1. Online kiszolgálóelhelyezés

A probléma általános definíciójának megadásához szükség van a metrikus tér fogalmára. Egy (M, d) párost, ahol M a metrikus tér pontjait tartalmazza, d pedig az $M \times M$ halmazon értelmezett távolságfüggvény metrikus térnek nevezünk, ha a távolságfüggvényre teljesülnek az alábbi tulajdonságok

- $d(x, y) \geq 0$ minden $x, y \in M$ esetén,
- $d(x, y) = d(y, x)$ minden $x, y \in M$ esetén,
- $d(x, y) + d(y, z) \geq d(x, z)$ minden $x, y, z \in M$ esetén,
- $d(x, y) = 0$ akkor és csak akkor teljesül, ha $x = y$.

A konstans költségű kiszolgálóelhelyezési feladatban adottak egy metrikus térben $s_1, \ldots, s_n$ kérések, amelyek a metrikus tér pontjai. Az algoritmusnak kiszolgálókat kell elhelyezni a metrikus tér pontjaiba. A cél a kiszolgálás teljes költségét minimalizálni, amely költség az alábbi két részköltség összege:

- A kiszolgálók elhelyezési költsége: egy f konstans szorozva a kiszolgálók számával.
- A kérések kiszolgálási költsége: $\sum_{i=1}^{n} min_{j=1,\ldots,k} d(s_i, p_j)$, ahol a kiszolgálók a $p_1, \ldots, p_k$ pontokban vannak. (Egy kérés kiszolgálásának a költsége a legközelebbi ponttól való távolsága.)

Az online kiszolgálóelhelyezési feladatban a kérések egyenként jönnek és az egyes kérések érkezése után kell eldöntenünk, hogy veszünk e új kiszolgálót. A feladat megoldására Meyerson a következő egyszerű véletlenített algoritmust javasolta.ű

Meyerson algoritmusa Legyen a kérés távolsága a legközelebbi kiszolgálótól d, és legyen $p = \min\{d/f, 1\}$. Vegyünk a kérés helyén egy új kiszolgálót p valószínűséggel.

Véletlenített algoritmusok esetén egy adott inputra az erdemény nem egy adott érték, hanem egy valószínűségi változó, amely függ az algoritmus által használt véletlen döntések kimenetelétől. Ennek megfelelően a versenyképesség fogalmát is módosítani kell, az eerdmény várható értékét felhazsnálva. Egy R véletlenített online algoritmus c-versenyképes, ha $E(R(I)) \leq COPT(I)$ teljesül minden I inputra. Az algoritmusra a következő állítások teljesülnek:

Tétel: Meyerson algoritmusa $O(\log n)$ versenyképes, ahol n a kérések száma.

Bizonyítás: Megtalálható [1]-ben.

Az alábbi állítás is igaz, amelynek bizonyítása túlmutat a kurzus keretein.

Tétel Amennyiben az inputsorozaton a végrehajtás előtt egy egyenletes eloszlás alapján választott permutációt is végrehajtunk, akkor ezen inputokra a Meyerson féle algoritmus konstans versenyképes.

Bizonyos online alkalmazásokban előfordulhat, hogy a megvásásárolt kiszolgáló helyét nem kell véglegesen rögzítenünk, az mozgatható. Amennyiben a mozgatás ingyenes a következő algoritmust használhatjuk a kiszolgálás során.

OPTMASOL algoritmus Határozzuk meg az inputra az optimális megoldást.

Amennyiben az optimális megoldás legalább annyi kiszolgálót használ, mint ahányat eddig az OPTMASOL algoritmus vásárolt, akkor vegyünk új kiszolgálókat, annyit ahány hiányzik az optimális megoldás által használt kiszolgálószámhoz. Majd mozgassuk ezeket az optimális megoldásban szereplő helyekre.

Amennyiben az optimális megoldás kevesebb kiszolgálót használ, mint ahányat eddig az OPTMASOL algoritmus vásárolt, akkor oldjuk meg az adott kiszolgálószám melletti optimális kiszolgálás feladatát (k-median feladat) az online algoritmus által már megvett kiszolgálószámra, és mozgassuk a kiszolgálókat a megoldásban szereplő helyekre.

Az algoritmusra a következő állítások teljesülnek.

Tétel OPTMASOL 2-versenyképes. Amennyiben a metrikus tér a egyenes OPTMASOL 3/2-versenyképes.

Igazolható az alábbi alsó korlát ebben a modellben.

tétel Nincs olyan online algoritmus, ami C-versenyképes valamely $C < (\sqrt{13}+1)/4 \approx 1.15$ esetén.

Bizonyítás: Vegyünk egy A online algoritmust és tegyük fel, hogy C-versenyképes valamely $C < (\sqrt{13}+1)/4$ számra. Vegyük a következő inputot: $s_1 = s_2 = 0$, $s_3 = s_4 = r = (\sqrt{13}-1)/4$ (ekkor $1/2 < r < 1$). Ha A csak egy kiszolgálót vesz, akkor a teljes költsége legalább $2r+1$, az optimális költség 2. Ekkor a sorozat véget ér és $C_A(\sigma_4)/C_{OPT}(\sigma_4) = (2r+1)/2 = (\sqrt{13}+1)/4 > C$, ami ellentmondás.

Tehát feltehetjük, hogy A legalább két kiszolgálót vesz. Ekkor az inputot $s_5 = s_6 = s_7 = r/2$ fejezi be. Az optimális megoldás egyetlen kis-

zolgálót használ az $r/2$ pontban, így $C_{OPT}(\sigma_7) = 2r + 1$. Másrészt A-nak már van legalább két kiszolgálója, így a költsége legalább $2 + r$. Tehát $C_A(\sigma_7)/C_{OPT}(\sigma_7) = (2 + r)/(2r + 1) = (\sqrt{13} + 1)/4 > C$, ami ismét ellentmondás.

2. A k-szerver probléma

Az egyik legismertebb on-line probléma a k-szerver probléma. A k-szerver problémában adott egy metrikus tér, és van k darab szerverünk, amelyek a térben mozoghatnak. A probléma során a tér pontjaiból álló kérések egy listáját kell kiszolgálni azáltal, hogy a megfelelő kérések helyére odaküldünk egy-egy szervert.

A probléma on-line, ami azt jelenti, hogy a kéréseket egyenként kapjuk meg, és az egyes kéréseket a további kérések ismerete nélkül azok érkezése előtt kell kiszolgálnunk. A cél a szerverek által megtett össztávolság minimalizálása. Ezen modellnek és speciális eseteinek számos alkalmazása van. A továbbiakban azt a metrikus tér pontjaiból álló multihalmaz, amely megadja mely pontokban helyezkednek el a szerverek (azért kell multihalmazokat használnunk, mert egy pontban több szerver is lehet) a *szerverek konfigurációjának* nevezzük.

Az első fontos eredményeket a k szerver problémára Manasse, McGeoch és Sleator érték el. A probléma megoldására javasolt első eljárás a következő *Egyensúly algoritmus*, amelyet a továbbiakban ES-el jelölünk. Az eljárás során a szerverek mindig különböző pontokban helyezkednek el. Az algoritmus futása során minden szerverre számon tartja, hogy az aktuális időpontig összesen mekkora távolságot tett meg. Jelölje rendre $s_1, \ldots, s_k$ a szervereket és a pontokat is, ahol a szerverek elhelyezkednek. Továbbá jelölje $D_1, \ldots, D_k$ rendre a szerverek által az adott időpontig megtett összutat. Ekkor, amennyiben egy P pontban megjelenik egy kérés akkor a ES algoritmus azt az i szervert választja a kérés kiszolgálására, ahol a $D_i + d(s_i, P)$ érték minimális. Tehát az algoritmus számon tartja a szerverek $S = \{s_1, \ldots, s_k\}$ szerver konfigurációját, és a szerverekhez rendelt távolságokat, amelyek kezdeti értékei $D_1 = \ldots = D_k = 0$. Ezt követően a algoritmus egy $I = P_1, \ldots, P_n$ pontsorozatra a következőképpen fut

ES(I)
for $j = 1$ **to** n
    $i = \text{argmin}\{D_i + d(s_i, P_j)\}$

szolgáljuk ki a kérést az i-edik szerverrel
$D_i := D_i + d(s_i, P_j)$
$s_i := P_j$

Példa Tekintsük a kétdimenziós Euklédeszi teret, mint metrikus teret. A pontok (x, y) valós számpárokból állnak, két pontnak a, b c, d-nek a távolsága $\sqrt{(a-c)^2 + (b-d)^2}$. Legyen két szerverünk kezdetben a $(0, 0)$ és $(1, 1)$ pontokban. Kezdetben $D_1 = D_2 = 0$, $s_1 = (0, 0)$, $s_2 = (1, 1)$. Az első kérés legyen az $(1, 4)$ pontban. Ekkor $D_1 + d((0, 0)(1, 4)) = \sqrt{17} > D_2 + d((1, 1)(1, 4) = 3$, így a második szervert használjuk és a kérés kiszolgálása után $D_1 = 0, D_2 = 3$, $s_1 = (0, 0)$, $s_2 = (1, 4)$ teljesül. Legyen a második kérés $(2, 4)$, ekkor $D_1 + d((0, 0)(2, 4)) = \sqrt{20} > D_2 + d((1, 4)(2, 4) = 3+1 = 4$, így ismét a második szervert használjuk, és a kérés kiszolgálása után $D_1 = 0, D_2 = 4$, $s_1 = (0, 0)$, $s_2 = (2, 4)$ teljesül. A harmadik kérés legyen ismét az $(1, 4)$ pontban, ekkor $D_1 + d((0, 0)(1, 4)) = \sqrt{17} < D_2 + d((2, 4)(1, 4) = 4 + 1 = 5$, így az első szervert használjuk és a kérés kiszolgálása után $D_1 = \sqrt{17}, D_2 = 4$, $s_1 = (1, 4)$, $s_2 = (2, 4)$ teljesül.

Az algoritmus hatékony speciális terek esetén, miként ezt a következő állítás mutatja. A tétel bizonyítását nem ismertetjük.

tétel *Amennyiben a metrikus tér $k + 1$ pontot tartalmaz, akkor az* ES *algoritmus enyhén k-versenyképes.*

Az alábbi állítás mutatja, hogy a k-szerver problémára általában nem adható meg k-versenyképesnél jobb algoritmus.

tétel *Nincs olyan legalább $k + 1$ pontból álló metrikus tér, ahol megadható olyan on-line algoritmus, amelynek kisebb a versenyképességi hányadosa, mint k.*

Bizonyítás Tekintsünk egy tetszőleges legalább $k + 1$ pontból álló teret, és egy tetszőleges on-line algoritmust. Jelölje az algoritmust ONL, a pontokat ahol kezdetben ONL szerverei állnak $P_1, P_2, \ldots, P_k$, a térnek egy további pontját jelölje P_{k+1}. Vegyük kéréseknek egy hosszú $I = Q_1, \ldots, Q_n$ sorozatát, amelyet úgy kapunk, hogy a következő kérés mindig a $P_1, P_2, \ldots, P_{k+1}$ pontok közül abban a pontban keletkezik, ahol a ONL algoritmusnak nem tartózkodik szervere.

Vizsgáljuk elsőként a ONL(I) költséget. Mivel a Q_j pont kiszolgálása után a Q_{j+1} pont lesz szabad, ezért a Q_j pontot mindig a Q_{j+1} pontban álló szerver szolgálja ki, így a kiszolgálás költsége $d(Q_j, Q_{j+1})$. Következésképpen

$$\mathrm{ONL}(I) = \sum_{j=1}^{n} d(Q_j, Q_{j+1}),$$

ahol Q_{n+1} azt a pontot jelöli, ahol az $n+1$-edik kérés lenne, azaz azt a pontot, amelyről kiszolgáltuk az n-edik kérést.

Most vizsgáljuk a $\mathrm{OPT}(I)$ költséget. Az optimális off-line algoritmus meghatározása helyett definiálunk k darab off-line algoritmust, és ezek költségeinek az átlagát használjuk, mivel mindegyik algoritmus költsége legalább akkora mint a minimális költség, ezért a költségek átlaga is felső korlátja lesz az optimális költségnek.

Definiáljunk k darab off-line algoritmust, jelölje őket $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$. Tegyük fel, hogy a kiindulási állapotban az OFF_j algoritmus szerverei a $P_1, P_2, \ldots, P_{k+1}$ pontok közül a P_j pontot szabadon hagyják, és a halmaz további pontjainak mindegyikén egy szerver helyezkedik el. Ez a kiindulási állapot elérhető egy C_j konstans extra költség felhasználásával.

A kérések kiszolgálása a következőképpen történik. Ha a Q_i ponton van az OFF_j algoritmusnak szervere, akkor nem mozgat egyetlen szervert sem, ha nincs, akkor a Q_{i-1} ponton levő szervert használja. Az algoritmusok jól definiáltak, hiszen ha nincs szerver a Q_i ponton, akkor a $P_1, P_2, \ldots, P_{k+1}$ pontok mindegyikén, így Q_{i-1}-en is van szerver. Továbbá a $Q_1 = P_{k+1}$ ponton az OFF_j algoritmusok mindegyikének áll szervere a kezdeti konfigurációban.

Vegyük észre, hogy az $\mathrm{OFF}_1, \ldots, \mathrm{OFF}_k$ algoritmusok szerverei rendre különböző konfigurációkban vannak. Kezdetben ez az állítás a definíció alapján igaz. Utána a tulajdonság azért marad fenn, mert amely algoritmusok nem mozdítanak szervert a kérés kiszolgálására, azoknak a szerver konfigurációja nem változik. Amely algoritmusok mozgatnak szervert, azok a Q_{i-1} pontról viszik el a szervert, amely pont az előző kérés volt, így ott minden algoritmusnak van szervere. Következésképp ezen algoritmusok szerver konfigurációja nem állítódhat azonos pozícióba olyan algoritmus szerver konfigurációjával, amely nem mozdított szervert. Másrészt, ha több algoritmus is mozgatná Q_{i-1}-ről Q_i-be a szervert azok szerver konfigurációja se válhat azonossá, hisz a mozgatást megelőzőleg különböző volt.

Következésképp egy Q_i kérés esetén, minden OFF_j algoritmusra más a szerver konfiguráció. Továbbá minden konfigurációnak tartalmaznia kell Q_{i-1}-et, tehát pontosan egy olyan OFF_j algoritmus van, amelynek nincsen szervere a Q_i ponton. Tehát a Q_i kérés kiszolgálásának a költsége az OFF_j algoritmusok egyikénél $d(Q_{i-1}, Q_i)$ a többi algoritmus esetén 0.

Következésképp

$$\sum_{j=1}^{k} \mathrm{OFF}_j(I) = C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}),$$

ahol $C = \sum_{j=1}^{k} C_j$ egy az input sorozattól független konstans, amely az off-line algoritmusok kezdeti konfigurációinak beállításának költsége.

Másrészt az off-line optimális algoritmus költsége nem lehet nagyobb semelyik off-line algoritmus költségénél sem, így $k \cdot \mathrm{OPT}(I) \leq \sum_{j=1}^{k} \mathrm{OFF}_j(I)$. Következésképpen

$$k \cdot \mathrm{OPT}(I) \leq C + \sum_{i=2}^{n} d(Q_i, Q_{i-1}) \leq C + \mathrm{ONL}(I),$$

amely egyenlőtlenségből következik, hogy az ONL algoritmus versenyképességi hányadosa nem lehet kisebb, mint k, hiszen az inputsorozat hosszúságának növelésével a $\mathrm{OPT}(I)$ érték tetszőlegesen nagy lehet.

Az általános probléma vizsgálata mellett a probléma speciális eseteit számos dolgozatban vizsgálták. Amennyiben bármely két pont távolsága 1, akkor a lapozási probléma on-line változatához jutunk, amely a számítógépek memóriakezelését modellezi. Egy másik vizsgált speciális tér az egyenes. Az egyenes pontjait a valós számoknak feleltetjük meg, és két pontnak a-nak és b-nek a távolsága $|a - b|$. Erre az esetre, ahol a metrikus tér egy egyenes, Chrobak és Larmore kifejlesztett egy k-versenyképes algoritmust, amelyet *dupla lefedő algoritmusnak* nevezünk. Az algoritmus a P kérést a P-hez legközelebb eső s szerverrel szolgálja ki, és amennyiben vannak szerverek P-nek az s-el átellenes oldalán is, akkor azon szerverek közül a P-hez legközelebbit $d(s, P)$ egységnyit mozdítja P felé. A továbbiakban a dupla lefedő algoritmust DL-el jelöljük.

Példa Tegyük fel, hogy három szerverünk van, s_1, s_2, s_3 amelyek az egyenes $0, 1, 2$ pontjaiban helyezkednek el. Amennyiben a következő kérés a 4 pontban jelenik meg, akkor DL a legközelebbi s_3 szervert küldi a kérés kiszolgálására a többi szerver helye nem változik, a költség 2 és a kérés kiszolgálása után a szerverek a $0, 1, 4$ pontokban lesznek. Amennyiben ezt követően a következő kérés a 2 pontban jelenik meg, akkor DL a legközelebbi s_2 szervert küldi a kérés kiszolgálására, de mivel a kérés másik oldalán is van szerver, ezért s_3 is megtesz egy egységnyi utat a kérés felé, így a költség 2 és a kérés kiszolgálása után a szerverek a $0, 2, 3$ pontokban lesznek.

A DL algoritmusra teljesül a következő állítás.

tétel *A* DL *algoritmus k-versenyképes ha a metrikus tér egy egyenes.*

Bizonyítás Vegyünk egy tetszőleges sorozatát a kéréseknek, jelölje ezt az inputot I. Az eljárás elemzése során feltételezzük, hogy párhuzamosan fut a DL algoritmus és egy optimális off-line algoritmus. Szintén feltesszük, hogy minden kérést elsőként az off-line algoritmus szolgál ki, utána pedig az on-line algoritmus. Az on-line algoritmus szervereit és egyben a szerverek pozícióit, (amelyek valós számok az egyenesen) $s_1, \ldots, s_k$ jelöli, az optimális off-line algoritmus szervereit és egyben a szerverek pozícióit $x_1, \ldots, x_k$ jelöli. Mivel a szerverek rendszeres átjelölésével ez elérhető feltételezzük, hogy $s_1 \leq s_2 \leq \ldots \leq s_k$ és $x_1 \leq x_2 \leq \ldots \leq x_k$ mindig teljesül.

A tétel állítását a potenciálfüggvény technikájával igazoljuk. A potenciálfüggvény a szerverek aktuális pozíciójához rendel egy értéket, az on-line és az off-line költségeket a potenciálfüggvény változásainak alapján hasonlítjuk össze. Legyen a potenciálfüggvény

$$\Phi = k \sum_{i=1}^{k} |x_i - s_i| + \sum_{i<j} (s_j - s_i).$$

Az alábbiakban igazoljuk, hogy a potenciálfüggvényre teljesülnek az alábbi állítások.

- Amíg OPT szolgálja ki a kérést, addig a potenciálfüggvény növekedése legfeljebb k-szor az OPT szerverei által megtett távolság.
- Amíg DL szolgálja ki a kérést, addig Φ legalább annyival csökken, mint amennyi a kérés kiszolgálásának költsége.

Amennyiben a fenti tulajdonságok teljesülnek, akkor a tétel állítása következik, hiszen ebben az esetben adódik, hogy $\Phi_v - \Phi_0 \leq k \cdot \mathrm{OPT}(I) - \mathrm{DL}(I)$, ahol Φ_v és Φ_0 a potenciálfüggvény kezdeti és végső értékei. Mivel a potenciálfüggvény nemnegatív, ezért adódik, hogy $\mathrm{DL}(I) \leq k\mathrm{OPT}(I) + \Phi_0$, azaz azt kapjuk, hogy a DL algoritmus enyhén k-versenyképes.

Most igazoljuk a potenciálfüggvény tulajdonságait.

Elsőként vizsgáljuk azt az esetet, amikor OPT valamely szervere d távolságot mozog. Ekkor a potenciálfüggvényben szereplő első rész legfeljebb kd-vel növekszik a második rész nem változik, tehát az első tulajdonsága a potenciálfüggvénynek valóban fennáll.

Most vizsgáljuk DL szervereit. Legyen P a kérés melyet ki kell szolgálni. Mivel elsőként OPT szolgálta ki a kérést, ezért $x_j = P$ valamely szerverre. Most különböztessünk meg két esetet DL szervereinek elhelyezkedésétől függően.

Elsőként tegyük fel, hogy minden szerver P-nek ugyanarra az oldalára esik. Feltehetjük, hogy minden szerver pozíciója nagyobb P-nél, a másik eset teljesen hasonló. Ekkor s_1 a legközelebbi szerver P-hez és DL s_1-et küldi P-be, más szervert nem mozgat. Tehát DL költsége $d(s_1, P)$. A potenciálfüggvényben szereplő első összegben csak az $|x_1 - s_1|$ tag változik és ez csökken $d(s_1, P)$ egységgel, tehát az első rész csökken $kd(s_1, P)$ egységgel. A második tag növekszik $(k-1)d(s_1, P)$ egységgel, így Φ értéke csökken $d(s_1, P)$ egységgel.

Most tekintsük a másik esetet! Ekkor P-nek mindkét oldalára esik szerver, legyenek ezek a szerverek s_i és s_{i+1}. Tegyük fel, hogy s_i esik közelebb P-hez, a másik eset teljesen hasonló. Tehát DL költsége $2d(s_i, P)$. Vizsgáljuk a potenciálfüggvény első részének változásait! Az i-edik és az $i+1$-edik tag változik. Az egyik tag növekszik, a másik csökken $d(s_i, P)$ egységgel, tehát az első rész összességében nem változik. A Φ függvény második részének változása

$$d(s_i, P)(- (k-i) + (i-1) - (i) + (k-(i+1))) = -2d(s_i, P).$$

Tehát ebben az esetben is fennáll a potenciálfüggvény második tulajdonsága. Mivel több eset nem lehetséges ezért igazoltuk a potenciálfüggvény tulajdonságainak fennállását, amivel a tétel állítását is bebizonyítottuk.

[1] A. Meyerson: Online Facility Location. *Prooceedings of FOCS 2001*, 426–431, 2001.

`http://www.cs.ucla.edu/~awm/papers/ofl.ps`