Monoton függvény fixpontjának algoritmikus keresése

2022. május 13.

Bursics Balázs, Zólomy Kristóf

1. Bevezető

Banach fixponttétele szerint egy teljes hálón (azaz olyan részbenrendezett halmazon, melyben minden részhalmaznak van legkisebb közös felső korlátja, és legnagyobb közös alsó korlátja) megadott monoton függvénynek létezik fixpontja. A tétel bizonyítása viszont az algoritmikus bonyolultságot nem adja meg, az e félévi egyéni kutatómunkánkban ezzel kapcsolatos kérdésekkel foglalkoztunk. Az általunk vizsgált esetben adott d dimenzióban valamilyen $N \in \mathbb{N}$-re az $L = \{0, 1, \ldots, N\}^d$ rács, amelyen részbenrendezés az alábbi szabály szerint van: $(a_1, \ldots, a_d) \leq (b_1, \ldots, b_d)$ pontosan akkor, ha minden $i \leq d$-re $a_i \leq b_i$.

A feladat egy orákulummal adott $f : L \to L$ függvény egy fixpontjának megtalálása, vagy két olyan pont megmutatása, amelyek miatt a függvény nem lehet monoton.

Az általános eset megoldatlan, d dimenzió esetén adódik egy $\log^d(N)$ idejű algoritmus, Fearnley, Pálvölgyi és Savani viszont 3 dimenzióban találtak egy $O(\log^2(N))$ idejű algoritmust([2]), amellyel d dimenzióban is $O(\log^{2\lceil d/3 \rceil} N)$-re javították a becslést, amit Chen és Li nemrég megjelent cikkükben $O(\log^{\lceil (d+1)/2 \rceil} N)$-re javítottak ([3]).

A legjobb ismert alsó becslés minden $d \geq 2$ dimenzióra $\Omega(\log^2(N))$, azaz létezik olyan f monoton függvény, amely fixpontját legrosszabb esetben ennél kisebb nagyságrendű lépésszámban nem lehet megtalálni. Ez a becslés 2 és 3 dimenzióban éles, magasabb dimenzióban azt sejtjük, hogy nem az.

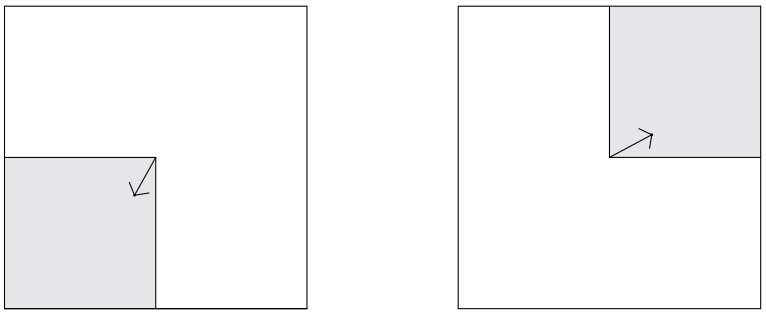
Ez a konstrukció megtalálható az [1] cikkben.

Az alábbiakban bemutatjuk a fent említett eredményeket, és a saját munkánkat.

2. Felső becslések

A kérdésre adott felső becslések egyik fő alapgondolata a következő megfigyelés:

2.1. Állítás. *Legyenek a és b olyan pontok, melyekre $f(a) \geq a$, és $f(b) \leq b$. Ekkor f a $\{c : a \leq c \leq b\}$ részrácsot önmagába képezi, és ezért létezik fixpontja ezen a halmazon.*



1. ábra. A rács csökkentése

A d-dimenziós feladat megoldható rekurzív bináris kereséssel $\log^d(N)$ lépésben: Először is, 1 dimenzióban egyszerű bináris kereséssel kaphatunk fixpontot, hiszen $f(N/2) < N/2$ esetén a $[0, N/2]$ intervallumban (sőt, a $[0, f(N/2)]$ intervallumban is) található fixpont, mert a függvény a monotonitás miatt ezt az intervallumot önmagába képzi. Hasonlóan, $f(N/2) < N/2$ esetén pedig az $[N/2, N]$ intervallumban található fixpont. Az így kapott $\leq N/2$ hosszú intervallumot további kérdésekkel újra és újra továbbfelezhetjük, így $O(\log N)$ lépésben találunk egy fixpontot. A d dimenziós esetben rögzítsük az utolsó koordinátát $N/2$-re, az így kapott $(d-1)$ dimenziós rácsban $\log^{d-1}(N)$ lépésből találhatunk olyan pontot, aminek az első $d-1$ koordinátája fixen marad (hiszen az első $d-1$ koordinátára szorítkozva is monoton a függvény, erre alkalmazhatjuk a rekurzióból adódó keresési algoritmust). Ezzel a 2.1. Állítás alapján megint megfelezhetjük a rácsot, így összesen $O(\log^d N)$ lépésből találhatunk fixpontot.

Többen azt a sejtést fogalmazták meg, hogy ez a felső becslés éles ([1], [4]), de mint a következő tételből kiderül, már 3 dimenzióban sem az:

2.2. Tétel (Fearnley, Pálvölgyi, Savani)**.** *A $d = 3$ esetben $O(\log^2(N))$ lépéssel találhatunk fixpontot.*

Az algoritmus azon alapul, hogy az aktuálisan vizsgált rácsrész csökkentéséhez nem szükséges olyan pontot találnunk, aminek majdnem minden koordinátája fixen marad (mint az iménti rekurzív algoritmusban), hanem elegendő olyat, amit egy vele összehasonlítható pontba visz a függvény.

Az algoritmus egy szubrutinja a következő problémát oldja meg: A rácsnak egy koordináta rögzítésével kapott részrácsán keresünk olyan pontot, amelyet a függvény egy vele összehasonlítható pontba visz.

A [2] cikkben megtalálható ennek a részletes leírása (inner algorithm néven), 3 dimenzióban az esetek megfelelő szétválasztásával $O(\log N)$ lépésből található ilyen pont. Ebből pedig úgy adódik az $O(\log^2 N)$-es felső korlát, hogy az éppen vizsgált rácsot a leghosszabb oldala mentén megfelezzük, és ezt a koordinátát rögzítve alkalmazzuk a szubrutint egy olyan pont keresésére a kapott hipersíkon, melyet vele összehasonlítható pontba visz a függvény, egy ilyen pont segítségével pedig megfelezhetjük a vizsgált rács méretét.

A következő tétel segítségével a meglévő algoritmusainkból nagyobb dimenziós algoritmusokat is kaphatunk:

2.3. Tétel (Fearnley, Pálvölgyi, Savani)**.** *Ha az a-dimenziós rácson q_a, a b-dimenziós rácson pedig q_b lépésben találhatunk fixpontot, akkor az $a + b$ dimenziós esetet $q_a \cdot (q_b + 2)$ lépésben megoldhatjuk.*

Ennek segítségével a 3 dimenziós becslés a magasabb dimenziós eseteket is megjavítja:

2.4. Következmény. *A d-dimenziós esetben $O(\log^{2\lceil d/3 \rceil} N)$ lépésben találhatunk fixpontot.*

A kisebb dimenziós algoritmusok összeillesztésének alapötlete a következő: futtassuk az a dimenziós algoritmust az első a darab koordinátán olyan módon, hogy az orákulum használatakor előbb az utolsó b koordinátában futtassuk a másik algoritmust, majd az így kapott pont függvényértékét használjuk fel. Azaz ha az első algoritmus a $c = (c_1, \ldots, c_a)$ pontot kérdezi le, akkor először az $\{x = (x_1 \ldots, x_{a+b}) : \forall i \leq a \;\; x_i = c_i\}$ altéren futtassuk az algoritmust, majd a kapott pont (melynek utolsó b koordinátáját f nem változtatja) f szerinti képének első a koordinátáját adjuk vissza függvényértékként. Ha így eljárva az első a koordinátában változatlan pontot kapunk, az egyben egy fixpontot is meghatároz, hiszen az eljárásunk szerint kaptunk egy olyan pontot, melynek az első koordinátái egyeznek, és az utolsó b koordinátája sem változik.

Érdekes módon a következő, erősebb becslés is az előbbi 3 dimenziós eredményen alapul:

2.5. Tétel (Chen, Li)**.** *A d dimenziós esetben $O(\log^{\lceil (d+1)/2 \rceil} N)$ lépésből található fixpont.*

Ebben az esetben a dimenzióugrás nem a teljes algoritmusnál, hanem a korábban leírt szubrutinnál (a [3] cikkben: Tarski*) történik, azaz a keresési algoritmus ugyanazon segédproblémára használ egy szubrutint, majd a már látott módon a lépésszámot $\log N$-szeresére növelve megtalál egy fixpontot.

A segédprobléma megoldása viszont egy következő, formálisan erősebb, de valójában lényegében ugyanolyan nehéz segédprobléma segítségével történik (a [3] cikkben: RefinedTarski*): Az egy koordináta rögzítésével kapott síkban két pontot keresünk, p^l-et és p^r-et, úgy, hogy $f(p^l) \geq p^l$, $f(p^r) \leq p^r$, és az alábbi három feltétel közül legalább egy teljesüljön:

1. p^l a rögzített koordinátában szigorúan nő
2. p^r a rögzített koordinátában szigorúan csökken
3. mindkét pont a rögzített koordinátában változatlanul marad

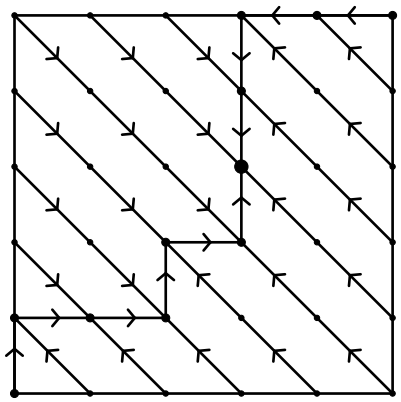
Ez a segédprobléma az előző, egyszerűbb szubrutin segítségével konstansszoros időben megoldható, továbbá a segítségével a szubrutin dimenzióját is növelhetjük, a következő állítás szerint:

2.6. Állítás. *Ha RefinedTarski* $a + 1$ dimenzióban q_a, $b + 1$ dimenzióban pedig q_b lépésben megoldható, akkor az $a + b + 1$ dimenziós Tarski* $O(q_a q_b)$ lépésben megoldható.*

Mivel 3 dimenzióban a Tarski* szubrutin $O(\log N)$ lépésben megoldható, így Tarski* d dimenzióban $O(\log^{\lceil (k-1)/2 \rceil} N)$ lépésben megoldható, amiből már következik a tétel állítása.

3. Az alsó becslés

Az alábbiakban bemutatjuk az orákulum egy lehetséges működését, amellyel a fixpont megtalálása $\Omega(\log^2(N))$ kérdést igényel, ez az alsó korlát [1]-ben van leírva. Az ötlet az, hogy a függvényünk az alábbi formátumú lesz: Lesz a $(0,0)$ és az (N,N) pontok között egy út, amelyen valahol van egy fixpont, és az úton minden pontból a a vele szomszédos, úton lévő, fixponthoz közelebb lévő pontba menjen a függvény. Az úton kívül pedig átlósan közeledjen a függvény 1 mezőnyit az úthoz, ezzel csak az út hollétéről ad információt, az úton belül a fixpont elhelyezkedéséről nem.



2. ábra. Az alsó becslés konstrukciója

Osszuk fel a négyzetet $\sqrt{N}$ régióra,

$$R_a := \{(x,y) \in (0,\dots,N) \times (0,\dots,N) : a\sqrt{N} \leq x+y < (a+1)\sqrt{N}\}$$

ezekben a régiókban egymástól majdnem függetlenül megy az út: Minden régiót osszunk tovább $\Theta(N^{1/4})$ alrégióra,

$$S_b := \{(x,y) \in (1,\dots,N) \times (1,\dots,N) : 2bN^{1/4} \leq x+y < (2b+2)N^{1/4}\},$$

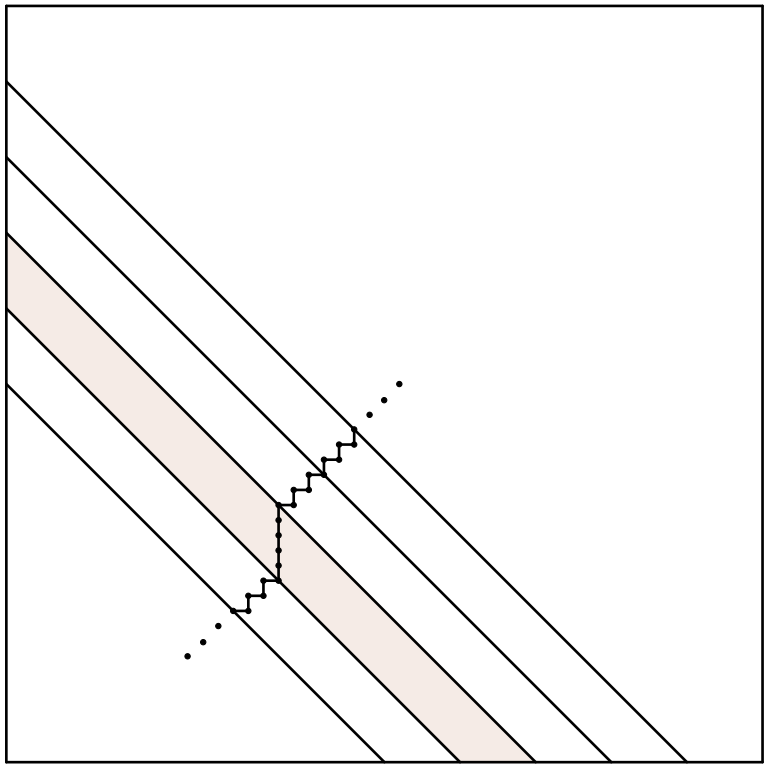
és minden régióból válasszunk ki egy speciális alrégiót egyenletes eloszlással, a nem speciális alrégiókban menjen az út felváltva jobbra és felfelé, azaz a jobb felső sarok felé „átlósan". Továbbá minden R_a régióhoz válasszunk egyenletes eloszlással egy $l_a \in [-N^{1/4}, N^{1/4}]$ értéket, és a speciális alrégióban úgy menjen út, hogy az alrégió végén $x-y = l_a$ teljesüljön, ez bármilyen érték esetén lehetséges, hiszen az $x-y$ érték az alrégióban $2N^{1/4}$-et változhat, és előtte is a $[-N^{1/4}, N^{1/4}]$ intervallumban van a két koordináta különbsége.

Ezzel megadtuk a monoton függvényt, erről belátjuk, hogy a fixpontját $\Omega(\log^2(N))$ idő megtalálni.

3.1. Állítás. *Egy adott R_a régióbeli speciális alrégió megtalálásához kell $\Omega(\log(N))$ kérdés ebből a régióból.*

Bizonyítás. A régión kívüli függvényértékek nem árulnak el semmit sem a speciális alrégió helyzetéről, így a régió két végén az út helyzetét ismerve is egy bináris keresési feladat az alrégió megtalálása, ezért $\Omega(\log(N^{1/4})) = \Omega(\log(N))$ időt igényel. □

Legyen S_a és S_b két szomszédos régió speciális alrégiója, és T a közéjük eső alrégiók uniója. Vegyük észre, hogy az út T-n belüli részén $x - y$ értéke fix marad 1 híján, és ez az érték csak a két szomszédos alrégiótól függ.



3. ábra. A kiválasztott út haladása a régiókon belül

3.2. Állítás. *Ahhoz, hogy az út T-n belül részének megtaláljuk egy pontját, szükséges $\Omega(\log(N))$ kérdés T-ben, vagy egy pont ismerete a két szomszédos speciális szubrégió valamelyikében az útról.*

Bizonyítás. Ha a speciális alrégiókból nem ismerünk egy pontot sem az útról, akkor a T-n kívüli kérdések nem segítenek, T-n belül pedig szükség van $\Omega(\log(N^{1/4})) = \Omega(\log(N))$ kérdésre. □

A fenti két állítás együttesen adja a következőt:

3.3. Következmény. *Egy R_a régióban egy úton lévő pont megtalálásához szükség van $\Omega(\log(N))$ kérdésre R_a-ból, vagy a szomszédos régiókból.*

3.4. Következmény. *Ahhoz, hogy egy algoritmus találjon az úton $\Omega(\log(N))$ pontot, amelyek egymástól páronként legalább $\Omega(\sqrt{N})$ távolságra vannak, szükség van $\Omega(\log^2(N))$ kérdésre.*

Bizonyítás. Két (vagy adott konstansnál több) ilyen pont nem lehet egy rögzített régióban és szomszédaiban, így egymástól függetlenül $\Omega(\log(N))$ darab $\Omega(\log(N))$ kérdésből álló algoritmust kell futtatni. □

Ebből pedig következik, hogy a fixpontot $\Omega(\log^2(N))$ kérdés megtalálni: ha minden régiót összehúzunk egy pontba, és tekintjük ezen a fixpontkeresési problémát (ugyanúgy irányítva, ahogy az eredeti utat), és az eredeti úton vett kérdéseket az összehúzotton is kérdésnek tekintjük (válasznak pedig vagy eláruljuk, hogy a régióban van a fixpont, vagy megmondjuk, melyik irányban), akkor itt $\Omega(\log(N))$ kérdésre továbbra is szükség van, és ha az eredeti úton egy algoritmus megtalálja a fixpontot, akkor itt is. Továbbá az itteni kérdések egymástól $\Omega(\sqrt{N})$-re lévő pontok kérdezéseinek felelnek meg, tehát valóban szükség van találni az úton $\Omega(\log(N))$ pontot, amelyek egymástól páronként legalább $\Omega(\sqrt{N})$ távolságra vannak.

4. Saját kutatás: Konstans sok változó koordináta esete és egyéb észrevételek

A keresési stratégia keresése során a monoton függvényről feltehető, hogy tetszőleges pont és a képe között minden koordináta legfeljebb 1-et változik, mivel ha egy általános monoton függvénynek vesszük ezt a levágását, akkor az továbbra is monoton marad, a fixpontja helyben marad, és a kereső algoritmus az eredeti függvényből elő tudja állítani a levágott függvényt (tehát az eredeti függvény fixpontját biztosan nem könnyebb megtalálni). Egy f függvény f_l levágásán a következőt értjük: $f_l(x)$ $i.$ koordinátája legyen $f(x)$ $i.$ koordinátája, eggyel növelve ha $f(x)$ $i.$ koordinátája nagyobb, mint x $i.$ koordinátája, eggyel csökkentve, ha kisebb, egyébként pedig ugyanannyi.

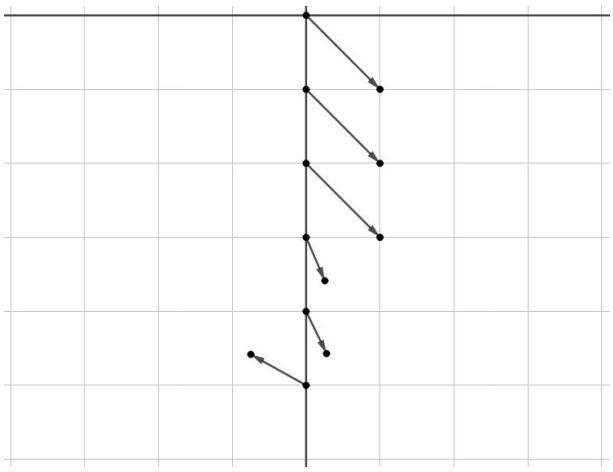
A keresési algoritmus keresése során ezután felmerült, hogy mi az algoritmikus bonyolultsága annak az esetnek, ha a függvény minden pontban csak rögzített számú koordinátában változhat.

Ha minden pontban legfeljebb 1 irányban változhat, akkor $O(\log(N))$ időben megtalálható a fixpont, mivel minden kérdésnél vagy megtaláljuk a fixpontot, vagy felezhető a megmaradt rács mérete.

Ha minden irányban legfeljebb 2 irányban változhat, akkor a sejtésünk az, hogy továbbra is a $\log^c(N)$ időben megtalálható a fixpont valamilyen c konstansra (a dimenziótól függetlenül), mivel minden pontban összesen 1 fajta függvényérték van, amelyek esetén nem tudjuk felezni a vizsgálandó rács méretét, így a függvény jóval kevésbé lehet bonyolult, mint az általános esetben:

Az első kérdés legyen a rács középső pontja, tegyük fel, hogy innen x_1 koordinátában csökken, x_2-ben nő. Az x_1 irány legyen a vízszintes, x_2 a függőleges, ekkor a középpontból jobbra lefele megy a függvény. Ezután nézzük meg a középpont alatti pontban a függvényértéket. Ha egy itteni kérdés után nem tudunk felezni, akkor itt is nő, illetve csökken 1-1 koordinátában. Monotonitás miatt nőni csak x_1-ben tud, csökkeni x_2-ben (ekkor ugyanolyan marad a függvény), vagy egy ezektől eltérő x_3-ban.

Tegyük fel ez utóbbi esetet, és nézzük meg az ezalatti ponton vett függvényértéket. Itt is tegyük fel, hogy nem felez, így egy koordinátában nő, egyben csökken. Mivel a felette lévő pont x_3-ban csökken, ezért neki is kell, nőni pedig szintén a felette lévő pont miatt csak x_1-ben (ekkor ugyanolyan marad a függvény), vagy x_2-ben tud. Tegyük fel az utóbbi esetet, és nézzük meg az alatta lévő pontot. Ennek a felette



4. ábra. Egy lehetséges függvény a függőleges szakaszon

lévő pont miatt x_3-ban csökkennie kell, és tegyük fel, hogy nem felez. Növekedni ez csak x_2-ben tud, azaz ha nem felez egy pont, akkor innentől lefele mindenképpen x_3-ban kell csökkennie, és x_2-ben nőnie. Tehát ha ezen a szakaszon egyik pont sem felező, akkor felbontható 3 szakaszra, a felső szakaszon x_1-ben nő, x_2-ben csökken a függvény, a középsőn x_1-ben nő, x_3-ban csökken, az alsón x_2-ben nő, x_3-ban csökken.

Hasonlóan végignézhető a középpontból jobbra menő szakasz, így ezekben a pontokban is kevés féle függvényérték lehet. Ezekből a jobb alsó negyedsíkra is levonható egy következtetés: ha egy pont nem felez, akkor majdnem egyértelműen meg van határozva ott a függvényérték, ha a két középpontól induló, feljebb tárgyalt szakaszon rögzítettük a függvényt: legfeljebb 2 lehetséges irány van, amerre nőhet, x_2 irányában, illetve a középső vízszintes szakaszon felette lévő pont növekedési irányában, a monotonitás miatt. Hasonlóan csökkenni csak x_1 irányában tud, vagy abban, amerre a függőleges középső szakaszon a vele egy magasságban lévő pont csökkent.

Mindezen észrevételek ellenére nem található a síkban biztosan 'felező' pont, ezért ez nem elég a $\log^c(N)$ időhöz.

Jobb alsó korlát 4 dimenzióban

Jelenleg minden $d \geq 2$ dimenzióban a legjobb ismert alsó becslés $\log^2(N)$, ez $d = 2, 3$ esetén éles, mi 4 dimenzióban próbáltunk ezen javítani. A 2 dimenziós rejtést próbáltuk meg átvinni 4 dimenzióba, de nem sikerült megfelelő monoton függvényt találni. Ha ugyanúgy egy $(0, 0, 0, 0)$ és (N, N, N, N) közti monoton úton akarjuk elrejteni a fixpontot, és egyébként az út fele irányítani a függvényt, akkor az $x_1 + x_2 + x_3 + x_4 = c$ hipersíkokon meg kellene adni egy olyan „természetes" függvényt, ami egy adott pont felé visz, és ilyen hipersíkok összerakva monoton függvényt adnak.

Hivatkozások

[1] Kousha Etessami, Christos Papadimitriou, Aviad Rubinstein, Mihalis Yannakakis: Tarski's theorem, supermodular games, and the complexity of equilibria. In 11th Innovations in Theoretical Computer Science Conference (ITCS 2020). Schloss Dagstuhl- Leibniz-Zentrum für Informatik, 2020.

https://arxiv.org/abs/1909.03210

[2] John Fearnley, Dömötör Pálvölgyi and Rahul Savani. A faster algorithm for finding Tarski fixed points. ACM Transactions on Algorithms, to appear.

https://arxiv.org/abs/2010.02618

[3] Xi Chen, Yuhao Li: Improved Upper Bounds for Finding Tarski Fixed Points. arXiv preprint arXiv:2202.05913, 2022.

https://arxiv.org/abs/2202.05913

[4] Chuangyin Dang, Qi Qi, Yinyu Ye: Computations and Complexities of Tarski's Fixed Points and Supermodular Games. arXix preprint arXiv:2005.09836, 2020.

https://arxiv.org/abs/2005.09836