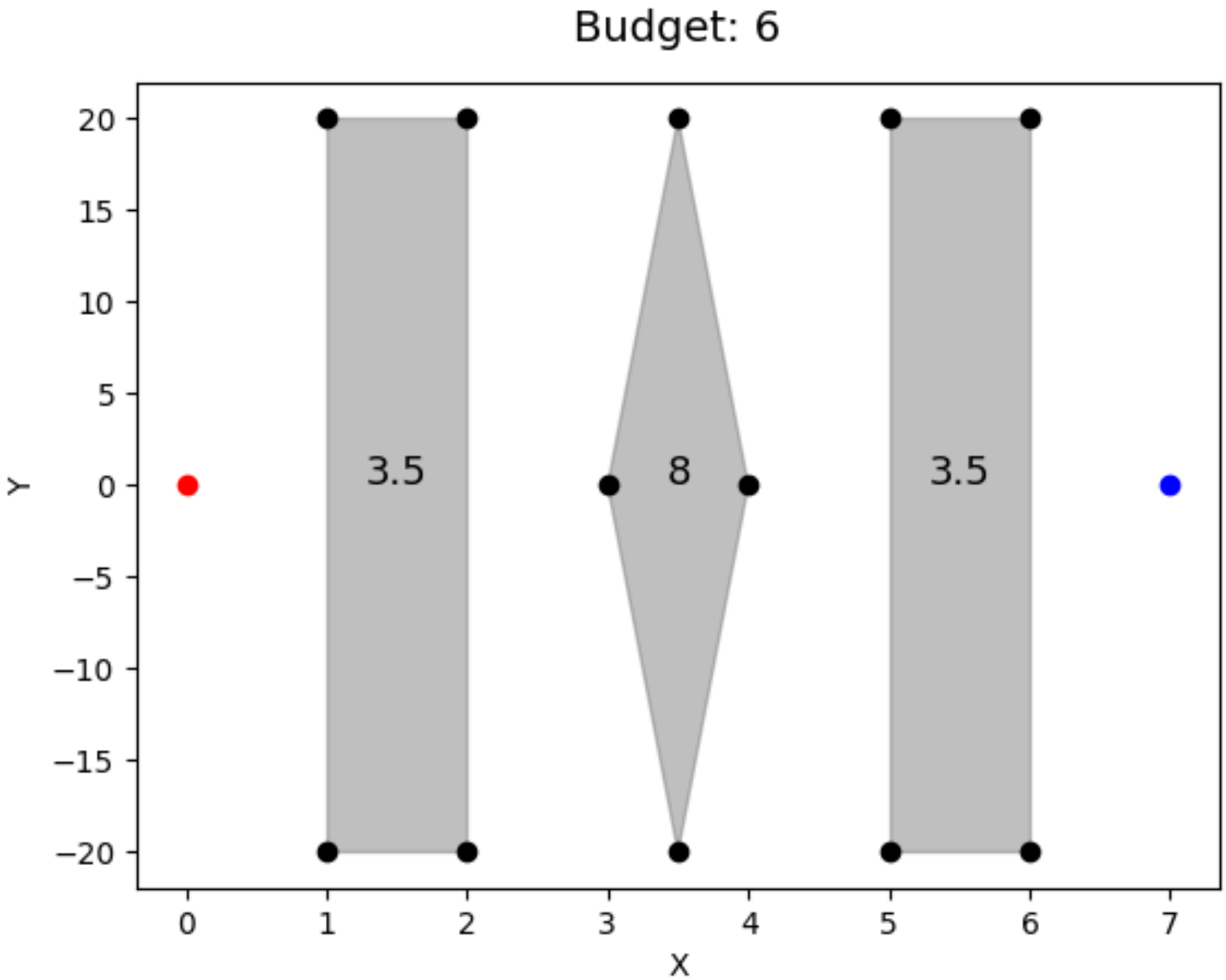
Najkrajša pot z odstranljivimi ovirami

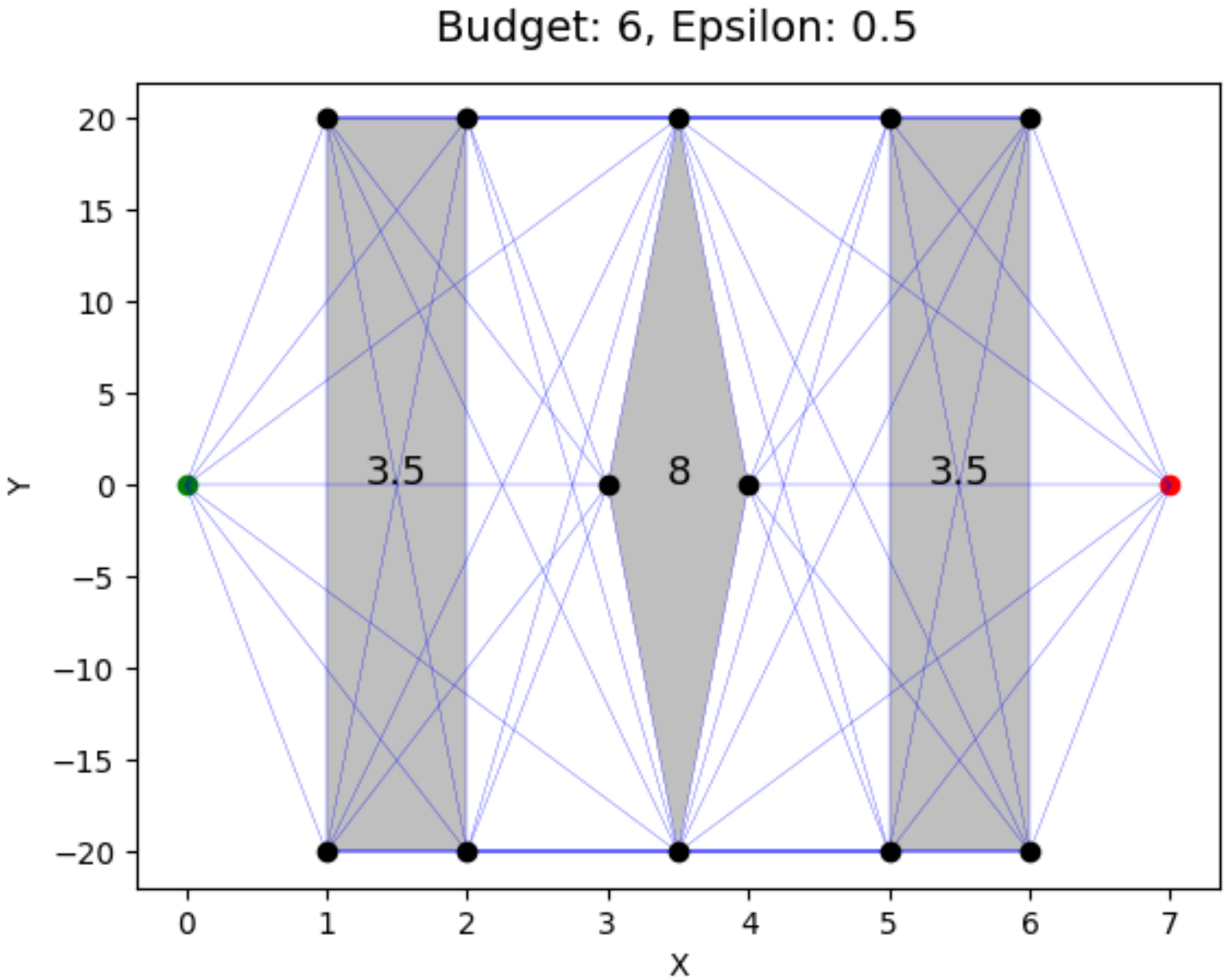
Gašper Terglav

20. junij 2024

Primer problema



Viability graf

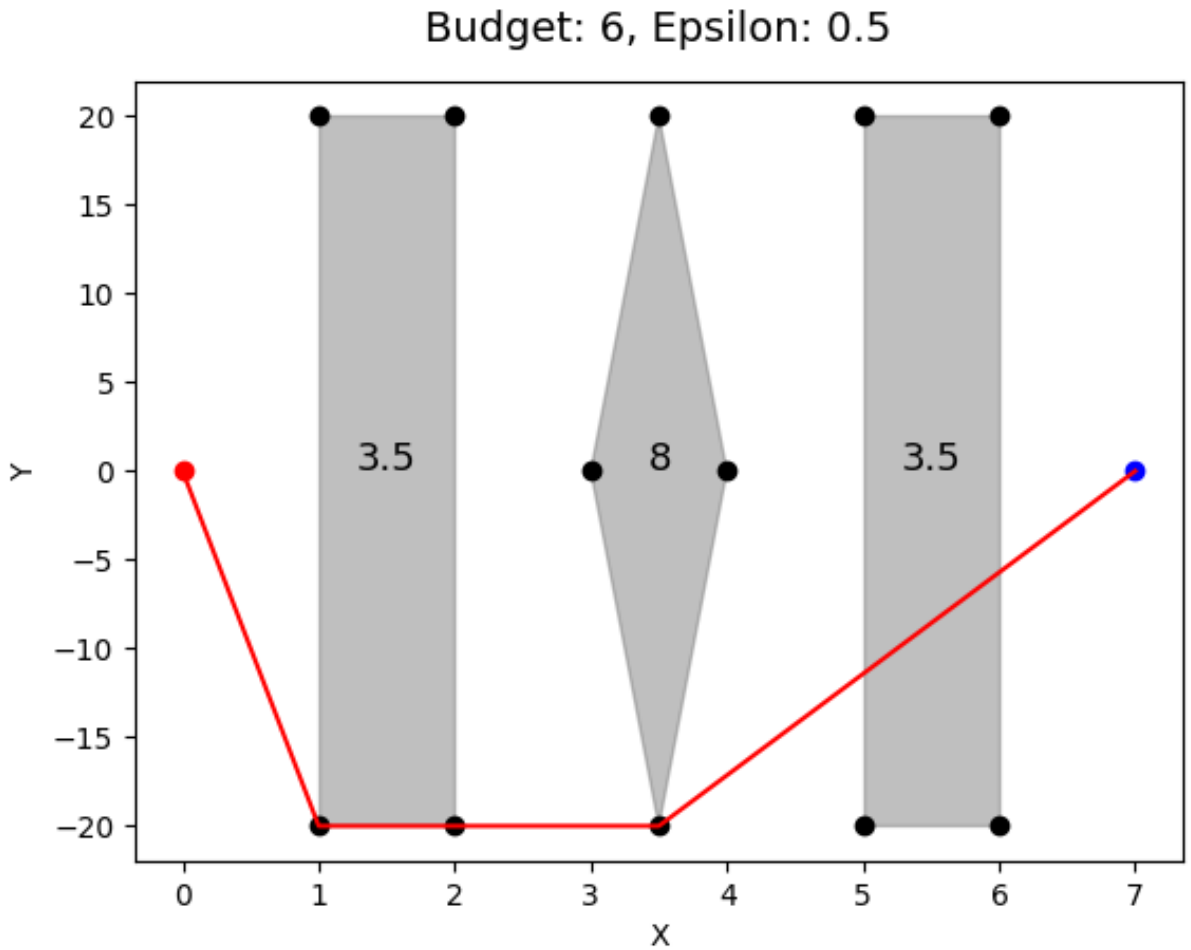


Rešitev

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

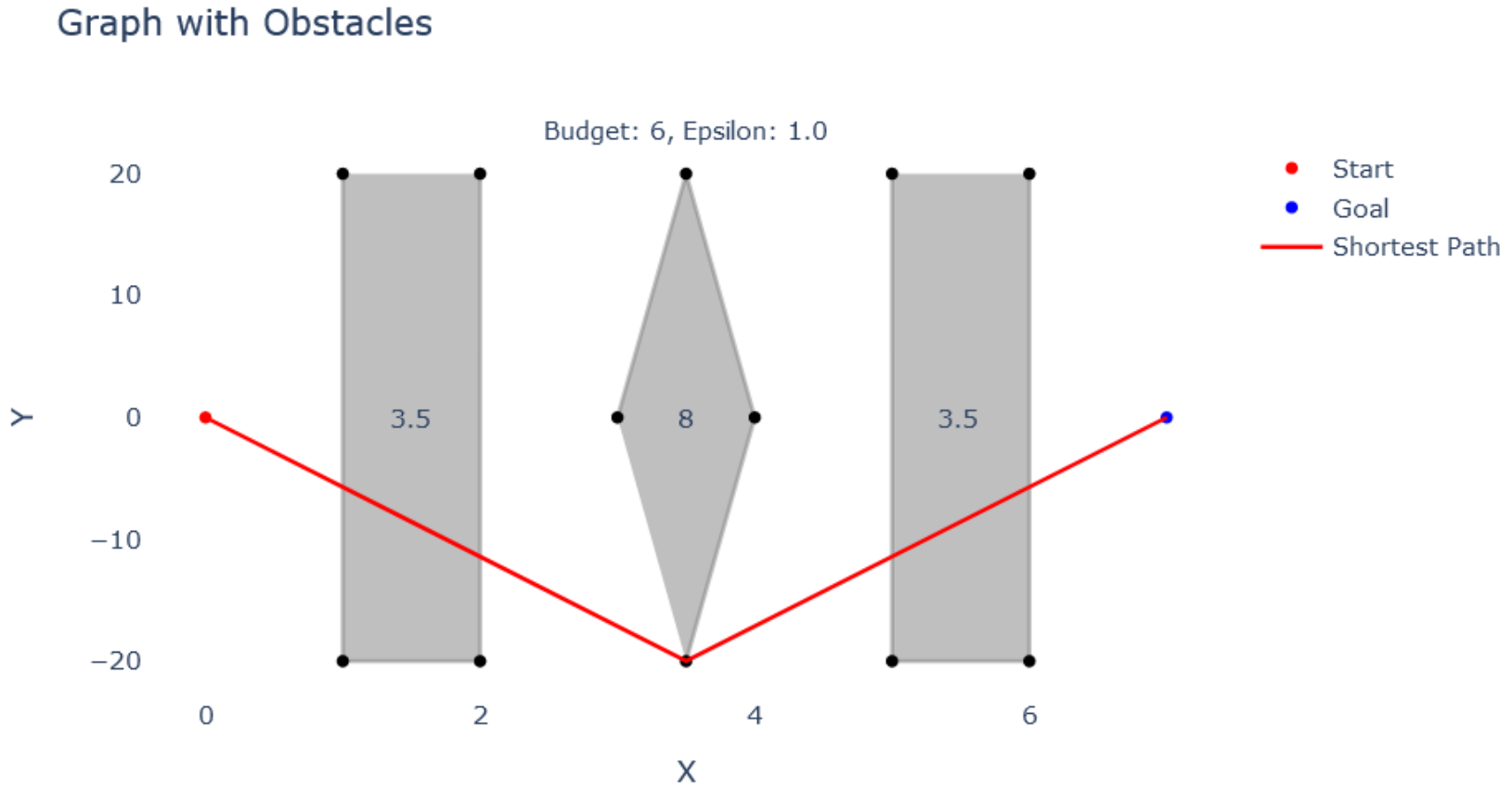
Rešitev

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



Rešitev

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

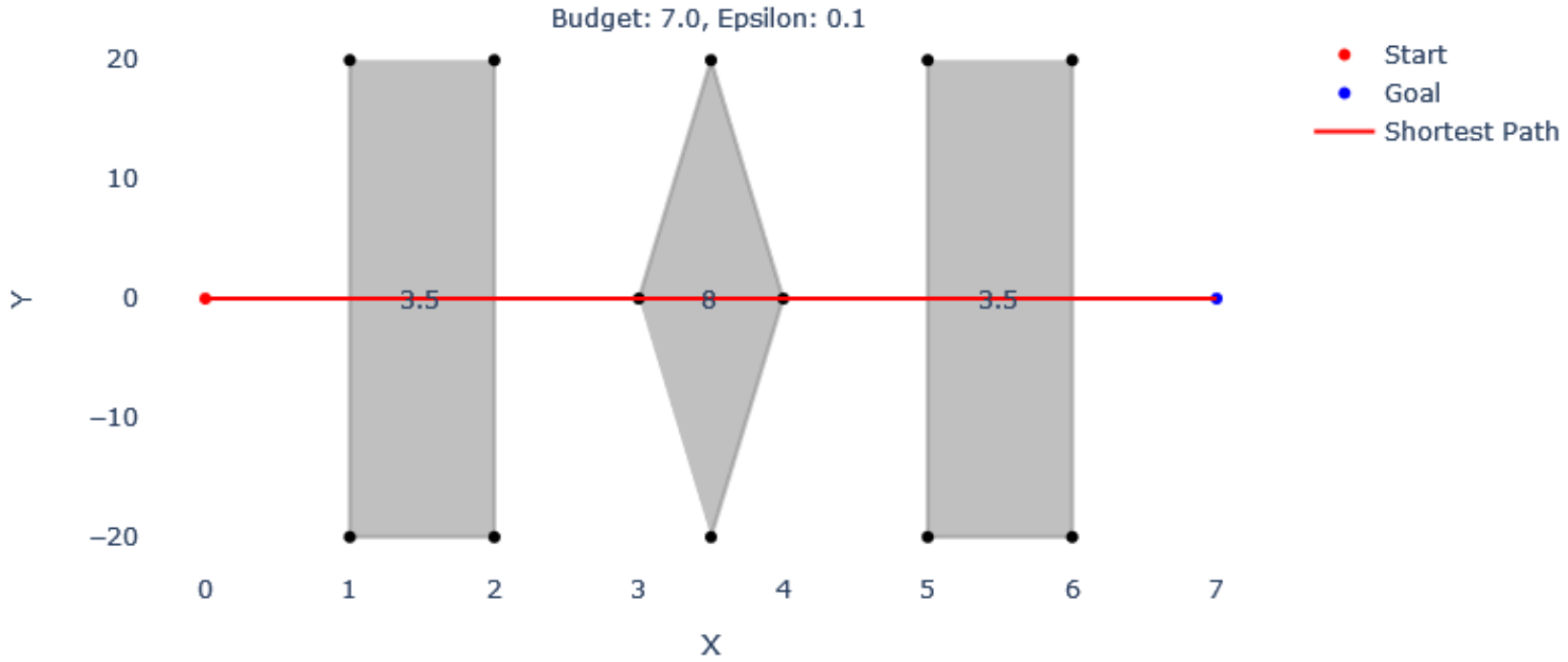
Naivni algoritem

	ϵ			
	0.01	10^{-3}	10^{-4}	10^{-5}
Čas	0.5s	5s	57s	memory full

Tabela: Vpliv vrednosti ϵ na čas iskanja poti

Naivni algoritem

Problem:

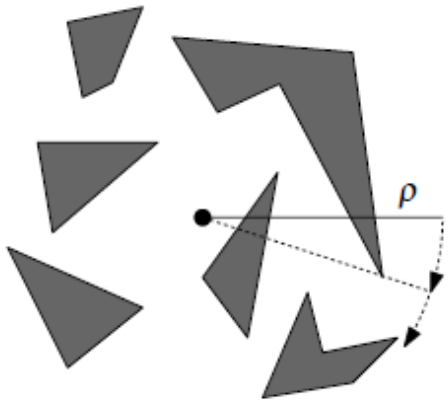


Sweep algoritem

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.

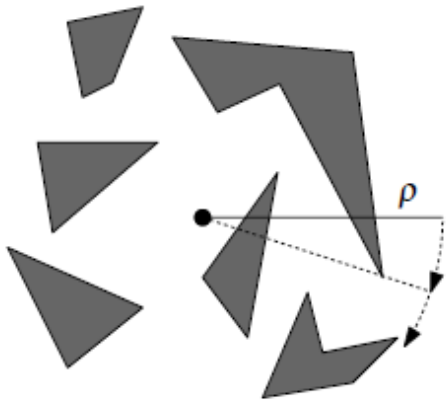
Sweep algoritem

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.



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.



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	1s	11s	126s	memory full

Tabela: Vpliv vrednosti ϵ na čas iskanja poti

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

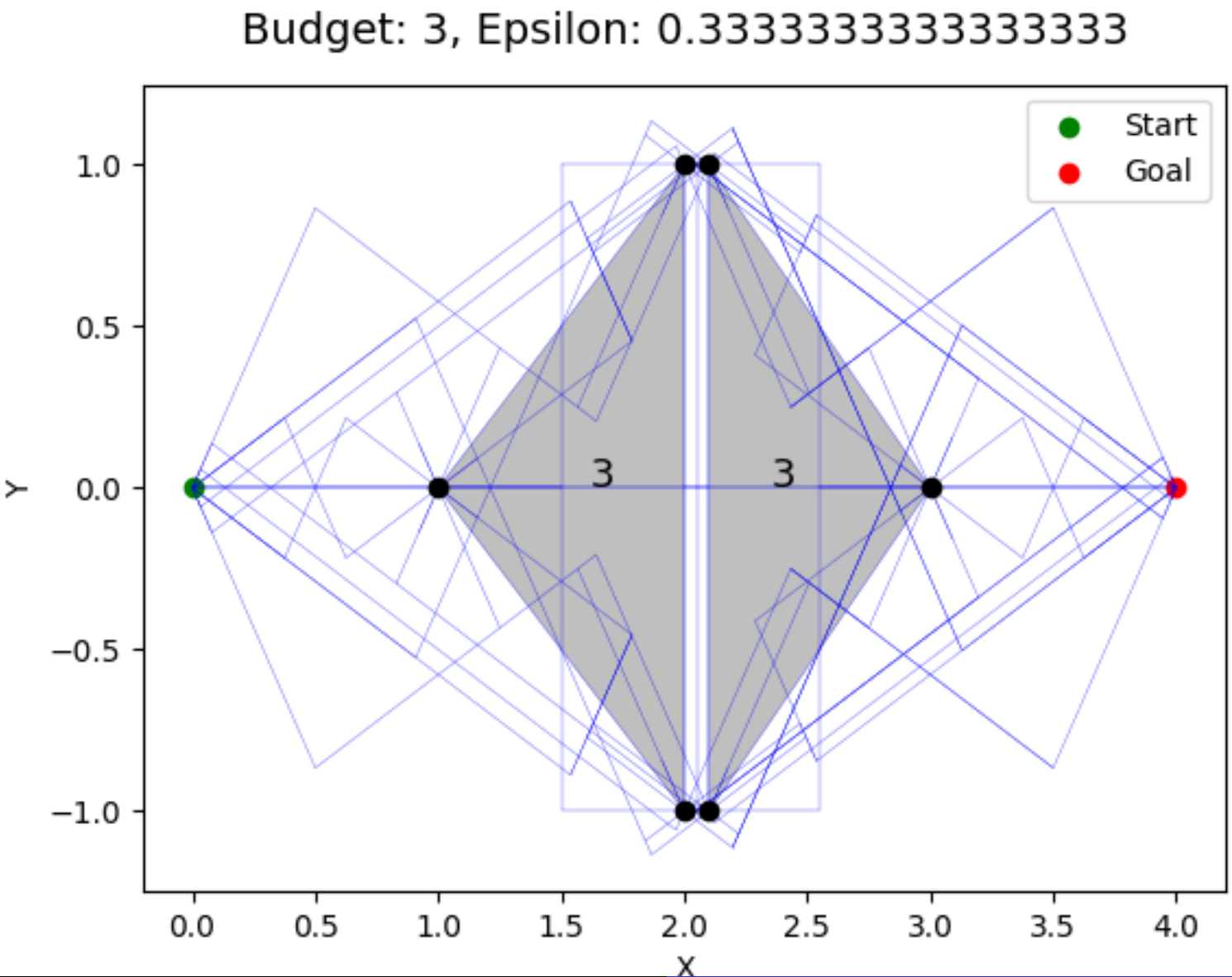
- Določimo navpično premico p, ki razdeli točke na dva množici približno enake moči. Za vsako točko v je v' njena projekcija na p.
- Poiščemo prvi rob ovire z navpičnim naklonom, ki seka $\overline{vv'}$. Označimo presečišče z x. Če obstaja presečišče roba s p (označimo ga z y). Dodamo v graf $\overline{xy}$.
- 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 $2\pi\epsilon$.

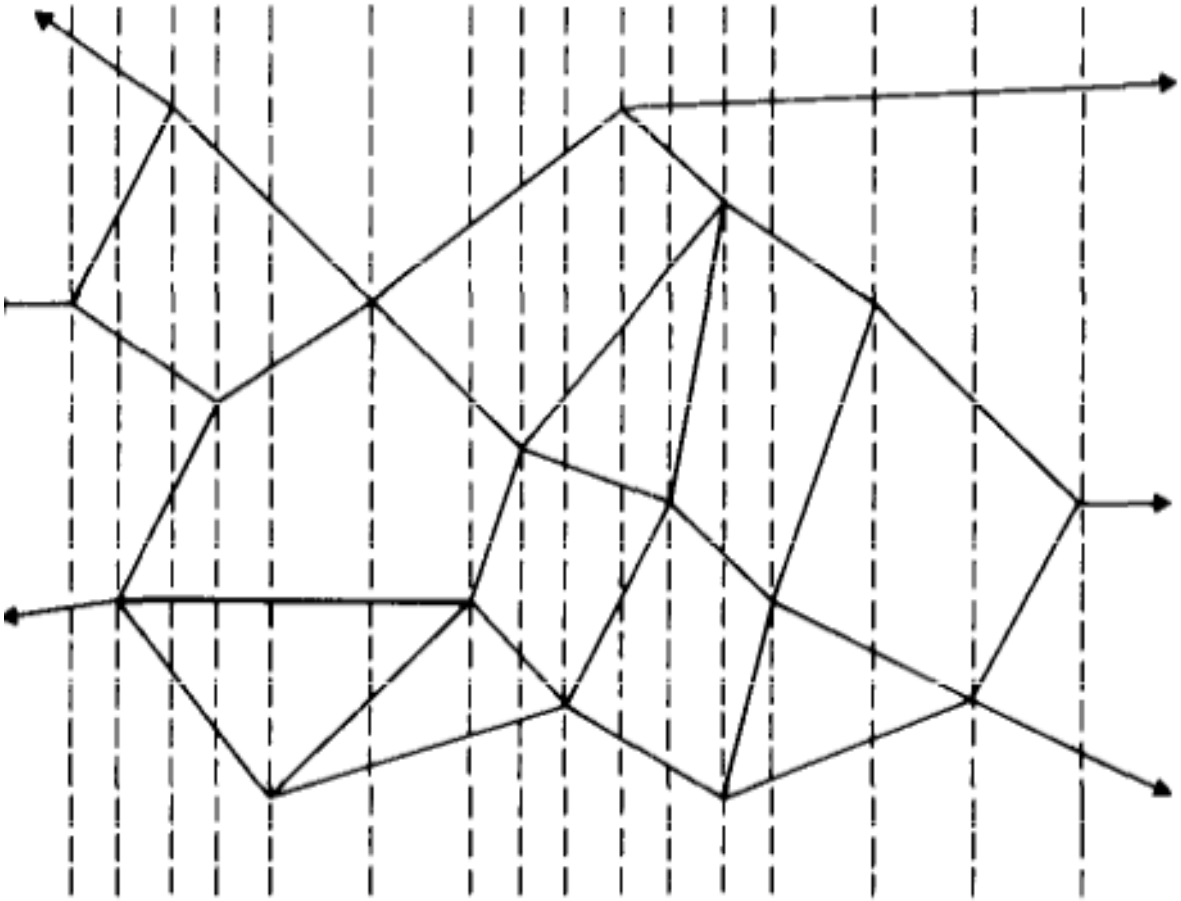
Sparse algoritem

Primer:

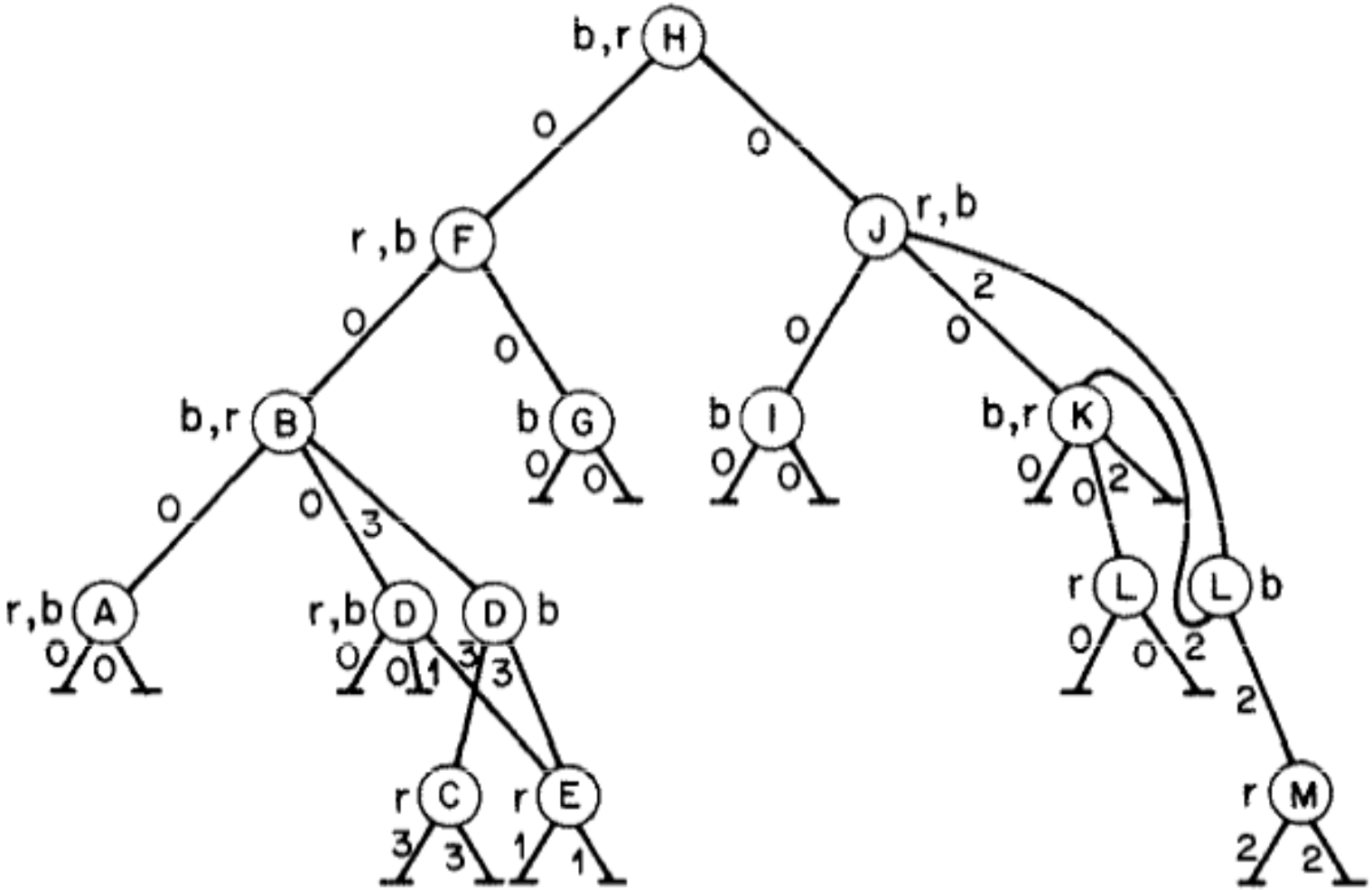


Persistent search tree

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



Persistent search tree



Persistent search tree

```
769    tree.insert(10, "A", 1)
770    tree.insert(20, "B", 1)
771    tree.insert(15, "C", 1)
772
773    tree.delete(10, 2)
774    tree.insert(30, "E", 2)
775    tree.insert(40, "F", 2)
776
777    tree.delete(20, 3)
778    tree.insert(35, "G", 3)
779    tree.insert(45, "H", 3)
780
781    tree.delete(15, 4)
782    tree.delete(30, 4)
783    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)
```

Sparse algoritem

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

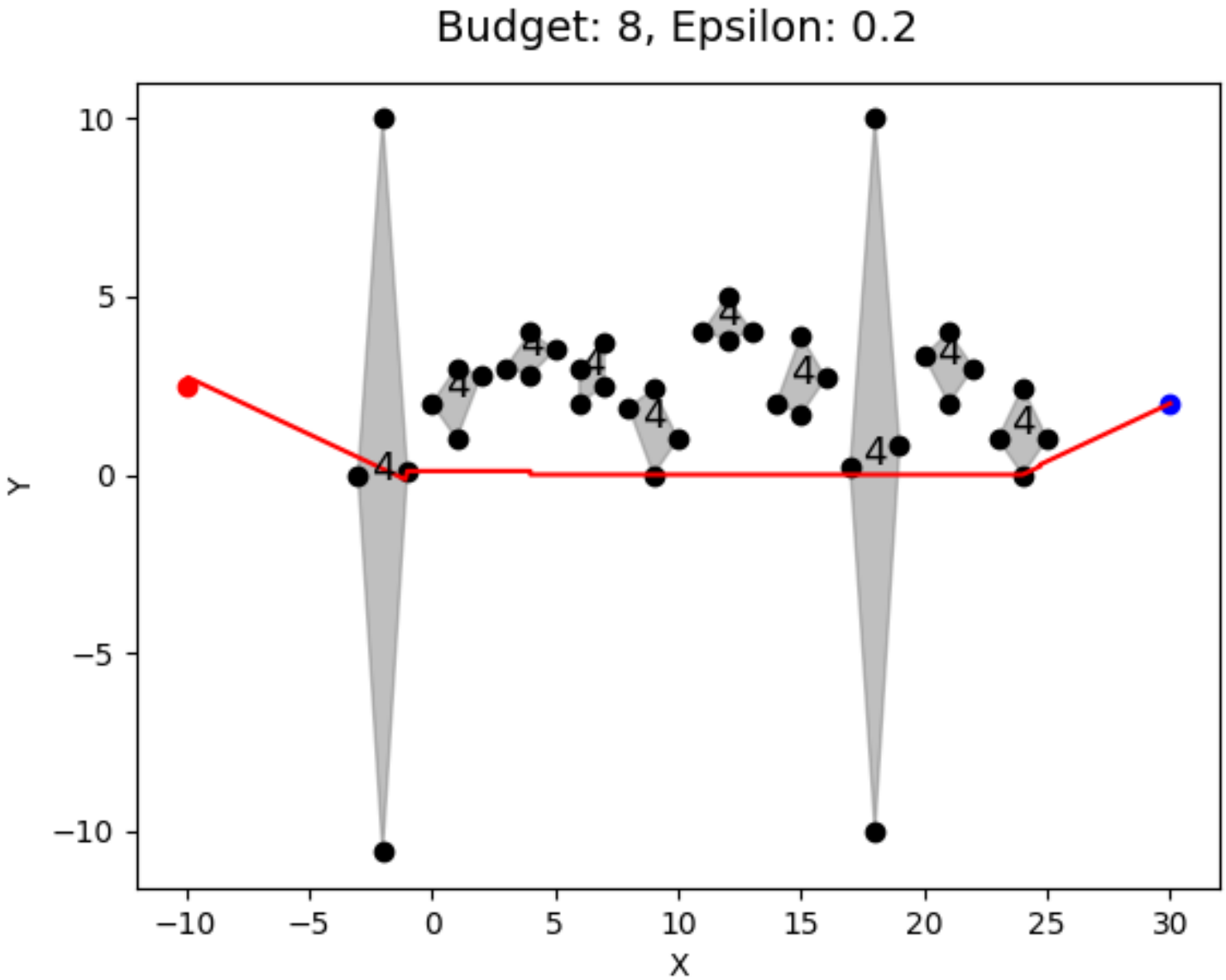
	ϵ				
	0.5	0.25	0.1	0.01	0.001
Čas	0.01s	0.03s	0.5s	31s	4651s

Tabela: Vpliv vrednosti ϵ na čas iskanja poti

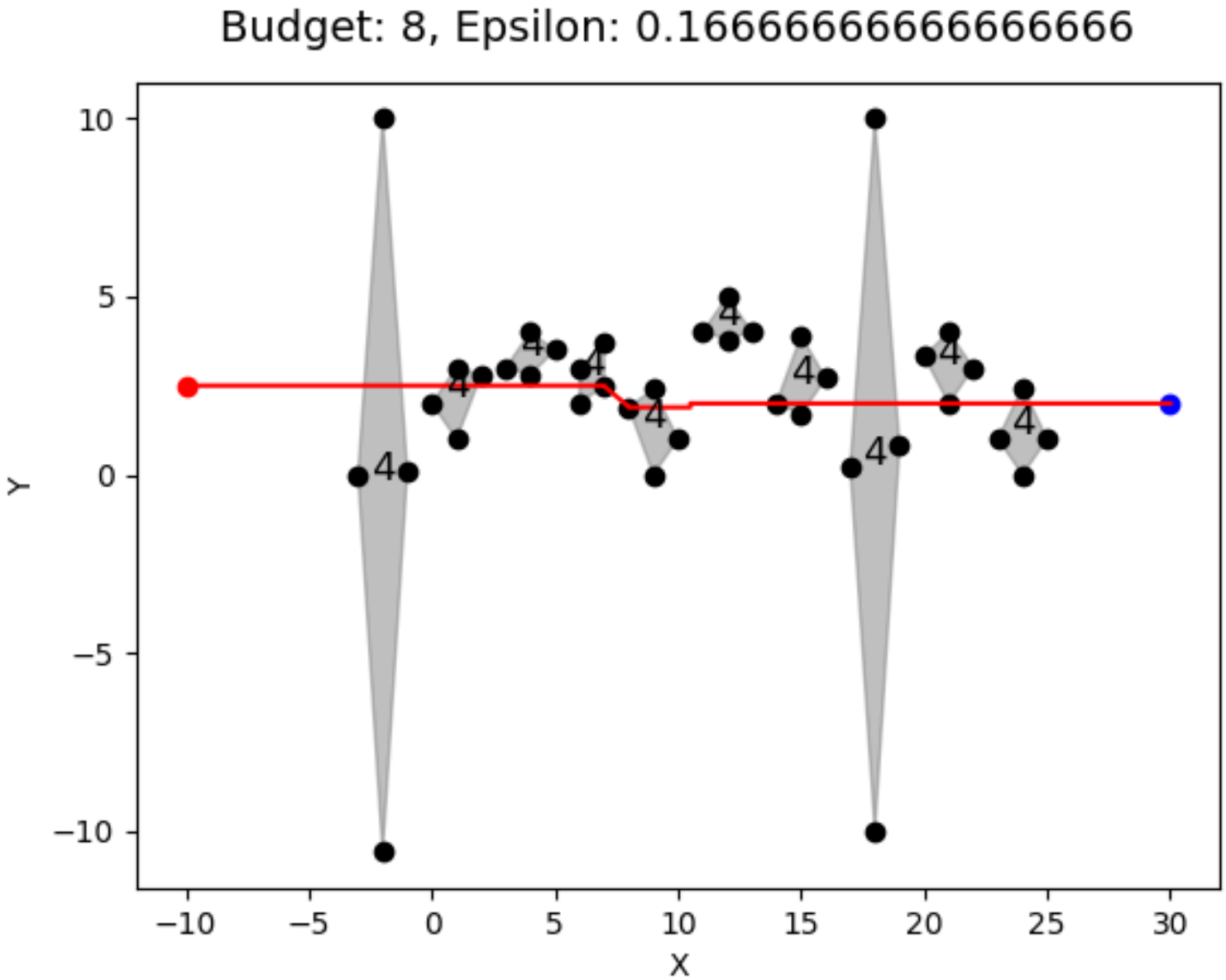
	n		
	15	40	180
Čas	0.37s	24s	905s

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

Sparse algoritem



Sparse algoritem



Sparse algoritem