Univerza v Ljubljani
Fakulteta za elektrotehniko

Laboratorij za Modeliranje, Simulacijo in Vodenje

Laboratorij za Avtonomne Mobilne Sisteme

Avtonomni mobilni sistemi

Izr. prof. dr. Gregor Klančar

gregor.klancar@fe.uni-lj.si

Planiranje poti

2013/2014

Uvod

- Robot deluje v okolju
- Mobilnost zahteva:
  - načrtovanje premika iz točka A do točke B
  - izračun potrebnih vhodov (regulacija) za aktuatorje
  - upravljanje aktuatorjev
- Okolje je lahko:
  - Znano, podano z zemljevidom okolja (znane pozicije ovir in prosti prostor) -> možno je planiranje poti (predhodna določitev pred izvedbo), robot na podlagi zaznav senzorjev in zemljevida planira pot do cilja.
  - Neznano, robot se odloča le glede na zaznave senzorjev in izvaja regulacijo, tako da se usmerja proti cilju.

Uvod...

- **Stanje** mobilnega sistema oz. **konfiguracija** je katerakoli točka v kateri se sistem lahko znajde
- iz enega stanja v drugega pride sistem z **akcijami**
- **Pot** je opisana z zaporedjem akcij, ki vodijo sistem od začetnega do končnega stanja
- Med začetnim in ciljnim stanjem mimo ovir lahko obstaja več poti, pot definirajo dodatne zahteve (kriterij optimalnosti):
  - Pot naj bo najkrajša,
  - Pot naj bo izvedena v najkrajšem času,
  - Pot naj bo čim bolj oddaljena od ovir,
  - Pot naj bo gladka,
  - Pot naj upošteva omejitve gibanja,...

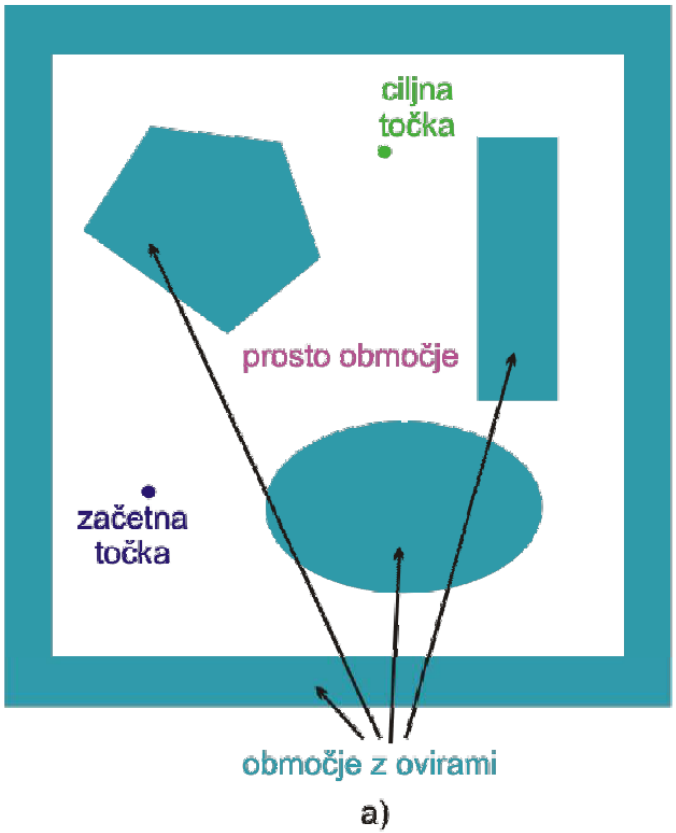
Planiranje poti

- Pristopi, ki zahtevajo znan zemljevid okolja:
  - Matematična predstavitev okolja (graf prehajnaja stanj, potencialno polje, razcep na celice,...)
  - Planiranje želene poti na podlagi cilja, zemljevida in trenutnih informacij senzorjev (lege)
  - Vodenje po želeni poti na podlagi informacije senzorjev (lega, detekcija neznanih ovir)
  - Lastnosti: bolj optimalna pot, zahtevnejši izračun

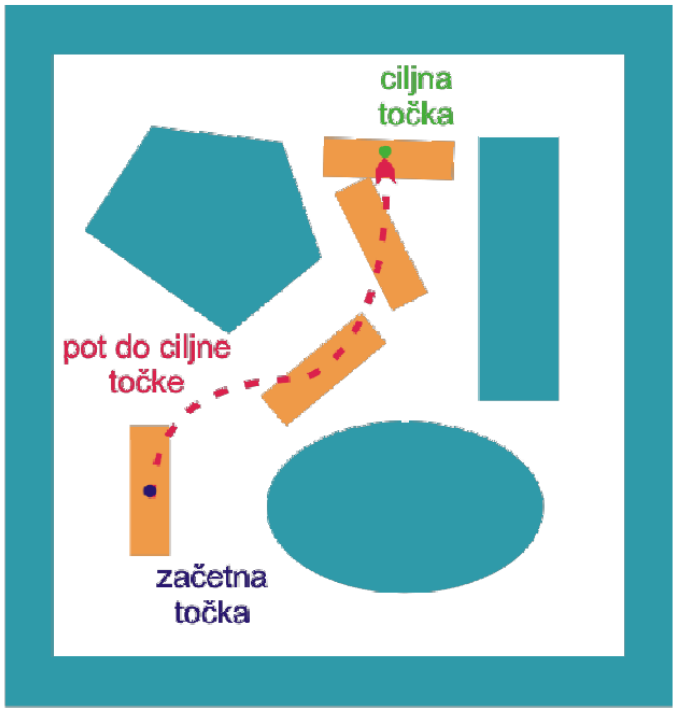
- Enostavni pristopi, ki ne rabijo zemljevida okolja:
  - Vodenje na podlagi znanega cilja in informacije senzorjev (lega in detekcija ovir)
  - Lastnosti: pot je suboptimalna, možna osciliranja, lahko obtiči v lokalnem minimumu

Planiranje poti

- Proces iskanja zvezne poti, od začetne do končne točke. Pot mora leži v prostem območju.



a)



b)

Konfiguracija in konfiguracijski prostor

- **Konfiguracija** je stanje MS v okolju, ki ga opišemo z vektorjem stanj $\mathbf{q} = [q_1, \cdots, q_n]^T$, kjer je *n* število prostostnih stopenj.

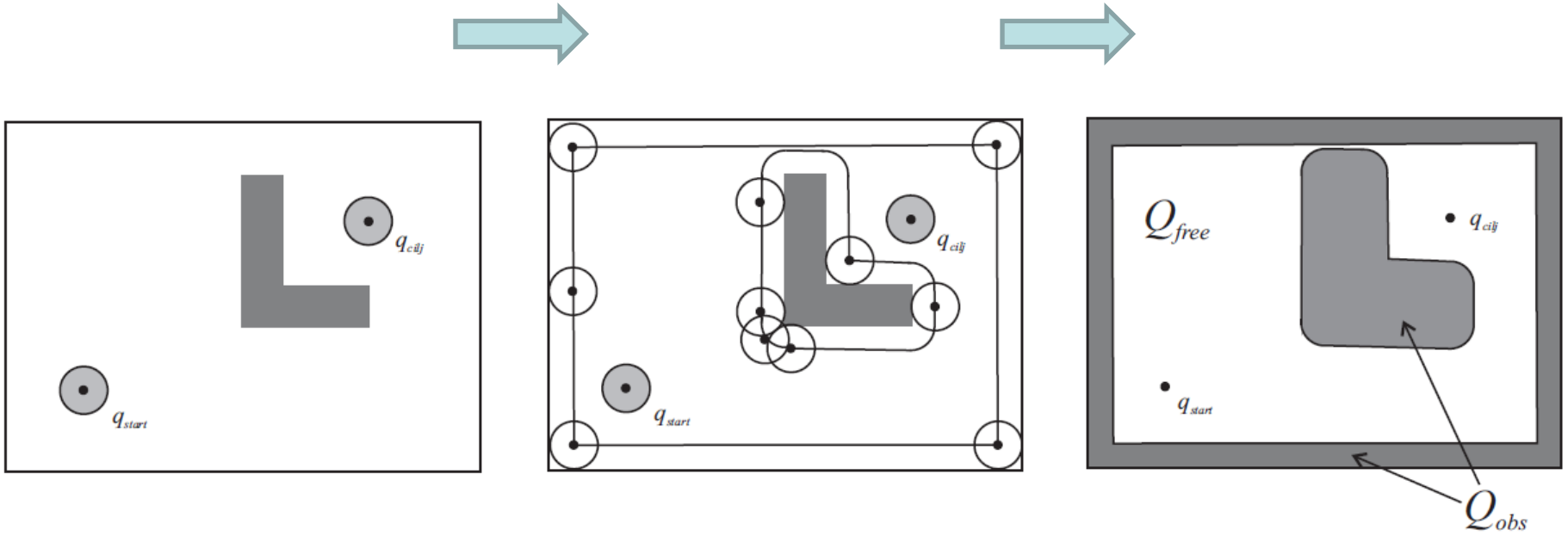
- **q** je točka v **konfiguracijskem prostoru** *Q*, ki predstavlja vse možne konfuguracije MS (upoštevajoč kinematiko).

- konf. prostor *Q* vsebuje prostor zaseden z ovirami Q_{obst} in prosti prostor Q_{free}, kjer se MS lahko nahaja.

$$Q_{free} = Q - Q_{obst}$$

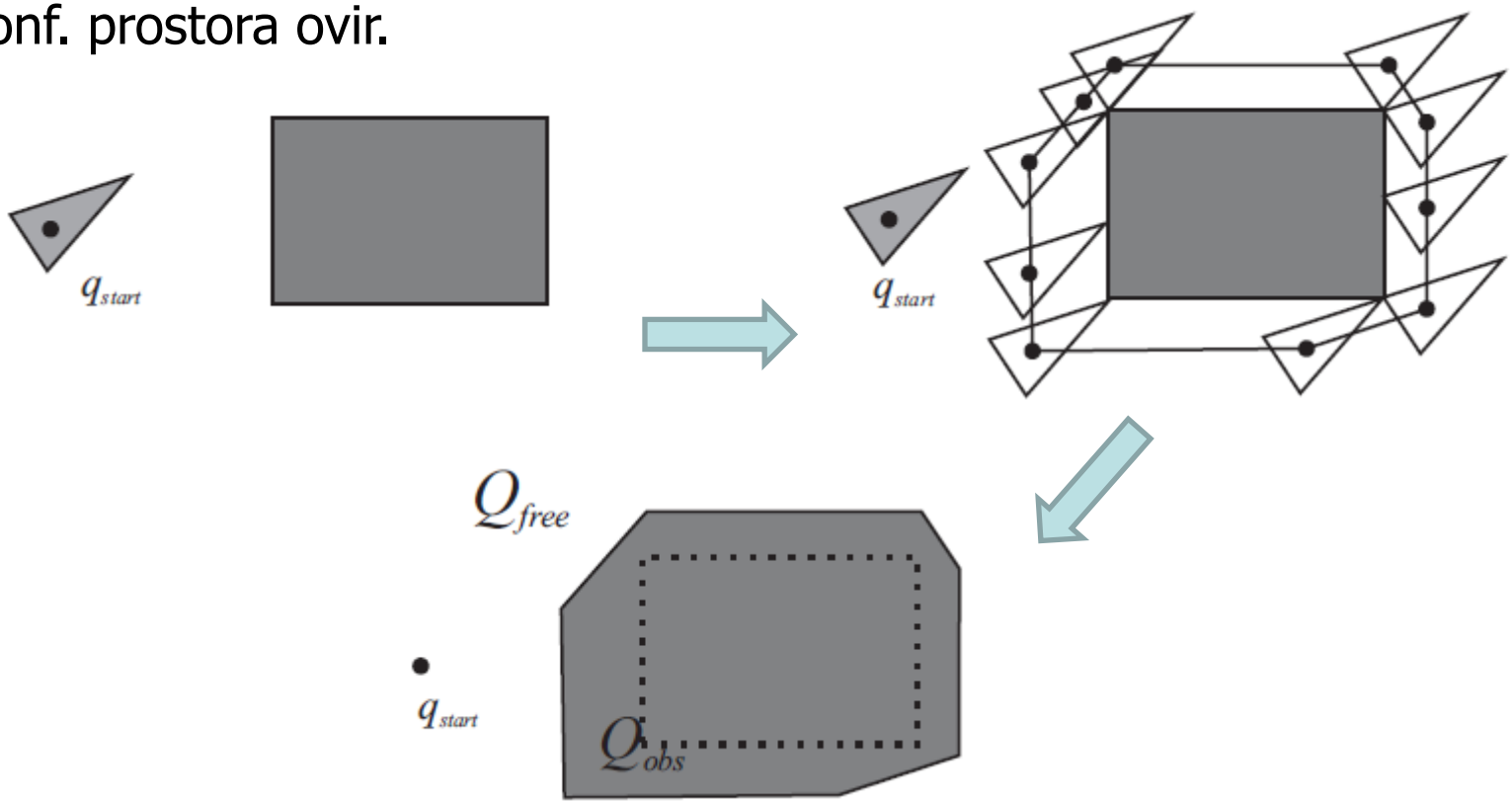
Primer določitve konfiguracijskega prostora

- Krožni robot, ki se lahko premika v ravnini z translacijo $q=[x, y]$ (orientacija nas ne zanima).
- Točkovna obravnava konfiguracije MS-> ovire razširimo za dimenzije MS
- To nam omogoča, da MS obravnavamo kot točko

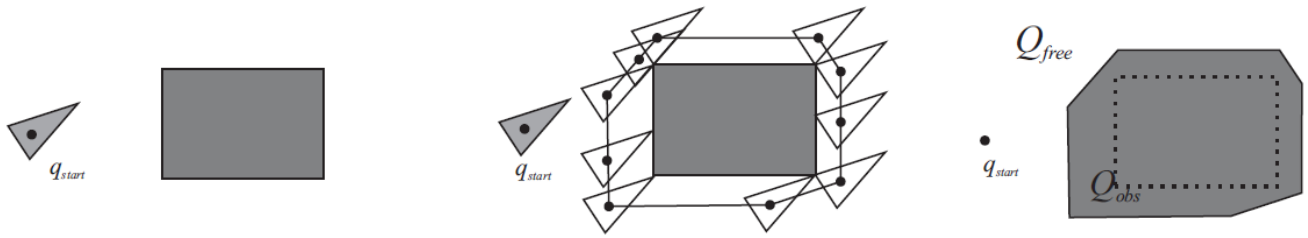


Primer določitve konfiguracijskega prostora...

- MS trikotne oblike, možne le translacije v ravnini $q=[x, y]$ (orientacija nas ne zanima).
- Točkovna obravnava MS zahteva določitev prostega konf. prostora in konf. prostora ovir.



Primer določitve konfiguracijskega prostora...



- Če bi MS lahko še rotiral je konfiguracija $q=[x, y, \varphi]$ in konf. prostor 3D

Poenostavitev določite konfiguracijskega prostora:

- Robotu določene oblike očrtamo krog (gibanje v ravnini) (oz. kroglo za 3D), ki ima središče v točki robota, ki predstavlja njegovo pozicijo.
- Dobljeni Q_{free} je manjši od dejanskega.

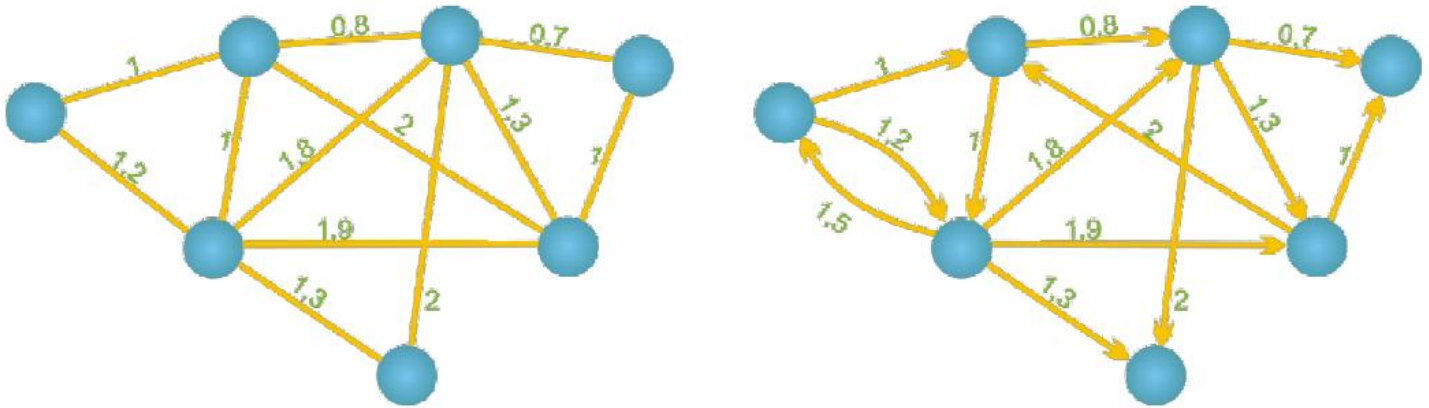
Predstavitve okolja za planiranje poti

Okolje moramo predstaviti na matematičen način. Možne predstavitve so:

- Graf prehajanja stanj
- Razcep na celice
- Zemljevid cest
- Potencialna polja
- Metode vzorčenja prostora

Graf prehajanja stanj

- Prosto okolje predstavimo z le neko podmnožico konfiguracij oz. stanj (npr. središča področij ali celic).
- Imamo začetno konfiguracijo, ciljno konfiguracijo, vmesne konfiguracije in možne povezave (prehode) med njimi.
- Graf stanj vsebuje vozlišča (stanja) in povezave (črte).
- **Utežen graf** ima za vsako povezavo določeno utež oz. ceno
- **Usmerjen graf** dodatno določa še smeri možnih prehodov med vozlišči.
- Iskanje poti v grafu prehajanja stanj se izvede z enim od algoritmom iskanja poti v grafu (A*, Dijkstrov,…)

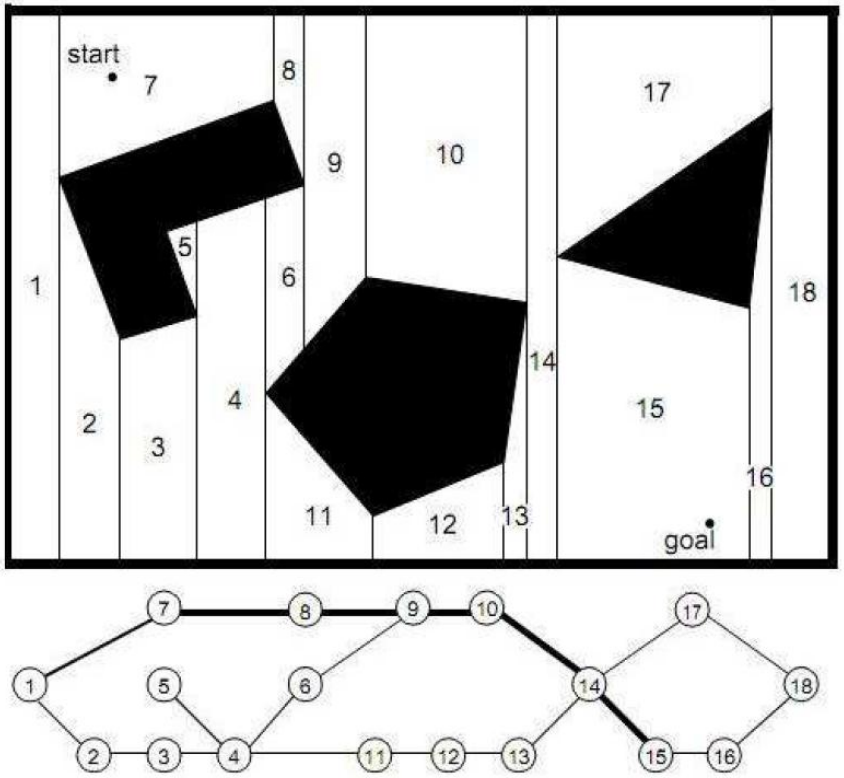


Razcep na celice

- Okolje razdelimo na celice (enostavne geometrijske like)
- Na celice razcepljeno okolje lahko predstavimo z grafom prehajanja stanj. Vozlišča grafa so določene točke celic (npr. težišča), povezave grafa pa so možne med sosednjimi celicami (skupna stranica).
- Možen je natančen ali približen razcep na celice

Natančen razcep na celice

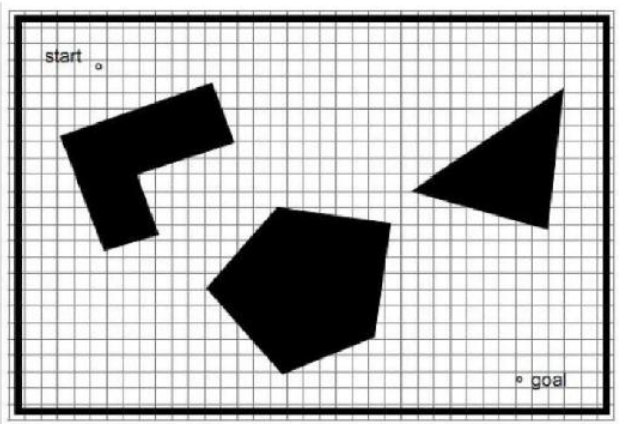
- Celice v celoti ležijo le v prostem ali le v področju z ovirami.
- Brezizguben razcep, unija vseh prostih celic enak *Qfree*
- Primer: navpični razcep in predstavitev z grafom prehajanja stanj (vozlišča so centri celic, povezave med sosednjimi celicami).

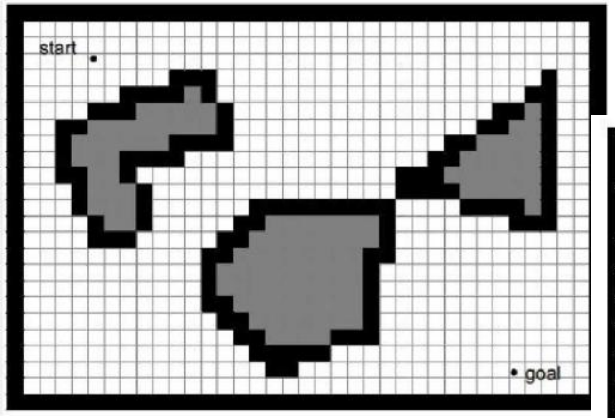


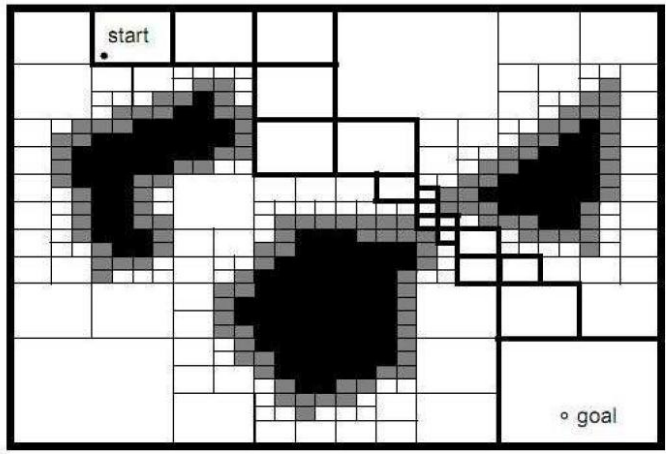
- Omejena Delaunayeva triangulacija (primer v Matlabu)

Približen razcep na celice

- Možno je, da je v celici prosti konfiguracijski prostor, kot tudi del ovire.
- Celice, ki vsebujejo ovire ali le del ovire označimo kot zasedene, ostale pa proste.
- (+) Enostavno za uporabo.
- (-) Izguba informacij – ozke poti se lahko izgubijo, (konstantno velike celice ali navzdol omejena velikost celic).
- Primera: konstantna velikost celic in spremenljiva velikost celic - štiriško drevo (Matlab). Zasedene celice delimo na 4 manjše…

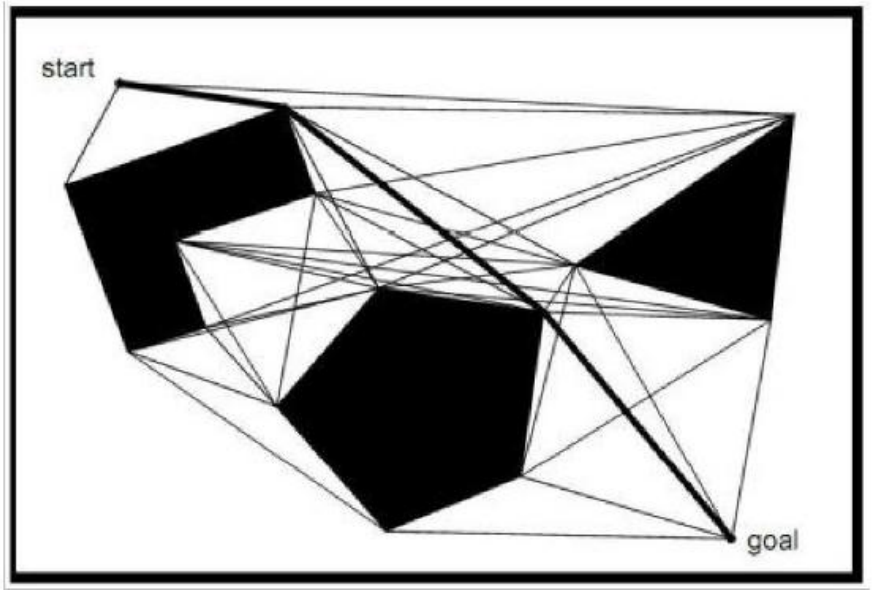


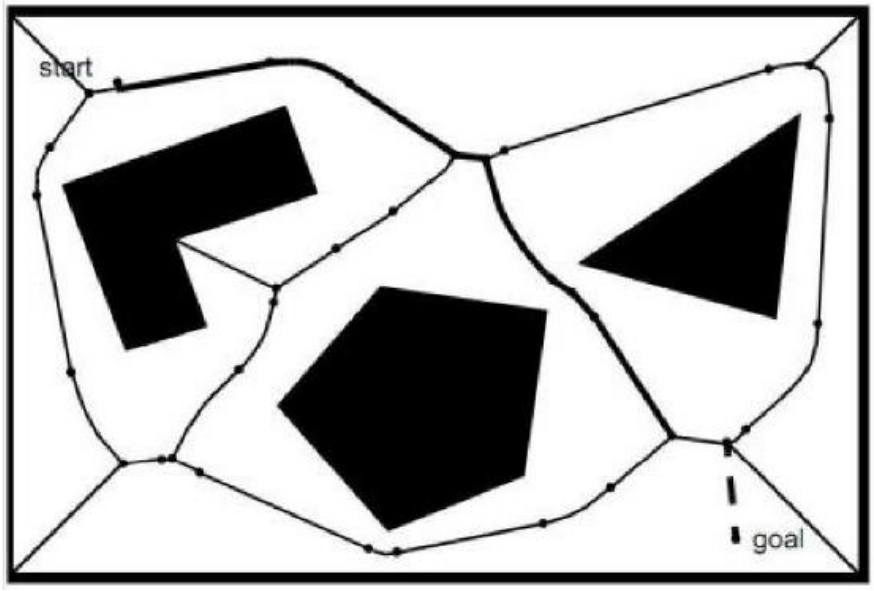




Zemljevid cest

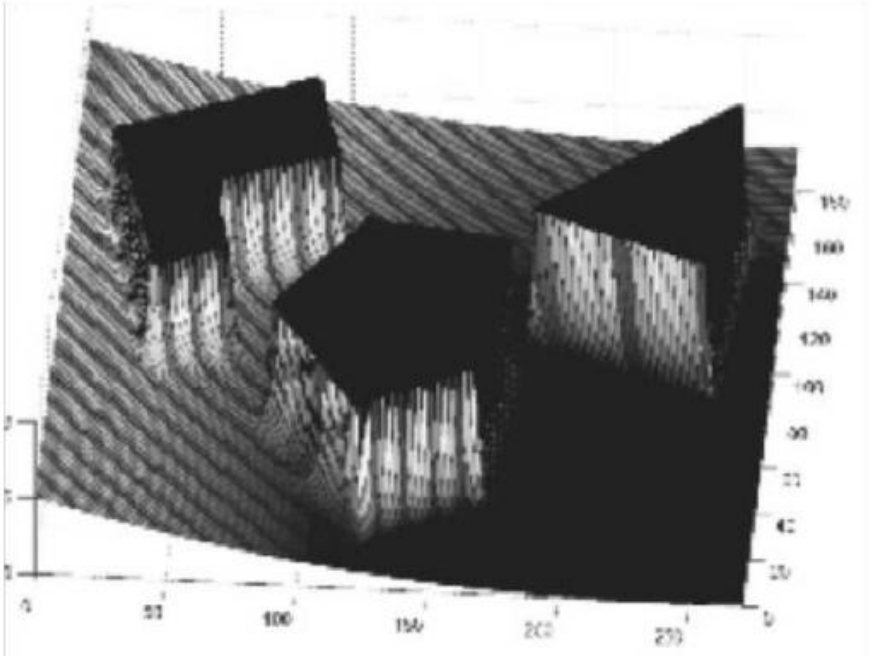
- Zemljevid cest (road map) podaja možno povezljivost prostega območja okolja
- Planiranje poti poišče povezujoče zaporedje cest.
- **Graf vidljivosti** povezuje oglišča ovir preko prostega prostora,. Dobljena pot je najkrajša, ker so ceste speljane kar se da blizu ovir
- **Veronoiev graf** podaja odseke poti, katerih oddaljenost od ovir je največja (poti med dvema ovirama je enako oddaljena od obeh). Dobljena pot je maksimalno oddaljena od ovir, ni pa kratka.

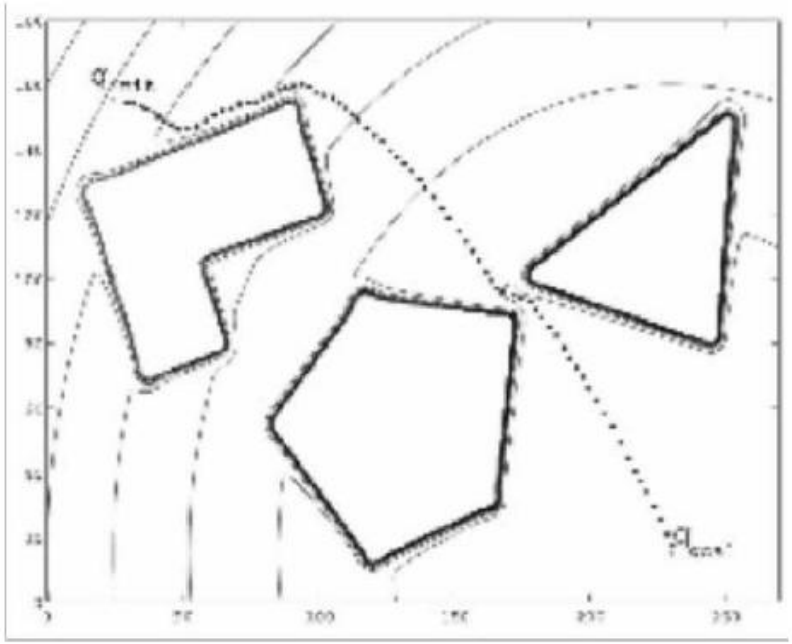




Planiranje poti s potencialnim poljem

- Definiramo potencialno polje, vsoto privlačnega potenciala zaradi cilja in odbojnega potenciala zaradi ovir.
- Potencial si predstavljamo kot relief, kjer je relief nagnjen k cilju, ki je najnižje v dolini, ovire pa so višje.
- Pot robota je definirana z negativnim gradientom potencialnega polja (smer največje spremembe). Robot sledi neg. gradientu in doseže cilj

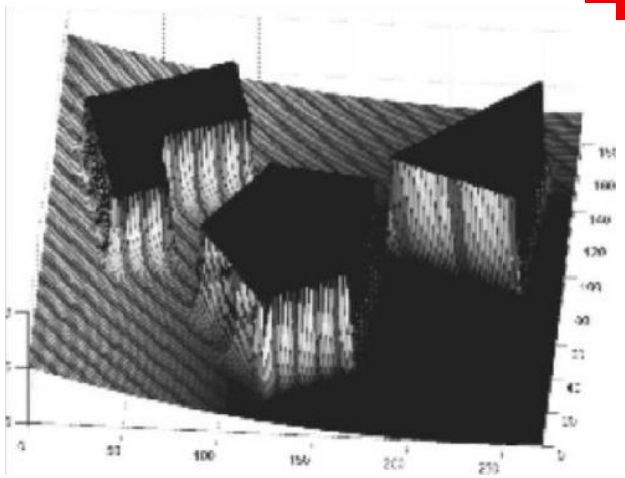




Potencialno polje...

- Definiramo potencialno polje

$$U(\mathbf{q}) = U_{atr}(\mathbf{q}) + U_{rep}(\mathbf{q})$$



- Privlačno polje

$$U_{atr}(\mathbf{q}) = k_{atr}\frac{1}{2}D^2(\mathbf{q}, \mathbf{q}_{goal})$$

$$D(\mathbf{q}, \mathbf{q}_{goal}) = \sqrt{(x - x_{goal})^2 + (y - y_{goal})^2}$$

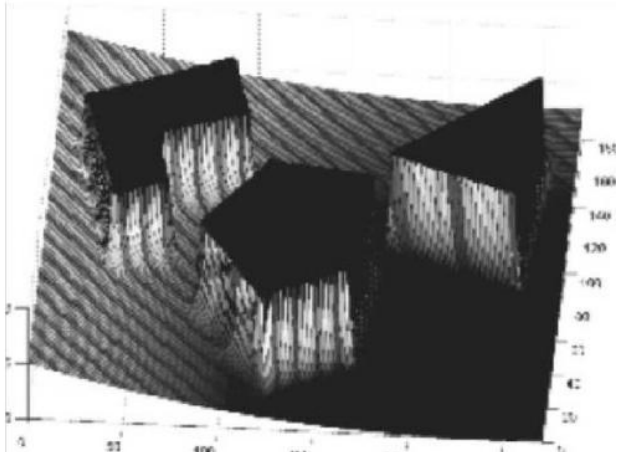
- Odbojno polje

$$U_{rep}(\mathbf{q}) = \begin{cases} \frac{1}{2}k_{rep}\left(\frac{1}{D(\mathbf{q},\mathbf{q}_{obst})} - \frac{1}{D_0}\right)^2 & ; \quad D(\mathbf{q}) \leq D_0 \\ 0 & ; \quad D(\mathbf{q}) > D_0 \end{cases}$$

Potencialno polje...

- Premikamo se v smeri negativnega gradienta potencialnega polja

$$-\nabla U(\mathbf{q})$$



- Gradient privlačnega polja

$$-\nabla U_{atr}(\mathbf{q}) = -k_{atr}\frac{1}{2}[2(x - x_{goal}), 2(y - y_{goal})]^T = k_{atr}(\mathbf{q}_{goal} - \mathbf{q})$$

- Gradient odbojnega polja (za $D(\boldsymbol{q}) < Do$)

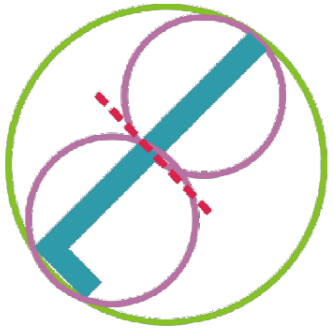
$$\begin{aligned} -\nabla U_{rep}(\mathbf{q}) &= -k_{rep}\left(\frac{1}{D_{obst}} - \frac{1}{D_0}\right)\frac{-1}{D_{obst}^2}\nabla D_{obst} \\ &= k_{rep}\left(\frac{1}{D_{obst}} - \frac{1}{D_0}\right)\frac{1}{D_{obst}^2}(\mathbf{q} - \mathbf{q}_{obst}) \end{aligned}$$

Potencialno polje...

- Uporaba:
  - Za planiranje želene poti, če imamo znan zemljevid okolja, začetno točko in podano ciljno točko. Nato želeni poti sledimo v vodenjem in upoštevanjem informacije senzorjev.
  - Zemljevid ni znan, zato izvajamo neposredno vodenje na podlagi znanega cilja in informacije senzorjev (lega vozila in detekcija ovir). Želene poti ne poznamo, opravljena pot vozila je tako implicitno določena s potencialnim poljem in uporabljenim regulatorjem.
- Možno je , da vozilo obtiči v lokalnem minimumu, potrebna je primerna izbira funkcij za privlačno in odbojno potencialno polje.

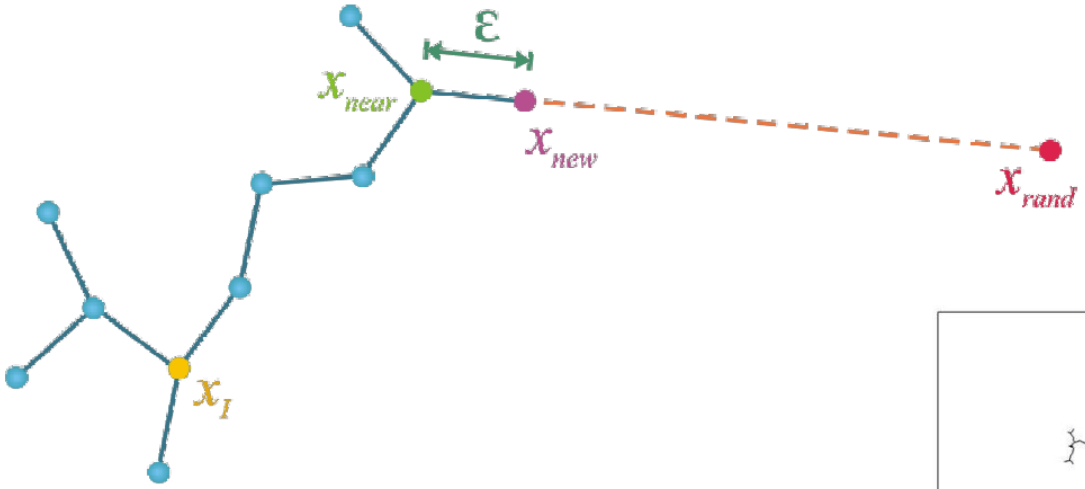
Metode vzorčenja prostora

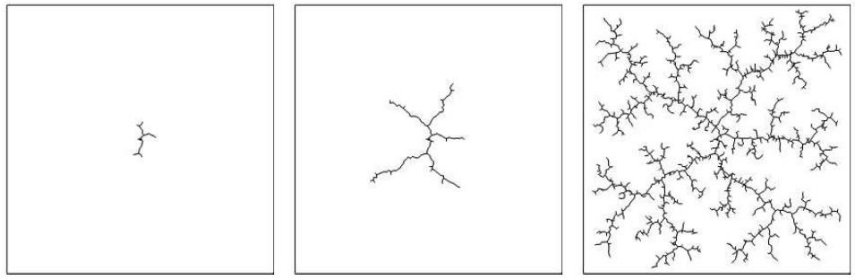
- Za velika okolja so metode za eksplicitno predstavitev prostega okolja (graf, delitev na celice, zemljevid cest,…) zahtevna in zato časovno potratne
- Metode vzorčenja prostora naključno zajemajo točke iz prostora in preverjajo, če so v prostem območju. Iz več takih točk poiščemo pot med začetno in ciljno točko.
- (+) Pri velikih okoljih je računsko učinkovit, saj izračun prostega konfiguracijskega prostora Q_{free} ni potreben in tudi ne delitev na veliko število celic. Omili tudi problem lokalnih minimumom, ker so premiki naključni.
- (-) Rešitev je nedeterministična, potrebno izvesti zaznavanje trka (možno izvesti hierarhično).



RRT - metoda vzorčenja prostora

- RRT (rapidly-exploring random tree), poišče pot med začetno in končna točko
- Algoritem: Ob vsakem koraku izberemo naključno točko *Xrand*, poiščemo najbližjo točko *Xnear*, ki je že v grafu. V smeri od *Xnear* do *Xrand* se na razdalji ε določi točka *Xnew*. Če *Xnew* leži v prostem območju, potem graf razširimo z novim vozliščem *Xnew* in povezavo le-tega z *Xnear*.

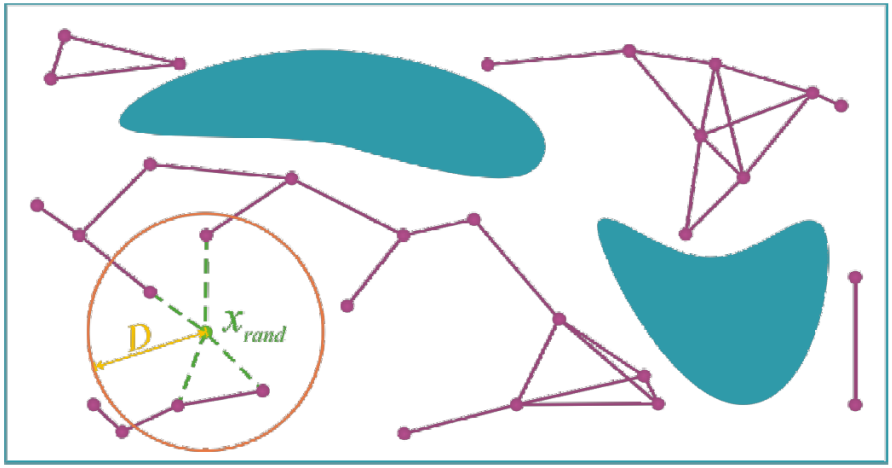




- Drevo dobljeno z RRT se hitro širi po neraziskanem področju

PRM - metoda vzorčenja prostora

- PRM (probalistic roadmap) je metoda iskanja poti med več začetnimi in več ciljnimi točkami
- Algoritem:
  - Naključno izberemo konfiguracijo *Xrand* v prostem območju dodamo v zemljevid
  - Določimo vozlišča *Xn* za razširitev zemljevida (oddaljena razdaljo *D* od *Xrand*)
  - V zemljevid dodamo vse enostavne povezave od *Xrand* do vozlišč *Xn*, ki so v prostem območju
  - Nadaljujemo do želenega števila vozlišč

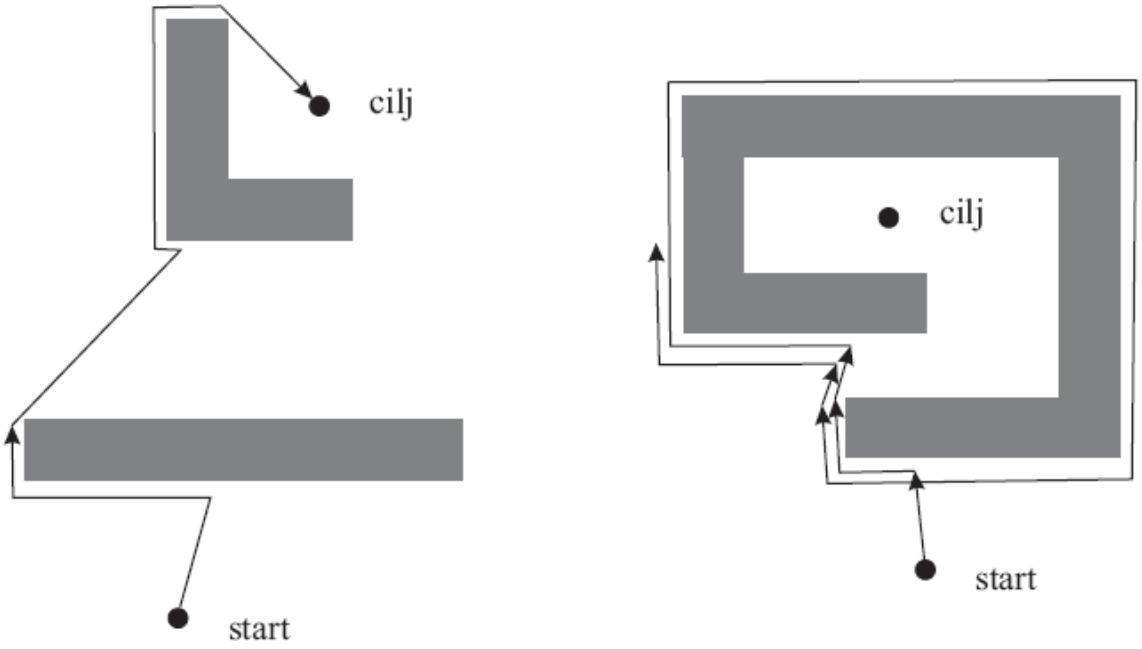


Enostavni algoritmi- hrošč

- Lokalna informacija okolja (senzorji)
- Zemljevid okolja ni potreben
- Če je zemljevid znan, ga lahko uporabimo za planiranje poti
- Enostavno delovanje:
  - Gibanje proti cilju v direktni liniji
  - Sledenje obrisu ovire
- Oponašanje gibanja enostavnih živih bitij
- Primeri: Hrošč 0, Hrošč 1, Hrošč 2

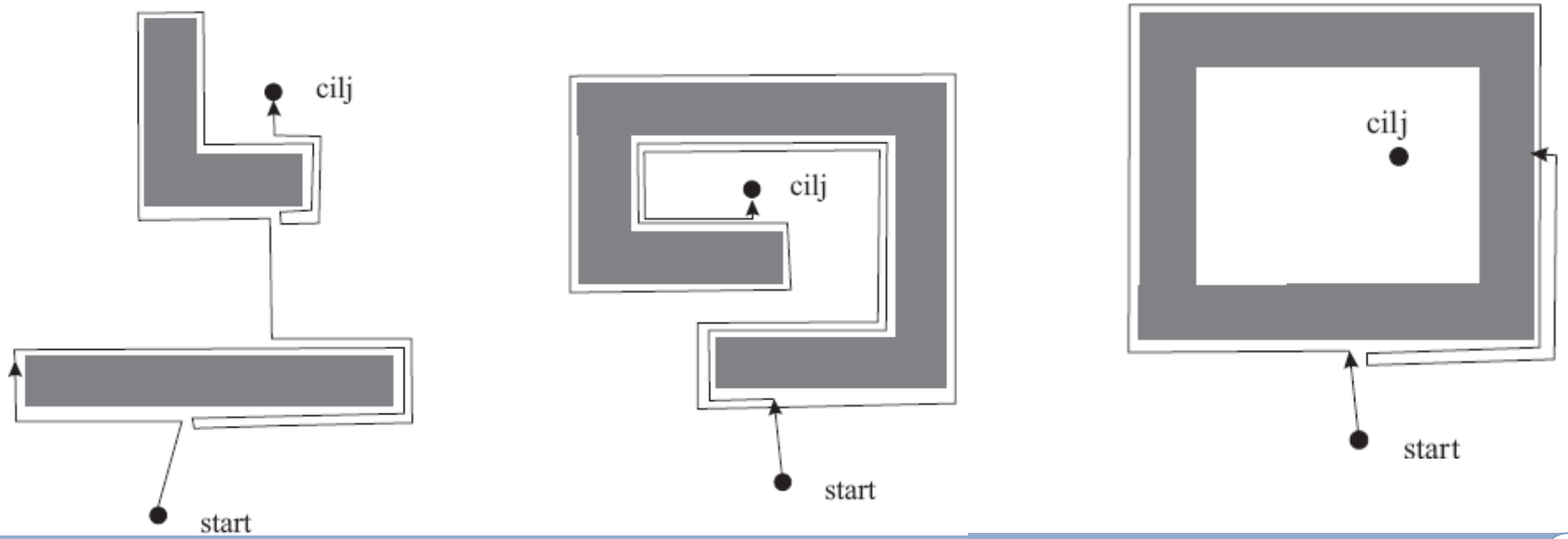
Hrošč 0

- Algoritem:
  - V ravni liniji se premika proti cilju, dokler ne naleti na oviro ali cilj.
  - Če naleti na oviro ob oviri zavije levo (oz. vedno desno, če je tako določeno v algoritmu) in sledi obrisu ovire, dokler ni možno ponovno nadaljevati proti cilju.



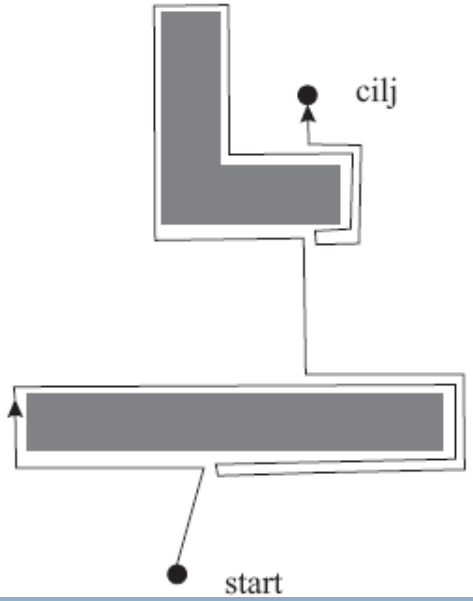
Hrošč 1

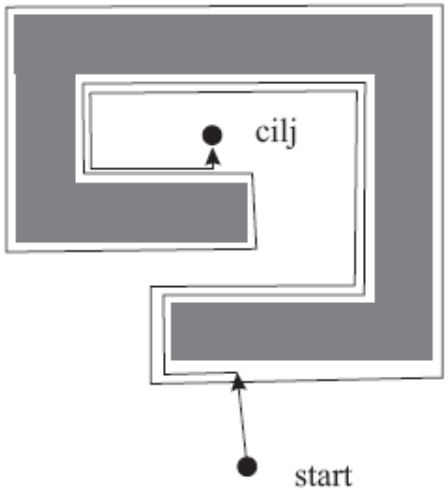
- Rabi nekaj spomina in več računanja (izračun razdalje do cilja)
- Algoritem:
  - V ravni liniji se premika proti cilju, dokler ne naleti na oviro ali cilj.
  - Če naleti na oviro ob oviri zavije levo in sledi celotnemu obrisu ovire ter ves čas meri Evklidsko razdaljo do cilja. Ko prispe do točke, kjer je naletel na oviro, gre po krajši poti ob obodu ovire do točke, ki je bila najbliže ciliu.

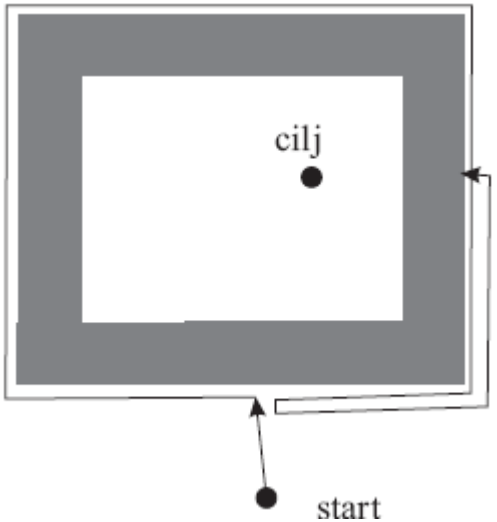


Hrošč 1...

- Dobljena pot ni optimalna .
- Pot je lahko je za 3/2 obsega vseh ovir daljša od direktne linije do cilja.
- Na oviro naleti največ enkrat, ker na oviro naleti le v eni točki in jo zapusti le v eni točki.
- Zna ugotoviti nedosegljiv cilj.

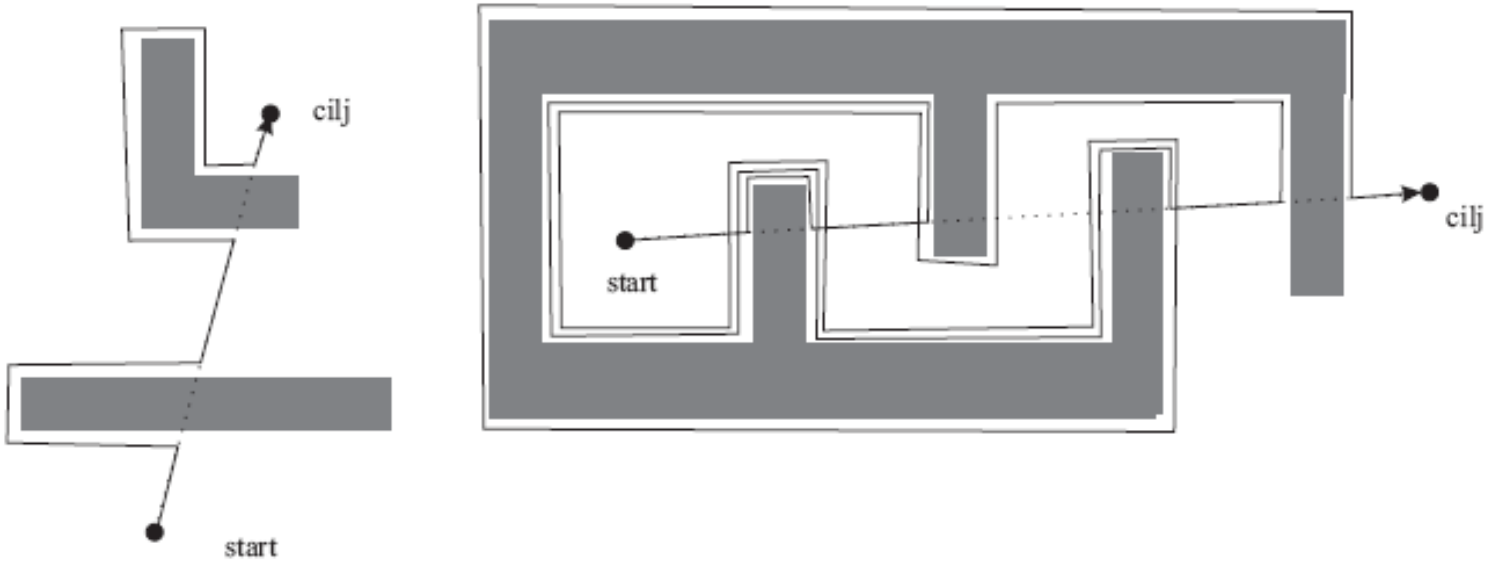






Hrošč 2

- Poskuša se gibati po ravni (glavni) liniji med začetno in ciljno točko.
- Algoritem:
  - Robot naj se premika po glavni liniji, dokler ne naleti na oviro ali ciljno točko.
  - Robot sledi obodu ovire toliko časa, da doseže glavno linijo, kjer je Evklidska razdalja do cilja manjša kot Evklidska razdalja točke, kjer je trenutno (zadnjič) naletel na oviro.



Hrošč 2...

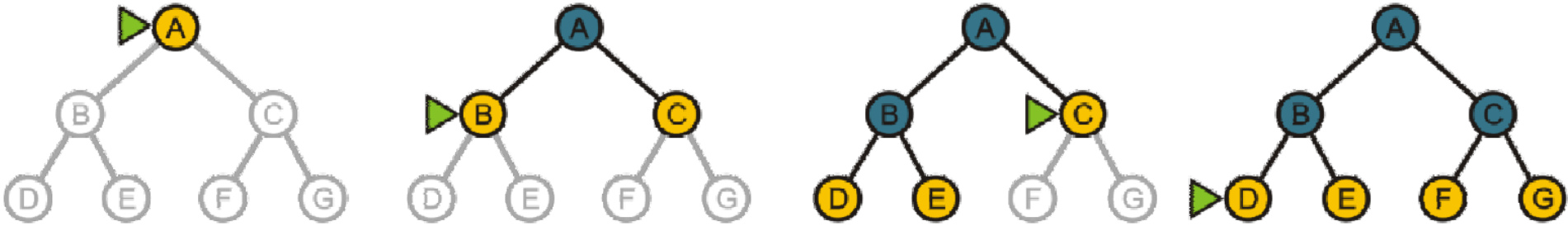
- Je v večini primerov bolj učinkovit kot hrošč 1
- Je (kot hrošč 1) temeljit, preišče vse možnosti pred izvedbo naslednjega koraka
- Lahko dalj časa po nepotrebnem kroži okoli ovir preden pride do cilja (glede na hrošč 1)
- Zna razbrati, kdaj cilj ni dosegljiv (če večkrat v isti točki naleti na isto oviro)
- Nima tako predvidljivega obnašanja kot hrošč1

Metode iskanja poti v grafu

- Poiščemo pot v grafu (prostor stanj, razcep na celice, zemljevid cest,…)
- Pri iskanju vodimo sezname že obiskanih vozlišč. **Seznam odprtih vozlišč** (vsebuje še neraziskane naslednike) in **seznam zaprtih vozlišč** (nimajo neraziskanih naslednikov)
- Iskanje poti v grafu je lahko **neinformirano** (slepo iskanje v grafu) ali **informirano** (imamo še dodatno informacijo, s pomočjo katere izbiramo bolj obetavne celice)
- Iskalni algoritem je lahko **popoln** (če obstaja pot jo najde) in je **optimalen** (najdena pot je najkrajša oz. najcenejša).
- Podanih je nekaj metod iskanja poti v grafu

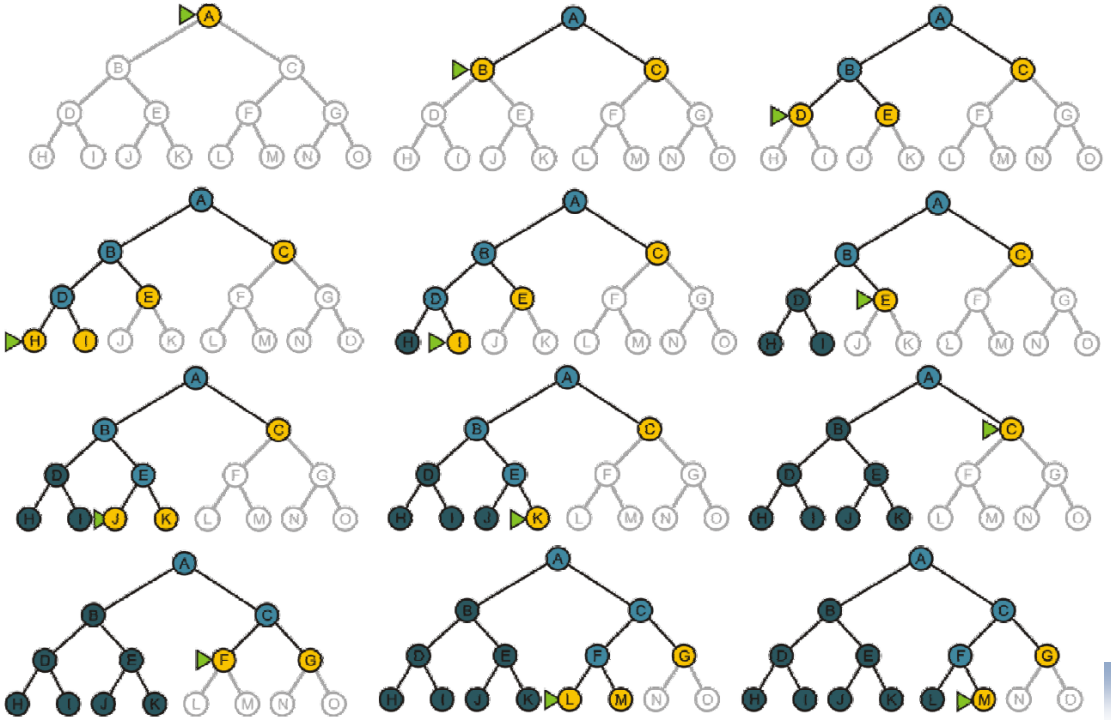
Iskanje v širino

- Najprej raziščemo plitva vozlišča (tista bližje začetnemu vozlišču)
- Seznam odprtih vozlišč vodimo po metodi FIFO. Vozlišča za razširitev jemljemo iz začetka vrste, nova vozlišča pa pripenjamo na konec vrste
- Algoritem je neinformiran, popoln in ni optimalen
- Ima velika porabo spomina in dolg računski čas - naraščata eksponentno z razvejanostjo drevesa



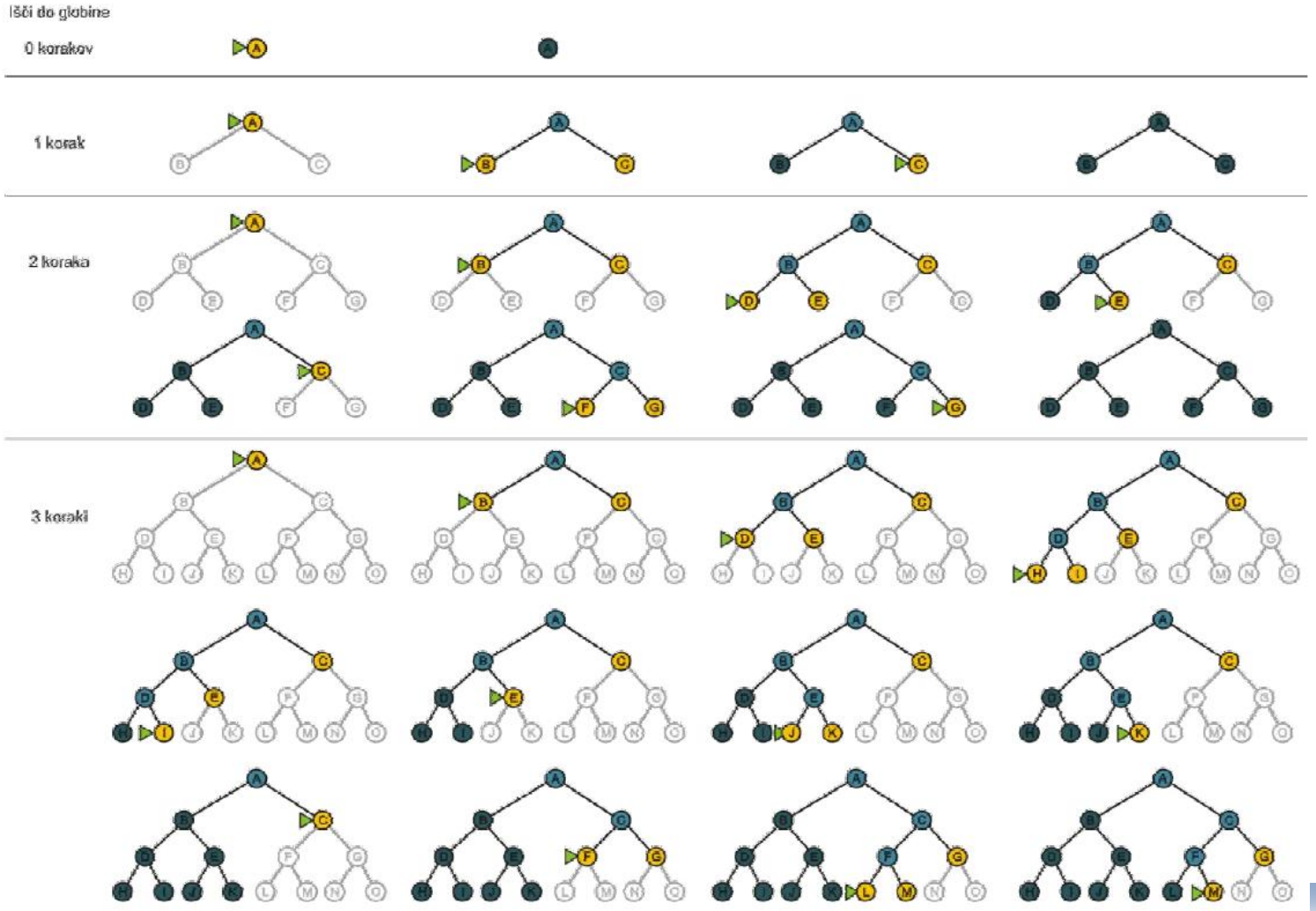
Iskanje v globino

- Iskanje širimo v globino. Razširimo vozlišče, ki je najbolj oddaljeno od začetnega.
- Odprti seznam vodimo po metodi LIFO. Nova odprta vozlišča dodajamo na začetek vrste od koder tudi jemljemo vozlišča za razširitev iskanja.
- Algoritem je neinformiran, nepopoln (primer neomejene globine) in ni optimalen.
- Ima majhno porabo spomina (hranimo le trenutno pot od začetnega do trenutnega vozlišča in vmesna vozlišča, s katerimi še nismo nadaljevali iskanja).



Iterativno poglabljanje iskanja v globino

- Združi prednosti iskanja v širino in globino.
- Po korakih večamo globino, do katere raziskujemo v globino
- Je popoln in optimalen (če so cene prehodov enake)



Dijkstrov algoritem

- Neinformiran algoritem, poišče najkrajše poti od enega začetnega vozlišča do vseh preostalih vozlišč.
- Računamo ceno poti od začetnega do trenutnega vozlišča (cena-do-sem)
- Za vsako obiskano vozlišče shranimo ceno trenutno najkrajše poti do njega in povezavo po kateri smo prišli do tega vozlišča.

- Algoritem je popoln in optimalen.

Dijkstrov algoritem - algoritem

1. Iz seznama odprtih vozlišč vzamemo prvo vozlišče, to naj bo trenutno vozlišče. Seznam naj bo urejen naraščajoče glede na ceno-do-sem, torej: prvo vozlišče je tisto z najmanjšo ceno-do-sem.

2. Vsem vozliščem, do katerih lahko pridemo iz trenutnega vozlišča in niso na seznamu zaprtih vozlišč, s pomočjo cene-do-sem trenutnega vozlišča in vsote cene vmesne povezave izračunamo ceno-do-sem.

3. Za vsako izmed teh vozlišč, ki izračunane cene-do-sem in ustrezne vmesne povezave od trenutnega vozlišča še nima shranjene, jo shranimo.

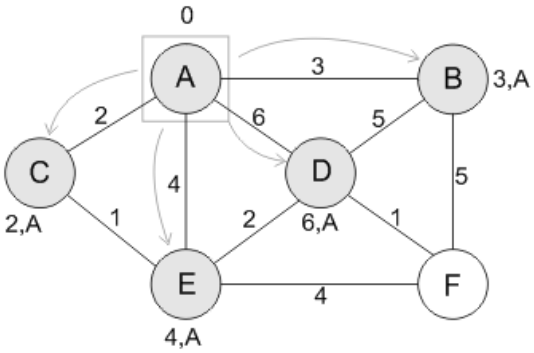
4. Če je v prejšnjem koraku katero izmed teh vozlišč že imelo shranjeno ceno-do-sem in ustrezno povezavo v grafu iz katere od prejšnjih iteracij, ti dve ceni primerjamo in kot končen podatek shranimo manjšo ceno in ustrezno povezavo.

5. Vozlišča dodamo na seznam odprtih vozlišč in ga uredimo po naraščajoči vrednosti cene-do-sem.

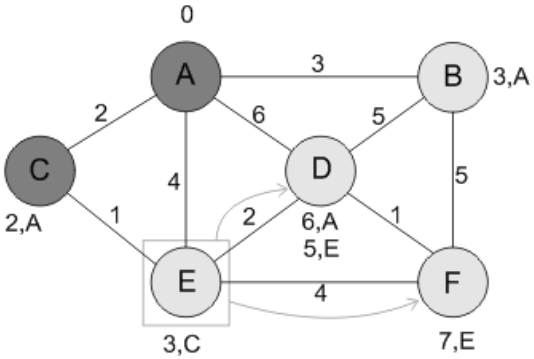
   Trenutno vozlišče premaknemo na seznam zaprtih vozlišč.

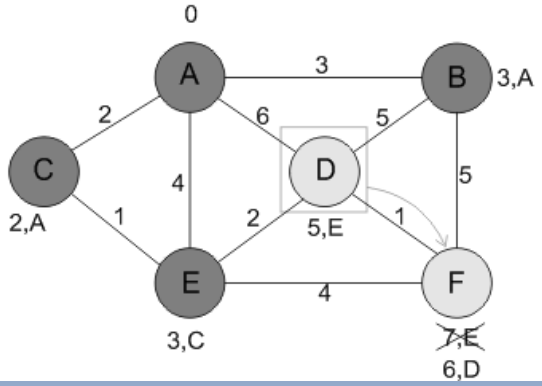
Dijkstrov algoritem - primer

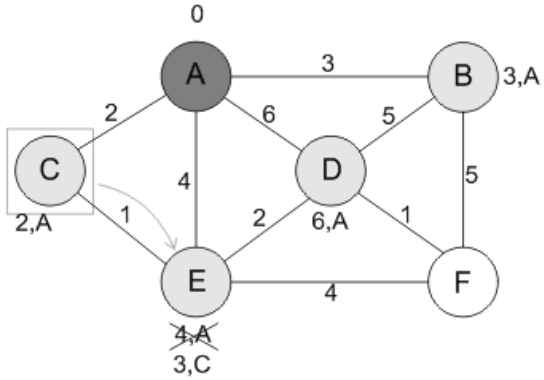
Poiščemo vse možne naslednike iz trenutnega vozlišča, določimo jim ceno in povezave

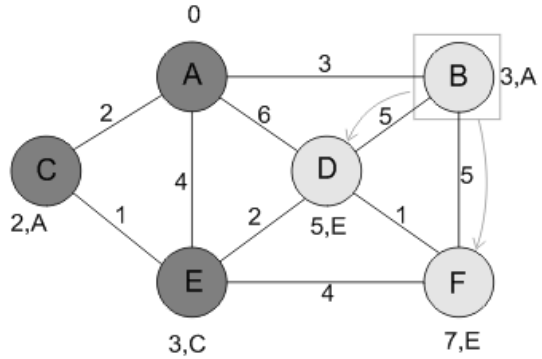


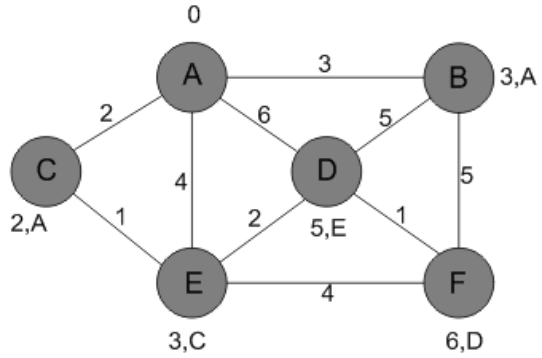
Izberemo novo trenutno vozlišče, prejšnje damo na seznam zaprtih…











Trenutno vozlišče damo na seznam zaprtih vozlišč. Za novo trenutno vozlišče izberemo tisto z najnižjo ceno. Določimo naslednike – je le eden, zapomnimo si novo ceno, ki je manjša.

Algoritem A*

- Je informiran algoritem iskanja, vsebuje dodatno informacijo – oceno poti od trenutnega vozlišča do cilja (cena-do-cilja)
- Zaradi te hevristike lahko loči med bolj obetavnimi in manj obetavnimi vozlišči (je učinkovitejši kot Dijkstrov).
- Med izvajanjem se za vsako vozlišče oceni *cena celotne poti* = *cena-do-sem* + *cena-do-cilja*

- Algoritem je popoln in optimalen, če je hevristika optimistična. Porabi veliko spomina.

- Hevristika je optimistična, če je ocena *cene-do-cilja* manjša oziroma kvečjemu enaka kot dejanska (Evklidova razdalja, oz. dolžina zračne linije do cilja ustreza tem pogoju). Če *cene-do-cilja* damo na 0 dobimo Dijkstrov algoritem.

Algoritem A* - algoritem

1. Iz seznama odprtih vozlišč vzamemo prvo vozlišče, to naj bo trenutno vozlišče. Seznam naj bo urejen naraščajoče glede na ceno-celotne-poti, torej: prvo vozlišče je tisto z najmanjšo ceno-celotne-poti.

2. Vsem vozliščem, do katerih lahko pridemo iz trenutnega vozlišča izračunamo

   - ceno-do-cilja,
   - ceno-do-sem kot vsoto cene do-sem trenutnega vozlišča in vmesne povezave ter
   - ceno-celotne-poti kot vsoto cene-do-sem in cene-do-cilja.

3. Za vsako izmed teh vozlišč, ki izračunane cene-do-sem, ustrezne vmesne povezave od trenutnega vozlišča, cene-do-cilja in cene-celotne-poti še nima shranjene, jo shranimo.

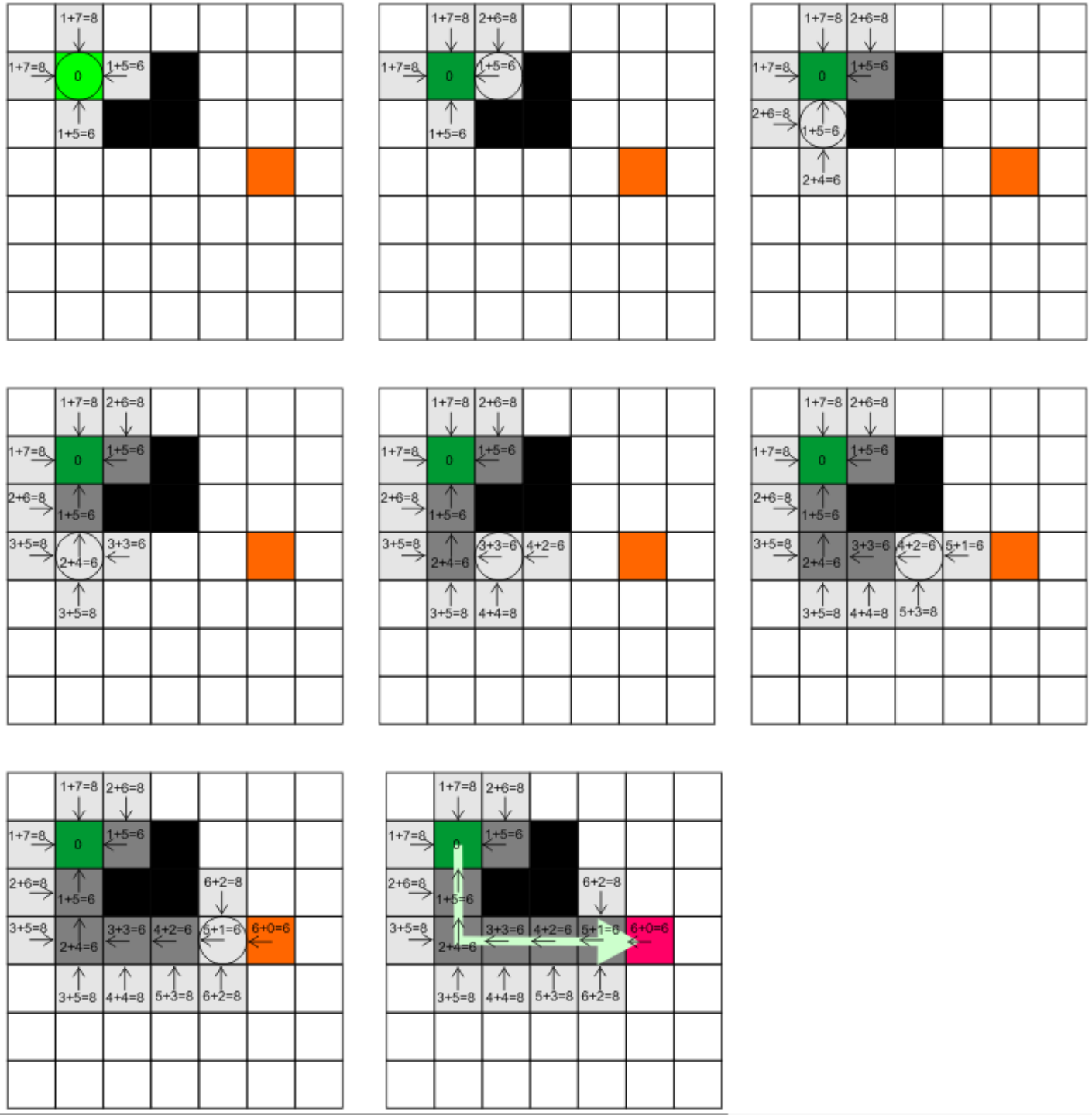
4. Če je v prejšnjem koraku katero izmed teh vozlišč že imelo shranjene izračunane vrednosti iz katere od prejšnjih iteracij, primerjamo obe ceni-do-sem in kot končen podatek shranimo manjšo ceno-do-sem, temu dodamo ustrezno povezavo in ustrezno ceno-celotne-poti.

5. Vozlišča, katerih vrednosti smo računali prvič, dodamo na seznam odprtih vozlišč. Vozlišča, ki smo jim posodobili vrednosti in so že bila na seznamu odprtih vozlišč, jih tam tudi obdržimo. Vozlišča, ki so bila na seznamu zaprtih vozlišč in katerih vrednosti smo posodobili (do njega smo našli pot z manjšo ceno-do-sem), premaknemo na seznam odprtih vozlišč, ki ga nato uredimo po naraščajoči vrednosti cene-celotne-poti. Trenutno vozlišče premaknemo na seznam zaprtih vozlišč.

Algoritem A* - primer

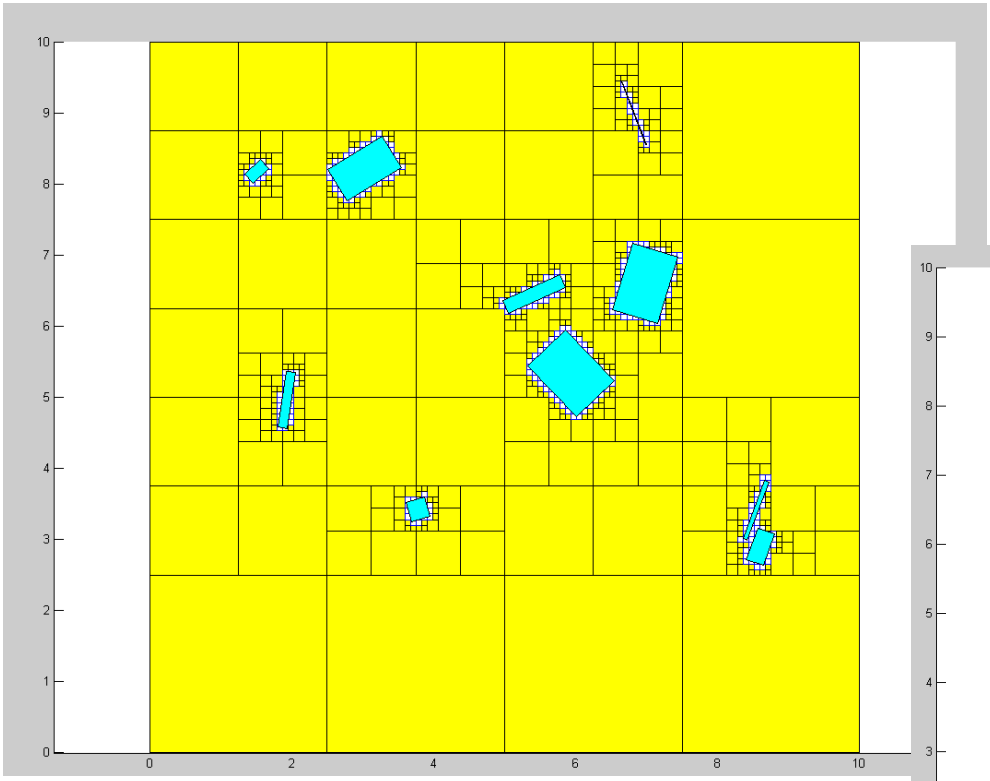
Za naslednike trenutnega vozlišča določimo ceno celotne poti. Za hevristiko je uporabljena razdalja Manhattan.

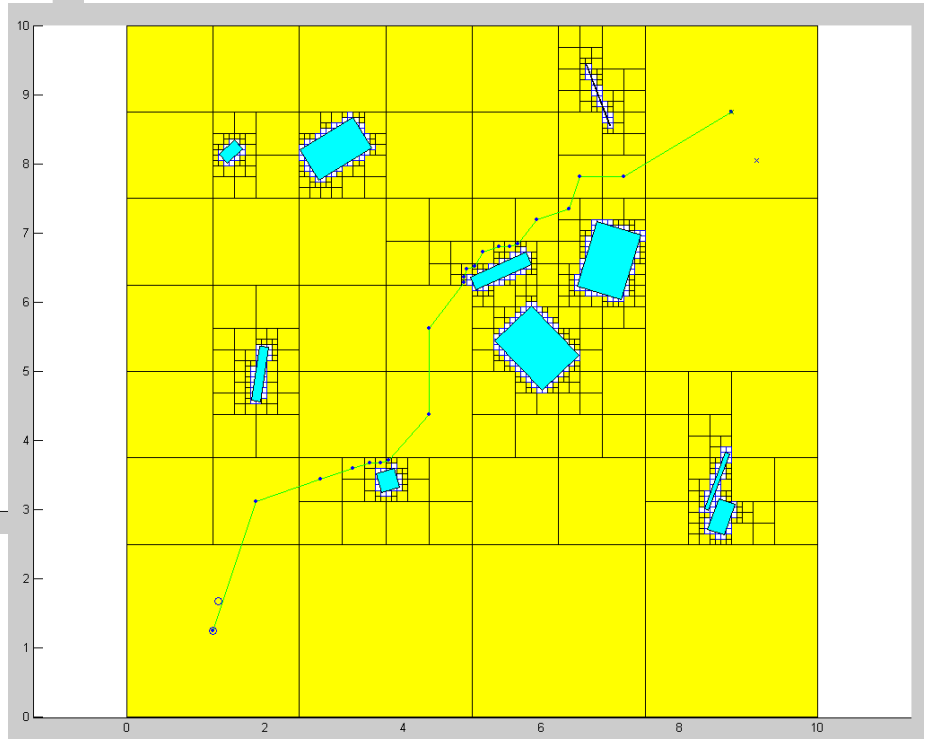
Novo trenutno vozlišče ima najnižjo ceno celotne poti



Planiranje poti z razcepom na celice in A*

- Določimo predstavitev okolja s štiriškim drevesom, poiščemo optimalno pot z A* (skozi središča celic).





Planiranje poti z razcepom na celice in A*

- Določimo predstavitev okolja z omejeno Delaunayevo triangulacijo, poiščemo optimalno pot z A* (zelena črta, ki gre skozi središča stranic trikotnikov).