Nyugtázás

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.

Azt az algoritmust, amely úgy állítja be az ébresztőt, hogy az első nyugtázatlan csomagtól legyen a teljes késedelem költsége 1 ÉBRESZT algoritmusnak nevezzük. A fenti szabály azt jelenti, hogy az általános definícióban az e_j érték a következő egyenlet megoldása:

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

Tétel Az ÉBRESZT algoritmus 2-versenyképes.

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.

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

Vegyünk egy tetszőleges online algoritmust, jelölje ONL. 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.

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.

ODD a páros sorszámú csomagok után küld nyugtát, EV pedig a páratlan sorszámú csomagok és közvetlenül az utolsó, $2n$-edik csomag után. Ekkor

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

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

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 $\mathrm{ONL}(I_{2n})/\mathrm{OPT}(I_{2n}) \geq 2-1/\mathrm{OPT}(I_{2n})$.

Egy tanuló algoritmus fázisokban dolgozik, minden fázis k nyugtáig tart. Minden fázisban egy ébresztő algoritmust használ, ahol az ébresztőt úgy állítjuk be, hogy az időpontjában a teljes késedelem p legyen, jelölje ezt az algoritmus $\textsc{Ébreszt}_p$.

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 $q \cdot k$ csomagra nézve.

Az optimális p érték meghatározásához az alábbi lemma ad segítséget.

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

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 $\textsc{Ébreszt}_p$ algoritmust az inputon, legyen a nyugták száma k.
  - Ha $k(1+p)<Z$ akkor $p^* := p$ és $Z := k(1+p)$.
- Return p^*

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.

Weblap letöltési feladat

A számítógépek memóriakezelésében a lapozás cserélési problémájának az általánosítása a lapletöltési probléma. 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.

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 $h \leq k$.

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

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

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 $(k-h+1)\mathrm{HÁZIÚR}_k(I) \leq k\mathrm{OPT}_h(I) + \Phi_0$.

Online forgalomirányítás

A hálózatot egy gráf reprezentálja és minden e élnek van egy $u(e)$ maximális felhasználható sávszélessége, az élek számát m-el jelöljük. A feladat az, hogy sorban kérések érkeznek, a j-edik kérés egy $(s_j, t_j, r_j, d_j, b_j)$ vektor, a feladat pedig az s_j pontból a t_j pontba egy kiválasztott úton r_j sávszélességet lefoglalni d_j időtartamra a kérés megjelenésétől kezdve. Amennyiben a kérést elfogadjuk, a nyereség b_j. A továbbiakban mindig feltesszük, hogy $d_j = \infty$ minden j kérés esetén. A feladat on-line, ami azt jelenti, hogy a kérés időpontjában nem tudunk semmit a későbbi kérésekről. Két különböző modellt ismertetünk.

Terhelést kiegyensúlyozó modell: Ebben a modellben minden kérést el kell fogadni és a célfüggvény az élek túltelítettségének - ami a hozzájuk rendelt teljes sávszélességnek és a megengedett maximális felhasználható sávszélességnek a hányadosa - a maximumának a minimalizálása.

Nyereség maximalizáló modell: Ebben a modellben visszautasíthatunk kéréseket, a teljes sávszélesség egyetlen élen sem lehet nagyobb, mint a megengedett maximális sávszélesség, a cél az elfogadott kérésekhez rendelt nyereségek összegének maximalizálása.

Az alábbiakban ismertetjük az exponenciális algoritmust. Az algoritmus pontos megfogalmazásához és elemzéséhez szükségünk lesz az alábbi jelölésekre. Minden elfogadott i kérésre a kéréshez lefoglalt utat P_i jelöli. Legyen A az algoritmus által elfogadott kérések halmaza. Ekkor az $l_e(j) = (\sum_{i \in A, i<j, e \in P_i} r_i)/u(e)$ érték azt adja meg, hogy a j-edik kérés előtt az e-n keresztül lefoglalt sávszélesség a megengedett sávszélességnek mekkora része.

Az exponenciális algoritmusok alapötlete, hogy minden élhez egy $l_e(j)$-ben exponenciális költséget rendelnek, és minimális költségű utat választanak. Az alábbiakban részletesen ismertetjük és elemezzük az exponenciális algoritmust a nyereségmaximalizáló modellre.

EXP algoritmus a nyereség modellre:

Egy j kérés elbírálása:

1. Legyen $c_e(j) = \mu^{l_e(j)}$, ahol μ egy a feladat paramétereitől függő konstans.
2. Vegyük azt a P_j utat s_j és t_j között, amelyre

$$C(P_j) = \sum_{e \in P_j} \frac{r_j}{u(e)} c_e(j)$$

minimális.

3. Ha $C(P_j) \leq 2mb_j$, akkor elfogadjuk a kérést és P_j-n foglaljuk le a sávszélességet, ha nem, akkor visszautasítjuk.

Megjegyzés: Amennyiben az algoritmusból csak az 1 és 2 lépéseket hajtjuk végre és minden kérést elfogadunk, akkor a terhelést kiegyensúlyozó modellre kapjuk meg az exponenciális algoritmust.

Tétel Az EXP algoritmus $1/O(\log\mu)$-versenyképes, ha $\mu = 4mPB$, ahol B felső korlátja a nyereségeknek, továbbá minden kérésre és élre teljesül

$$\frac{1}{P} \leq \frac{r(j)}{u(e)} \leq \frac{1}{\log_2 \mu}.$$