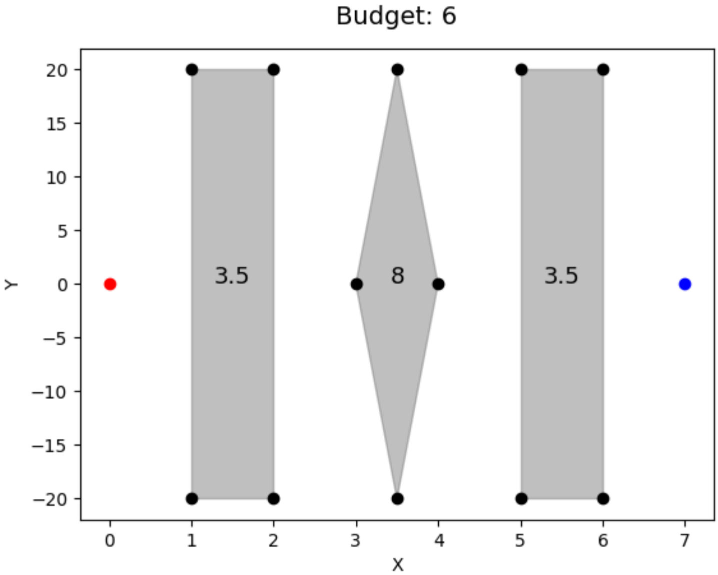
Najkrajša pot z odstranljivimi ovirami

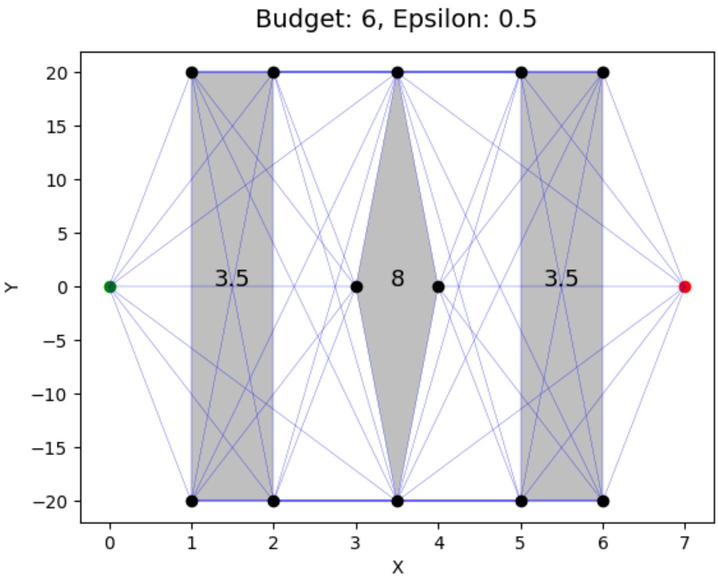
Gašper Terglav

20. junij 2024

Primer problema



Viability graf

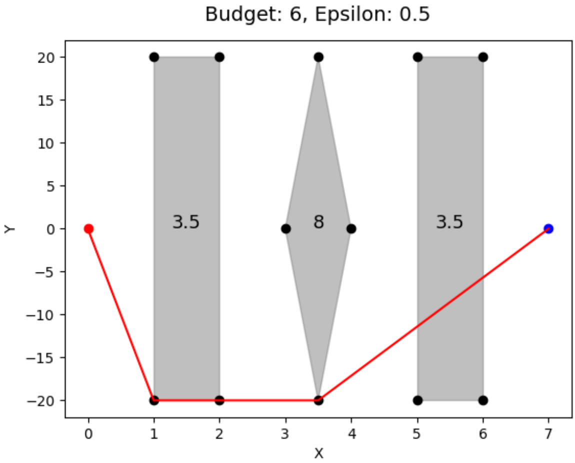


Rešitev

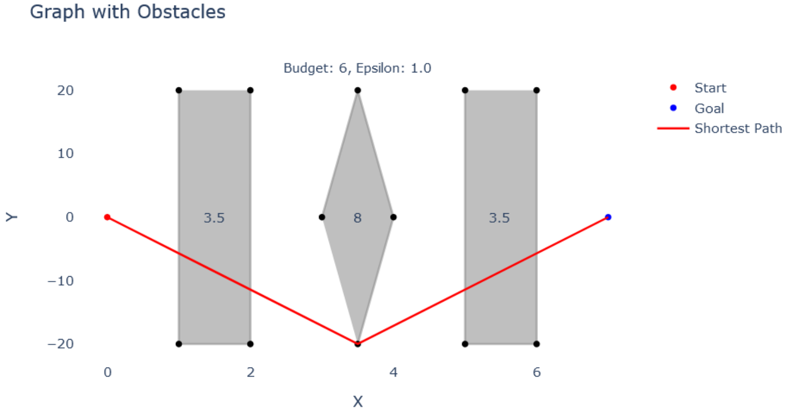
Izberemo parameter natančnosti ϵ. Če ima viabilty graf vozlišča $v \in V$, potem ima nov graf vozlišča oblike $v_i \in V$, kjer $i = 0, \epsilon, 2\epsilon, \ldots, \lceil budget/\epsilon \rceil$.

Rešitev

Izberemo parameter natančnosti ϵ. Če ima viabilty graf vozlišča $v \in V$, potem ima nov graf vozlišča oblike $v_i \in V$, kjer $i = 0, \epsilon, 2\epsilon, \ldots, \lceil budget/\epsilon \rceil$.



Napaka algoritma je $1 + 2\epsilon$ v ceni.



Naivni algoritem

Za vsak par točk v, u in vsak rob ovire e, pogledam,če $\overline{vu}$ seka e. Če ja, dodam ceno ovire e k ceni $\overline{vu}$. V graf dodam vse daljice s ceno manj od budgeta.

Naivni algoritem

Za vsak par točk v, u in vsak rob ovire e, pogledam,če $\overline{vu}$ seka e. Če ja, dodam ceno ovire e k ceni $\overline{vu}$. V graf dodam vse daljice s ceno manj od budgeta. Časovna zahtevnost $O(n^3)$. Zahtevnost celotnega algoritma je potem $O(n^3/\epsilon)$

Naivni algoritem

Za vsak par točk v, u in vsak rob ovire e, pogledam,če $\overline{vu}$ seka e. Če ja, dodam ceno ovire e k ceni $\overline{vu}$. V graf dodam vse daljice s ceno manj od budgeta. Časovna zahtevnost $O(n^3)$. Zahtevnost celotnega algoritma je potem $O(n^3/\epsilon)$

	n (število oglišč)			
	40	180	360	860
Čas	0.5s	36s	301s	1ura 16min

Tabela: Vpliv n na čas iskanja poti

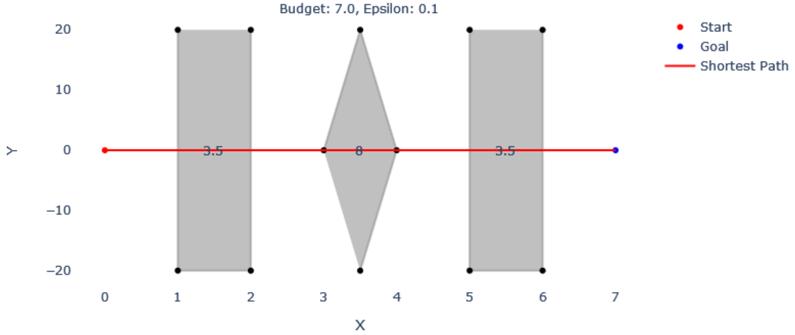
Naivni algoritem

	ϵ			
	0.01	10^{-3}	10^{-4}	10^{-5}
Čas	0.9s	5.8s	57s	memory full

Tabela: Vpliv vrednosti ϵ na čas iskanja poti ($n = 40$)

Naivni algoritem

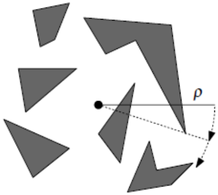
Problem:



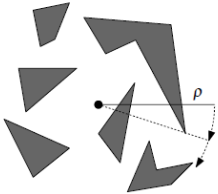
Sweep algoritem

Za vsako točko v, ostale točke uredim glede na kot z vodoravno premico in jih nato pregledam po vrsti. Robove ovir shranjujem v AVL drevo.

Za vsako točko v, ostale točke uredim glede na kot z vodoravno premico in jih nato pregledam po vrsti. Robove ovir shranjujem v AVL drevo.



Za vsako točko v, ostale točke uredim glede na kot z vodoravno premico in jih nato pregledam po vrsti. Robove ovir shranjujem v AVL drevo.



Časovna zahtevnost $O(n^3)$. Zahtevnost celotnega algoritma je potem $O(n^3/\epsilon)$

Sweep algoritem

	n (število oglišč)			
	40	180	360	860
Čas	0.7s	50s	393s	1ura 28min

Tabela: Vpliv n na čas iskanja poti

	ϵ			
	0.01	10^{-3}	10^{-4}	10^{-5}
Čas	1.5s	9.2s	99s	memory full

Tabela: Vpliv vrednosti ϵ na čas iskanja poti ($n = 40$)

Sparse algoritem

- Določimo navpično premico p, ki razdeli točke na dva množici približno enake moči. Za vsako točko v je v' njena projekcija na p.

Sparse algoritem

- Določimo navpično premico p, ki razdeli točke na dva množici približno enake moči. Za vsako točko v je v' njena projekcija na p.
- Poiščemo prvi rob ovire z navpičnim naklonom, ki seka $\overline{vv'}$. Označimo presečišče z x. Če obstaja presečišče roba s p (označimo ga z y). Dodamo v graf $\overline{xy}$.

Sparse algoritem

- Določimo navpično premico p, ki razdeli točke na dva množici približno enake moči. Za vsako točko v je v' njena projekcija na p.
- Poiščemo prvi rob ovire z navpičnim naklonom, ki seka $\overline{vv'}$. Označimo presečišče z x. Če obstaja presečišče roba s p (označimo ga z y). Dodamo v graf $\overline{xy}$.
- Ponovimo za prvi negativen rob.

Sparse algoritem

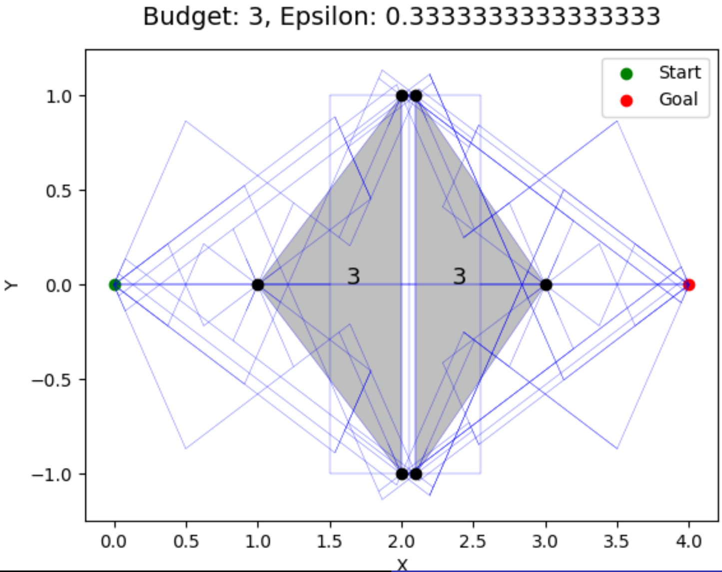
- Določimo navpično premico p, ki razdeli točke na dva množici približno enake moči. Za vsako točko v je v' njena projekcija na p.
- Poiščemo prvi rob ovire z navpičnim naklonom, ki seka $\overline{vv'}$. Označimo presečišče z x. Če obstaja presečišče roba s p (označimo ga z y). Dodamo v graf $\overline{xy}$.
- Ponovimo za prvi negativen rob.
- Rekurzivno ponovimo na levi in desni strani p.

Sparse algoritem

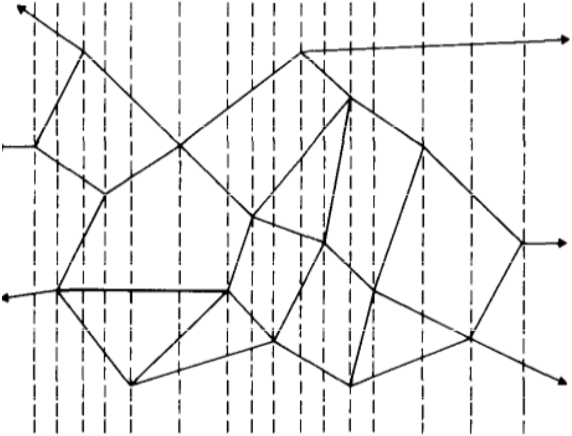
- Določimo navpično premico p, ki razdeli točke na dva množici približno enake moči. Za vsako točko v je v' njena projekcija na p.
- Poiščemo prvi rob ovire z navpičnim naklonom, ki seka $\overline{vv'}$. Označimo presečišče z x. Če obstaja presečišče roba s p (označimo ga z y). Dodamo v graf $\overline{xy}$.
- Ponovimo za prvi negativen rob.
- Rekurzivno ponovimo na levi in desni strani p.
- Ponovimo vse do sedaj $\lceil 1/\epsilon \rceil$-krat, le da vsakič zarotiramo ravnino za kot $2\pi\epsilon$.

Sparse algoritem

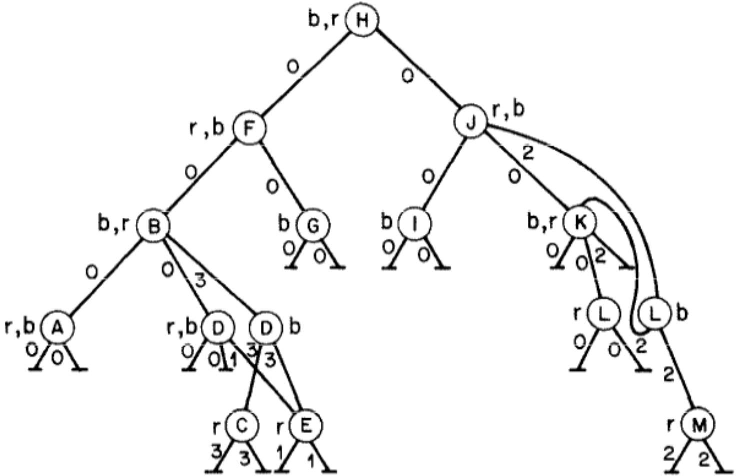
Primer:



Za algoritem je treba določiti samo cene navpičnih in vodoravnih poti. Rešitev: persistent search tree.



Persistent search tree



Persistent search tree

```
tree.insert(10, "A", 1)
tree.insert(20, "B", 1)
tree.insert(15, "C", 1)

tree.delete(10, 2)
tree.insert(30, "E", 2)
tree.insert(40, "F", 2)

tree.delete(20, 3)
tree.insert(35, "G", 3)
tree.insert(45, "H", 3)

tree.delete(15, 4)
tree.delete(30, 4)
tree.insert(7, "I", 4)
```

```
Tree at time 1:
R--- 15(black)
    L--- 10(red)
    R--- 20(red)
Tree at time 2:
R--- 20(black)
    L--- 15(black)
    R--- 30(black)
        R--- 40(red)
Tree at time 3:
R--- 30(black)
    L--- 15(black)
    R--- 40(black)
        L--- 35(red)
        R--- 45(red)
Tree at time 4:
R--- 40(black)
    L--- 35(black)
    |   L--- 7(red)
    R--- 45(black)
```

Sparse algoritem

Časpvna zahtevnost bi morala biti $O(\frac{nh}{\epsilon^2} \log n \log \frac{n}{\epsilon})$.

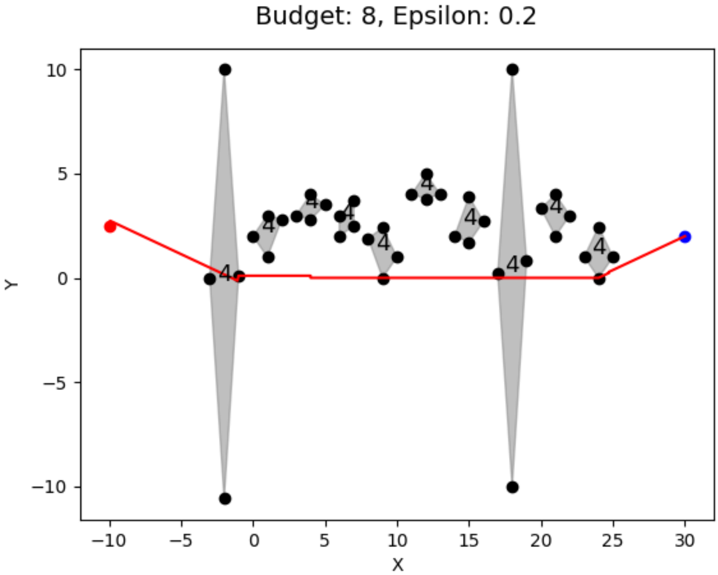
	ϵ				
	0.5	0.25	0.1	0.01	0.001
Čas	0.01s	0.03s	0.5s	31s	4651s

Tabela: Vpliv vrednosti ϵ na čas iskanja poti

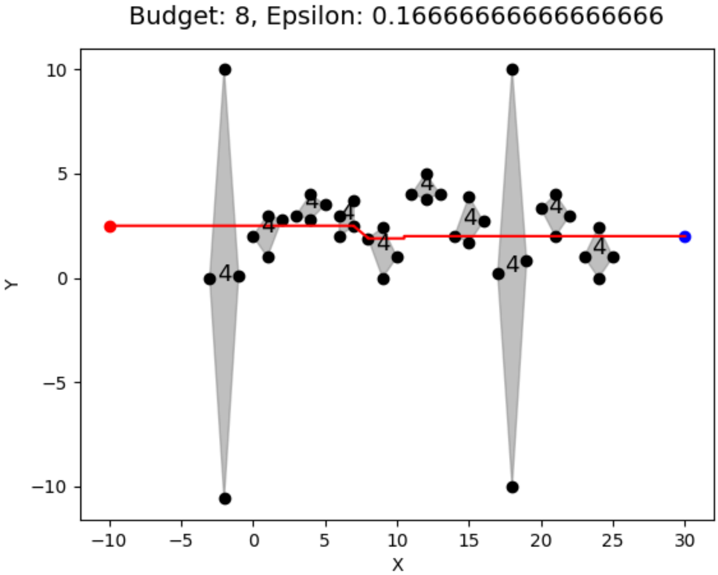
	n		
	15	40	180
Čas	0.37s	24s	905s

Tabela: Vpliv n na čas iskanja poti ($\epsilon = 0.2$)

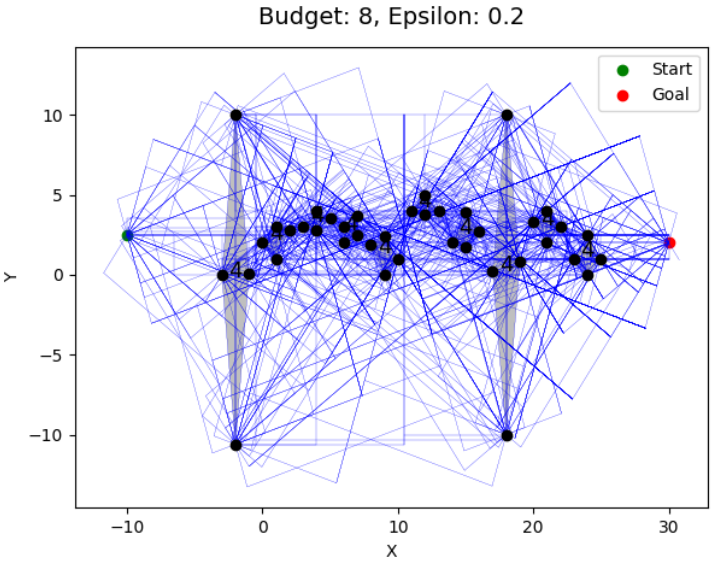
Sparse algoritem



Sparse algoritem



Sparse algoritem



Primerjava algoritmov glede na n

	40	180	360	860
Naive	0.5s	36s	301s	1ura 16min
Sweep	0.7s	50s	393s	1ura 28min
Sparse	25s	905s	-	-

Tabela: Vpliv n na čas iskanja poti

Primerjava algoritmov glede na ϵ

	0.01	10^{-3}	10^{-4}	10^{-5}
Naive	0.9s	5.8s	57s	memory full
Sweep	1.5s	9.2s	99s	memory full
Sparse	31s	4651s	-	-

Tabela: Vpliv vrednosti ϵ na čas iskanja poti