Szegedi Tudományegyetem
Számítógépes Algoritmusok és Mesterséges Intelligencia Tanszék

Online algoritmusok klaszterezési problémákra

Ph.D. értekezés tézisei

Divéki Gabriella

Témavezető:
Dr. Imreh Csanád
tanszékvezető egyetemi docens

SZTE TTIK, Informatika Doktori Iskola

Szeged
2014

-

1. Bevezetés

A gyakorlatban sűrűn találkozhatunk olyan optimizációs problémával, ahol a bemenetet (vagy inputot: a számokat, amelyek meghatározzák a problémát) részenként ismerjük meg, még azt sem tudjuk előre, hogy van-e még további rész. Ezeket a problémákat *online problémáknak* hívják, és az ún. *online algoritmusok*, amelyek megoldják őket, a bemenetet sorjában dolgozzák fel anélkül, hogy kezdettől rendelkezésre állna a teljes bemenet, és nem is "láthatják előre" a jövőt. Ezzel ellentétben az optimális offline algoritmus előre ismeri a kérések sorozatát. Az online algoritmus arra kényszerül, hogy döntéseket hozzon, amelyekről később kiderülhet, hogy közel sem optimálisak. Az online algoritmusok tanulmányozása az ilyen körülmények közötti döntéshozatal minőségére összpontosul.

Két alap metódust használnak az online algoritmusok hatékonyságának a mérésére. Az egyik lehetőség átlagos esetet vizsgálni. Valamely valószínűségi eloszlást kell feltételezni a lehetséges bemeneten, és erre az eloszlásra vizsgáljuk az algoritmus várható értékét. E metódus fő hátránya az, hogy a bemenet eloszlása általában ismeretlen.

A hatékonyság mérésének a másik megközelítése a legrosszabb eset elemzése, amelyet versenyképességi elemzésnek nevezünk. Ez ugyanazon a bemeneten összehasonlítja az online és az offline algoritmus teljesítményét. Egy algoritmus *versenyképességi hányadosa* a legrosszabb eset hányadosa, amikor az algoritmus költségét osztjuk az optimális költséggel minden lehetséges bemenetet figyelembe véve. Egy online probléma versenyképességi hányadosa a legjobb versenyképességi hányados, amelyet egy online algoritmus elérhet.

Jelölje $A(I)$ egy tetszőleges online algoritmus célfüggvényértékét az I inputon, az optimális offline algoritmus célfüggvényértéke pedig legyen $OPT(I)$. Ezt a jelölésrendszert használva a következőképpen definiálhatjuk a versenyképességet: az A algoritmus c-versenyképes, ha $A(I) \leq c \cdot OPT(I)$ érvényes minden I inputra. A c számot az A versenyképességi hányadosának nevezzük, ha c a legkisebb olyan szám, amelyre ez az egyenlőtlenség igaz.

E munkában az online algoritmusok széles területének egy részét tanulmányozzuk, a klaszterezési problémákat. A kérési pontokat csoportosítani kell: klaszterhez vagy kiszolgálóhoz kell őket rendelni úgy, hogy minimizáljuk a célfüggvényt, amely az egy klaszteren belüli pontok egymás közötti távolságától függ. Az online klaszterező algoritmus klaszterek halmazát kezeli, ahol a klasztereket nevükkel és a már hozzájuk rendelt

pontok halmazával azonosítjuk. A pontokat érkezésükkor klaszterhez kell rendelni; a kiválasztott klasztert ekkor rögzítjük. A klasztereket nem szabad egybevonni vagy szétválasztani. A költségek alapjául a klaszterek száma és tulajdonságai szolgálnak, a probléma pontos meghatározásától függenek.

Sokféle célfüggvény definiálható, ezeknek két fő csoportját vizsgáljuk: ahol a költség a klaszterek méreteitől függ, és ahol a kiszolgáló és a kérési pontok közötti távolságtól függ a költség. Ebben a munkában új modelleket tanulmányozunk, amelyeket még nem vizsgáltak. Minden új modellre bemutatunk egy megoldó algoritmust és meghatározzuk a versenyképességi hányadosát, továbbá megadunk alsó korlátokat az adott modellt megoldó bármely online algoritmus versenyképességi hányadosára.

2. Klaszterezési problémák

Az online klaszterezési problémáknál a kutatók sokáig egységméretű klasztereket használó algoritmusokat tanulmányoztak. A probléma első általánosabb megközelítését a [2] cikkben találjuk. A szerzők változó méretű klasztereket vizsgáltak 1 dimenzióban, ahol a klaszter költsége egy konstans beállítási költség és a klaszter átmérője. Ebben a munkában ennek a modellnek a kiterjesztéseit vizsgáljuk.

Az online klaszterezés változó klaszterméretekkel probléma 1- és 2-dimenziós változatait tanulmányozzuk; az eredményeket a [3], [5] és [6] cikkek mutatják be. A klaszterek intervallumok (2 dimenzióban négyzetek, mivel az l_∞ normát használjuk), a klaszterek költsége a konstans beállítási költség 1-re skálázva plusz az intervallum hossza (illetve 2 dimenzióban a négyzet oldala) négyzetre emelve. A cél a klaszterek összköltségének a minimalizálása.

Két változatot vizsgálunk, mindkettő jellemzője, hogy egy adott klaszterhez rendelt pont e klaszterben is marad, és a klasztereket nem szabad egybevonni vagy szétválasztani. A szigorú változatban a klaszter méretét és helyét készítésekor rögzítjük. A flexibilis változatban az algoritmus eltolhatja vagy nyújthatja a klasztert, de csak annyira, hogy az eddig hozzárendelt pontokat továbbra is tartalmazza.

Sok algoritmusunk alapötlete a $GRID$ algoritmus, amely a [7] cikkben lett definiálva: a d-dimenziós teret d-dimenziós egységkockákkal fedjük, és amikor egy olyan kérési pont érkezik, amely nem tartozik egy már meglévő klaszterhez, új klasztert nyit: a zárt d-dimenziós kockát, ami tartalmazza a pontot.

2.1. Klaszterezési problémák 1 dimenzióban

Először az offline problémát vizsgáljuk, amelynél a teljes bemenet előre adott. A probléma optimális megoldására ajánljuk az egyszerű dinamikus programozáson alapuló DP algoritmust.

A bemenet n kérési pontból áll $(x_1, ..., x_n)$. A dinamikus programozáson alapuló DP algoritmus a k-medián problémát megoldó algoritmus egy változatát használja:

Algoritmus 1 DP algoritmus

- A kérési pontokat növekvő sorrendbe rendezzük koordinátáik szerint.
- Definiáljuk az $F(i, r)$ $(i \geq r)$ részproblémát: az első i kérési pont r klaszterre lett felosztva. Ekkor a klaszterezési probléma optimális költsége $min_r(F(n, r) + r)$.
- Az $F(i, r)$ értékeket a következő rekurziókból számíthatjuk ki:
  $F(i, 1) = (x_i - x_1)^2$
  $F(i, r) = min_{j=r}^{i}\{F(j - 1, r - 1) + (x_i - x_j)^2\}$

A klaszterezési probléma szigorú változatánál a klaszter méretét és pontos helyét rögzíteni kell akkor, amikor létrehozzuk. Vizsgáljuk az online és a félig online verziót is, ahol a "félig online" kifejezés azt jelenti, hogy az algoritmus ismer valamit az ezután következő kérési pontokról. A mi esetünkben növekvő sorrendben vannak a kérési pontok. Az online probléma megoldására bemutatjuk a $GRID_a$ algoritmust, amelynél az egyenest fedő intervallumok hossza az a paraméter. Amikor egy x kérési pont érkezik, amely nem tartozik már meglévő klaszterhez, az algoritmus nyit egy újat, a $[ka, (k + 1)a]$ intervallumot, ahol $k = \lfloor x/a - 1 \rfloor$. A félig online szigorú modellben a pontok növekvő sorrendben érkeznek. E probléma megoldására a $SOSM_a$ algoritmust javasoljuk.

A flexibilis változatban az algoritmus eltolhatja vagy nyújthatja a klasztert mindaddig, amíg tartalmazza az összes eddig hozzárendelt pontot. Bemutatjuk az $FGRID_a$ algoritmust a probléma megoldására.

Érdemes megemlíteni, hogy a $SOSM_a$ algoritmus változtatásával az online esethez hasonlóan (ahogyan a $GRID_a$ algoritmust változtattuk az $FGRID_a$ algoritmussá) nem érhető el 2-nél jobb versenyképességi hányados.

Továbba alsó korlátokat határozunk meg az adott modellt megoldó bármely online algoritmus versenyképességére.

Algoritmus 2 $SOSM_a$ algoritmus

1. Legyen p az újonnan érkezett pont.
2. Ha már létezik klaszter, amely tartalmazza p-t, rendeljük p-t ehhez a klaszterhez.
3. Ellenkező esetben új klasztert nyitunk a $[p,\, p+a]$ intervallumon és p-t ehhez az új klaszterhez rendeljük.

Algoritmus 3 $FGRID_a$ algoritmus

1. Legyen p az új pont.
2. Ha már létezik olyan klaszter, amely tartalmazza p-t, rendeljük p-t ehhez a klaszterhez változtatás nélkül.
3. Ellenkező esetben vizsgáljuk azt a rácscellát, amely a p pontot tartalmazza.
   (a) Ha ebben a cellában még nincs klaszter, nyissunk újat és rendeljük p-t az új klaszterhez. Az új klaszter maga a p pont.
   (b) Egyébként nyújtsuk az intervallumban található klasztert annyira, hogy tartalmazza p-t.

A szigorú modellre elért eredményeinket az első, a flexibilis modellre pedig a második tézisben összegezzük. Ezek az eredmények a [3] cikkben lettek bemutatva.

1. Tézis

1. Lemma. *Az offline probléma optimálisan megoldható a DP dinamikus programozási algoritmus segítségével.*

1. Tétel. *A $GRID_a$ algoritmus versenyképességi hányadosa*

$$\max\{F(\lfloor -2+\sqrt{4+\frac{1}{a^2}}\rfloor), F(\lceil -2+\sqrt{4+\frac{1}{a^2}}\rceil), 2+2a^2\}$$

ahol $F(k)=\frac{(k+2)(1+a^2)}{1+k^2a^2}$, $k\geq 1$.

1. Következmény. *A $GRID_a$ algoritmus a legkisebb versenyképességi hányadosát akkor éri el, ha* $\frac{1}{2\sqrt{2}} \leq a \leq \frac{1}{\sqrt{2}}$*; ekkor a versenyképességi hányados* 3.

2. Tétel. *Bármely online algoritmus versenyképességi hányadosa a szigorú modellre legalább* 2.3243.

3. Tétel. *A $SOSM_a$ algoritmus versenyképességi hányadosa*

$$\max\{F(\lfloor -1+\sqrt{1+\frac{1}{a^2}}\rfloor), F(\lceil -1+\sqrt{1+\frac{1}{a^2}}\rceil), 1+a^2\}$$

ahol $F(k)=\frac{(k+1)(1+a^2)}{1+k^2a^2}$, $k\geq 1$.

2. Következmény. *A $SOSM_a$ algoritmus a legkisebb versenyképességi hányadosát akkor éri el, ha* $\frac{1}{\sqrt{5}}\leq a\leq 1$*; ekkor a versenyképességi hányados* 2.

4. Tétel. *Bármely félig online algoritmus versenyképességi hányadosa a szigorú modellben legalább* 1.6481.

2. Tézis

5. Tétel. *Az $FGRID_a$ algoritmus versenyképességi hányadosa 2, ha* $\frac{1}{\sqrt{5}}\leq a\leq 1$.

6. Tétel. *Bármely online algoritmus versenyképességi hányadosa a flexibilis modellben legalább* 1.2993.

7. Tétel. *Bármely félig online algoritmus versenyképességi hányadosa a flexibilis modellben legalább* 1.1991144.

2.2. Klaszterezési problémák magasabb dimenziókban

2.2.1. Klaszterezési problémák 2-dimenzióban

Hasonlóan az előző szakaszhoz a most már 2-dimenziós euklideszi térben az egyenként érkező pontok számára az algoritmusnak meg kell határozni egy klasztert: vagy egy már nyitott klasztert, vagy egy újat. Minden klaszter költsége a konstans beállítási költség 1-re skálázva plusz a klaszter (négyzet) oldalhosszának a négyzete, a cél pedig minimalizálni a klaszterek összköltségét.

Először a szigorú modellt tanulmányozzuk. Az algoritmus a teret $a\times a$ nagyságú négyzetekkel fedi. Ha érkezik egy olyan kérési pont, amelyet nem fed már létező klaszter, akkor az algoritmus nyit egy újat: a

zárt négyzetet, amely a kérést tartalmazza. Ezt az egyszerű négyzetrácsot alkalmazva 9-versenyképes algoritmushoz jutunk az a paraméter megfelelő kiválasztása esetén. Ha $a/2$ nagyságú eltolásokat alkalmazunk (minden sor négyzetet $a/2$-vel eltolunk az előző sorhoz képest) a rácsban, akkor 8-versenyképes algoritmust kapunk az a legjobb kiválasztásával. Részletesebben tanulmányozzuk az előzőekből továbbfejlesztett $Shift(1/3)GRID_a$-nak nevezett algoritmust, amely $a/3$ nagyságú eltolásokat alkalmaz. Ez a változtatás még jobb végeredményhez vezet: az algoritmus így 7-versenyképes.

A flexibilis 2-dimenziós modell megoldására bemutatjuk a $Shift(1/3)FGRID_a$ algoritmust: ha a rácscella, amelyben az új p kérési pont van nem tartalmaz klasztert, akkor újat nyit (ebben az esetben a klaszter a p pontból áll), egyébként nyújtja a már meglévő klasztert annyira, hogy fedje a p-t is.

A versenyképességi elemzésben azt kaptuk, hogy a $Shift(1/3)GRID_a$ algoritmus az a paraméter egy intervallumában éri el a legjobb versenyképességi hányadosát. Vizsgáltuk, hogy hogyan viselkednek az algoritmusok átlagos esetben ettől a paramétertől függően. Egyenletes eloszlású különböző hosszú véletlenszerű kéréspont-sorozatokat generáltunk egy 30×30 nagyságú négyzetben. Ugyanazon a pontsorozaton vizsgáltuk a $Shift(1/3)GRID_a$ és a $Shift(1/3)FGRID_a$ algoritmusokat, ahol az a paramétert abból az intervallumból vettük, ahol a $Shift(1/3)GRID_a$ algoritmus versenyképességi hányadosa a legjobb, továbbá egy kicsit kisebb és egy kicsit nagyobb paraméter értékeket is felhasználtunk. A kísérleti eredmények elemzése során az alábbi következtetésekre jutottunk. A $Shift(1/3)GRID_a$ algoritmus esetében a legjobb eredményt hozó paraméter érték a kérési pontok számától függ, amely a pontok sűrűségével függ össze. Ahogy a sűrűség nő, jobb nagyobb rácscellákat használni. A $Shift(1/3)FGRID_a$ algoritmus esetében szintén függőséget találunk a pontok sűrűsége és a jobb paraméter értékek között, de ez a függőség nem olyan erős, mint a szigorú változat algoritmusáé. Ennek az okát abban kereshetjük, hogy a $Shift(1/3)FGRID_a$ nem használja a teljes rácscellát. A sűrűségtől való függőség nagyon hasznos lehet, ha van ilyen előzetes információnk a pontokról. Ahogyan vártuk is, a $Shift(1/3)FGRID_a$ algoritmusnak jelentősen kisebb a költsége, mint a $Shift(1/3)GRID_a$-é, nem csak a versenyképességi hányadosa kisebb.

Eredményeinket, melyek a [5] cikkben lettek bemutatva, a 3. tézisben összegezzük.

3. Tézis

8. Tétel. *Ha* $\sqrt{1/2} \leq a \leq \sqrt{27/29}$*, akkor a* $\textsc{Shift}(1/3)\text{GRID}_a$ *algoritmus versenyképességi hányadosa* 7*.*

9. Tétel. *Bármely online algoritmus versenyképességi hányadosa a szigorú modellben legalább* 2.768*.*

10. Tétel. *Legyen* $x = (31 + \sqrt{10529})/59 \approx 2.2748$ *a következő egyenlet pozitív gyöke*

$$\frac{2+2x}{1+x/9} = \frac{6+44x/9}{1+x}.$$

Ha $a = \sqrt{x} \approx 1.508$*, akkor a* $\textsc{Shift}(1/3)\text{FGRID}_a$ *algoritmus* C*-versenyképes, ahol*

$$C = \frac{2+2x}{1+x/9} = \frac{6+44x/9}{1+x} \approx 5.228.$$

11. Tétel. *Bármely online algoritmus versenyképességi hányadosa a flexibilis modellben legalább* $C = (x+4)/(x+1) \approx 1.743$*, ahol* $x \approx 3.0351$ *valós gyöke a* $4x^3 + 4x^2 - 48x - 3 = 0$ *egyenletnek.*

2.2.2. Kiterjesztések: d-dimenziós tér és négyzetes költség helyett általános hatványú költség

A változtatható nagyságú klaszterezési probléma általánosabb változatát is vizsgáltuk: d-dimenziós euklideszi tereket és általánosabb költségfüggvényt. Modellünkben a klaszter költsége egy egységes beállítási költség plusz a rácscella oldalának a p-edik hatványa. A probléma szigorú változatát tanulmányoztuk, ahol a klaszter készítésekor rögzíteni kell a pontos helyét. Elemezzük a $GRID_a$ algoritmust általános esetben, és bebizonyítjuk, hogy nem konstans versenyképes, ha $p < d$, és 3^d-versenyképes, ha $p \geq d$.

2-dimenziós euklideszi térben vizsgáljuk a $\textsc{Shift}(1/3)\text{GRID}_a$ algoritmust, és bebizonyítjuk, hogy 7-versenyképes az a paraméter megfelelő értékeire.

A 4. tézisben bemutatott eredményeink nagy része a [6] cikkben került bemutatásra.

4. Tézis

12. Tétel. *Ha* $p < d$*, a* $GRID_a$ *algoritmus nem konstans versenyképes, ha a költség a d-dimenziós kocka élének a p-edik hatványa.*

13. Tétel. *Ha $p \geq d$, létezik olyan a paraméter, hogy a $GRID_a$ algoritmus 3^d-versenyképes. Továbbá a $GRID_a$ algoritmusnak nem lehet 3^d-től kisebb versenyképességi hányadosa.*

14. Tétel. *A* $\textsc{Shift}(1/3)\text{GRID}_a$ *algoritmus 7-versenyképes, ha*

$$\max\left\{\sqrt[p]{\frac{1}{2^p-2}}, \sqrt[p]{\frac{1}{7\cdot(4/3)^p-8}}\right\} \leq a \leq \sqrt[p]{\frac{27}{29}}.$$

$\sqrt[p]{\frac{1}{2^p-2}}$ *nagyobb* $2 < p \leq x$*-re és* $\sqrt[p]{\frac{1}{7\cdot(4/3)^p-8}}$ *nagyobb* $p > x$*-re, ahol* $x \approx 4.0257$ *gyöke a következő egyenletnek:*

$$\frac{1}{2^p-2} = \frac{1}{7\cdot(4/3)^p-8}$$

.

3. Online kiszolgáló elhelyezés

A kiszolgáló elhelyezés problémánál adott egy metrikus tér $\mathbf{M} = (M, d)$, ahol M a pontok halmaza. Ahhoz, hogy kiszolgáljuk a kéréseket, kiszolgálókat kell nyitni az M halmaz pontjainál. A bemenet kérési pontok sorozata $s_1, \ldots, s_n$, minden kérés az M halmaz egy pontja. Az $I = s_1, \ldots, s_n$ kéréspont-sorozatra jelöljük I_i-vel az első i pontot $(s_1, \ldots, s_i)$. A cél megtalálni a kiszolgálók helyzetének olyan halmazát, amely minimizálja a kiszolgálók költségeinek és a pontok kiszolgálókhoz való rendelésének összegét. A megfelelő offline változat a kombinatorikus optimizáció egy jól ismert problémája. Némely alkalmazásban (pl. számítógépek vagy szenzorok hálózatának létrehozása, néhány klaszterezési probléma) a kéréspontok halmaza nem ismert előre. Egyenként érkeznek, és ekkor az algoritmusnak el kell döntenie hogy nyisson-e új kiszolgálót vagy rendelje a kérési pontot már meglévő kiszolgálóhoz úgy, hogy semmi információ nem áll rendelkezésre a további kéréspontokról. A kiszolgáló elhelyezés online verzióját először Meyerson tanulmányozta a [11] cikkben, amelyben nem engedélyezett mozgatni az algoritmus által kinyitott kiszolgálókat.

A problémát tovább vizsgálta Fotakis [9]-ben, és adott egy determinisztikus $O(\frac{\log n}{\log\log n})$-versenyképes algoritmust egységes és nem egységes kiszolgáló költségre. Továbbá bebizonyította, hogy nincs determinisztikus vagy véletlenített algoritmus, amelynek kisebb lenne a versenyképessége, mint $\Omega(\frac{\log n}{\log\log n})$.

Ebben a munkában bevezetünk egy olyan modellt, ahol az algoritmus más helyre mozgathatja a kiszolgálókat egy kérési pont megérkezése után, de már megnyitott kiszolgálókat nem zárhatunk be, így a kiszolgálók számának növelése visszavonhatatlan döntés, amely később rossz választásnak bizonyulhat.

Egységes f kiszolgáló költséget vizsgálunk, ahol ez ugyanaz minden kiszolgáló számára. Eredményeinket a [4] cikkben mutatjuk be. Ha egy SOL megoldás az $a_1, \ldots, a_k$ pontoknál nyit kiszolgálókat, e megoldás összköltsége

$$c(SOL) = k \cdot f + \sum_{i=1}^{n} \min_{j=1,\ldots,k} d(s_i, a_j).$$

A cél megtalálni azt a megoldást, amelynél ez a költség minimális. Bármely A algoritmusra jelölje $C_A(I)$ az összköltséget, $S_A(I)$ a kiszolgálás költségét és $F_A(I)$ a kiszolgálók nyitásának a költségét az I bemeneten.

Az online probléma megoldására javasoljuk az OFW (Optfollow) algoritmust. Az algoritmus alapötlete az, hogy utánozza az optimális offline algoritmus viselkedését. Az OFW a következő szabályokat alkalmazza az s_i sorozat i-edik pontjának a megérkezése után:

Algoritmus 4 OFW algoritmus

- 1. lépés: Határozzuk meg az I_i bemenetre az optimális offline megoldást.
- 2/a. lépés: Ha $F_{OFW}(I_{i-1}) \leq F_{OPT}(I_i)$, akkor nyissunk $(F_{OPT}(I_i) - F_{OFW}(I_{i-1}))/f$ számú új kiszolgálót, a $F_{OPT}(I_i)/f$ kiszolgálót pedig mozgassuk az optimális pozíciókra.
- 2/b. lépés: Ha $F_{OFW}(I_{i-1}) > F_{OPT}(I_i)$, akkor ne nyissunk új kiszolgálót, hanem oldjuk meg az offline $F_{OFW}(I_{i-1})/f$-medián problémát az I_i bemenetre, majd a kiszolgálókat mozgassuk a kapott pozíciókra.

Általános metrikus terek esetén az OFW algoritmus NP-nehéz problémát old meg az 1. lépésben, szintén a 2/b. lépésben. Ezeket az NP-nehéz problémákat átfogóan tanulmányozták, és néhány pontos megoldó algoritmust is kifejlesztettek (lásd a részleteket itt: [1], [8], [10]). Másrészt, ha a metrikus tér az egyenes, akkor mindkét probléma megoldható polinomiális időben.

Ha approximációs algoritmusokat használunk az NP-nehéz problémák

megoldására, definiálhatjuk a $POFW$ algoritmust, ami az OFW polinomiális változata.

Algoritmus 5 $POFW$ algoritmus

- 1. lépés: Használjunk polinomiális idejű approximációs offline $FAPPR$ algoritmust az I_i bemenetre a kiszolgáló elhelyezési problémában.
- 2/a. lépés: Ha $F_{POFW}(I_{i-1}) \leq F_{APPR}(I_i)$, akkor nyissunk új $(F_{FAPPR}(I_i) - F_{POFW}(I_{i-1}))/f$ számú kiszolgálót, és mozgassuk a $F_{FAPPR}(I_i)/f$ kiszolgálót az $FAPPR$ által eredményezett pozíciókra.
- 2/b. lépés: Ha $F_{POFW}(I_{i-1}) > F_{FAPPR}(I_i)$, akkor ne nyissunk új kiszolgálót, hanem használjunk polinomiális idejű approximációs offline $MAPPR$ algoritmust a $F_{POFW}(I_{i-1})/f$-medián probléma megoldására az I_i bemeneten, majd mozgassuk a kapott pozíciókba a kiszolgálókat.

A $POFW$ algoritmus approximációs algoritmusokat használ a kiszolgáló elhelyezés és k-medián problémákra, ilyen algoritmusok találhatóak a [12] és [13] cikkekben.

A [4] cikkben bemutatott eredményeinket az 5. tézisben összegezzük.

5. Tézis

15. Tétel. *Az OFW algoritmus 2-versenyképes.*

16. Tétel. *Ha az $FAPPR$ algoritmus c_1-approximációs algoritmus, az $MAPPR$ pedig c_2-approximációs algoritmus, akkor a $POFW$ algoritmus $c_1(1+c_2)$-versenyképes.*

17. Tétel. *Az egyenesen euklideszi távolságot használva az OFW algoritmus $\frac{3}{2}$-versenyképes, ahol azt az optimális megoldást használjuk, amely a legkevesebb kiszolgálót alkalmazza.*

18. Tétel. *Az egyenesre nincs C-versenyképes online algoritmus bármely $C < (\sqrt{13}+1)/4 \approx 1.15$-re.*

Hivatkozások

[1] Christofides, N., and Beasley, J.E. A tree search algorithm for the p-median problem. *European Journal of Operational Research*, **10(2)**, pp. 196–204, 1982.

[2] Csirik, J., Epstein, L., Imreh, Cs., and Levin, A. Online Clustering with Variable Sized Clusters. *Algorithmica*, **65(2)**, pp. 251–274, 2013.

[3] Divéki, G. Online Clustering on the Line with Square Cost Variable Sized Clusters. *Acta Cybernetica*, **21(1)**, pp. 75–88, 2013.

[4] Divéki, G. and Imreh, Cs. Online facility location with facility movements. *Central European Journal on Operations Research*, **19(2)**, pp. 191–200, 2011.

[5] Divéki, G. and Imreh, Cs. An Online 2-dimensional Clustering Problem with Variable Sized Clusters. *Optimization and Engineering*, **14**, pp. 575–593, 2013.

[6] Divéki, G. and Imreh, Cs. Grid based online algorithms for clustering problems. *CINTI 2014*, accepted for publication, 2014.

[7] Epstein, L., Levin, A., and van Stee, R. Online unit clustering: Variations on a theme. *Theoretical Computer Science*, **407(1-3)**, pp. 85–96, 2008.

[8] Erlenkotter, D. A Dual-Based Procedure for Uncapacitated Facility Location. *Operations Research* **26(6)**, pp. 992–1009, 1978.

[9] Fotakis, D. On the Competitive Ratio for Online Facility Location. *Algorithmica*, **50(1)**, pp. 1–57, 2008.

[10] Garfinkel, R.S., Neebe, A.W., and Rao, M.R. An Algorithm for the M-Median Plant Location Problem. *Transportation Science*, **8**, pp. 217–231, 1974.

[11] Meyerson, A. Online facility location. *In Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS2001)*, pp. 426–431, 2001.

[12] Shmoys, D. Approximation algorithms for facility location problems. *Proceedings of 3rd International Workshop of Approximation Algorithms for Combinatorial Optimization, Springer-Verlag LNCS vol. 1913 (2000)*, pp. 27–33, 2000.

[13] Solis-Oba, R. Approximation Algorithms for the k-Median Problem. *In Efficient Approximation and Online Algorithms, LNCS 3484*, pp. 292–320, 2006.