čas t_s. t_2 je čas konca zasedenega intervala in predstavlja čas prispetja vozila v vozlišče povečan za varnosti čas. Primer časovnice vozlišča prikazuje slika 1.



Slika 1: Časovnica vozlišča.

2.3 SIPP algoritem

Kot že omenjeno, je algoritem SIPP nadgradnja algoritma A* z dodatnim upoštevanjem varnih intervalov vozlišč. Za cenilko poti SIPP uporablja čas. Glavna razlika med obema algoritmoma je v načinu iskanja naslednikov.

V algoritmu SIPP so uporabljene naslednje predpostavke [3]:

- Robot je sposoben čakati na mestu,
- robot se lahko ustavi in pospeši v trenutku.

Stanje *s* v sledečih algoritmih predstavlja skupek vozlišča in enega njegovih varnih intervalov. Vsako vozlišče ima torej vsaj eno stanje (en varni interval ali več).

2.4 Iskanje naslednikov

Pri iskanju naslednikov uporabljamo akcije »čakanje in premik«. To pomeni, da za vsak varni interval v naslednjem vozlišču čakamo v trenutnem vozlišču čim krajši čas. Ko je mogoč premik v naslednje vozlišče, nam to omogoča, da pridemo čim prej v enega izmed njegovih varnih intervalov [3].

Sledi opis iskanja naslednikov s pomočjo algoritma 1:

```
1  getSuccessors(s)
2    successors = ∅;
3    for each m in M(s)
4      cfg =configuration of m applied to s
5      m_time =time to execute m
6      start_t = time(s) + m_time
7      end_t = endTime(interval(s)) + m_time
8      for each safe interval i in cfg
9        if startTime(i) > end_t or endTime(i) < start_t
10           continue
11       t =earliest arrival time at cfg during interval i with no collisions
12       if t does not exist
13         continue
14       s' = state of configuration cfg with interval i and time t
15       insert s' into successors
16   return successors;
```

Algoritem 1: Iskanje naslednikov [3].

Funkcija *M(s)* vrne premike *m*, ki se lahko izvedejo iz trenutnega vozlišča *s*. Premiki imajo določen čas, ki je potreben za njihovo izvedbo *m_time*, kar uporabimo za pomoč pri določanju, v katere varne intervale lahko vstopimo. Funkcija *startTime(i)* vrne začetni čas varnega intervala *i*, *endTime(i)* pa konec varnega intervala *i*.

Najprej ustvarimo prazen seznam naslednikov *sucessors*. Za vsak možen premik *m* v naslednje vozlišče iz trenutnega vozlišča *s* izračunamo čas izvedbe *m_time*, najzgodnejši možen prihod v vozlišče *start_t* in najkasnejši možen prihod v vozlišče *end_t*. Najzgodnejši prihod je vsota prihoda v trenutno vozlišče *time(s)* in časa izvedbe premika *m_time*. To velja v primeru, ko se vozilo pelje skozi vozlišče *s*. Najkasnejši prihod je omejen s koncem varnega intervala trenutnega vozlišča *endTime(interval(s))* kateremu prištejemo čas izvedbe *m_time*. To velja v primeru, ko v vozlišču *s* vozilo tudi čaka.

Nato za vsak varni interval *i* naslednjega vozlišča preverimo pogoja:

- začetni čas *startTime(i)* je večji od najkasnejšega možnega prihoda v vozlišče *end_t*,
- končni čas *endTime(i)* je manjši od najzgodnejšega možnega prihoda v vozlišče *start_t*.

V primeru, da sta izpolnjena oba pogoja, to pomeni, da to ni veljaven naslednik in nadaljujemo iskanje z naslednjim varnim intervalom. Če pogoja nista izpolnjena preverimo, če obstaja čas najzgodnejšega možnega prihoda v naslednje vozlišče *t* znotraj njegovega varnega intervala *i* in ga vpišemo v seznam naslednikov *sucessors*. V nasprotnem primeru nadaljujemo z iskanjem. Čas najzgodnejšega prihoda v naslednje vozlišče *t* hkrati predstavlja tudi cena-do-sem.

V seznamu naslednikov se tako lahko nahaja več naslednikov, ki predstavljajo isto vozlišče, ampak imajo različne varne intervale.

2.5 A z varnimi intervali (SIPP)*

Sledi opis nadgrajenega algoritma A* z varnimi intervali s pomočjo algoritma 2:

```
1  g(s_start) = 0; OPEN = ∅;
2  insert s_start into OPEN with f(s_start) = h(s_start);
3  while(s_goal is not expanded)
4    remove s with the smallest f-value from OPEN;
5    successors = getSuccessors(s);
6    for each s' in successors
7      if s' was not visited before then
8        f(s') = g(s') = ∞;
9      if g(s') > g(s) + c(s, s')
10       g(s') = g(s) + c(s, s');
11       updateTime(s');
12       f(s') = g(s') + h(s');
13       insert s' into OPEN with f(s');
```

Algoritem 2: A z varnimi intervali [3].*

Hevristična funkcija *h(s)* je ocena cene poti od trenutnega vozlišča *s* do ciljnega vozlišča s_{goal} (cena-do-cilja), izračunamo jo po enačbi (3), kjer $d(s,s_{goal})$ predstavlja evklidsko razdaljo.

$$h(s) = \frac{d(s, s_{goal})}{v} \qquad (3)$$

Spremenljivka *g(s)* je cena poti od začetnega vozlišča s_{start} do trenutnega vozliča *s*. Funkcija *getSucessors(s')* vrne možne naslednike. *c(s, s')* je čas izvedbe premika od vozlišča *s* do naslednika *s'*.

Najprej ustvarimo prazen odprti seznam *OPEN*, in vanj vnesemo začetno vozlišče s ceno-do-sem *g(s)=0* in cena-celotne-poti, ki je enaka cena-do-cilja. Naslednje korake ponavljamo, dokler ne naletimo na ciljno vozlišče, ali zmanjka vozlišč na *OPEN* seznamu, kar pomeni, da poti ni mogoče najti.

Iz seznama *OPEN* odstranimo vozlišče *s* z najmanjšo ceno-celotne-poti *f(s)*, in poiščemo njegove možne naslednike. Za vsakega naslednika *s'* preverimo ali smo ga že obiskali. Če ga še nismo, nastavimo obe ceni *g(s')=f(s')=∞*. V primeru, ko je cena-do-sem naslednika *s'* večja kot vsota cena-do-sem *g(s)* in časa izvedbe premika *c(s, s')*, to vsoto nastavimo kot novo vrednost cena-do-sem naslednika *g(s')*, izračunamo hevristiko *h(s')* in cena-celotne-poti *f(s')*, ter naslednika *s'* vstavimo v seznam *OPEN*.

2.6 *Večagentno planiranje poti s prioritetami*

Algoritem SIPP se lahko izvaja na vsakem vozilu ločeno, podatke o poteh (gibanju) ostalih vozil pa si lahko izmenjujejo med seboj ali preko centralnega komunikacijskega sistema [4].

Da dosežemo hierarhijo, uvedemo prioritetni sistem. Vozilo z višjo prioriteto ima tako absolutno prednost pred manj prioritetnimi. Iz tega sledi, da se najprej določi pot za vozilo z najbolj prioritetno nalogo, nato se išče pot za vozila z nižjo prioriteto, tako da se ta izognejo trkom predhodnikov z višjo prioriteto. Prioritete so npr. lahko določene s časom objave naloge (vozila, ki so prej prejela nalogo, imajo višjo prioriteto) ali pa so določene glede na potrebe procesa (tovor, ki mora biti čimprej dostavljen, ima višjo prioriteto).

3 Simulacijsko okolje

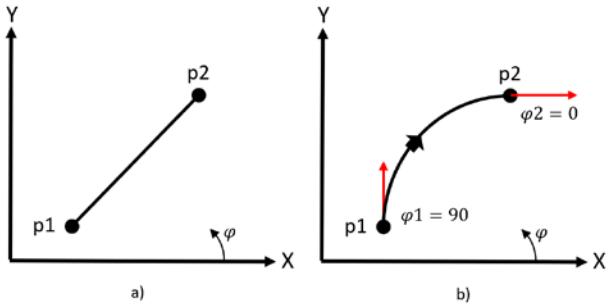
V laboratoriju za avtomatiko in kibernetiko na Fakulteti za elektrotehniko v Ljubljani imajo razvito simulacijsko okolje v programskem paketu Matlab. Namenjeno je razvijanju in testiranju algoritmov za večagentno planiranje poti vozil AGV [5].

Okolje omogoča kreiranje zemljevida z vozlišči in povezavami – cestami. Prav tako omogoča vizualizacijo vožnje AGV vozil po zemljevidu cest in vsebuje mehanizem za detekcijo trkov med vozili tekom simulacije.

3.1 *Zemljevid cest*

Okolje omogoča izgradnjo zemljevida s pomočjo vozlišč in cest, ki jih povezujejo. Na obeh koncih ceste se mora nahajati vozlišče. Ceste so lahko eno ali dvosmerne. Cesta je lahko sestavljena iz enega ali več odsekov različnih oblik, ki so lahko linearni ali pa krožni.

Pri izgradnji zemljevida za linearni odsek ceste potrebujemo podatke [*p1, p2, dist*], kjer je *p1* začetna točka, *p2* končna točka, *dist* pa razdalja med njima, slika 2 a). Za krožni odsek ceste pa potrebujemo podatke [*p1, φ1, Δφ, arclen*], kjer je *p1* začetna točka, *φ1* začetni kot, *Δφ* sprememba kota in *arclen* dolžina krožnega loka, slika 2 b). Spremembo kota izračunamo po enačbi (4), dolžino krožnega loka pa po enačbi (5).



Slika 2: Odseki cest. a) linearen in b) krožni.

$$\Delta\varphi = \varphi 2 - \varphi 1 \tag{4}$$

$$arclen = \frac{d(p1, p2)}{2\sin\left(\frac{\Delta\varphi}{2}\right)}\Delta\varphi \tag{5}$$

3.2 *Časovni načrt*

Za vizualizacijo oz. simulacijo vožnje vozil AGV po zemljevidu cest potrebujemo časovni načrt. Načrt za vsako vozilo je urejen seznam *n* skupkov podatkov, ki skupaj tvorijo pot vozila.

En skupek podatkov [*t, roadID, loc*] lahko interpretiramo kot čas *t*, ob katerem se vozilo nahaja na lokaciji *loc* ceste *roadID*. Podatek *roadID* je oznaka ceste, *loc* pa je lokacija na tej cesti v procentih (0 do 1). Lokacija 0 predstavlja začetno vozlišče ceste, 1 pa končno.

Primer časovnega načrta za eno vozilo je v tabeli 1. Do trenutka *t*=1 čakamo v začetnem vozlišču ceste 10. Sledi 2,5–1 s=1,5 s vožnja do končnega vozlišča ceste 20, nato še 3,5–2,5 s=1 s vožnja do ciljnega vozlišča na koncu ceste 22.

n	***t***	***roadID***	***loc***
1	1	10	0
2	2.5	20	1
3	3.5	22	1

Tabela 1: Primer časovnega načrta.

4 Nadgradnje planiranja poti

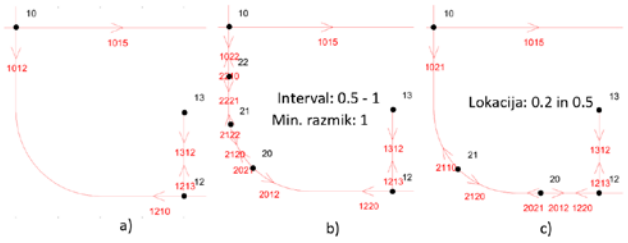
SIPP algoritem lahko v izogib trku manj prioritetnemu vozilu določi čakanje v vozlišču ali pa nadaljevanje vožnje po drugi, alternativni poti (če le-ta obstaja in je cenejša kot čakanje). V primeru čakanja v vozlišču, kjer se stika več cest (križišča), se tako lahko zgodi, da višjeprioritetno vozilo zapre pot drugim, manj prioritetnim, kar močno zmanjša pretočnost zemljevida. V nadaljevanju sta opisana dva pristopa za izboljšanje algoritma.

4.1 Vmesna vozlišča

Prvi pristop za rešitev omenjenega problema je dokaj preprost - na cesto dodamo vmesna vozlišča. Vozila tako lahko čakajo na novih, vmesnih vozliščih in ne zasedajo končnih vozlišč (križišč).

Pri dodajanju vmesnih vozlišč moramo paziti na razdaljo med njimi. Premajhna razdalja (glede na velikost AGV vozil), bi lahko privedla do situacije, ko bi center vozila sicer bil v pravem vozlišču, vozilo pa bi zaradi svoje velikosti zasedlo tudi sosednja vozlišča, kar bi lahko privedlo do trka. V primeru dolge ceste je smiselno nekaj vmesnih vozlišč postaviti le proti koncu in začetku ceste. V nasprotnem primeru lahko hitro pride do izjemno velikega števila vozlišč v zemljevidu, kar neugodno daljša računski čas SIPP algoritma.

Naša rešitev omogoča dodajanje vmesnih vozlišč na obstoječo cesto, slika 3 a), glede na podano minimalno razdaljo med vozlišči in interval na cesti (v procentih dolžine ceste oz. med 0 in 1), slika 3 b). Omogočeno je tudi ročno dodajanje vozlišč z seznamom željenih lokacij na cesti (v procentih dolžine ceste oz. med 0 in 1), slika 3 c).



Slika 3: a) s prikazom osnovne ceste, b) z dodanimi vozlišči na predpisanem intervalu in c) z dodanimi vozlišči na želenih lokacijah.

Najprej izračunamo število novih vmesnih cest N_{roads}, ki je zaokroženo navzdol, enačba (6), glede na dolžino ceste in podano minimalno razdaljo d_{min}. Število vmesnih vozlišč je za eno manjše, kot število cest.

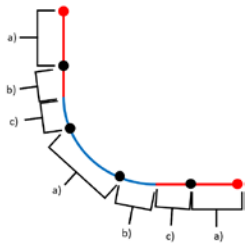
$$N_{roads} = \left\lfloor \frac{road_{len}}{d_{min}} \right\rfloor \qquad (6)$$

Izračunamo tudi lokacijo vozlišč na cesti. Sledi izbris obstoječe ceste iz seznama cest in vozlišč. Glede na obliko odsekov obstoječe ceste moramo sestavljati odseke novih vmesnih cest. Pri tem lahko naletimo na tri različne situacije, ki jih moramo zaradi pravilnega sestavljanja novih odsekov (začetna in končna točka ter dolžina odseka) obravnavati ločeno:

- Odsek ceste med dvema vozliščema, slika 4 oznaka a),
- odsek med vozliščem in mejo odseka obstoječe ceste, slika 4 oznaka b),
- odsek med mejo odseka obstoječe ceste in vozliščem, slika 4 oznaka c).

Slika 4 prikazuje obstoječo cesto z obstoječima končnima vozliščema (rdeči piki), ki je sestavljena iz dveh linearnih (rdeča črta) in enega krožnega odseka (modra črta). Ko dodamo

vmesna vozlišča (črne pike), tako nastanejo nove vmesne ceste, ki so sestavljene iz različnih delov odsekov obstoječe ceste.

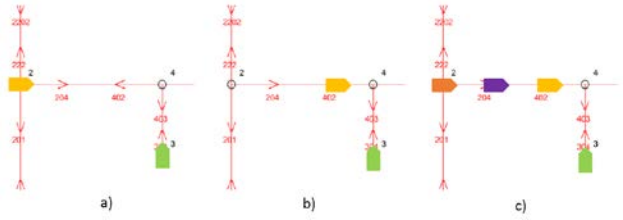


Slika 4: Odseki cest z različnimi možnostmi: a) cesta med vozlišči (pike) vsebuje le en odsek, b) in c) ceste sestavljene iz več odsekov.

4.2 Čakanje na cesti

Drugi pristop je ta, da vozilom namesto v vozliščih dovolimo čakanje na cesti. Lokacijo čakanja vozila tako premaknemo iz začetnega vozišča ceste, slika 5 a) v lokacijo na cesti pred končnim vozliščem ceste, slika 5 b). Če je na cesti dovolj prostora, lahko na njej čaka več vozil, slika 5 c).

Najprej pregledamo časovne načrte že planiranih vozil in iščemo prekrivanje intervalov voženj po istih cestah s trenutnim vozilom. Sledi preverjanje s kakšno hitrostjo se pelje predhodnik. Izračunamo lokacijo čakanja in čas vožnje do lokacije in konca ceste. Preverimo tudi če je na cesti dovolj prostora.



Slika 5: a) osnovna verzija čaka v vozlišču, b) nadgrajena verzija čaka na cesti in c) čakanje več vozil, če je prostor na cesti.

V obstoječem stanju se čas čakanja izračuna s pomočjo enačb (6) in (7), kjer $t_a(V_e)$ predstavlja čas prispetja v končno vozlišče, $t_a(V_s)$ pa čas prispetja v začetno vozlišče ceste. Čas vožnje po cesti izračunamo po enačbi (7). Sledi preverjanje pogoja $t > t_d$. Če je pogoj izpolnjen pomeni, da moramo za to časovno razliko čakati v začetnem vozlišču, lokacija 0.

$$t_a(V_e) - t_a(V_s) = t \tag{6}$$

$$t_d = \frac{road_{len}}{v} \tag{7}$$

V nadgradnji se najprej izračuna lokacija čakanja na cesti po enačbi (8). Podatek $road_{free}$ predstavlja prosto dolžino ceste, ki jo določimo glede na lokacijo zadnjega čakajočega vozila, ki že čaka na tej cesti. $road_{len}$ je dolžina ceste, R_s pa varnostni radij. Čas vožnje do izračunane lokacije izračunamo po enačbi (9), čas vožnje od lokacije čakanja do konca ceste pa po enačbi (10).

$$loc = \frac{road_{free} - 2R_s}{road_{len}} \tag{8}$$

$$t_{d\_loc} = \frac{road_{free} - 2R_s}{v} \tag{9}$$

$$t_{d\_end} = \frac{road_{len}}{v} - t_{d\_loc} \tag{10}$$

V primeru, če na cesti ni dovolj prostora, ali pa je prekratka, določimo čakanje v začetnem vozlišču ceste, lokacija 0.

Če se spredaj vozi ali čaka počasnejše vozilo, moramo hitrost prilagoditi počasnejšemu, sicer lahko pride do trka. To vodi v krajše čakanje ali pa ga celo odpravi. V redkih primerih se lahko zgodi, da s počasnejšo hitrostjo ne uspemo prevoziti ceste dovolj hitro kakor nam določi SIPP. V tem primeru hitrost vožnje malce povečamo, da zadostimo SIPP algoritmu.

Primer obstoječega stanja in izboljšave časovnega načrta je prikazan v tabeli 2.

Obstoječa rešitev		Nadgradnja	
[0, 120, 0]	čakanje	[0, 120, 0]	čakanje
[10, 120, 1]	vožnja	[10, 120, 1]	vožnja
[12, 130, 0]	čakanje	[11.6, 130, 0.8]	vožnja
[14, 130, 1]	vožnja	[13.6, 130, 0.8]	čakanje
/	/	[14, 130, 1]	vožnja

Tabela 2: Primerjava časovnih načrtov.

V obeh primerih začnemo vožnjo ob času t=0 s, in prispemo v končno vozlišče ceste 120 ob času t=10 s. V obstoječem stanju sledi 12–10 s=2 s čakanja v začetnem vozlišču ceste 130

ter 14–12 s=2 s vožnje do končnega vozlišča ceste 130. V nadgradnji pa 11,6–10 s=1,6 s vožnje na lokacijo 0,8 ceste 130, 13,6–11,6 s=2 s čakanja na lokaciji 0,8 in 14–13,6 s=0,4 s vožnje do končnega vozlišča.

Kot lahko opazimo je čas čakanja v obeh primerih enak, prav tako čas prispetja v končno vozlišče vseh cest iz načrta, saj moramo upoštevati zahteve SIPP algoritma.

5 Rezultati

Sledi primerjava osnovnega algoritma in njegovih nadgradenj na primeru. Na sliki 6 a) so prikazane poti vozil. V tabeli 3 so primerjani časi prispetja v ciljna vozlišča vozil iz vseh treh različic algoritma in skupni čas poti. Na sliki 7 je prikazan računski čas SIPP algoritma in časovnega načrta.

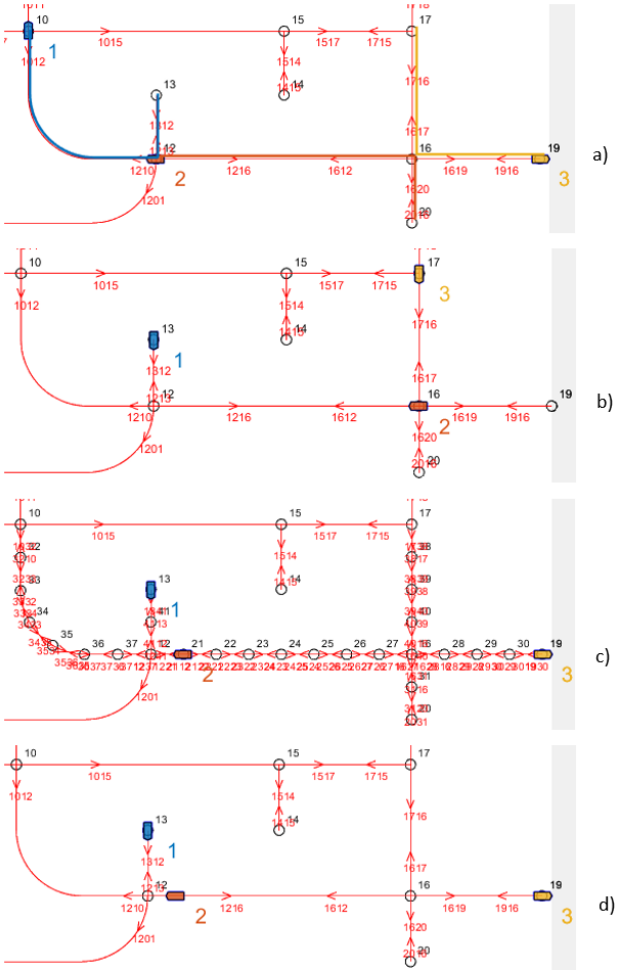
Vozilo 1 začne vožnjo v vozlišču 13 ob času t=30 s in ima cilj v vozlišču 10. Vozilo 2 začne vožnjo v vozlišču 20 ob času t=0 s in ima cilj v vozlišču 12. Vozilo 3 začne vožnjo v vozlišču 17 ob času t=1 s in ima cilj v vozlišču 19.

V primeru na sliki 6 vozilo 2 čaka vozilo 1, ki ima višjo prioriteto v vozlišču 16, slika 6 b). To onemogoči pot vozilu 3 z najnižjo prioriteto, kar vodi v nepotrebno čakanje tega vozila. V obeh nadgradnjah vozilo 2 čaka v bližini vozlišča 12, s tem pa sprosti vozlišče 16 za vozilo 3, kateremu sedaj ni potrebno čakati. Trenutek prihoda vozila 2 na lokacijo čakanja je viden na sliki 6 c) in 6 d).

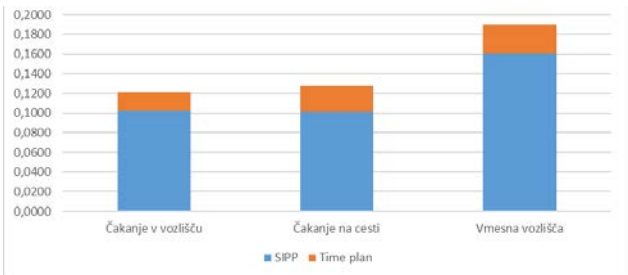
Vozilo	Čakanje v vozlišču	Vmesna vozlišča	Čakanje na cesti
1	48,30 s	48,30 s	48,30 s
2	35,80 s	35,80 s	35,80 s
3	29,50 s	17,00 s	17,00 s
Skupaj	113,60 s	101,10 s	101,10 s

Tabela 3: Primerjava časov za izvedbo planov osnovnega algoritma in nadgrajenega s čakanjem v dodanih vozliščih ali s čakanjem na cesti.

Opazimo, da smo s premikom čakanja izven križišča zmanjšali čas ki, ga potrebuje vozilo 3 za izvršitev naloge in sicer za 12,5 sekunde.



Slika 6: Primerjava koordiniranih voženj. a) opravljene poti, b) koordinacija s čakanjem v vozlišču, c) s čakanjem v dodanih vozliščih in d) s čakanjem na cesti.



Slika 7: Primerjava računskih časov osnovnega algoritma in njegovih nadgradenj.

Računski čas osnovnega SIPP algoritma je relativno kratek cca. 0,1 s in se z večanjem števila vozlišč v grafu povečuje. Ovrednotena računska zahtevnost predstavljenih nadgradenj v tem primeru kaže, da v primeru čakanja na cesti algoritem potrebuje 36 % več računskega časa za

izračun časovnega načrta. Pri rešitvi z vmesnimi vozlišči pa zaradi 21 dodatnih vozlišč (skupaj 28, prvotno 7) na poti vseh treh vozil potrebuje SIPP 57 % več računskega časa, za izračun časovnega načrta pa 51 % več.

6 Zaključek

V prispevku je opisano delovanje algoritma SIPP, ki je nadgradnja algoritma A* z uporabo varnih intervalov in iskanje naslednikov, kakor tudi večagentno planirane poti s prioritetami ter simulacijsko okolje za razvijanje algoritmov.

Obe predstavljeni izvedbi nadgradnje osnovnega planiranja poti, tako dodajanje vmesnih vozlišč kakor čakanje na cesti izboljšata pretočnost zemljevida, saj premaknemo čakanje višjeprioritetnih vozil iz končnih vozlišč ceste in jih tako sprostimo za tiste z nižjo prioriteto, ki lahko posledično hitreje dosežejo cilj. Kot je prikazano na primeru, veliko število vmesnih vozlišč pričakovano daljša računski čas, tako SIPP algoritma kakor izračun časovnega plana.

Rešitev za dodajanje vozlišč uporabniku zelo olajša delo z zemljevidom, saj bi drugače moral ročno spreminjati obstoječi zemljevid (vnašanje novih vmesnih cest). Prav tako nakazuje možnosti nadaljnje uporabe, kot so npr. dodajanje odstavnih niš in polnilnih postaj ob obstoječe ceste.

7 Literatura

[1] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, R. Bartak. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. Symposium on Combinatorial Search (SoCS), str 151-159, 2019.

[2] G. Klančar, A. Zdešar, S. Blažič, I. Škrjanc. Avtonomni mobilni sistemi: kolesna vozila. Fakulteta za elektrotehniko, 2021.

[3] M. Phillips, M. Likhachev. SIPP: Safe interval path planning for dynamic environments. Proceedings - IEEE International Conference on Robotics and Automation, str. 5628-5635, 2011.

[4] D. Herrero-Perez, H. Martinez-Barbera. Coordination of AGVs in an industrial environment. Proceedings - 7th international joint conference on Autonomous agents and multiagent systems: demo papers, str. 1709-1710, 2008.

[5] A. Zdešar, M. Bošnak in G. Klančar. Cyber-physical platform with miniature robotic vehicles for research and development of autonomous mobile systems. V I. Petrovic, E. Menegatti in I. Marković (Uredniki), Intelligent Autonomous Systems 17, str. 897-908. Springer Nature Switzerland, Cham, 2023.