1. Fogalmak, definíciók

Mivel egy online algoritmusnak részenként kell meghozni a döntéseit a teljes bemenet ismerete nélkül, ezért egy ilyen algoritmustól nem várhatjuk el, hogy a teljes információval rendelkező algoritmusok által megkapható optimális megoldást szolgáltassa. Azon algoritmusokat, amelyek ismerik a teljes inputot *off-line algoritmusoknak* nevezzük.

Az online algoritmusok hatékonyságának vizsgálatára két alapvető módszert használnak. Az egyik lehetőség az *átlagos eset elemzése,* ebben az esetben fel kell tételeznünk valamilyen valószínűségi eloszlást a lehetséges bemenetek terén, és az erre az eloszlásra vonatkozó várható értékét vizsgáljuk a célfüggvénynek.

Ezen megközelítés hátránya, hogy általában nincs információnk arról, hogy a lehetséges inputok milyen valószínűségi eloszlást követnek. Mi az átlagos eset elemzésének témakörével nem foglalkozunk, hanem az elterjedtebb versenyképességi analízis módszerét használjuk.

A másik megközelítés egy legrosszabb-eset korlát elemzés, amelyet *versenyképességi elemzésnek* nevezünk. Ebben az esetben az online algoritmus által kapott megoldás célfüggvényértékét hasonlítjuk össze az optimális off-line célfüggvényértékkel.

Egy online minimalizálási probléma esetén egy online algoritmust C-*versenyképesnek* nevezünk, ha tetszőleges inputra teljesül, hogy az algoritmus által kapott megoldás költsége nem nagyobb mint C-szer az optimális off-line költség. Egy *algoritmus versenyképességi hányadosa* a legkisebb olyan C szám, amelyre az algoritmus C-versenyképes.

A továbbiakban egy tetszőleges ALG online algoritmusra az I inputon felvett célfüggvényértéket $\mathrm{ALG}(I)$-vel jelöljük. Az I inputon felvett optimális off-line célfüggvényértéket $\mathrm{OPT}(I)$-vel jelöljük. Használva ezt a jelölésrendszert a versenyképességet minimalizálási problémákra a következőképpen definiálhatjuk.

Az ALG algoritmus C-versenyképes ha $\mathrm{ALG}(I) \leq C \cdot \mathrm{OPT}(I)$ teljesül minden I input esetén.

Szokás használni a versenyképesség további két változatát. Egy minimalizálási probléma esetén az ALG algoritmus *enyhén* C*-versenyképes* ha van olyan B konstans, hogy $\mathrm{ALG}(I) \leq C \cdot \mathrm{OPT}(I) + B$ teljesül minden I input esetén.

Egy *algoritmus enyhe versenyképességi hányadosa* a legkisebb olyan C szám, amelyre az algoritmus enyhén C-versenyképes.

A fentiekben a minimalizálási problémákra definiáltuk a versenyképességi analízis fogalmait. A definíciók hasonlóan értelmezhetőek maximalizálási problémák esetén is. Ekkor az ALG algoritmus C-versenyképes ha $\mathrm{ALG}(I) \geq C \cdot \mathrm{OPT}(I)$ teljesül minden I input esetén, illetve enyhén C-versenyképes ha valamely B konstans mellett $\mathrm{ALG}(I) \geq C \cdot \mathrm{OPT}(I) + B$ teljesül minden I inputra.

2. Nyugtázás

A számítógépes hálózatok kommunikációja során a küldő a címzettnek adatcsomagokat küld. Amennyiben a kommunikációs csatorna nem teljesen megbízható (pl vezeték nélküli), akkor lényeges a csomagok megérkezéséről a címzettnek nyugtát küldeni, így lehetővé tehető az elveszett adatcsomagok újraküldése. A nyugtázási probléma során felmerülő kérdések egyike, hogy mikor nyugtázzuk a csomag megérkezését. Egy nyugtával több csomag megérkezését igazolhatjuk. Minden csomagra külön nyugta küldése nagy mértékben növelné a kommunikációs csatornák telítettségét. Másrészt, a nyugta küldésével hosszan várakozni sem lehet, hiszen az igazolás késedelme a csomag újraküldéséhez vezethet, ami ismét a csatorna túltelítettségét eredményezheti. A nyugtaküldések idejének megállapítására az első optimalizálási modellt Dooly, Goldman és Scott fejlesztették ki 1998-ban. Az alábbiakban ismertetjük az általuk kifejlesztett modell lényegét és az alapvető eredményeket.

A nyugtázási probléma matematikai modelljében az inputot a csomagok $a_1, \ldots, a_n$ érkezési idejei adják. Az algoritmusnak meg kell határoznia mikor küld nyugtákat, ezeket az időpontokat $t_1, \ldots, t_k$ jelöli. A nyugtázási probléma költségfüggvénye:

$$k + \sum_{j=1}^{k} \nu_j \ ,$$

ahol k a nyugták száma és $\nu_j = \sum_{t_{j-1} < a_i \leq t_j} (t_j - a_i)$, a j-edik nyugta által összegyűjtött teljes késedelem. A probléma online, azaz egy adott t időpontban csak a t-ig megérkezett csomagok érkezési idejeit ismerjük és nincs semmi információnk a további csomagokról.

A probléma megoldására az ébresztő beállításán alapuló algoritmusokat dolgoztak ki. Egy *ébresztő algoritmus* a következőképpen működik. Az a_j csomag érkezésekor beállítunk egy ébresztőt valamilyen $a_j + e_j$ időpontra.

Ha az $a_j + e_j$ időpontig nem érkezik új csomag, akkor az $a_j + e_j$ időpontban nyugtát küldünk, egyébként az a_{j+1} érkezésekor átállítjuk az ébresztőt, egy $a_{j+1} + e_{j+1}$ időpontra. Az alábbiakban egy ébresztő algoritmust elemzünk részletesebben, azt az algoritmust, amely úgy állítja be az ébresztőt, hogy az első nyugtázatlan csomagtól legyen a teljes késedelem költsége 1. Ezt az algoritmust ÉBRESZT algoritmusnak nevezzük. A fenti szabály azt jelenti, hogy az általános definícióban az e_j érték a következő egyenlet megoldása:

$$1 = |\sigma_j| e_j + \sum_{a_i \in \sigma_j} (a_j - a_i) \ .$$

Az algoritmus versenyképességére teljesül a következő állítás.

Tétel *Az* ÉBRESZT *algoritmus 2-versenyképes.*

Biz Tegyük fel, hogy az ÉBRESZT algoritmus k darab nyugtát küld. Ezek a nyugták k darab intervallumot határoznak meg. Az algoritmus költsége legfeljebb $2k$, hiszen k a nyugtákból keletkező költség, és az algoritmus úgy állítja be az ébresztőt, hogy a teljes késedelemből adódó költség minden nyugtára pontosan 1 legyen.

Legyen k^* az optimális off-line algoritmus által küldött nyugták száma. Ha $k^* \geq k$, akkor $\mathrm{OPT}(I) \geq k = \text{ÉBRESZT}(I)/2$ nyilvánvalóan teljesül és adódik, hogy az algoritmus valóban 2-versenyképes. Amennyiben $k^* < k$, akkor az ÉBRESZT algoritmus nyugtái által meghatározott k közül legalább $k - k^*$ intervallumban OPT-nak nincs nyugtája ez legalább $k - k^*$ késedelemből származó költséget jelent OPT számára, így ismét $\mathrm{OPT}(I) \geq k$ adódik, amely egyenlőtlenségből következik, hogy az algoritmus 2-versenyképes.

A fentiekben ismertetett ÉBRESZT algoritmus a lehetséges legjobb algoritmus a versenyképességi analízis szempontjából, hiszen miként a következő állítás mutatja nem létezik olyan algoritmus, amelynek kisebb lenne a versenyképességi hányadosa.

Tétel *Nem létezik olyan online algoritmus az online nyugtázási problémára, amelynek kisebb a versenyképességi hányadosa, mint 2.*

Biz Vegyünk egy tetszőleges online algoritmust, jelölje ONL. Tekintsük a következő inputot. Vegyük csomagok egy hosszú sorozatát, amelyet úgy kapunk, hogy minden esetben, amikor ONL egy nyugtát küld azonnal egy új csomag érkezik. A számítások egyszerűsítéséhez feltesszük, hogy az új csomag érkezéséig eltelő nagyon rövid idő 0, ennek ellenére az új csomagot a nyugta nem nyugtázza. (Ez a feltétel elkerülhető lenne kicsi $\varepsilon > 0$ értékek

használatával.) Ekkor egy $2n$ csomagból álló sorozat esetén az online algoritmus költsége $\mathrm{ONL}(I_{2n}) = 2n + t_{2n}$, hiszen a nyugtákból adódó költsége $2n$, és az i-edik nyugtánál fellépő késedelem $t_i - t_{i-1}$, ahol a $t_0 = 0$ értéket használjuk.

Vizsgáljuk meg a következő két algoritmust, ODD a páros sorszámú csomagok után küld nyugtát, EV pedig a páratlan sorszámú csomagok és közvetlenül az utolsó, $2n$-edik csomag után.

Ekkor ezen algoritmusok költségei

$$\mathrm{EV}(I_{2n}) = n + \sum_{i=0}^{n-1}(t_{2i+1} - t_{2i}) + 1 \ ,$$

és

$$\mathrm{ODD} = n + \sum_{i=1}^{n}(t_{2i} - t_{2i-1}) \ .$$

Következésképpen $\mathrm{EV}(I_{2n}) + \mathrm{ODD}(I_{2n}) = \mathrm{ONL}(I_{2n}) + 1$. Másrészt az optimális off-line algoritmusra $\mathrm{OPT}(I_{2n}) \leq \min\{\mathrm{EV}(I_{2n}), \mathrm{ODD}(I_{2n})\}$, így azt kapjuk, hogy $\mathrm{ONL}(I_{2n})/\mathrm{OPT}(I_{2n}) \geq 2 - 1/\mathrm{OPT}(I_{2n})$. Másrészt ezen egyenlőtlenség alapján adódik, hogy ONL nem lehet jobb, mint 2-versenyképes, hiszen elegendően hosszú sorozatát véve a csomagoknak az $\mathrm{OPT}(I_{2n})$ érték tetszőlegesen nagy lehet.

A fenti állítások mutatják, hogy az ÉBRESZT algoritmus a lehető legkisebb versenyképességi hányadossal rendelkezik. Ezt az okozza, hogy az ébresztőt arra az értékre állítjuk be, amely minimalizálja a versenyképességi hányadost. Felmerül a kérdés, hogy más értékek nem adnának- e jobb eredményt az átlagos esetben, valós adatokon. A következő algoritmus megpróbálja megtanulni az optimális értékét az ébresztő beállításának.

Az algoritmus fázisokban dolgozik, minden fázis k nyugtáig tart. Minden fázisban egy ébresztő típusú algoritmust használunk. Az ébresztőt úgy állítjuk be, hogy az ébresztő időpontjában a teljes késedelem p legyen, jelölje ezt az algoritmus ÉBRESZT_p. Az első fázisban $p = 1$, azaz az ÉBRESZT algoritmust használjuk, utána p-t mindig arra az értékre állítjuk be, amelyik a legjobb megoldást adja az utolsó $10k$ csomagra nézve.

Az optimális p érték meghatározásához az alábbi lemma ad segítséget.

Lemma *Az optimális p értékre teljesül, hogy van olyan nyugta, amelyet közvetlenül valamely csomag érkezésekor küldünk.*

Biz: Vegyünk egy tetszőleges I input sorozatot és tekintsünk egy olyan p értéket, amelyre nem teljesül a lemmában megadott tulajdonság. Jelölje k az ÉBRESZT$_p$ algoritmus által küldött nyugták számát. Az algoritmus definíciója alapján adódik, hogy minden nyugta teljes késedelme p, így a teljes költség $k(1+p)$.

Mivel egyetlen nyugtát sem küldtünk valamely csomag érkezési időpontjában, ezért létezik olyan kicsi $\varepsilon > 0$, hogy az ÉBRESZT$_{p-\varepsilon}$ algoritmus esetén minden nyugta pontosan azokat a csomagot nyugtázza, mint az ÉBRESZT$_p$ algoritmusnál. Másrészt ezen algoritmus esetén a teljes költség $k(1+p-\varepsilon)$, így p nem lehetett optimális.

Tehát az alábbi algoritmus meghatározza az optimális p értéket.

SimpleOpt Algoritmus

- *Inicializálás* Legyen $p^* = 1$ és $Z = \infty$.
- *Keresés* Minden $1 \leq i < j \leq n$ esetén
  - Legyen $p := \sum_{k=i}^{j}(a_j - a_i)$.
  - Hajtsuk végre az ÉBRESZT$_p$ algoritmust az inputon, legyen a nyugták száma k.
  - Ha $k(1+p) < Z$ akkor $p^* := p$ és $Z := k(1+p)$.
- Return p^*

A keresés $\Theta(n^2)$ iterációt hajt végre és minden iterációban a második lépés időigénye $\Theta(n)$ a többi konstans, tehát az algoritmus időigénye $\Theta(n^3)$.

A tanuló algoritmus majdnem 20 százalékkal kisebb költséget adott valós inputokon végrehajtott tesztekben, mint az ÉBRESZT algoritmus.

Az offline feladat megoldható dinamikus proramozással. Az $I = a_1, \ldots, a_n$ inputra jelölje $F(r,k)$ az első r kérésnek k nyugtával való nyugtázása során elérhető minimális késedelem értékét. Az $F(n,k)$ értékek alapján könnyen megkapható az optimális költség: venni kell a $k + F(n,k)$ értékek minimumát, ahol $1 \geq k \geq n$ egész. Az optimális megoldás pedig mint dinamikus programozásnál általában visszafejtéssel vagy a feljegyzéses módszerrel megkapható. Tehát célunk az $F(r,k)$ értékek kiszámítására egy dinamikus programozási algoritmust megadni. Ehhez kezdeti értékeket kell megadnunk és egy rekurzív összefüggést.

Ha $k = 1$, akkor az egyetlen nyugtát az utolsó csomag érkezésekor kell küldeni. Tehát

$$F(r, 1) = \sum_{i=1}^{r} (a_r - a_i).$$

Amennyiben $k > 1$, akkor az utolsó nyugtát az a_r időpontban küldjük. Az utolsó előtti, (k-1)-dik nyugtát küldhetjük az $a_{k-1}, a_k, \ldots, a_{r-1}$ időpontokban. Ha valamely a_q időpontban küldjük, akkor az első $k-1$ nyugta minimális késedelme a definíció alapján $F(q, k-1)$ a k-adik nyugta késedelme pedig $\sum_{i=q+1}^{r}(a_r - a_i)$. Következésképpen, ha $k > 1$, akkor

$$F(r, 1) = \min_{k-1 \leq q \leq r-1} \{F(q, k-1) + \sum_{i=q+1}^{r} (a_r - a_i)\}.$$

A kezdeti feltételek és a rekurzív összefüggés alapján soronkénti táblázatkitöltéssel megkaphatóak a $F(r, k)$ értékek.

Érdekes kérdés, hogy lehet -e 2-nél kisebb versenyképességet elérni, ha az algoritmus valamennyi extra információt kap. Igazolható, hogy tetszőleges N konstansra nem elegendő információ az elkövetkező N csomag érkezési idejét ismerni ahhoz, hogy 2 versenyképesnél jobb hányadosú algoritmust kapjunk. Az alábbiakban egy olyan algoritmust (LIP) adunk meg, amely egy adott c hosszú intervallumban előre látja a csomagok érkezési idejét.

Az algoritmus blokkokra bontja az inputot és minden blokkra a csomagokat az optimális offline megoldás alapján nyugtázza. A blokk mindig az első nyugtázatlan csomagnál kezdődik. Elsőként megvizsgáljuk, hogy van -e két egymást követő csomag az a_i, a_{i+1} a c hosszú intervallumban, amelyekre $a_{i+1} - a_i \geq 1$ teljesül. Ha van ilyen pár, akkor a blokk az első ilyen a_i csomagnál véget ér, egyébként a blokk hossza c. LIP meghatározza az optimális offline megoldást a blokk csomagjaira és ennek megfelelően küldi a nyugtákat.

Tétel *LIP* $1 + 1/c$*-versenyképes.*

Biz Vegyünk egy tetszőleges I inputot és osszuk fázisokra. Legyen $S_1 = \{a_1, \ldots, a_{k(1)}\}$, ahol $k(1)$ az első olyan index, amelyre $a_{k(1)+1} - a_{k(1)} \geq 1$. A többi fázis hasonlóan definiálható $S_{j+1} = \{a_{k(j)+1}, \ldots, a_{k(j+1)}\}$, ahol $k(j+1)$ az első olyan index $k(j)$ után, amelyre $a_{k(j+1)+1} - a_{k(j+1)} \geq 1$. Az utolsó fázist az utolsó munka zárja. Ekkor van olyan optimális offline algoritmus, amely

minden fázis utolsó csomagjánál nyugtát küld. (Ha a fázis utolsó csomagját a következő fázisban nyugtázza, akkor a késedelmi költség növekedése legalább 1.) Másrészt egy fázis utolsó csomagja szintén utolsó csomagja valamelyik blokknak is, így LIP is küld nyugtát a csomag érkezésekor.

Vegyünk egy tetszőleges S_i fázist. Legyen r a benne szereplő blokkok számát. Vegyünk egy optimális megoldását a fázisnak. Ha kiegészítjük $r-1$ további nyugtával, akkor egy olyan megoldást kapunk, amely minden blokk végén nyugtát küld. Másrészt ezek közül LIP a legkisebb költségű megoldást adja meg, így $(r-1) + OPT(S_i) \geq LIP(S_i)$, azaz $LIP(S_i)/OPT(S_i) \leq 1 + (r-1)/OPT(S_i)$.

Mivel minden blokk ugyanabban a fázisban van, ezért az első $r-1$ blokk hossza c, így a fázis hossza legalább $(r-1)c$. Tegyük fel, hogy egy offline algoritmus k nyugtát küld a fázisban. Ekkor az első $k-1$ nyugta mindegyike után egy legfeljebb 1 hosszú csomagmentes időintervallum van. Ebből adódik, hogy a teljes késedelem legalább $(r-1)c-(k-1)$. Következésképpen $OPT(S_i) \geq k + (r-1)c - (k-1) = (r-1)c + 1$. Tehát azt kaptuk, hogy $LIP(S_i)/OPT(S_i) \leq 1 + 1/c$.

Másrészt 1-versenyképes algoritmus nem konstruálható, amint azt az alábbi állítás mutatja.

Tétel *Tetszőleges c-hosszú intervallumra előrenéző algoritmus versenyképességi hányadosa legalább* $1 + \Omega(1/c^2)$.

3. A lapletöltési probléma

A számítógépek memóriakezelésében a lapozás cserélési problémájának az általánosítása a lapletöltési probléma. A világhálón a böngészők is használnak egy memóriát, amelyben a letöltött lapokat tárolják, hogy amennyiben egy oldalt rövid időn belül többször meg akar nézni a felhasználó ne kelljen minden alkalommal letölteni. Amennyiben a memória megtelik és az új lap nem helyezhető el benne, akkor valamilyen stratégia alapján ki kell rakni bizonyos lapokat a memóriából. A lapletöltési probléma a megfelelő cserélési stratégiák megkeresése. A különbség a lapozás problémájához képest, hogy nem minden lap mérete egyforma, továbbá az egyes lapok letöltési költsége is különböző. Tehát a problémát a következőképpen fogalmazhatjuk meg.

Adott egy k méretű memória. Az input a letöltendő lapok sorozata. Minden p lapnak van egy *lapmérete* $s(p)$ és egy *letöltési költsége* $c(p)$. A

letöltendő lapot a memóriába kell tennünk. Ha nem fér el, akkor fel kell szabadítani elegendő helyet a memóriában szereplő lapok kihelyezésével. Ha a kért lap a memóriában van, akkor a letöltés költsége 0 egyébként $c(p)$. Célunk az összköltség minimalizálása. A probléma online, ami azt jelenti, hogy a döntéseket (telített memória esetén mely lapokat dobjuk ki), csak az adott kérésig megtörtént kérések ismerete alapján kell meghozni, a további kérésekre vonatkozó információk nélkül. A továbbiakban feltesszük, hogy mind a memória mérete mind pedig a lapok méretei pozitív egész számok.

A probléma és a speciális eseteinek megoldására több eljárást is javasoltak. Az alábbiakban a Young által kifejlesztett k-versenyképes HÁZIÚR algoritmust és annak elemzését ismertetjük.

Az algoritmus minden lapra, amely az algoritmus memóriájában van, számon tart egy $0 \leq cr(f) \leq c(f)$ értéket. Az algoritmus futása során a HÁZIÚR algoritmus memóriájában aktuálisan szereplő lapok halmazát HA jelöli. Amennyiben egy g fájlt kell letöltenünk, a következő lépéseket hajtjuk végre:

Háziúr(HA, g)

if g nincs a memóriában
then amíg nincs elég hely
{legyen $\Delta = \min_{f \in HA} cr(f)/s(f)$
legyen minden $f \in HA$-ra $cr(f) = cr(f) - \Delta \cdot s(f)$
tegyünk ki olyan lapokat, akikre $cr(f) = 0$ }
tegyük be g-t a HA memóriába, legyen $cr(g) = c(g)$
else (ha benne volt) állítsuk át $cr(g)$ értékét valamelyik $cr(g)$ és $c(g)$ közötti értékre

Az algoritmus enyhén k-versenyképes, de ennél erősebb állítás is igaz. A lapletöltési problémára egy ALG online algoritmust enyhén C-(k, h)versenyképesnek nevezünk, ha van olyan B konstans, hogy $\mathrm{ALG}_k(I) \leq C \cdot \mathrm{OPT}_h(I) + B$ minden inputra teljesül, ahol $\mathrm{ALG}_k(I)$ az algoritmus által elvégzett összes letöltés költsége k méretű memória mellett, $\mathrm{OPT}_h(I)$ pedig a minimális elérhető teljes letöltési összeg h-méretű memória mellett. A HÁZIÚR algoritmusra teljesül a következő állítás.

tétel *A* HÁZIÚR *algoritmus* $k/(k-h+1)$*-(k,h)versenyképes a lapletöltési problémára, ha* $h \leq k$.

Biz Tekintsük kérések egy tetszőleges sorozatát, jelölje ezt az inputot I. A potenciálfüggvény technikát alkalmazzuk. Párhuzamosan hajtjuk végre az

OPT és a HÁZIÚR algoritmusokat, minden kérésnél először az OPT és utána a HÁZIÚR algoritmus lépését hajtjuk végre.

Jelölje az optimális off-line algoritmus memóriájában aktuálisan szereplő lapjainak halmazát OPT. Tekintsük a következő potenciálfüggvényt.

$$\Phi = (h-1)\sum_{f\in HA} cr(f) + k\sum_{f\in OPT}(c(f) - cr(f)) \ .$$

Vizsgáljuk a potenciálfüggvény változásait egy g lap letöltése során!

- OPT elhelyez egy g lapot a memóriájában.

  Ekkor OPT költsége $c(g)$, a potenciálfüggvénynek csak a második része változhat, mivel $cr(g) \geq 0$, ezért a potenciálfüggvény növekedése legfeljebb $k \cdot c(g)$

- HÁZIÚR csökkenti minden $f \in HA$-ra a $cr(f)$ értéket.

  Ekkor minden $f \in HA$ esetén a $cr(f)$ érték csökkenése $\Delta \cdot s(f)$, így összességében Φ a

  $$\Delta((h-1)s(HA) - ks(OPT \cap HA))$$

  értékkel csökken, ahol $s(HA)$ és $s(OPT \cap HA)$ rendre a HA illetve az $OPT \cap HA$ halmazokban levő lapoknak a méreteinek az összege. Abban az időpontban mikor ez a lépés bekövetkezik OPT már elhelyezte a g lapot OPT-ban de a lap még nincs a HA halmazban. Következésképp $s(OPT \cap HA) \leq h - s(g)$. Másrészt ez a lépés azért hajtódik végre, mert nem fért el a g lap a HA memóriában, tehát $s(HA) > k - s(g)$, és így, mivel a lapok méretei egészek, ezért $s(HA) \geq k - s(g) + 1$. Következésképpen azt kapjuk, hogy a Φ potenciálfüggvény csökkenése legalább

  $$\Delta((h-1)(k - s(g) + 1) - k(h - s(g))) \ .$$

  Mivel $s(g) \geq 1$ és $k \geq h$, ezért a fenti érték legalább $\Delta((h-1)(k-1+1) - k(h-1)) = 0$.

- HÁZIÚR kirak egy f lapot a HA memóriából.

  Mivel HÁZIÚR csak akkor rak ki egy f lapot a memóriából, ha arra $cr(f) = 0$, ezért ebben a részben nem változik Φ.

- HÁZIÚR elhelyezi a g lapot a HA memóriában és beállítja a $cr(g) = c(g)$ értéket.

  Ekkor HÁZIÚR költsége $c(g)$. Másrészt g nem volt eddig benne a HA memóriában így $cr(g) = 0$ teljesült. Továbbá elsőként OPT helyezte el a lapot, így $g \in OPT$ teljesül. Következésképpen a Φ függvény értékének csökkenése ebben a lépésben $-(h-1)c(g)+kc(g) = (k-h+1)c(g)$.

- HÁZIÚR átállítja a $g \in HA$ lapra a $cr(g)$ értéket, valamely $cr(g)$ és $c(g)$ közötti értékre.

  Ebben az esetben is fennáll $g \in OPT$, hisz OPT már hamarabb elhelyezte g-t a memóriájába. Mivel $cr(g)$ nem csökkenhet, és $k > h-1$, ezért ebben az esetben Φ nem növekedhet.

Végignéztük az algoritmusok lehetséges lépéseit és a Φ függvény következő tulajdonságai adódtak.

- Ha OPT elhelyez egy lapot a memóriájában, akkor a potenciálfüggvény növekedése legfeljebb k-szor az OPT algoritmusnál fellépő költség.

- Ha HÁZIÚR elhelyez egy lapot a memóriájában, akkor Φ $(k-h+1)$-szer annyival csökken, mint amennyi az algoritmusnál fellépő költség.

- A többi esetben Φ nem növekszik.

A fentiek alapján azt kapjuk, hogy $\Phi_v - \Phi_0 \leq k \cdot \mathrm{OPT}_h(I) - (k-h+1) \cdot \mathrm{HÁZIÚR}_k(I)$, ahol Φ_v és Φ_0 a potenciálfüggvény kezdeti és végső értékei. Mivel a potenciálfüggvény nemnegatív, ezért adódik, hogy $(k-h+1)\mathrm{HÁZIÚR}_k(I) \leq k\mathrm{OPT}_h(I) + \Phi_0$, azaz azt kapjuk, hogy a HÁZIÚR algoritmus enyhén $k/(k-h+1)$-(k,h)versenyképes.

Megjegyezzük, hogy ennél kisebb versenyképességű hányados nem érhető el, még a legegyszerűbb speciális esetben sem, amint azt az alábbi tétel mutatja.

Tétel *Ha minden lap mérete és letöltési költsége 1, akkor nincs olyan online algoritmus, amelynek a (k,h)versenyképességi hányadosa kisebb mint $k/(k-h+1)$.*