Najkrajša pot z odstranljivimi ovirami – poročilo

Gašper Terglav

22. maj 2024

Predstavitev problema

Problem je posplošitev klasične verzije, v kateri se lahko oviram samo izogibamo. V $\mathbb{R}^2$ imamo dve točki s in t iščemo najkrajšo evklidsko pot med njima. Ovire v ravnini so konveksni večkotniki, vsak od njih ima ceno $c_i > 0$. Če imamo na voljo C "denarja", katere ovire se splača odstraniti, da dosežemo najkrajšo pot? V resničem življenju lahko npr. načrtovalci mest spremenijo cestno omrežje, da dosežejo boljšo pretočnost in tako odstranijo ovire za neko ceno. Še en primer omenjen v članku so skladišča v katerih delajo roboti. Kako spremeniti postavitev ovir v skladišču, da se roboti hitreje premikajo okoli?

Naš problem je torej: ali obstaja pot dolžine L in cene C med začetkom in koncem? Ta problem je na žalost NP-težek, zato se bomo za polinomski čas morali zadovoljiti z algoritmom, ki ima neko napako vsaj v ceni. Še pred algoritmi iz članka pa je treba narediti t. i. "viability graph" za naš problem.

Viability graph

Začnimo s pomembnima opazkama o obnašanju najkrajše poti. Če bo šla naša pot skozi oviro, bo zaradi konveksnosti sekala samo dve stranici večkotnika. Smer bo pot spremenila samo v ogliščih večkotnikov. Tako bomo lahko problem prevedli na iskanje najkrajše poti v grafu, katerega vozlišča so vsa oglišča ovir ter točki s in t. Označimo ta graf z $G = (V, E)$, kjer $(u, v) \in E$, če je vsota cen vseh ovir, ki jih prečka daljica uv manjša ali enaka od našega proračuna C. Vsaka povezava v grafu ima dva parametra, dolžino in ceno.

Najprej sem se konstrukcije lotil naivno. Za vsak par točk sem za vsako oviro pogledal, če jo pot seka. Ta pristop ima časovno zahtevnost $O(n^3)$. Je pa tudi veliko robnih primerov, ki otežijo implementacijo. Trenutna implementacija ima problem, ko je na poti med ogliščema še tretje oglišče. Če sta drugo in tretje oglišče del iste ovire in gre pot po njeni notranjosti, tega nisem znal na lep način zaznati. Zato bi imela ta povezava premajhno ceno. Implementacija je v datoteki "viabilityGraph.py"

Viabilty graph je v resnici posplošitev t. i. "visibility graph", s katerim se rešuje osnovni problem kjer je $C = 0$. Zato sem izgradnjo visibility grapha iz knjige [2] v poglavju 15 posplošil na viability graph. Izberimo oglišče v. Ugotoviti moramo, do katerih oglišč lahko iz v pridemo s ceno največ C. Pri

naivnem pristopu smo oglišča pregledali kakor so padla. Sedaj pa jih hočemo v nekem pametnem vrstnem redu, da bomo lahko nekaj informacij uporabili za naprej. Predstavljajmo si poltrak p, ki ima izhodišče v v in naredi en krog v smeri urinega kazalca. Oglišča bomo pregledali v vrstnem redu, ki ga določa ta sprehod poltraka (ang. plane sweep). Če p oglišča seka istočasno, imajo prednost tista, ki so bližje v. Robove ovir, ki jih p seka, bomo shranili v dvojiško drevo, kar omogoča hitrejše dodajanje in brisanje.

Algorithm 1 Vrnem seznam oglišč, ki jih lahko dosežemo iz v

1: Uredimo preostala oglišča glede na kot, ki ga daljica med ogliščem in v naredi s pozitivno x-osjo.
2: Dodamo v drevo vse robove, ki jih p seka. Prvi rob je tisti, ki ga poltrak najprej seka.
3: $W = [\ ]$
4: **for** w_i v urejenih ogliščih **do**
5: **if** (cena poti med v in w_i) $\leq C$ **then**
6: Dodamo w_i v W.
7: **end if**
8: Zavrtimo poltrak p, da gre sedaj še skozi w_i.
9: Odstranimo iz drevesa robove ovir, ki se končajo v w_i ter ležijo nad p.
10: Dodamo v drevo robove, ki se končajo v w_i in ležijo pod njim.
11: **end for**
12: **return** W

Ta algoritem poženemo za vsako oglišče, ki ga imamo in tako lahko sestavimo naš viability graph. Potrebujemo pa še funkcijo, ki nam v 5. koraku izračuna ceno poti. Kot argument ji moramo podati točke v, w_i, w_{i-1} ter ceno poti vw_{i-1} in še naše dvojiške drevo ter vse ovire. Ključno je, da če w_i in w_{i-1} ležita na isti premici, lahko uporabimo to kar vemo o w_{i-1}, zato se nam problem iz naivnega algoritma ne ponovi.

Algoritem v polinomskem času z napako $(1+\epsilon)$ v ceni

Ta graf bi radi modificirali tako, da bi na novem grafu samo pognali Dijkstra algoritem in dobili rešitev, zato hočemo odstraniti parameter cene iz povezav. Naj bo $K = min(\frac{C}{min_i\ c_i}, h)$, kjer je h število ovir, c_i pa njihove cene ter $\sum c_i = C$. Delimo vse cene z K/C, da je naš proračun za ovire na tako enak K. Potem naredimo $\lceil \frac{2K}{\epsilon} \rceil + 1$ kopij vsakega vozlišča v grafu G in jih označimo z $v_0, v_{\epsilon/2}, v_{\epsilon}..., v_K$. Za vsak rob $(u, v) \in E$ s ceno c in za vsak $0 \leq i \leq \lceil \frac{2K}{\epsilon} \rceil$, dodamo rob $(u_{i\epsilon/2}, v_{j\epsilon/2})$, kjer je $j \leq \lceil \frac{2K}{\epsilon} \rceil$ največje celo število da velja $j\frac{\epsilon}{2} \leq i\frac{\epsilon}{2} + c$. Dolžina povezav je enaka kot v originalnem grafu. Tako smo ceno zakodirali v vozlišča. Če se nahajamo v vozlišču v_i to pomeni, da smo za odstranitev ovir do sedaj plačali i približno denarja. Na koncu dodamo še novi vozlišči s in t, ter vsakega od njiju povežemo z vsemi njunimi kopijami. Te povezave imajo dolžino 0. Ker imajo povezave samo še dolžino, lahko sedaj

za iskanje najkrajše poti uporabimo Dijkstro. Novi graf ima $O(|V|K/\epsilon)$ vozlišč in $O(|E|K/\epsilon)$ robov. Ker je Dijkstra najbolj časovno zahtevna stvar, je potem časovna zahtevnost celotnega algoritma enaka $O(\frac{K}{\epsilon}(|E| + |V| \log \frac{|V|}{\epsilon}))$. Če je n število vseh oglišč ovir, je to v najboljšem primeru $\Omega(\frac{n^3}{\epsilon})$.

Hitrejši algoritem

Ta algoritem je bolj zapleten in doseže večjo hitrost z zmanjšanjem povezav v grafu G. V zameno pa je tukaj lahko napaka $(1 + \epsilon)$ tudi pri dolžini poti. Ozaničmo nov graf z manj povezavami s $H = (X, \Gamma)$. Zanimivo je da ima H več vozlišč kot G, bolj natančno $|X|, |\Gamma| = O(n \log n)$. Ko enkrat skonstruiramo H pa lahko spet uporabimo idejo iz prejšnjega algoritma in naredimo $O(1/\epsilon)$ kopij ter uporabimo Dijkstro. Ta algoritem ima časovno zahtevnost $O(\frac{nh}{\epsilon^2} \log n \log \frac{n}{\epsilon})$ kar je približno za potenco n manj kot prejšnji algoritem.

Okviren načrt dela

Vir

[1] P. K. Agarwal, N. Kumar, S. Sintos in S. Suri. *Computing Shortest Paths in the Plane with Removable Obstacles*. In 16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018). Leibniz International Proceedings in Informatics (LIPIcs), Volume 101, pp. 5:1-5:15, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2018)

[2] M. Berg, O. Cheong, M. Kreveld in M. Overmars *Computational Geometry: Algorithms and Applications* Springer Berlin, Heidelberg, 2010