Szegedi Tudományegyetem
Számítógépes Algoritmusok és Mesterséges Intelligencia Tanszék

On-line előrenéző és paraméter tanuló algoritmusok nyugtázási és ütemezési problémákra

Ph.D. értekezés

Németh Tamás

témavezető:

Dr. Imreh Csanád

tanszékvezető egyetemi docens

SZTE TTIK, Informatika Doktori Iskola
Szeged, 2013

Köszönetnyilvánítás

Mindenekelőtt szeretnék köszönetet mondani Dr. Imreh Csanád tanszékvezető egyetemi docensnek, témavezetőmnek, aki nagy segítséget nyújtott a kezdetektől fogva kutatásaim során, mind szakmai, mind kutatás-módszertani kérdésekben. A segítsége nélkül ez a mű nem születhetett volna meg.

Továbbá szeretnék köszönetet mondani Dr. Csendes Tibor professzor úrnak, az Informatika Doktori Program vezetőjének, és Dr. Csirik János professzor úrnak, hogy támogattak a doktori program évei alatt, és azóta is.

Köszönet illeti Nagy Sándor urat és Dr. Kókai Gabriellát†, a Fraunhofer Intézet munkatársait, hogy doktori tanulmányaim alatt, majd később is lehetőséget biztosítottak, hogy részt vegyek az Intézet által végzett kutatásokban, és külön köszönöm Sándornak, a kutatásainkban való aktív részvételt és a sok segítséget. Köszönet illeti továbbá Gyekiczki Balázs tanítványomat, aki a félig on-line paraméter tanuló algoritmusok kutatásában segédkezett.

Köszönöm doktorandusz társaimnak és minden kollégámnak a sok segítő szándékú kritikát, javaslatot és támogatást.

Végül, de nem utolsó sorban szeretnék köszönetet mondani feleségemnek: Németh-Szabados Klára Viktóriának a tanulmányi és eddigi kutatási éveim alatt nyújtott rengeteg erkölcsi támogatásért és kislányomnak: Németh-Szabados Abigélnek, amiért jól tűrte az apa nélkülözését a hétköznapokban és a kutatással töltött hétvégéken is.

Tartalomjegyzék

1. Fejezet

Bevezetés

Disszertációmban a 2006 óta, az on-line klaszterezési és ütemezési problémák területén végzett kutatásaim eredményeit gyűjtöttem össze. Munkatársaimmal ezeken a területeken alapvetően két részterületet vizsgáltunk részletesebben. Az első az on-line klaszterezés témaköre, ezen belül is első sorban az on-line (és félig on-line) nyugtázási feladatot valamint egy, a mérési adatfolyamok valós idejű feldolgozásához definiált klaszterezési feladatot vizsgáltunk. A másik vizsgált témakör a párhuzamos gépes, visszautasításos on-line ütemezés problémaköre. Dolgozatomban számos új elméleti eredmény és sok alkalmazás, valamint az új algoritmusok valós körülmények közötti vizsgálata és azok eredményeinek ismertetése egyaránt megtalálható.

A gyakorlati problémákban gyakran fordulnak elő olyan optimalizálási feladatok, ahol a bemenetet (vagyis a feladatot definiáló számadatot) csak részenként ismerjük meg, és a döntéseinket a már megkapott információ alapján, a további adatok ismerete nélkül kell meghoznunk. Ilyen feladatok esetén on-line problémáról beszélünk. Az on-line algoritmusok elméletének igen sok alkalmazása van a számítástudomány, a közgazdaságtan és az operációkutatás különböző területein. Az első eredmények az on-line algoritmusok elméletének területéről az 1970-es évekből származnak, majd a 90-es évek elejétől kezdve egyre több kutató kezdett el a területhez

kapcsolódó problémák vizsgálatával foglalkozni. Számos részterület alakult ki és napjainkban is a legfontosabb, algoritmusokkal, operációkutatással foglalkozó konferenciákon rendszeresen ismertetnek új eredményeket ezen témakörökből.

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

Az on-line 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 a célfüggvénynek az erre az eloszlásra vonatkozó várható értékét vizsgáljuk. Ezen megközelítés hátránya, hogy általában nincs információnk arról, hogy a lehetséges bemenetek milyen valószínűségi eloszlást követnek.

A másik megközelítés egy legrosszabb eset elemzés, amelyet versenyképességi elemzésnek nevezünk. Ebben az esetben az on-line algoritmus által kapott megoldás célfüggvényértékét hasonlítjuk össze az optimális off-line célfüggvényértékkel. Egy on-line minimalizálási probléma esetén egy on-line algoritmust C-versenyképesnek nevezünk, ha tetszőleges bemenetre teljesül, hogy az algoritmus által kapott megoldás költsége nem nagyobb, mint az optimális off-line költség C-szerese. Egy algoritmus versenyképességi hányadosa a legkisebb olyan C szám, amelyre az algoritmus C-versenyképes.

A továbbiakban egy tetszőleges ALG on-line algoritmusra az I bemeneten felvett célfüggvényértéket $ALG(I)$-vel jelöljük, az off-line célfüggvényértéket pedig $OPT(I)$-vel. Ezt a jelölésrendszert használva a versenyképességet minimalizálási problémákra a következésképpen definiálhatjuk. [7, 13, 26] Az ALG algoritmus C-versenyképes, ha $ALG(I) \leq C \cdot OPT(I)$ teljesül minden I bemenet esetén. Szokás használni a versenyképesség egy további változatát is. Egy minimalizálási probléma esetén az ALG algoritmus enyhén C-versenyképes, ha van olyan B konstans, hogy $ALG(I) \leq C \cdot OPT(I) + B$ teljesül minden I bemenet esetén. Egy algoritmus enyhe versenyképességi hányadosa a legkisebb olyan C szám, amelyre az algoritmus enyhén C-versenyképes. Természetesen igaz, hogy ha egy algoritmus erősen versenyképes valamely C konstanssal, akkor ezzel egyidejűleg ugyanezzel a konstanssal gyengén is 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 $OPT(I) \leq C \cdot ALG(I)$ teljesül minden I bemenet esetén, illetve enyhén C-versenyképes, ha valamely B konstans mellett $OPT(I) \leq C \cdot ALG(I) + B$ teljesül minden I bemenetre.

A versenyképességi elemzésen túl, további, a gyakorlat szempontjából igen fontos hatékonysági vizsgálati módszerek is léteznek. Az egyik ilyen, a gyakorlatban is jól alkalmazható módszer, az algoritmusok viselkedésének, hatékonyságának vizsgálata generált és valós inputokon. Ez a módszer megfelelő mérési módszer alkalmazása esetén az egyébként a valós körülmények között bonyolultsága miatt általában nehezen vagy

egyáltalán nem alkalmazható átlagos eset elemzést helyettesítheti. A módszer lényege, hogy az adott célfüggvényértékeket kiszámítjuk a vizsgált algoritmusok futása során az adott inputokon, és kiszámítjuk rendre az adott inputra az optimális off-line célfüggvényértéket is. Az így kapott értékeket rendre összevetjük az on-line algoritmusok különböző inputon generált célfüggvényértékeivel. Ez a módszer úgy is alkalmazható, ha nem ismerjük az optimális off-line megoldást, ilyenkor a különböző on-line algoritmusok által generált célfüggvény-értékeket hasonlíthatjuk össze.

2. Fejezet

A nyugtázási probléma

A számítógépes hálózatok elmélete az alkalmazások rendkívüli fontosságából adódóan az informatika egyik legjelentősebb kutatási területévé vált. A hálózatok tervezésénél és működtetésük során számos optimalizálási feladat merül fel, mely feladatok többsége lényegében on-line, hiszen sem a forgalom, sem a hálózati topológia és egyéb paramétereinek esetleges változása nem látható előre. A 90-es évek végétől az on-line algoritmusok területén dolgozó szakemberek on-line matematikai modelleket dolgoztak ki a számítógépes hálózatok témakörébe tartozó on-line feladatokra. A következő két fejezetben (és egy keveset az utolsó fejezetben is) ezzel a területtel foglalkozunk.

Az informatikai hálózatokon a kommunikáció során elküldött információ, csomagok formájában továbbítódik. Az esetek többségében azonban a kommunikációs csatorna nem teljesen megbízható (például az alkalmazott számítógépes hálózati infrastruktúrában vezeték nélküli komponensek is előfordulhatnak), ezért az egyes információcsomagok megérkezéséről a címzettnek nyugtát kell küldenie, így lehetővé tehető az elveszett csomagok újraküldése. A TCP implementációk is küldenek nyugtát a csomagok fogadásakor [38]. A nyugtázási probléma tárgyalása során azt próbáljuk meghatározni, hogy melyik időpontban érdemes egy nyugtát küldeni.

Abból indulunk ki, hogy egy nyugta több csomag megérkezését tudja igazolni, azonban ha túl sokáig várunk egy-egy nyugta elküldésével az a csomag újraküldéséhez vezethet, ami többletterhelést ró a hálózatra, míg ha minden csomagot azonnal nyugtázunk akkor maguk a nyugták terhelik feleslegesen a hálózatot.

Az első on-line optimalizációs modell, mely a nyugták küldésének optimális időpontját próbálja meghatározni Dolly, Goldman és Scott fejlesztették ki 1998-ban [14]. Ebben a modellben minden csomagnak van egy beérkezési időpontja és az algoritmusnak valós időben kell döntenie a nyugta küldéséről úgy, hogy a jövőben érkező csomagokról nincs egyéb információja.

2.1. A nyugtázási probléma matematikai modellje

A nyugtázási probléma matematikai modelljében a bemenet az egyes csomagok $a_1, ..., a_n$ érkezési ideje. Az algoritmusnak meg kell határoznia, hogy mikor küld nyugtákat, ezeket az időpontokat $t_1, ..., t_k$ jelöli. A nyugtázási probléma költségfüggvénye:

$$\gamma k + (1 - \gamma) \sum_{j=1}^{k} v_j,$$

ahol k a nyugták száma és v_j a nyugtákhoz tartozó késedelmet jelöli, γ pedig egy, a feladat által meghatározott paraméter. Amennyiben $v_j = \sum_{t_{j-1} < a_i \leq t_j} (t_j - a_i)$, a j-edik nyugta által összegyűjtött teljes késedelem, akkor az f_{sum} modellről beszélünk. Ha $v_j = \max_{t_{j-1} < a_i \leq t_j} (t_j - a_i)$, a

j-edik nyugta által összegyűjtött maximális késedelem, akkor az f_{max} modellről beszélünk. A probléma on-line, azaz egy adott t időpontban csak a t-ig megérkezett csomagok érkezési idejét ismerjük és nincs semmi információnk a további csomagokról.

2.2. Az optimális off-line algoritmus és az ébresztő algoritmus

Az off-line algoritmus

Az f_{sum} modellben az optimális off-line célfüggvény értéket az alábbi algoritmus szolgáltatja [14].

```
1:    M_min[0] ← 0
2:    M[1,1] ← γ + a_1(1 − γ)
3:    M_min[1] ← M[1,1]
4:    M_pt[1] ← 1
5:    minden i ∈ [2,n]-re
6:        M_min[i] ← ∞
7:        minden j ∈ [1,i]-re
8:            M[i,j] ← γ + (1 − γ) · j · a_i + M_min[i − j]
9:            Ha M[i,j] < M_min[i], akkor
10:               M_min[i] ← M[i,j]
11:               M_pt[i] ← j
```

$$f'_{sum} = M_{min}[n] = min_{l \in [1,n]} M[n,l]$$

$$f_{sum} = f'_{sum} - (1-\gamma)\sum_{i=1}^{n} a_i$$

Az off-line algoritmust valós körülmények között nem tudjuk használni, hiszen nincs információnk előre az egyes csomagok érkezési időpontjairól. A tesztek végrehajtásához azonban szükséges a célfüggvény optimális értékét is kiszámolnunk, hogy az on-line algoritmus által elért célfüggvény értéket össze tudjuk vetni a célfüggvény optimális off-line értékével. Ezt a gyakorlatban úgy lehet kivitelezni, hogy a tesztek futása során az on-line érkező adatcsomagok érkezési időpontjait mentjük egy fájlba vagy adatbázisba, és később ezekre az értékekre számítjuk ki, hogy mi lett volna az optimális off-line célfüggvény érték.

Az on-line ébresztő algoritmus

Az on-line probléma megoldására az ébresztő beállításán alapuló algoritmusokat [14] 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 ébresztő algoritmusnak nevezzük. A fenti szabály azt jelenti, hogy az általános definícióban az e_j érték a következő egyenlet megoldása:

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

ahol σ_j az a_j csomag érkezése után meglevő még nyugtázatlan csomagok halmaza.

Az ébresztő algoritmusra az f_{sum} modellben a következő tulajdonságok teljesülnek. Az alábbiakban a bizonyítások alapötletét is bemutatjuk a speciális $\gamma = 1/2$ esetre, mivel a későbbiekben ezeket a bizonyításokat fogjuk kiterjeszteni az általánosabb modellre.

1. Prepozíció. *([14]) Az f_{sum} modellben az ébresztő algoritmus versenyképességi hányadosa 2.*

Bizonyítás: Vegyünk egy tetszőleges inputot és tegyük fel, hogy az ébresztő 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 $OPT(I) \geq k$ és adódik, hogy az algoritmus valóban 2-versenyképes. Amennyiben $k^* < k$, akkor az ébresztő 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 $OPT(I) \geq k$ adódik, amely egyenlőtlenségből következik, hogy az algoritmus 2-versenyképes. ■

2. Prepozíció. *([14]) A nyugtázási problémára az f_{sum} modellben nem létezik olyan on-line algoritmus melynek 2-nél kisebb a versenyképességi hányadosa.*

Bizonyítás: Vegyünk egy tetszőleges on-line algoritmust, jelölje ONL. Tekintsük a következő bemenetet. Vegyük csomagok egy hosszú sorozatát, amelyet úgy kapunk, hogy minden esetben, amikor ONL egy nyugtát küld azonnal egy új csomag érkezik, de addig nem jött új csomag. 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 on-line algoritmus költsége $ONL(I_{2n}) = 2n + t_{2n}$, 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

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

és

$$ODD(I_{2n}) = n + \sum_{i=0}^{n-1}(t_{2i} - t_{2i-1}).$$

Innen következik, hogy $EV(I_{2n}) + ODD(I_{2n}) = ONL(I_{2n}) + 1$. Másrészt az optimális off-line algoritmusra

$$OPT(I_{2n}) \leq min\{EV(I_{2n}), ODD(I_{2n})\},$$

így azt kapjuk, hogy

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

Ezen egyenlőtlenség alapján adódik, hogy ONL nem lehet jobb, mint

2-versenyképes, hiszen a csomagok egy elegendően hosszú sorozatát véve az $OPT(I_{2n})$ érték tetszőlegesen nagy lehet. ■

Hasonló módon igazolhatók az f_{max} modellre az alábbi eredmények.

3. Prepozíció. *([14]) Az f_{max} modellben az ébresztő algoritmus versenyképességi hányadosa 2.*

4. Prepozíció. *([14]) A nyugtázási problémára az f_{max} modellben nem létezik olyan on-line algoritmus melynek 2-nél kisebb a versenyképességi hányadosa.*

2.2.1. További eredmények a nyugtázási problémához

Számos további eredményt publikáltak a nyugtázás területéről. A probléma off-line verziójában az f_{sum} célfüggvényt vizsgálták [34] ahol egy gyors, lineáris futási idejű algoritmust mutattak be. Véletlenített on-line algoritmusokat is vizsgáltak [29, 35] a területen, továbbá publikáltak [29] egy $e/(e-1)$-versenyképes algoritmust is a probléma megoldására. Bizonyították, hogy nem létezik olyan on-line véletlenített algoritmus [35] a probléma megoldására, aminek $e/(e-1)$-nél kisebb a versenyképességi hányadosa.

Néhány további célfüggvényt is vizsgáltak [1] és [22].

Az [1] cikkben olyan matematikai modellt mutattak be, melyben a célfüggvény értéke a nyugták számának és a nyugták által összegyűjtött maximális késedelemnek az összege. Az általános modellben a maximális

késedelem p-edik hatványát vizsgálták. Mindkét esetre optimális on-line algoritmust mutattak be. A $p = 1$ esetben az algoritmus versenyképességi hányadosa $\pi^2/6$, az általános esethez a versenyképességi arányt egy Riemann zéta függvény adja meg, mely 1.5-höz tart, ha p tart végtelenbe. A cikk a véletlenített algoritmusokkal elérhető versenyképességi hányados alsó korlátját is bizonyítja. További célfüggvényt is bemutattak [22] az $f_{\max}$ általánosítására, mely szintén használja a nyugták küldése közötti időtartamok felső korlátját. Ebből a cikkből egy optimális $(1+\sqrt{5})/2$-versenyképes determinisztikus, és szintén optimális $(\sqrt{3}+1)/2$-versenyképes véletlenített algoritmust is megismerhetünk, továbbá bemutatnak az algoritmusok egy olyan osztályát, mely algoritmusok számára csak korlátozott számú véletlen bit használata megengedett. A nyugtázási probléma egy még általánosabb modelljét [17] is megismerhetjük, mely on-line probléma motivációja a híváskezelő rendszerek működtetéséből származik, ahol a két ügyfél kezelésének esete a nyugtázási probléma egy általánosítása.

2.3. Előrenéző változatok versenyképességi elemzése

A félig on-line algoritmusok néhány előretekintő tulajdonságot is figyelembe vesznek. A legtöbb alkalmazásban az algoritmusok on-line-ok, azaz a további csomagokat illetően nem rendelkeznek előretekintő információval. Ennek ellenére az előretekintő algoritmusok vizsgálata is fontos probléma [14]. Az ilyen algoritmusok vizsgálata annak a megértésében is segít, hogy hogyan használhatják az egyes tanuló algoritmusok a további csomagok érkezésének becslését. Az f_{max} célfüggvény esetén ha

tudjuk a következő csomag érkezési idejét, úgy 1 versenyképes algoritmus adható, viszont az f_{sum} célfüggvény esetében, a következő k csomag érkezési idejének ismerete sem elég (bármely k konstansra), hogy 2-nél jobb versenyképességi hányadost kapjunk.

Vizsgáljunk meg egy másik előretekintő tulajdonságot! Tegyük fel, hogy az algoritmus az érkezési időket nem a következő k csomagra, hanem a következő c hosszúságú időintervallumra ismeri. Az ilyen típusú előretekintő tulajdonságot idő előretekintő tulajdonságnak nevezzük, ezt vizsgálták például on-line jármű útvonaltervező problémára [3] is.

Az on-line probléma t időpontjában az algoritmus ismeri a már megérkezett csomagok érkezési idejét, de nincs információja az érkezési időkről a további csomagokat illetően. Az alábbiakban bemutatunk egy idő-előretekintő tulajdonságú félig on-line modellt, ahol t időpontban a az algoritmus ismeri a már megérkezett csomagok érkezési idejét, továbbá a $(t, t + c]$ időintervallumban érkező csomagok érkezési idejét is. A továbbiakban σ_i-vel jelöljük az a_i időpontban még nyugtázatlan csomagok halmazát. A csomagok tetszőleges L sorozatára és egy A algoritmusra, $A(L)$ az algoritmus által küldött nyugták összes költségét jelöli, az optimális költséget L listára $OPT(L)$ jelöli. Az algoritmusok értékelésére versenyképességi elemzést alkalmazzuk.

2.3.1. Az f_{max} célfüggvény

Ahogy azt a fentiekben definiáltuk az f_{max} célfüggvényt úgy kapjuk, ha $L_j = max_{t_{j-1}<a_i\leq t_j}(t_j - a_j)$, azaz a j. nyugta által összegyűjtött maximális

késedelem. Így a célfüggvényünk:

$$\gamma k + (1-\gamma)\sum_{j=1}^{k} max_{t_{j-1}<a_i\leq t_j}(t_j - a_j).$$

Az algoritmus

Az f_{max} célfüggvény alkalmazása esetén optimális megoldást kapunk, ha a a_j időpontbeli nyugtát akkor és csak akkor küldjük, ha $a_{j+1} - a_j \geq \gamma/(1-\gamma)$. Így két esetet kell megvizsgálnunk, c értékétől függően. Ha $c \geq \gamma/(1-\gamma)$, akkor könnyen megadhatunk egy 1-versenyképes algoritmust is. Ekkor ugyanis az előretekintési intervallum mérete elég nagy ahhoz, hogy eldöntsük, az $a_{j+1} - a_j \geq \gamma/(1-\gamma)$ egyenlőtlenség igaz, vagy hamis, tehát az előrenéző algoritmusunk az optimális megoldást adja. Sokkal érdekesebb, amikor $c < \gamma/(1-\gamma)$. Ebben az esetben az ébresztő algoritmus egy kiterjesztett verzióját használjuk, amit a továbbiakban TLA algoritmusnak nevezünk (Time Lookhead Alarming algorithm). Az algoritmus a következőképpen működik. Az a_j időpontban beállítjuk az ébresztőt $a_j + \gamma/(1-\gamma) - c$ időpontra. Ha az a_{j+1} csomag az ébresztési időpont előtt megérkezik vagy ha az a_{j+1} csomagot az $a_j + \gamma/(1-\gamma) - c$ időpontban az $(a_j + \gamma/(1-\gamma) - c, a_j + \gamma/(1-\gamma)]$ intervallumban látjuk (előre) akkor átállítjuk az ébresztőt az $a_{j+1} + \gamma/(1-\gamma) - c$ időpontra. Ellenkező esetben, azaz ha nem érkezik meg a következő csomag az $(a_j, a_j + \gamma/(1-\gamma)]$ időintervallumban, akkor nyugtát küldünk az $a_j + \gamma/(1-\gamma) - c$ időpontban, mely az összes még nyugtázatlan csomagot nyugtázza.

1. Tétel. *([27]) A TLA algoritmus* $max\{1, 2 - \frac{1-\gamma}{\gamma}c\}$*-versenyképes.*

Bizonyítás: Először azt mutatjuk meg, hogy az algoritmus $2 - \frac{1-\gamma}{\gamma}c$ -versenyképes. Vegyünk egy tetszőleges $a_1, ..., a_n$ inputot. Particionáljuk az input sorozatot részsorozatokra (fázisokra) a következőképpen: Legyen $S_1 = \{a_1, \ldots, a_{k(1)}\}$ ahol $k(1)$ az első index melyre $a_{k(1)+1} - a_{k(1)} \geq \gamma/(1-\gamma)$. A többi fázist is hasonló módon definiáljuk, azaz $S_{j+1} = \{a_{k(j)+1}, \ldots, a_{k(j+1)}\}$, ahol $k(j+1)$ az első index a $k(j)$ után, melyre $a_{k(j+1)+1} - a_{k(j+1)} \geq \gamma/(1-\gamma)$. Az utolsó fázis az utolsó csomaggal zárul, továbbá legyen k(0)=0.

Ekkor az optimális off-line algoritmus minden fázis utolsó csomagjánál küld egy-egy nyugtát, ezért az algoritmus összes költsége a j. nyugtánál

$$OPT(S_j) = \gamma + (1-\gamma)(a_{k(j)} - a_{k(j-1)+1}).$$

A TLA algoritmus viszont $a_{k(j)} + \gamma/(1-\gamma) - c$ időpontban küldi a nyugtát, így a j. csomagnál az összes költség:

$$TLA(S_j) = \gamma + (1-\gamma)(a_{k(j)} + \gamma/(1-\gamma) - c - a_{k(j-1)+1}).$$

Mivel $TLA(S_j)/OPT(S_j) > 1$, ezért ha a számlálót és a nevezőt is csökkentjük ugyanazzal a konstanssal, akkor a hányados értéke növekszik, így azt kapjuk, hogy

$$\frac{TLA(S_j)}{OPT(S_j)} \leq \frac{\gamma + (1-\gamma)(\gamma/(1-\gamma) - c)}{\gamma} = 2 - \frac{1-\gamma}{\gamma}c.$$

A teljes költség viszont a partíciók költségeinek összege, így TLA $2 - \frac{1-\gamma}{\gamma}c$ -versenyképes. ■

Alsó korlát

A TLA algoritmus optimális, azaz nem létezik olyan algoritmus, amelynek kisebb a versenyképességi hányadosa.

2. Tétel. *Nem létezik olyan c-előrenéző félig on-line algoritmus a nyugtázási problémára az f_{max} modellben, amelynek kisebb a versenyképességi hányadosa, mint* $\max\{1, 2 - \frac{1-\gamma}{\gamma}c\}$.

Bizonyítás: Ha $c \geq \gamma/(1-\gamma)$ akkor az állítás nyilvánvalóan igaz, így feltételezhetjük, hogy $c < \gamma/(1-\gamma)$. Tekintsünk egy tetszőleges A online algoritmust. A következő inputot definiáljuk, jelöljük I_n-el. Az első csomag érkezési ideje $a_1 = 0$. Az i. csomag $(i = 2, \ldots, n)$ az $i-1$. csomag nyugtája után c időegység múlva érkezik $(a_i = t_{i-1} + c)$.

Partícionáljuk az inputot a 2. tételnél látottakhoz hasonlóan. Az egyes fázisokat $S_1, \ldots, S_j$-vel jelöljük. Most tekintsük S_i fázist. Az egyszerűség kedvéért $k(i) - k(i-1)$ értéket r_i-vel jelöljük. Az A algoritmus minden csomagot nyugtáz, így azt kapjuk, hogy

$$A(S_i) = \gamma r_i + (1-\gamma) \sum_{p=k(i-1)+1}^{k(i)} (t_p - a_p).$$

Az optimális megoldás ellenben mindössze egyetlen nyugtát küld, az $a_{k(i)}$ időpontban, így

$$OPT(S_i) = \gamma + (1-\gamma)\Big(\sum_{p=k(i-1)+1}^{k(i)-1} (t_p - a_p) + c(r_i - 1)\Big).$$

Most tegyük fel, hogy $i < j$. Ha kiszámoljuk az $A(S_i) - (2 - \frac{1-\gamma}{\gamma}c)OPT(S_i)$ értéket és felhasználjuk, hogy $t_p - a_p \leq \gamma/(1-\gamma) - c$, minden $p = k(i-1)+1, \ldots, k(i)-1$ és azt, hogy $t_{k(i)} - a_{k(i)} > \gamma/(1-\gamma) - c$, akkor azt kapjuk, hogy

$$A(S_i) - (2 - \frac{1-\gamma}{\gamma}c)OPT(S_i) \geq \gamma(r_i - 2 + \frac{1-\gamma}{\gamma}c) + (1-\gamma)$$

$$\Big(\big(\frac{1-\gamma}{\gamma}c - 2\big)(r_i - 1)c + (r_i - 1)\big(\frac{1-\gamma}{\gamma}c - 1\big)\big(\frac{\gamma}{1-\gamma} - c\big) + \frac{\gamma}{1-\gamma} - c \Big)$$

$$= \gamma(r_i - 2 + \frac{1-\gamma}{\gamma}c) - (1-\gamma)(r_i - 1)c + (r_i - 1)((1-\gamma)c - \gamma) + \gamma - c(1-\gamma) = 0.$$

Így bizonyítottuk, hogy $A(S_i) \geq (2 - \frac{1-\gamma}{\gamma}c)OPT(S_i)$ ha $i < j$. Az S_j esetében egyetlen különbség van. Ekkor a $t_{k(j)} - a_{k(j)} > \gamma/(1-\gamma) - c$ egyenlőtlenség nem áll fenn, viszont ugyanígy beláthatjuk, hogy

$$A(S_j) + \gamma/(1-\gamma) - c \geq (2 - \frac{1-\gamma}{\gamma}c)OPT(S_j).$$

Mivel a teljes költség az egyes fázisokon számított költségek összege, így $A(I_n) + \gamma/(1-\gamma) - c \geq (2 - \frac{1-\gamma}{\gamma}c)OPT(I_n)$. Továbbá ha n tart végtelenbe $OPT(I_n)$ is a végtelenbe tart, így a fenti egyenlőtlenség szerint $A(I_n)/OPT(I_n)$ hányados $2 - \frac{1-\gamma}{\gamma}c$-hez tart, tehát A versenyképességi hányadosa nem lehet kisebb mint $2 - \frac{1-\gamma}{\gamma}c$. ■

2.3.2. Az f_{sum} célfüggvény

Mint azt a fentiekben definiáltuk, az f_{sum} célfüggvényt úgy kapjuk, ha $L_j = \sum_{t_{j-1} < a_i \leq t_j}(t_j - a_j)$, azaz a j. nyugta által összegyűjtött összes késedelem. Így a célfüggvényünk:

$$\gamma k + (1-\gamma)\sum_{j=1}^{k}\sum_{t_{j-1} < a_i \leq t_j}(t_j - a_j).$$

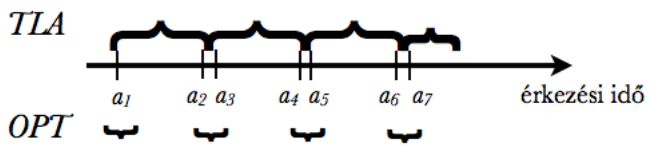
A TLA algoritmus

Ebben a részben is kézenfekvő ötlet az ébresztő algoritmus [14] idő-előretekintő kiterjesztését használni. Most a következőképpen definiáljuk

az ébresztő algoritmus idő-előretekintő változatát. Az a_j csomag megérkezésének pillanatában beállítjuk az ébresztőt $a_j + e_j$ időpontra, ahol $e_j = (\gamma/(1-\gamma) - \sum_{a_i \in \sigma_j}(a_j - a_i))/|\sigma_j|$. Ha a következő csomag (a_{j+1}) a $\max\{a_j, a_j + e_j - c\}$ időpont előtt megérkezik vagy látjuk a_{j+1} csomagot az előretekintő időintervallumban, akkor lépjünk következő csomaghoz és állítsuk be újra az ébresztőt a fent leírt módon. Ellenkező esetben (nem érkezik csomag az $(a_j, a_j + e_j]$) intervallumban, küldünk egy nyugtát a $\max\{a_j, a_j + e_j - c\}$ időpontban mely az összes még nyugtázatlan csomagot nyugtázza. Sajnos a fenti algoritmus nem ér el kisebb versenyképességi hányadost mint 2.

3. Tétel. *Tetszőleges c-re a TLA algoritmus versenyképességi hányadosa* 2.

Bizonyítás: Az algoritmus versenyképességi hányadosa nem nagyobb mint az ébresztő algoritmus versenyképességi hányadosa [14], tehát a TLA algoritmus versenyképesési hányadosa nem nagyobb mint 2, mivel az ébresztő algoritmus 2 versenyképes. Viszont nincs kisebb versenyképességi hányadosa sem, amit az alábbi inputon láthatunk.



1. ábra. A TLA és OPT viselkedése

Legyen $I_n = \{a_1, \ldots, a_{2n+1}\}$ ahol $a_1 = 0$ és $a_{2i} = i\gamma/(1-\gamma) + (i-1)\varepsilon$, $a_{2i+1} = i\gamma/(1-\gamma) + i\varepsilon$, $i = 1, \ldots, n$. Ekkor TLA a következő időpontokban küldi a nyugtákat: $a_2, \ldots, a_{2n}, \max\{a_{2n+1}, a_{2n+1} + \gamma/(1-\gamma) - c\}$. Így $TLA(I_n) = (n+1)\gamma + n\gamma + (1-\gamma)\max\{0, \gamma/(1-\gamma) - c\}$. Egy optimális off-line algoritmus a_1, a_3, a_{2n+1} időpontokban küldi a nyugtákat, ennek költsége $OPT(I_n) = (n+1)\gamma + (1-\gamma)n\varepsilon$. A versenyképességi hányados így 2-höz tart, ha ε tart a 0-hoz és n tart végtelenbe, ebből következik, hogy a versenyképességi hányados legalább 2. ■

A LIP algoritmus

Ha $c > \gamma/(1-\gamma)$ 2-nél kisebb versenyképességű algoritmust is adhatunk. A LIP algoritmus (Lookahead Interval Planning algorithm) a következő. Az algoritmus blokkokra bontja az inputot, majd minden blokkra meghatározza a nyugták küldésének időpontját az optimális off-line algoritmus segítségével. Egy-egy blokk mindig az első nyugtázatlan csomaggal kezdődik. Az algoritmus először meghatározza, hogy van-e 2 egymást követő csomag a c hosszúságú előretekintő intervallumban melyre $a_{i+1} - a_i > \gamma/(1-\gamma)$. Ha van ilyen pár, a blokkot lezárjuk az első ilyen a_i időpontnál, egyébként a blokk intervallumának hossza c. Ez után az algoritmus meghatározza a blokkra a nyugták küldésének időpontját az optimális megoldáshoz az off-line probléma optimális megoldásával [14], majd elküldi a nyugtákat a megoldás szerint és áttér a következő blokkra.

4. Tétel. *A LIP algoritmus* $1 + \frac{\gamma}{(1-\gamma)c}$*-versenyképes.*

Bizonyítás: A fenti tétel belátásához vegyünk egy tetszőleges I inputot! Bontsuk blokkokra az inputot úgy, ahogy azt az algoritmus leírásánál

megadtuk. Figyeljük meg, hogy létezik egy optimális off-line algoritmus mely minden blokkban küld egy nyugtát a blokk utolsó csomagjánál. (Mert ha a blokk utolsó csomagja késik, akkor a késés költsége több mint $(1-\gamma)(\gamma/(1-\gamma)) = \gamma$, és a nyugtázás költsége pontosan γ.) Továbbá vegyük észre, hogy az utolsó csomag mindig egy blokk utolsó csomagja egyben és LIP is küld nyugtát a blokk utolsó csomagjánál.

Tekintsünk egy tetszőleges fázist, S_i-t. Jelöljük r-rel a fázisban található blokkok számát. Vegyük a fázison az optimális off-line megoldást. Ha hozzáveszünk $r-1$ további nyugtát az első $r-1$ blokk végéhez, olyan megoldást kapunk, amely a blokkokat külön-külön nyugtázza. De LIP a legjobb ilyen megoldást adja, ezért azt kapjuk, hogy $(r-1)\gamma + OPT(S_i) \geq LIP(S_i)$, tehát $LIP(S_i)/OPT(S_i) \leq 1 + (r-1)\gamma/OPT(S_i)$.

Határozzuk meg $OPT(S_i)$ értékét. Mivel minden blokk azonos fázisban van és minden blokk hossza legalább c, a fázis hossza legalább $(r-1)c$. Tegyük fel, hogy az optimális off-line algoritmus k nyugtát küld ebben a fázisban. Ekkor az első $k-1$ nyugta után van egy legfeljebb $\gamma/(1-\gamma)$ hosszú időintervallum mely idő alatt nem érkezik csomag. Ebből következik, hogy a teljes optimális költség legalább $(1-\gamma)((r-1)c-(k-1)\gamma/(1-\gamma))$. Így $OPT(S_i) \geq k\gamma + (1-\gamma)((r-1)c-(k-1)\gamma/(1-\gamma)) = (1-\gamma)(r-1)c+\gamma$. Ezt használva kapjuk, hogy $LIP(S_i)/OPT(S_i) \leq 1 + \frac{\gamma}{(1-\gamma)c}$. ■

Alsó korlátok

Először adunk egy alsó korlátot a versenyképességi hányados nagyságrendjére. Ez megmutatja, hogy az f_{sum} célfüggvény esetében nem

létezik olyan konstans méretű előretekintés, amely segítségével elérhető az 1-versenyképesség.

5. Tétel. *Nem létezik olyan c-előrenéző félig on-line algoritmus a nyugtázási problémára az f_{sum} modellben, amelynek kisebb a versenyképességi hányadosa, mint* $1 + \Omega(1/c^2)$.

Bizonyítás:

A számítások egyszerűsítése érdekében feltesszük, hogy $c = \gamma(k - 1/4)/(1 - \gamma)$, ahol $k \geq 1$ egész. Ezt feltehetjük az általánosság megszorítása nélkül, mert egy nagyobb előretekintés nem tudja csökkenteni a versenyképességi hányadost. Tekintsünk egy tetszőleges c-előrenéző A algoritmust. Legyen $x = \frac{\gamma(2k+1)}{4k(1-\gamma)}$ és $y = \gamma/2(1 - \gamma)$. A következő sorozatokat definiáljuk minden $j = 1, \ldots, k$-ra.

- $S_{xjx} = \{a_1, a_2, \ldots, a_{2j}\}$, ahol $a_{2i-1} = (i-1)x + (i-1)y$ és $a_{2i} = ix + (i-1)y$, $i = 1, \ldots, j$. Vegyük észre, hogy $a_{2k} = c$ az S_{xkx} sorozatban.

- $S_{xjy} = \{a_1, a_2, \ldots, a_{2j+1}\}$, ahol $a_{2i-1} = (i-1)x + (i-1)y$, $i = 1, \ldots, j+1$ és $a_{2i} = ix + (i-1)y$, $i = 1, \ldots, j$.

- $S_{yjy} = \{a_1, a_2, \ldots, a_{2j}\}$, ahol $a_{2i-1} = (i-1)y + (i-1)x$ és $a_{2i} = iy + (i-1)x$, $i = 1, \ldots, j$.

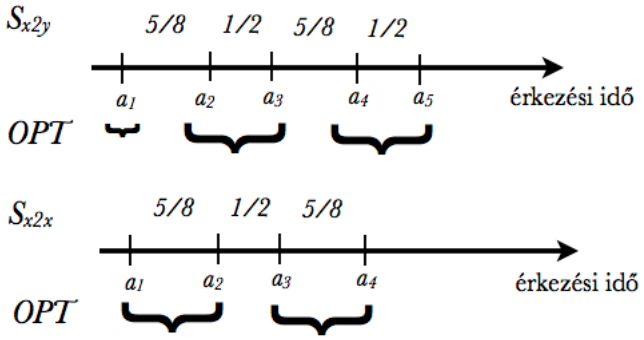
- $S_{yjx} = \{a_1, a_2, \ldots, a_{2j+1}\}$, ahol $a_{2i-1} = (i-1)y + (i-1)x$, $i = 1, \ldots, j+1$ és $a_{2i} = iy + (i-1)x$, $i = 1, \ldots, j$.

A $\gamma + (1-\gamma)(2y + x) = 2\gamma + (1-\gamma)x$ egyenletből azt kapjuk, hogy van olyan optimális megoldás a fent leírt sorozatokra, mely soha nem nyugtáz

2 csomagnál többet egyetlen nyugtával. Ezt a megfigyelést felhasználva igazoljuk az alábbi lemmát.

1. Lemma. *Minden j-re* $(1 \leq j \leq k)$*:* $OPT(S_{xjy}) = OPT(S_{yjx}) = \gamma(j+1) + (1-\gamma)jy$, $OPT(S_{xjx}) = \gamma j + (1-\gamma)jx$, $OPT(S_{yjy}) = \gamma j + (1-\gamma)jy$.

Bizonyítás: Az állítást teljes indukcióval igazoljuk. Elsőként vegyük a $k = 1$ esetet. Az S_{x1y} sorozatban az érkezési idők $0, x, x+y$, az S_{x1y} sorozatban az érkezési idők $0, y, x+y$ és az optimális megoldás mindkét esetben két nyugtát küld, S_{x1y} esetén a 0 és az $x+y$ időpontokban, S_{x1y} esetén pedig az y és az $x+y$ időpontokban. Így a késedelem mindkét esetben y. Az S_{x1x} és S_{y1y} sorozatok esetén mindkét esetben két csomag van, és az optimális megoldás egyben nyugtázza őket a második csomag érkezésekor. Most tegyük fel, hogy $i \leq k$ és a lemma teljesül minden $j < i$-re. Igazoljuk az állítást i-re is.



2. ábra. Az S_{x2y} és S_{x2x} inputok és optimális megoldásai $\gamma = 1/2$ esetén

Vegyük elsőként az S_{xix} sorozatot. Mint a fentiekben észrevettük van olyan optimális megoldás, ami soha nem nyugtáz két csomagnál többet, így két lehetőséget kell megvizsgálnunk az a_1 csomag nyugtázására vonatkozóan. Ha az a_1 csomagot az a_1 időpontban nyugtázzuk, akkor még marad egy $S_{y(i-1)x}$ sorozatunk, amit az indukciós feltevés alapján $i\gamma+(1-\gamma)(i-1)y$ költséggel tudunk nyugtázni. Tehát ebben az esetben a teljes költség $(i+1)\gamma+(1-\gamma)(i-1)y$. Ha az a_1 csomagot a_2-vel együtt az $a_2 = x$ időpontban nyugtázzuk, akkor marad még egy $S_{x(i-1)x}$ sorozatunk, amit az indukciós feltevés alapján $(i-1)\gamma + (1-\gamma)(i-1)x$ költséggel tudunk nyugtázni. Tehát ebben az esetben a teljes költség $i\gamma + i(1-\gamma)x$. Behelyettesítve x és y értékét adódik, hogy a második esetben kapunk kisebb költséget, így az állítást ezen részét igazoltuk i-re is.

Teljesen hasonlóan kezelhető az S_{yiy} sorozat. Ha az a_1 csomagot az a_1 időpontban nyugtázzuk, akkor még marad egy $S_{x(i-1)y}$ sorozatunk, amit az indukciós feltevés alapján $i\gamma + (1-\gamma)(i-1)y$ költséggel tudunk nyugtázni. Tehát ebben az esetben a teljes költség $(i+1)\gamma+(1-\gamma)(i-1)y$. Ha az a_1 csomagot a_2-vel együtt az $a_2 = y$ időpontban nyugtázzuk, akkor marad még egy $S_{y(i-1)y}$ sorozatunk, amit az indukciós feltevés alapján $(i-1)\gamma + (1-\gamma)(i-1)y$ költséggel tudunk nyugtázni. Tehát ebben az esetben a teljes költség $i\gamma + i(1-\gamma)y$. Behelyettesítve x és y értékét adódik, hogy a második esetben kapunk kisebb költséget, így az állítást ezen részét igazoltuk i-re is.

Vizsgáljuk most meg az S_{xiy} sorozatot. Ha az a_1 csomagot az a_1 időpontban nyugtázzuk, akkor még marad egy S_{yiy} sorozatunk, aminek az optimális nyugtázási költsége a fentiek alapján $i\gamma + i(1-\gamma)y$. Így a

teljes költség ebben az esetben $(i+1)\gamma + i(1-\gamma)y$. Ha az a_1 csomagot a_2-vel együtt az $a_2 = x$ időpontban nyugtázzuk, akkor marad még egy $S_{x(i-1)y}$ sorozatunk, amit az indukciós feltevés alapján $i\gamma + (1-\gamma)(i-1)y$ költséggel tudunk nyugtázni. Tehát ebben az esetben a teljes költség $(i+1)\gamma + (1-\gamma)(i-1)y + x$. Mivel $x > y$ ezért az első esetben kapunk kisebb költséget, így az állítást ezen részét is igazoltuk i-re is.

Végül vizsgáljuk meg az S_{yix} sorozatot. Ha az a_1 csomagot az a_1 időpontban nyugtázzuk, akkor még marad egy S_{xix} sorozatunk, aminek az optimális nyugtázási költsége a fentiek alapján $i\gamma + i(1-\gamma)x$. Így a teljes költség ebben az esetben $(i+1)\gamma + i(1-\gamma)x$. Ha az a_1 csomagot a_2-vel együtt az $a_2 = y$ időpontban nyugtázzuk, akkor marad még egy $S_{y(i-1)x}$ sorozatunk, amit az indukciós feltevés alapján $i\gamma + (1-\gamma)(i-1)y$ költséggel tudunk nyugtázni. Tehát ebben az esetben a teljes költség $(i+1)\gamma + i(1-\gamma)y$. Mivel $x > y$ ezért a második esetben kapunk kisebb költséget, így az állítást ezen részét is igazoltuk i-re is. ■

Legyen S_{xkx} az A algoritmus inputjának első része, majd várjunk y időpontig. Tegyük fel, hogy az algoritmus nyugtát küld egy $z \leq y$ időpontban. Ekkor nyugtázza a_1 csomagot, majd nyugtáznia kell még az $S_{y(k-1)x}$ sorozatba eső csomagokat is. Ezért az 1. lemából következően kapjuk, hogy $A(S_{xkx}) \geq \gamma + (1-\gamma)z + k\gamma + (1-\gamma)(k-1)y$. Tehát

$$\frac{A(S_{xkx})}{OPT(S_{xkx})} \geq \frac{(k+1)\gamma + (1-\gamma)(k-1)y}{k\gamma + k(1-\gamma)x} = 1 + \frac{1}{6k+1}.$$

Most tegyük fel, hogy A nem küld nyugtát az y időpont előtt. Ekkor az $y + c$ időpontban egy további csomag érkezik, így az input S_{xky}. Az

algoritmus észreveszi az új csomagot az y időpontban. Ha nyugtázza az első csomagot az x időpont előtt, akkor $A(S_{xky}) \geq \gamma+(1-\gamma)y+OPT(S_{yky})$. Így a lemmából következően az alábbi egyenlőtlenséget kapjuk:

$$\frac{A(S_{xky})}{OPT(S_{xky})} \geq \frac{(k+1)\gamma+(1-\gamma)(k+1)y}{(k+1)\gamma+(1-\gamma)ky} = 1+\frac{1}{3k+2}$$

Végül tegyük fel, hogy A nem küld nyugtát x időpont előtt. Ha az első nyugtát az x időpont után küldi, akkor a teljes költség növekszik, így feltehetjük, hogy az első nyugtát x időpontban küldi. Ha az algoritmus az első két csomagot nyugtázza és a megmaradt rész $S_{x(k-1)y}$, azt kapjuk, hogy $A(S_{xky}) \geq \gamma+(1-\gamma)x+OPT(S_{x(k-1)y})$. Így a lemmából következően az alábbi egyenlőtlenséget kapjuk:

$$\frac{A(S_{xky})}{OPT(S_{xky})} \geq \frac{(k+1)\gamma+(1-\gamma)(x+(k-1)y)}{(k+1)\gamma+(1-\gamma)ky} \geq 1+\frac{1}{6k^2+4k}$$

Mivel az összes lehetséges esetet megvizsgáltuk, a tételt bizonyítottuk.

■

A fenti eredmény azonban nem ad 1-nél jobb alsó korlátot arra az esetre, ha az algoritmus csak kicsi előretekintéssel rendelkezik. Mutatunk még egy alsó korlátot arra az esetre amikor $c \leq \gamma/(1-\gamma)$ annak a technikának [14] a kiterjesztésével, melyet a 2. fejezetben már olvashattunk.

6. Tétel. *Nem létezik olyan c-előrenéző félig on-line algoritmus a nyugtázási problémára az f_{sum} modellben, amelynek kisebb a versenyképességi hányadosa, mint $2\gamma/(c(1-\gamma)+\gamma)$, ha $c \leq \gamma/(1-\gamma)$.*

Bizonyítás: Tekintsünk egy tetszőleges on-line algoritmust, jelöljük A-val. Elemezzük a következő inputot. Vegyük a csomagok egy hosszú sorozatát

melyben a csomagok rendre pontosan c időegységgel azután érkeznek, amikor A egy nyugtát küld ($t_j + c = a_{j+1}$). Ekkor az on-line költség a $2n+1$ csomagot tartalmazó inputra $A(I_{2n+1}) = \gamma(2n+1) + (1-\gamma)\sum_{i=1}^{2n+1}(t_i - a_i)$. Most tekintsük a következő két off-line algoritmust. ODD csak a páratlan sorszámú csomagot és az utolsó csomagot nyugtázza, EV viszont csak a páros sorszámú csomagokat nyugtázza.

Ezen algoritmusok költségei:

$$\mathrm{EV}(I_{2n+1}) = (n+1)\gamma + (1-\gamma)\sum_{i=1}^{n}(a_{2i+1} - a_{2i}) =$$

$$(n+1)\gamma + (1-\gamma)(nc + \sum_{i=1}^{n}(t_{2i} - a_{2i}))$$

és

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

$$(n+1)\gamma + (1-\gamma)(nc + \sum_{i=1}^{n}(t_{2i-1} - a_{2i-1})).$$

Másfelől sem az ODD sem EV nem ér el kisebb költséget mint az optimális off-line költség, így

$$OPT(I_{2n+1}) \leq \min\{EV(I_{2n+1}), \mathrm{ODD}(I_{2n+1})\} \leq$$

$$(EV(I_{2n+1}) + ODD(I_{2n+1}))/2.$$

Ezért

$$\frac{A(I_{2n+1})}{OPT(I_{2n+1})} \geq \frac{2(\gamma(2n+1) + (1-\gamma)\sum_{i=1}^{2n+1}(t_i - a_i))}{\gamma(2n+2) + (1-\gamma)(\sum_{i=1}^{2n}(t_i - a_i) + 2nc)}$$

$$\geq 2 - \frac{2\gamma + 4nc(1-\gamma)}{\gamma(2n+2) + 2nc(1-\gamma)}.$$

Az alsó korlát, amit az $A(I_{2n+1})/OPT(I_{2n+1})$ hányadosra kaptunk, a $2\gamma/(c(1-\gamma)+\gamma)$ értékhez tart, ha n tart végtelenhez. Ezzel a tételt beláttuk. ■

3. Fejezet

Paraméter tanuló algoritmusok a nyugtázási problémához

Általában egy on-line algoritmus hatékonyságát versenyképességi elemzéssel vizsgálják. A versenyképességi hányados a nyugtázási problémára teljesen megoldott [14], az ébresztő algoritmus optimális, azaz a problémában elérhető legkisebb versenyképességi hányadossal rendelkezik, ez azonban nem jelenti azt, hogy egy másik (esetleg nem optimális versenyképességgel rendelkező algoritmus) ne működhetne hatékonyabban valós adatokon.

Ebben a fejezetben bemutatunk egy új on-line algoritmust és egy új félig on-line algoritmust is a nyugtázási problémára. Az alapötlet mindkét esetben az ébresztő algoritmus továbbfejlesztése. Ezt úgy is fel lehet fogni, mint egy paraméteres algoritmust, ahol a paramétert, úgy választjuk, hogy a versenyképességi hányadost minimalizálja. Az új algoritmus szakaszokban működik és minden szakaszban megkeresi a paraméter optimális értékét az input utolsó szakaszára. A paraméter ezen új értékét használjuk a következő szakaszban. Teszteket végeztünk az új algoritmus teljesítményének meghatározására és megmutatjuk, hogy az új algoritmus jobb teljesítmény nyújt a teszteken, mint az ébresztő algoritmus.

3.1. Jelölések és előzmények

Most is a korábban definiált f_{sum} matematikai modellt [14] használjuk. Az ébresztő algoritmus [14] egy kiterjesztését fogjuk használni. 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 1. és 4. prepozíciókból következik, hogy nem létezik a problémára kisebb versenyképességi hányadosú on-line algoritmus ebben a modellben, de ennek ellenére egy új algoritmus jobb lehet átlagos esetben. Egy algoritmusra A és inputra I egy $A(I)$ alakú kifejezés az A algoritmus költségét jelenti I inputon az f_{sum} célfüggvényen.

3.2. On-line paraméter tanuló algoritmus

Most bemutatjuk a paraméter tanuló algoritmust. Az algoritmus szakaszokban működik és az ébresztő algoritmust használja a csomagok nyugtázására. Először definiáljuk az ébresztő algoritmus egy verzióját melyet használni fogunk, az $Alarm_p$ algoritmust, mely egy pozitív p paramétert használ. $Alarm_p$ algoritmus $e_j = (p\gamma/(1-\gamma) - \sum_{a_i \in \sigma_j}(a_j - a_i))/|\sigma_j|$ értéket használja minden j-hez. Ez az e_j érték azt jelenti, hogy ha nem érkezik új csomag $a_j + e_j$ időpontig, az algoritmus $a_j + e_j$ időpontban

nyugtázza az összes még nyugtázatlan csomagot. A késedelmi költség $\sum_{a_i \in \sigma_j}(a_j - a_i)) + |\sigma_j| \cdot e_j = p \cdot \gamma/(1-\gamma)$. Meg kell jegyeznünk, hogy $Alarm_1$ az előzőekben már tárgyalt 2-versenyképes ébresztő algoritmus [14].

Az új paraméter tanuló algoritmus ezen p paraméter tanulásán alapszik, továbbá használja az r és q paramétereket, amelyek a tanulási fázis hosszát határozzák meg. Az algoritmus a következőképpen működik.

LearnAlarm(r,q) algoritmus

- *1. szakasz* Használjuk az $Alarm_1$ algoritmust az első r csomag nyugtázásához. Ugrás a 2. szakaszra.

- *j. szakasz ($j > 1$)*

  - Jelölje I_j az inputot, ami az utolsó $r \cdot q$ csomag. Ha kevesebb csomag érkezett, akkor I_j mindet tartalmazza.

  - A SimpleOpt algoritmust használjuk a p azon értékének meghatározására, mely minimalizálja az $Alarm_p$ algoritmus költségét az I_j inputon. Jelölje ezt az értéket p^*.

  - Az $Alarm_{p^*}$ algoritmust használjuk a következő r csomag nyugtázására majd folytassuk a $j+1$. szakaszon.

3.3. Az optimális p érték meghatározása az $Alarm_p$ algoritmushoz

A következő lemma fontos szerepet játszik a p paraméter optimális értékét meghatározó algoritmus kifejlesztésében.

2. Lemma. *A p optimális értéke olyan, hogy a nyugtázás időpontja pontosan valamely csomag beérkezési időpontjával egyenlő.*

Bizonyítás: Tekintsünk a csomagok egy tetszőleges sorozatát, jelöljük I-vel és egy p értéket, amelyik nem teljesíti a lemma által definiált tulajdonságot. Megmutatjuk, hogy ez nem optimális érték I-hez. Jelölje k az $Alarm_p$ algoritmus által küldött nyugták számát. Az $Alarm_p$ definíciójából következik, hogy minden nyugta teljes késedelme $p\gamma/(1-\gamma)$, ezért $Alarm_p(I) = k\gamma + (1-\gamma)kp\gamma/(1-\gamma) = k\gamma(1+p)$. Mivel feltevésünk szerint $Alarm_p$ nem pontosan egy csomag megérkezésekor küldi a nyugtát, ezért létezik egy $\varepsilon > 0$, amire egy másik algoritmus a p helyett egy $p-\varepsilon$ paramétert használva ugyanazokat a csomagokat nyugtázza mint $Alarm_p$, de korábban. Ez azt jelentené, hogy az $Alarm_{p-\varepsilon}$ ugyanúgy k nyugtát használná továbbá azt is, hogy $Alarm_{p-\varepsilon}(I) = k\gamma(1+p-\varepsilon)$ ami azt mutatja, hogy ekkor p nem optimális választás. ■

Emiatt a következő algoritmust használhatjuk a p paraméter optimális értékének meghatározására. Az algoritmus csak azokat az értékeket vizsgálja, melyek a lemma által támasztott kritériumnak megfelelnek.

A SimpleOptAlarm Algoritmus

- *Inicializálás:* Legyen $p^* = 1$ és $Z = \infty$
- *Keresés:* Minden $1 \leq i < j \leq n$ -re
  - Legyen $s := \sum_{k=i}^{j}(a_j - a_i)$, és $p := (1-\gamma)s/\gamma$.
  - Futtassuk p-re $Alarm_p$-t az inputon és határozzuk meg a nyugtát számát (k)
  - Ha $k\gamma(1+p) < Z$, akkor $p^* := p$ és $Z := k\gamma(1+p)$
- Return p^*

A keresés $\Theta(n^2)$ iterációt igényel és minden iterációban a második lépés futási ideje $\Theta(n)$, így az algoritmus futási ideje $\Theta(n^3)$.

3.4. Az $LearnAlarm(r, q)$ algoritmus hatékonysági vizsgálata

Az új paraméter tanuló algorimusunkat teszteknek vetettük alá, hogy a hatékonyságát értékeljük. A tesztek végrehajtásához valós adatokat és véletlen generált inputot is felhasználtunk. Hogy jobban megértsük az egyes paraméterek szerepét és hatásait az adatokon több különböző paraméter beállítással is futtattuk az algoritmust. Az $A1, A2, \ldots, A6$ algoritmusok a $q = 10$ és rendre a $r = 100, 200, 500, 1000, 2000, 4000$ értékeket használták a tesztek futtatása során. A $B1, B2, \ldots, B6$, $C1, \ldots, C6$, $D1, \ldots, D6$, $E1, \ldots, E6$ algoritmusok rendre a $q = 25, 50, 75, 100$ és $r = 100, 200, 500, 1000, 2000, 4000$ értékekkel futottak.

3.4.1. Tesztek valós adatokon

Két különböző webszerver forgalmát vizsgáltuk: a http://www.sziszszi.hu-t, egy némileg kisebb forgalmú egy középiskola webszerverének forgalmát, melyre a tesztekben Szerver1-ként hivatkozunk, és egy nagyobb forgalmú szervert, mely több mint 30 általános- és középiskola intézményi adminisztrációját és elektronikus naplóját szolgálja ki, ezt Szerver2-vel jelöltük. Mindkét szerveren naplóztuk egy-egy adatbázisba azon események időpontjait, amihez adatcsomag küldése tartozott. Ezeket az adatokat használtuk fel a tesztek inputjaként. Mindkét adatbázisból 3 különböző méretű részinputot használtunk: small (10,000) medium (100,000) large (500,000), melyeket rendre S, M és L jelöl. Mindegyik méretből 10-10 tesztesetet generáltunk az adatbázisok különböző részeiből. Az inputok letölthetők a

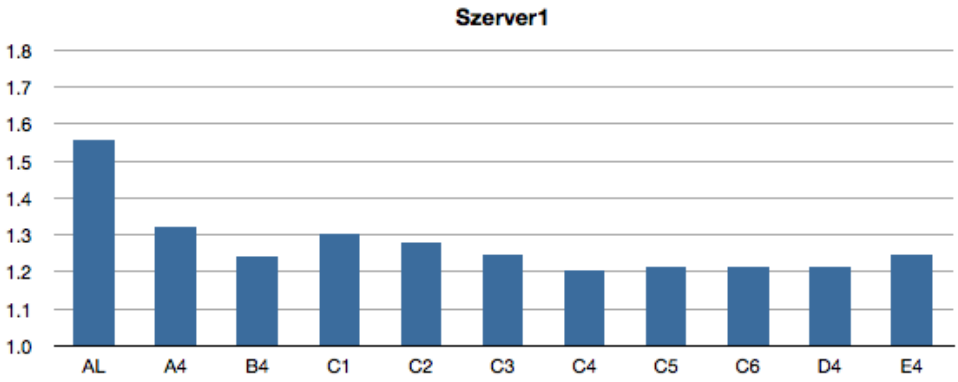
$$www.inf.u - szeged.hu/ \sim cimreh/dataack/acknowltest.zip$$

webcímről.

Az algoritmusokat 3 különböző célfüggvénnyel teszteltük, 0.25, 0.5 és 0.75 γ értékekkel. Mindegyik inputra egy dinamikus programozás elvű algoritmus segítségével [14] meghatároztuk az optimális off-line megoldást, ezt jelölje OPT. Tehát minden inputon 31 on-line algoritmust futtattunk:

$$Alarm, A1, \ldots, A6, B1, \ldots, B6, \ldots, E1, \ldots, E6$$

algoritmusokat. A egyes on-line algoritmusok költségeinek átlagát, az optimális off-line költségek átlagát és ezek arányait az 1. számú függelékben található táblázat tartalmazza. A 3. és 4. ábra az ALG/OPT értékek átlagát ábrázolja az egyes szerverek adataival néhány algoritmushoz.

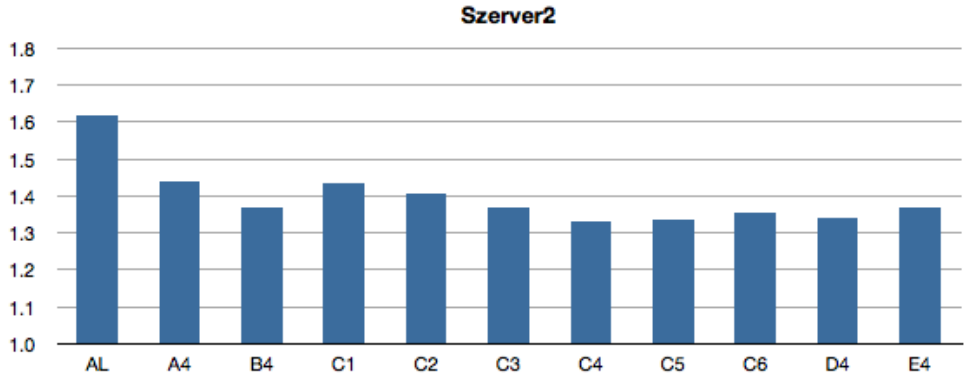


3. ábra. Néhány algoritmus teljesítménye a Szerver1-en

A $C1, C2, \ldots, C6$ algoritmusokat azért választottuk ki, hogy az r paraméter hatását be tudjuk mutatni, az A, B, D, E algoritmus osztályok legjobb eredményei azt mutatják be, hogy a q paraméter hogyan befolyásolja az algoritmus működését, és hogyan kell a megfelelő q értéket a problémához optimálisan beállítani. $Alarm(AL)$ lényegesen rosszabb értéke a diagramon jól mutatja a paraméter tanuló algoritmus gyakorlati hasznát.

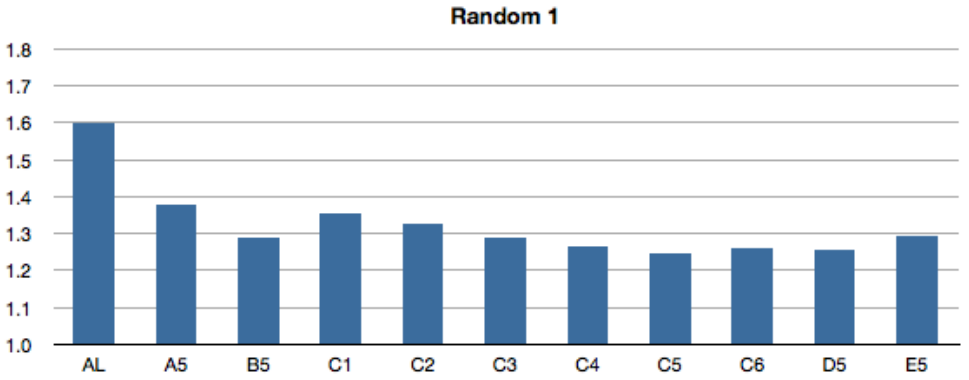
3.4.2. Tesztek végrehajtása generált adatokon

A valós adatokon végrehajtott tesztek mellett véletlen-generált inputokon is teszteltük az algoritmusokat. Két tesztesetet vizsgáltunk. Az első esetben egyenletes eloszlású érkezési időket, míg a másodikban exponenciális eloszlású inputot generáltunk. Ezt ez eloszlást egy homogén Poisson-folyamat eredményezi, mely gyakran előforduló telekommunikációs jelenség. Itt is minden inputon 31 on-line algoritmust futtattunk, az $Alarm$, $A1, \ldots, A6$, $B1, \ldots, B6, \ldots, E1, \ldots, E6$ algoritmusokat. A egyes



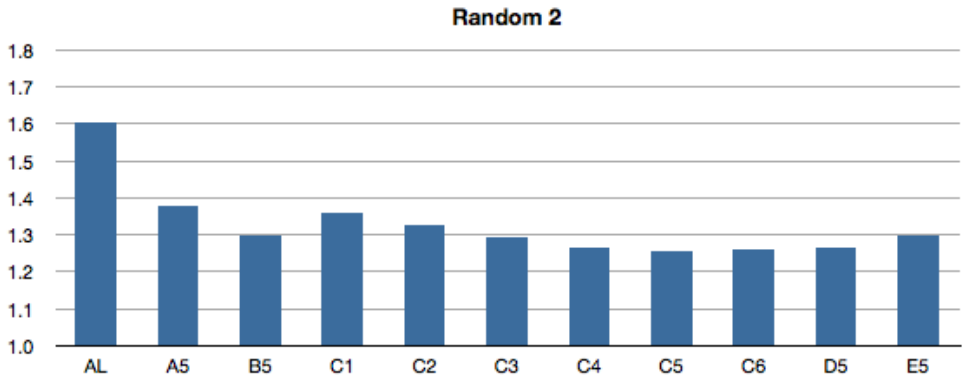
4. ábra. Néhány algoritmus teljesítménye a Szerver2-őn

on-line algoritmusok költségeinek átlagát, az optimális off-line költségek átlagát és ezek arányait a ... számú függelékben található táblázat tartalmazza.



5. ábra. Néhány algoritmus teljesítménye egyenletes eloszlású adatokon

Az 5. és 6. ábrán néhány algoritmus teszteredményeit láthatjuk a valós adatoknál ismertetett szempontok alapján. A véletlen-generált adatokon futtatott tesztek eredményeinek eredményeit az 1. számú függelékben



6. ábra. Néhány algoritmus teljesítménye exponenciális eloszlású adatokon

található táblázat tartalmazza.

3.4.3. Következtetések

A fenti eredmények alapján a következő következtetéseket vonhatjuk le:

- A legfontosabb megfigyelésünk, hogy a tesztek eredményei tisztán mutatják, hogy a paraméter tanulás jelentős hatékonyságjavulást eredményez az *Alarm* algoritmus működésében.

- A teszteredmények alapján az is jól látható, hogy az új algoritmus igen érzékeny a paraméterek beállításaira. Egy pontos paraméter-beállítás tovább csökkenti az ALG/OPT hányadost. Az is jól látszik, hogy a paraméterek optimális beállítása az input függvénye, de néhány általános következtetést levonhatunk. Úgy tűnik, hogy egy közepes hosszúságú tanulási szakasz jobb eredményt ad, mint egy

rövid, vagy egy hosszú. Ami a vizsgált szakaszok hosszát illeti (r paraméter), az látszik, hogy 1000 - 2000-ig növelve azt jelentős javulással számolhatunk, de a további növelés nem hoz szignifikáns javulást az algoritmus hatékonyságában. A véletlen-generált teszteken az $r = 2000$, míg valós adatokon az $r = 1000$ beállítás adott jobb eredményeket.

- Miután összehasonlítjuk a különböző tesztesetek eredményeit azt a következtetést is levonhatjuk, hogy az algoritmusok hasonló viselkedést mutatnak. A Szerver2-őn a javulás valamivel jobb volt, mint a többi esetben.

3.5. Félig on-line paraméter tanuló algoritmus

Ebben a fejezetben a nyugtázási probléma idő-előretekintő változatát [27] vizsgáljuk. A 2. fejezetben már megmutattuk, hogy a nyugtázási probléma idő-előretekintő modelljében definiált Ébresztő algoritmus kiterjesztése bármely c-re sem jobb mint 2-versenyképes. Most definiáljuk egy paraméter tanuló változatát ennek az algoritmusnak. Analizáljuk az algoritmust az előző fejezetben ismertetett valós adatokon [28], továbbá véletlen-generált adatokon is. Az eredmények megmutatják, hogy az átlagos esetben az idő-előretekintő tulajdonság lényegesen jobb eredményeket produkál, mint az Ébresztő algoritmus on-line változata, és az eredményeket a paraméter-tanulás ötlete tovább javítja.

A második fejezetben részletesen bemutatott félig on-line c idő-előretekintő modellt fogjuk alkalmazni, ahol a döntéshozó ismeri a t időpontban már megérkezett csomagok érkezési időpontját, továbbá a azon csomagok érkezési időpontját is amik a $(t, t + c]$ intervallumban érkeznek.

Most kiterjesztjük a TLA algoritmust egy paraméter tanuló algoritmussá a fejezetben korábban tárgyalt módszerrel.

Paraméter tanuló TLA algoritmus: Az algoritmus szakaszokban működik. Minden szakasz a TLA algoritmust használja a csomagok nyugtázására. Először definiáljuk az ébresztő algoritmus egy verzióját melyet használni fogunk, az TLA_p algoritmust, mely egy pozitív p paramétert használ. TLA_p algoritmus $e_j = (p\gamma/(1-\gamma) - \sum_{a_i \in \sigma_j}(a_j - a_i))/|\sigma_j|$ értéket használja minden j-hez. Ez az e_j érték azt jelenti, hogy ha nem érkezik új csomag $a_j + e_j$ időpontig, az algoritmus $a_j + e_j$ időpontbn nyugtázza az összes még nyugtázatlan csomagot. A késedelmi költség $\sum_{a_i \in \sigma_j}(a_j - a_i)) + |\sigma_j| \cdot e_j = p \cdot \gamma/(1-\gamma)$. Meg kell jegyeznünk, hogy TLA_1 algoritmus 2-versenyképes [28].

Az új paraméter tanuló algoritmus a p paraméter tanulásán alapszik, és az r és q paramétereket használja, amik a tanulási fázis hosszát határozzák meg. Az algoritmus a következőképpen működik.

LearnTLA(r,q) algoritmus

- *1. szakasz* Használjuk az TLA_1 algoritmust az első r csomag nyugtázásához. Ugrás a 2. szakaszra.
- *j. szakasz ($j > 1$)*
  - Jelölje I_j az inputot, ami az utolsó $r \cdot q$ csomag. Ha kevesebb csomag érkezett, akkor I_j mindet tartalmazza.
  - A SimpleOpt algoritmust használjuk a p azon értékének meghatározására, mely minimalizálja az TLA_p algoritmus költségét az I_j inputon. Jelölje ezt az értéket p^*.
  - Az $Alarm_{p^*}$ algoritmust használjuk a következő r csomag nyugtázására majd folytassuk a $j+1$. szakaszon.

A következő $\Theta(n^3)$ futási idejű algoritmus meghatározza p optimális értékét.

A SimpleOptTLA Algoritmus

- *Inicializálás:* Legyen $p^* = 1$ és $Z = \infty$
- *Keresés:* Minden $1 \leq i < j \leq n$ -re
  - Legyen $s := \sum_{k=i}^{j}(a_j - a_i)$, és $p := (1-\gamma)s/\gamma$.
  - Futtassuk p-re TLA_p-t az inputon és határozzuk meg a nyugtát számát (k)
  - Ha $k\gamma(1+p) < Z$, akkor $p^* := p$ és $Z := k\gamma(1+p)$
- Return p^*

3.6. Az algoritmusok értékelése

Az új paraméter tanuló algorimusunkat teszteknek vetettük alá, hogy a hatékonyságát értékeljük. A tesztek végrehajtásához itt is valós adatokat és véletlen generált inputot használtunk. Minden inputra futtattuk az optimális off-line megoldást a dinamikus programozás módszerével meghatározó [14] algoritmust, ezen a megoldáson felvett célfüggvény értéket jelöltük OPT-tal, továbbá minden inputra futtattuk a TLA algoritmust is. Ezen kívűl számos különböző paraméter beállítással futtattuk az új algoritmust is. Az $A1, A2, A3$ algoritmusok a $q = 25$ és rendre a $r = 200, 2000, 4000$ értékeket használták a tesztek futtatása során. A $B1, B2, B3$, $C1, C3, C3$, és $D1, D2, D3$ algoritmusok rendre a $q = 50, 75, 100$ és $r = 200, 2000, 4000$ értékekkel futottak. Az előretekintő algoritmusokat minden teszt bemenetnél két különböző előretekintő értéken vizsgáltuk, $c = 0.2$ és $c = 0.8$ értékekkel.

3.6.1. A valós tesztadatok

Itt is a szakoktatas.hu webszerver forgalmi eseményeinek egy adatbázisba rögzített (naplózott) időpontjait használtuk a tesztek futtatása során. Az adatbázisból 3 különböző méretű részinputot használtunk fel a tesztekhez az algoritmusok inputjaként: kis méretű input - small (10,000 adat), közepes méretű - medium (100,000 adat) és nagy méretű input - large (500,000 adat), melyeket rendre S, M és L jelöl. Minden méretthez 10 különböző inputot konstruáltunk. Az algoritmusokat 3 különböző célfüggvény mellett analizáltuk, 0.25, 0.5 és 0.75-ös γ értékek mellett. A

1. táblázat. teszt eredmények a $c = 0.2$ értékhez

	OPT	AL	TLA	B2
S(0,25)	2114	3798	2748	2575
S(0,5)	4271	6517	5566	5215
S(0,75)	5744	8187	7467	6993
M(0,25)	21735	38026	28212	26501
M(0,5)	43272	64534	56253	52791
M(0,75)	58552	82466	75883	71357
L(0,25)	107539	191773	139263	130982
L(0,5)	224571	323450	291268	273967
L(0,75)	298716	412946	386240	363448

teszteredmények átlagait az 1. és 2. táblázatban összesítettük, ahol az optimális off-line megoldás, az ébresztő algoritmus, a TLA algoritmus és a legjobb paraméter tanuló algoritmus eredményei szerepelnek. Kiszámoltuk az egyes algoritmusok optimális off-line költséghez viszonyított arányát is. Az ébresztő algoritmus esetében ez az arány 1.556, a TLA algoritmus esetében pedig rendre 1.298 és 1.211 a $c = 0.2$ és $c = 0.8$ értékekre. A paraméter tanuló algoritmusokra az arányt a paraméter beállítások függvényében a 3. és 4. táblázat mutatja.

3.6.2. Generált tesztadatok

A valós adatok használata mellett véletlen generált bemenetekkel is teszteltük az algoritmusokat. Két különböző esetet vizsgáltunk, az első esetben egyenletes eloszlású érkezési időket, a másodikban exponenciális eloszlású érkezési időket generáltunk. Itt is meghatároztuk az egyes algoritmusok és az optimális off-line megoldás költségének arányát. Az

2. táblázat. teszt eredmények a $c = 0.8$ értékhez

	OPT	AL	TLA	B2
S(0,25)	2114	3798	2555	2429
S(0,5)	4271	6517	5186	4920
S(0,75)	5744	8187	6967	6597
M(0,25)	21735	38026	26341	25001
M(0,5)	43272	64534	52488	49803
M(0,75)	58552	82466	71011	67335
L(0,25)	107539	191773	129390	123873
L(0,5)	224571	323450	272404	258892
L(0,75)	298716	412946	361167	342960

3. táblázat. ALG/OPT eredmények a $c = 0.2$ értékhez valós adatokon

	A	B	C	D
1	1.230	1.220	1.221	1.225
2	1.221	1.219	1.219	1.222
3	1.223	1.219	1.220	1.223

4. táblázat. ALG/OPT eredmények a $c = 0.8$ értékhez valós adatokon

	A	B	C	D
1	1.168	1.151	1.152	1.155
2	1.163	1.150	1.150	1.153
3	1.165	1.150	1.151	1.54

egyenletes eloszlású adatokon az ébresztő algoritmusnál átlagos arány 1.601, a TLA algoritmusnál 1.276 és 1.183 a $c = 0.2$ és $c = 0.8$ értékekre. A paramétertanuló algoritmusokra az arányt a paraméter beállítások függvényében az 5. és 6. táblázat mutatja.

Az tesztek az exponenciális eloszlású adatokon az ébresztő algoritmus

5. táblázat. ALG/OPT eredmények a $c = 0.2$ értékhez, egyenletes eloszlású inputon

	A	B	C	D
1	1.212	1.197	1.198	1.201
2	1.198	1.196	1.196	1.199
3	1.199	1.196	1.196	1.1.200

6. táblázat. ALG/OPT eredmények a $c = 0.8$ értékhez, egyenletes eloszlású inputon

	A	B	C	D
1	1.138	1.130	1.130	1.133
2	1.130	1.128	1.128	1.131
3	1.132	1.129	1.129	1.132

7. táblázat. ALG/OPT eredmények a $c = 0.2$ értékhez, exponenciális eloszlású inputon

	A	B	C	D
1	1.229	1.220	1.220	1.224
2	1.219	1.217	1.217	1.220
3	1.222	1.220	1.219	1.223

8. táblázat. ALG/OPT eredmények a $c = 0.8$ értékhez, exponenciális eloszlású inputon

	A	B	C	D
1	1.160	1.151	1.151	1.155
2	1.150	1.147	1.147	1.149
3	1.153	1.150	1.150	1.154

esetében 1.604, a TLA algoritmus esetében rendre 1.288 és 1.1214 arányt eredményeztek a $c = 0.2$ és $c = 0.8$ értékekre. Az exponenciális eloszlású inputokon a paraméter tanuló algoritmusokhoz kapott arányokat a 7. és 8. táblázat mutatja a paraméter beállítások függvényében.

3.6.3. Következtetések

Ebben a fejezetben az on-line nyugtázási probléma idő-előretekintő algoritmusainak egy empírikus analízisét mutattunk be. Összehasonlítottuk az ébresztő algoritmust és annak idő-előretekintő változatát valamint a paraméter tanuló on-line algoritmusunkat az optimális off-line algoritmus megoldásának eredményével. Az eredmények alapján az alábbi következtetéseket vonhatjuk le:

- A vizsgálatok alapján a fő megfigyelésünk, hogy a versenyképességi analízis eredményével ellentétben mind a paraméter tanulás, mind az előretekintő tulajdonság jelentős javulást eredményez a az ébresztő algoritmus átlagos teljesítményében. A $c = 0.2$ előretekintő értéknél az ALG/OPT hányados 17%-ot csökkent az előretekintés hatására, és további 7%-ot a paraméter tanulás ötletét használva. A $c = 0.8$ esetben az előretekintés hatására 22%-ot, a paraméter tanulás hatására további 5%-ot javult a hatékonyság.

- A teszteredményekből az is jól látszik, hogy az különböző inputok nem befolyásolják jelentősen az algoritmusok hatékonyságát, bár meg kell jegyeznünk, hogy egyenletes eloszlás mellett enyhén jobb eredményeket kapunk.

- Végül azt is észrevehetjük, hogy a paraméter tanuló algoritmus az idő-előretekintő esetben nem érzékeny annyira a különböző paraméter beállításokra, mint az on-line esetben, minden beállítás mellett hasonló eredményeket kapunk.

4. Fejezet

Visszautasításos on-line ütemezés

Ebben a fejezetben egy új algoritmust mutatunk be és elemzünk az on-line visszautasításos ütemezés problémájának megoldására, azonos gépek alkalmazása esetén.

Az ütemezési feladatokban munkák végrehajtását kell megszerveznünk. Az általános modellben egy gyártási folyamat során a munkadarabokon több különböző műveletet kell elvégezni. Ehhez meg kell határozni az egyes műveleteket végrehajtó gépeket és az időintervallumokat, melyekben az egyes műveletek végrehajtódnak.

A továbbiakban azt az egyszerűbb modellt vizsgáljuk, melyben minden munka egyetlen műveletből áll, tehát a feladatunk az, hogy a munkához, amelynek ismerjük a végrehajtási idejét hozzárendeljük azt a gépet, amelyen a munkát végrehajtjuk, továbbá a végrehajtás kezdeti és befejezési időpontját. A két időpont különbsége a végrehajtási idő. Azzal a modellel foglalkozunk, amikor a végrehajtás ideje minden gépen ugyanannyi, tehát párhuzamos azonos gépekről beszélünk. Az on-line modellben a munkák egy listáról érkeznek. Amikor egy munkát megkapunk a listáról, akkor ismerjük meg a szükséges végrehajtási időt. Ezt követően ütemeznünk kell a munkát, hozzárendelve a kezdési és befejezési időt, amelyet később már nem változtathatunk meg, és csak ezt követően kapjuk meg a listáról a következő munkát.

A visszautasításos ütemezés problémájában [4] az algoritmusnak lehetősége van az egyes munkákat visszautasítani. A munkák jellemzője a végrehajtási idő és a visszautasítás büntetése. Célunk itt az elfogadott munkák befejezési idejének és a visszautasítás büntetéseinek összegét minimalizálni. Létezik egy 2.618-versenyképes algoritmus az on-line esetre tetszőleges számú géphez. Ezt az algoritmust RTP algoritmusnak (Reject Total Penalty) nevezik, és az egyik alapötlete az, hogy minden munkára hasonlítsuk össze a büntetést és a terhelést (végrehajtási idő osztva a gépek számával), majd utasítsuk el azokat a munkákat, melyekre a büntetés kisebb mint a terhelés. Ez az alapötlet még csak a mohó algoritmust adja, ami akkor hoz rossz döntést, ha a gépek száma nagy, mert ez lehetővé teszi, hogy úgy tűnjön, hogy egy nagy munka kis terhelést jelent. Az RTP algoritmus ezeket a munkákat sokkal óvatosabban kezeli. Ismert továbbá 1.618-versenyképes algoritmus is a 2 gépes ütemezéshez [4] és ezzel egyező alsó határ is ismert. Az off-line esetet megoldhatjuk egy dinamikus programozáson alapuló teljesen polinomiális approximációs sémával, ha a gépek számát konstansnak tekintjük, és egy polinomiális idejű approximációs sémával, ha a gépek száma az input része [4]. Az on-line visszautasításos ütemezés megszakítható munkákkal modellt a [36] cikkben vizsgálták. A probléma további általánosítása, amikor a gépek használatának is van költsége [11, 30] és az algoritmusnak ezt ki kell fizetnie ha egy gépet használni szeretne.

Fontosnak tartjuk megjegyezni, hogy az ütemezési problémát követően egyéb on-line visszautasításos modelleket is vizsgáltak. Az on-line visszautasításos ládapakolást, ahol a költség a használt ládák számának

és a visszautasított tárgyak büntetésének az összege a [12] és [19] cikkekben vizsgálták. Az on-line visszautasításos gráfszínezést tárgyalja a [18] cikk. Az on-line visszautasításos lapozás a [20] cikkben vizsgálják, a visszautasításos k-szerver problémát pedig a [6] cikkben.

A feladat egy általánosítása az az ütemezési modell, ahol két géphalmaz van és minden munkának két végrehajtási ideje, attól függően, hogy melyik géphalmaz gépén hajtjuk végre. A célfüggvény a gépcsoportokon vett maximális befejezési idők összege. Ha az egyik csoport egyetlen gépből áll, akkor a visszautasításos ütemezési modellt kapjuk. Ezt a modellt a [24, 25] cikkekben vizsgálták, ahol elemezték az RTP algoritmus egy kiterjesztését és megadtak egy teljesen polinomiális approximációs sémát az off-line problémára.

Ebben a fejezetben bemutatunk néhány már ismert eredményt a visszautasításos on-line ütemezés területéről, majd bemutatjuk az új paraméter tanuló algoritmusunkat is. Az algoritmusokat hatékonysági vizsgálatnak vetjük alá valós és generált adatokból készített input szekvenciákon végzett tesztek alapján, majd kiértékeltük a kapott eredményeket.

4.1. Jelölések és előzmények

A problémában m azonos gépet használunk. Minden j munkának van egy p_j végrehajtási ideje és egy büntetése w_j. A munkák tetszőleges J sorozatára A algoritmus által választott ütemezés költsége $A(J)$. Az on-line ütemezési algoritmust eljárásként fogjuk használni. Számos az on-line ütemezési probléma megoldására kifejlesztett algoritmus [37] használ mohó

stratégiát a probléma megoldására [37] úgy, hogy az algoritmus mindig mohón ütemezi az érkező munkát az első szabad gépre. A visszautasítás lehetőségét is felhasználva kézenfekvő az ötlet, hogy utasítsuk el a munkát, ha a visszautasítás büntetése kisebb, mint a munka végrehajtása esetén járó haszon. Az alábbi mohó algoritmus egy (nem optimális) megoldást nyújt a problémára.

A Greedy algoritmus

Ha a j munkára $w_j \leq p_j/m$, utasítsuk el, egyébként ütemezzük a munkát arra a gépre, ahol a legkisebb az eddig odarendelt munkák végrehajtási idejének összege.

Sajnos a Greedy algoritmus mohó választása némely esetben nem jó. Ha csak egyetlen nagy végrehajtási idedű M munka érkezik és a visszautasításának büntetése $M/m + \varepsilon$, a mohó választás elfogadja és beütemezi a munkát. Így az algoritmus nem lehet m-versenyképesnél jobb, tehát a versenyképességi hányados nem konstans.

Az RTP algoritmus [4] ellenben konstans versenyképességi hányadossal rendelkezik. Ez a Greedy algoritmusnak egy kifinomultabb változata, ami szintén visszautasítja azokat a munkákat (amikre $w_j \leq p_j/m$), de a visszautasított munkákat összegyűjti egy R halmazba, és bizonyos nagy munkákat akkor is visszautasít, ha $w_j > p_j/m$.

Algorithm RTP(α)

- 1. *Inicializáció.* Legyen $R := \emptyset$.
- 2. Egy új j munka érkezésekor
  - (i) Ha $w_j \leq \frac{p_j}{m}$, visszautasítjuk a munkát.
  - (ii) Legyen $r = \sum_{i \in R} w_i + w_j$. Ha $r \leq \alpha \cdot p_j$, visszautasítjuk a j munkát és $R := R \cup \{j\}$.
  - (iii) Egyébként elfogadjuk j-t és ütemezzük a munkát arra a gépre, ahol a legkisebb az eddig odarendelt munkák végrehajtási idejének összege.

Az RTP algoritmus versenyképességi hányadosáról és a felső korlátról [4] a következők állíthatók.

5. Prepozíció. *Az RTP az* $\alpha = (\sqrt(5) - 1)/2$ *paraméterrel* $(3 + \sqrt(5))/2$ *versenyképes.*

6. Prepozíció. *Nem létezik olyan* β*-versenyképes on-line algoritmus tetszőleges* m*-re, ahol* $\beta < (3 + \sqrt{5})/2$

4.2. Paraméter tanuló algoritmus

A prepozíciók alapján a visszautasításos több processzoros ütemezés problémája megoldott az általános esetre, mivel versenyképesség szempontjából a lehetséges legjobb algoritmust ismerjük. A probléma

optimálisan megoldására az $RTP((\sqrt{5}-1)/2)$ algoritmust használhatjuk. Másrészt sokszor a jobb versenyképességgel rendelkező algoritmus rosszabb eredményt produkál az átlagos esetre, valós vagy generált adatokon. Az on-line ütemezés területén már láthattunk erre példát is [2], melyben a több processzoros ütemezési modell különböző algoritmusait tesztelték és azt az eredményt kapták, hogy az egyszerű LISTA algoritmus [23] jobb eredményeket ért el, mint néhány jóval komplikáltabb, jobb versenyképességű algoritmus. Ebben a problémában az α paraméter optimális értéke $(\sqrt{5}-1)/2$ a versenyképesség szempontjából, de kézenfekvő a kérdés, hogy a paraméter más értéke nem jobb-e az átlagos esetre.

Most bemutatunk egy új algoritmust, PAROLE (Parameter Online Learning), mely a futása során megpróbálja megtanulni a legjobb paramétert. Az algoritmus fázisokban működik, minden fázis után választ egy új paramétert az input addig megismert része alapján. Először egy keretalgoritmust definiálunk, mely az új paramétert választó algoritmust eljárásként használja, majd megadjuk azt az eljárást is, mely a paraméter optimális értékét próbálja határozni az adott fázison. A fázisokat a beérkezett munkák száma alapján határozzuk meg, a fázis a 250. beérkezett munkával ér véget.

A PAROLE algoritmus (i. fázis)

- Az i fázis kezdetekor a CHOOSE algoritmus meghatározza az α_i paraméter értékét.
- Hajtsuk végre $RTP(\alpha_i)$ algoritmust az inputon már ismert részén, R halmazt is módosítjuk.

- Használuk $RTP(\alpha_i)$-t a fázis alatt beérkező munkákra.

Azt az α_i értéket szeretnénk használni, mely mellett $RTP(\alpha_i)(I)$ költség minimális az input már ismert részére. Sajnos a paraméter optimális értékének meghatározása nehéz, mert az $RTP(\alpha)(I)$ nem monoton és nem is folytonos függvénye α-nak. A feltételezésünk, hogy az optimális α érték meghatározása egy NP-teljes probléma, de úgy tűnik ezt nehéz bizonyítani. Ezért a következő mintavételező algoritmust alkalmazzuk a paraméter új értékének meghatározására. Az algoritmus a paraméter megelőző értékét is felhasználja, amit α^*-al jelölünk.

A CHOOSE algoritmus

- Generálunk egy-egy értéket egyenletes eloszlással az $[(i-1)/10, i/10]$, $i = 1, \ldots, 10$ intervallumokból. A kiválasztott értékek halmazát jelöljük S_1-el. Válasszuk ki S_1 halmaz azon α elemét, melyre $RTP(\alpha)$-nak a legkisebb költsége van I-n. Jelöljük ezt az értéket $\bar{\alpha}$-val.
- Generálunk egy-egy értéket egyenletes eloszlással az $[\alpha^\star - i/100, \alpha^\star - (i-1)/100]$, $[\alpha^\star + (i-1)/100, \alpha^\star + i/100]$, $[\bar{\alpha} - i/100, \bar{\alpha} - (i-1)/100]$, $[\bar{\alpha} + (i-1)/100, \bar{\alpha} + i/100]$ intervallumokból is, ahol $i = 1, \ldots 10$. A kiválasztott értékek halmazát jelöljük S_2-vel. Legyen α az $S_2 \cup \{\alpha^*\} \cup \{\bar{\alpha}\}$ halmaz azon eleme melyre I inputra az $RTP(\alpha)$-nak a legkisebb költsége van.

A CHOOSE alapötlete, hogy válasszunk egy jó paraméterértéket. Két jelölt szomszédságát vizsgáljuk meg, az egyik az előző paraméter értéke, a másik egy további $\bar{\alpha}$ érték, mely a legjobb egy sor olyan érték közül, melyeket az paraméter előző értékétől függetlenül választottunk.

4.3. Hatékonysági vizsgálat

4.3.1. A végrehajtott tesztek ismertetése

Az új paraméter tanuló algorimusunkat teszteknek vetettük alá, hogy a hatékonyságát értékeljük. A tesztek végrehajtásához valós adatokat és véletlen generált inputot is felhasználtunk.

A tesztek végrehajtásához a következő algoritmusokat vizsgáltuk.

- a Greedy algoritmus
- az $RTP(\alpha)$ algoritmus az $\alpha = (\sqrt{5} - 1)/2$ paraméterrel
- a parameter tanuló algoritmus ($PAROLE$).

Algoritmusok tesztelésének végrehajtásához a következő típusú bemeneteket konstruáltuk. Használtunk valós adatokat az A és B teszthez, továbbá az irodalomban használt [2] and [5] véletlen eloszlású adatokat generáltunk. Ezen eloszlások egy részének tipikus jellemzője, hogy kis valószínűséggel nagyon nagy munkák is előfordulhatnak. Minden esethez elegendően nagy méretű inputot használtunk, hogy a PAROLE-nak elég lehetősége legyen a paraméter értékének tanulására és a legjobb érték használatára.

- Teszt A (valós adatok): A szakoktatas.hu szerveren naplóztuk az egyes feladatok végrehajtási idejét. A naplófájlban körülbelül 100000 munka végrehajtási idejét gyűjtöttük össze, amit egy adattáblába gyűjtöttünk. A munkák átlagos mérete 2597.1, a szórásuk 20129.96. A legkisebb munka mérete 1, a legnagyobbé 3124460. Így a munkák

mérete követte a fent leírt szituációt [2], azaz voltak nagyon nagy munkák is. A szerveren minden feladathoz egy prioritás érték volt rendelve, mely a feladat fontosságát és sürgősségét jellemezte. Ez a fontossági érték a következő jellemzőkből tevődik össze: feladat tulajdonosa (minden felhasználónak is fontossági értéke van), a feladat típusa (a munkák feladatosztályokhoz vannak rendelve, a feladatosztályhoz pedig prioritási érték), továbbá néhány munkának határideje volt, amit szintén figyelembe vettünk egy változóval. Ezen érték lineáris kombinációját használtuk a visszautasítás büntetésére. Az átlagos büntetés 272.83, a szórás 229.32 volt. A minimális büntetés 1, a maximális 2613 volt.

- Teszt B (valós adatok): A moravarosi.hu szerveren is naplóztuk az egyes feladatok végrehajtási idejét. Az adatbázis k.b. 200000 munkát tartalmazott, és a végrehajtott munkák nagyobbak voltak mint az előző tesztnél. A munkák átlagos mérete 15608.96, míg a szórásuk 120991.53 volt. A legkisebb munka mérete 1, a legnagyobbé 18839728 volt. A munkákhoz tartozó prioritásokat az előző teszthez hasonlóan használtuk fel, az átlagos büntetés 572.99 míg a szórás 481.58 volt. A minimális büntetés értéke itt is 1 míg a maximális érték 5487 volt.

- Teszt C (generált adatok): Ebben a tesztesetben a véletlent használva exponenciális eloszlású munkaméreteket generáltunk. Az eloszlás generálásához $\lambda = 1/1000$ paramétert használtuk, így a várható érték 1000, a variancia 1000000 volt. A büntetések szintén exponenciális eloszlásúak voltak $\lambda = 1/100$ a variancia itt 10000 volt.

- Teszt D (generált adatok): Ebben a tesztesetben hiperexponenciális eloszlású munkaméreteket és büntetéseket generáltunk, 2 különböző λ értékkel. $\lambda_1 = 1/800$ és $\lambda_1 = 1/1200$ értékeket használtunk a munkák generálására, mindkét esetben $p_2 = 1/2$ értéket használva. Ezért a munkák várható értéke itt $p_1/\lambda_1 + p_2/\lambda_2 = 1000$, a variancia 1080000. A büntetésekre vonatkozó paraméterek $\lambda_1 = 1/80$ és $p_1 = 1/2$ valamint $\lambda_2 = 1/120$ és $p_2 = 1/2$, így a várható érték 100, a szórás értéke 10800 volt.

- Teszt E (generált adatok): Ebben a tesztesetben az Erlang2 eloszlású munkaméreteket és büntetéseket generáltunk. A használt paraméterek a munkák esetében $\lambda = 500$, így a várható érték 1000, a szórás 500000 volt a büntetések esetében $\lambda = 1/50$, a várható érték 100, a szórás 5000 volt.

- Teszt F (generált adatok): Ebben a tesztesetben a véletlent használva korlátos Pareto-eloszlású munkaméreteket és büntetéseket generáltunk. A munkák generálásához $B(k, p, \alpha)$-ban k (a legkisebb munka méretének korlátja) 1, p (a legnagyobb munka méretének korlátja) 20000 volt. Az α-t úgy választottuk, hogy a várható érték 1000 legyen. A büntetéseket ugyanígy generáltuk, azzal a különbséggel, hogy a felső korlát 1000 és a várható érték 100 volt.

A valós adatokon végzett analízisnél kétféle tesztet végeztünk, egy nagy méretű (NAGY) inputon mely az összes munkát tartalmazta, és egy kis méretűt (KICSI), ami csak a munkák 20%-át. A tesztek végrehajtásánál végig 10 gépre ütemeztünk. Az eredmények összesítése a 9. táblázatban

található. A generált adatokon végzett teszteknél szintén kétféle input-méretet használtunk, a nagyobb méretű (NAGY) 1 000 000, a kisebb (KICSI) 10 000 munkát tartalmazott, és szintén 10 gépre ütemeztünk. Minkét esethez 100-100 generált adatsoron végeztük el a teszteket, a tesztek eredményeinek átlagát a 10. táblázatban összesítettük.

9. táblázat. Teszteredmények a valós adatokon

	A(KICSI)	A(NAGY)	B(KICSl)	B(NAGY)
Greedy	883549	4326745	2813479	13976483
RTP	843756	4117672	2889437	14245263
PAROLE	843679	4118935	2876559	14169854

10. táblázat. Átlagolt eredmények a generált adatokon

	C(KICSI)	C(NAGY)	D(KICSI)	D(NAGY)
Greedy	7487543	713456754	7964338	701341876
RTP	6822435	674523268	7657424	731735218
PAROLE	7014345	698764525	7776885	707823932
	E(KICSI)	E(NAGY)	F(KICSI)	F(NAGY)
Greedy	5698614	498238711	6718917	614492728
RTP	5254355	476521848	6396234	592718325
PAROLE	5317817	475632194	6512922	595598238

A visszautasított munkák számát is vizsgáltuk. Nagy különbséget tapasztaltunk az A és B teszt között, míg A tesztben a munkák 25%-a lett visszautasítva, addig B tesztnél ez az érték 42% körüli. Ez várható volt, hiszen a B tesztben a büntetések és a munkák végrehajtási idejének arányának várható értéke kisebb (az A tesztben 0.105, a B tesztben 0.037).

A generált adatokon végzett tesztek esetében a visszautasított munkák aránya 22 és 30 százalék körül mozgott.

Az eredmények alapján megfigyelhető, hogy a Gredy algoritmus kevesebb munkát utasított el, mint az RTP algoritmus. Ezt is vártuk, hiszen a Gredy algoritmus minden az RTP és PAROLE algoritmusok által visszautasított munkát visszautasít.

A költségek két típusára vonatkozóan megállapíthatjuk, hogy Gredy fizeti a legkevesebb büntetést és RTP a legtöbbet, viszont csak kis különbség mutatkozik a befizetett büntetések összegét illetően, a különbség az összes esetben 2%-nál kisebb. A különböző tesztek alatt a büntetések összege a teljes költség 35-45 százaléka volt, kivéve a B teszt esetében, ahol ez a szám 60%.

4.3.2. Az eredmények értékelése

Az eredmények értékelése alapján az alábbi következtetéseket vonhatjuk le:

- Az összes tesztesetben csak kicsi különbség mutatkozik az egyes algoritmusok hatékonyságában. A legnagyobb arány a C(Kicsi) teszt eredményében volt mérhető a Greedy/RTP hányados értékében, mely 1.098. A legtöbb esetben a legjobb és legrosszabb megoldás között 1.05 alatti.
- Az RTP algoritmus 6 teszt esetén adott legjobb megoldást ,a PAROLE algoritmus 4 tesztesetben míg Greedy 2 esetben. A PAROLE algoritmus viszont egyik esetben sem adta az algoritmusok közül a

legrosszabb megoldást. Így elmondható, hogy Parole vagy a legjobb teljesítményt nyújtja, vagy a hatékonysága a másik két algoritmusok között helyezkedik el.

- A tapasztalati vizsgálatok eredményei alapján az is elmondható, hogy a Gredy algoritmus jobb teljesítményt nyújt nagyobb inputon. Úgy gondoljuk ez azért van, mert a Gredy algoritmus nagy inputok esetén könnyeben javíthatja egy rossz döntésének következményeit (amikor egy nagy munkát fogad el, amikor azt el kellene utasítania).

- Feltételeztük, hogy a Gredy algoritmus kevesebb munkát utasít el, így kevesebb büntetést fizet, mint a másik két algoritmus és RTP fizeti a legtöbb büntetést. A tesztek igazolták, hogy feltételezésünk igaz. Úgy tűnik, hogy PAROLE óvatosabban utasítja el a munkákat mint RTP. Azt is láthatjuk azonban, hogy nincs jelentős különbség az visszautasított munkák számában a tesztesetek adatain és ebben a tekintetben mindhárom algoritmus hasonló viselkedést mutat.

Az elemzés eredményeit összefoglalva arra a következtetésre jutottunk, hogy általában a PAROLE algoritmus hatékonysága a többi algoritmus hatékonysága között van, sőt az algoritmus által generált célfüggvényérték mindig az optimális célfüggvényértékhez közelít, ezért abban az esetben mindenképpen jól használható az on-line feladatokon, amikor nincs információnk az input egyes paramétereiről.

5. Fejezet

On-line klaszterezés egy RTLS rendszerben

Ebben a fejezetben matematikai modellt és megoldásokat javaslunk és értékelünk egy on-line klaszterezési problémára. A probléma gyakran felvetődik a folytonos adatfeldolgozási rendszerekben, mint például a valós idejű helymeghatározási rendszerekben is. Cél a lehető legtöbb forrásból származó információt összegyűjteni, de minimalizálni az összegyűjtött információk továbbítása/feldolgozása előtti várakozási időt. A valós idejű helymeghatározó rendszerek adatgyűjtő moduljainak legfontosabb jellemzője, hogy az egy időpontban különböző mérésekből (távolságadatok, beesési szögek) származó adatokból egy csomagot kell kialakítani, amit a pozíciómeghatározáshoz fel tudunk használni. Megadunk egy általános matematikai modellt a probléma kezelésére, és két alapvető algoritmus az NWT és CWT algoritmust, melyeket versenyképességi elemzésnek vetünk alá, majd egy új algoritmust (VWT) mutatunk be a probléma megoldására, mely a problémát megszorítások nélkül kezeli. Ezt az algoritmust teszteljük valós körülmények között, hogy belássuk, ez az algoritmus megszorítások nélküli megoldást ad a problémára.

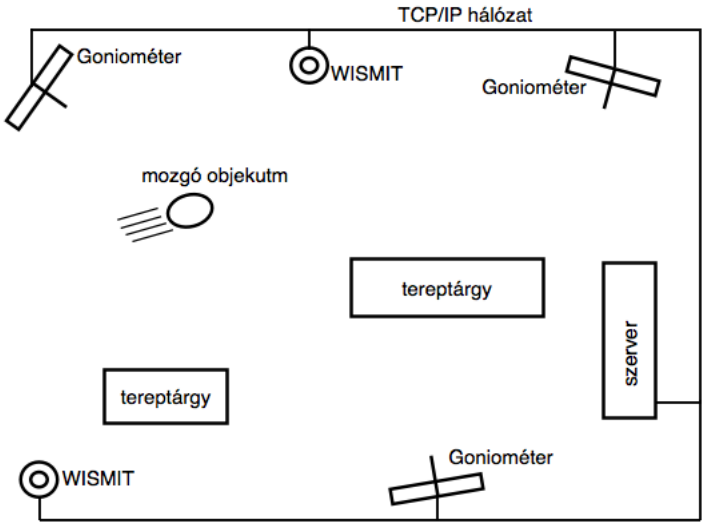
5.1. Előzmények

A fejezetben bemutatott probléma a Fraunhofer IIS intézetben folytatott on-line lokalizációs rendszer fejlesztése közben merült fel, de maga a probléma általános, így az itt bemutatott matematikai modell is az. Az, hogy a származási helyével együtt mutatjuk be a problémát, segít megérteni, hogy a bevezetett megszorítások mit jelentenek egy valós alkalmazásban.

A valós idejű helymeghatározó rendszerek feladata a különböző objektumok pozíciójának nagy pontosságú meghatározása, mely adatokat sok különböző alkalmazásban használnak fel. Ilyen alkalmazások például a szállítmányozás és logisztika, az otthoni idős és betegápolás, a sürgősségi betegellátás támogatása, vagy a különböző szórakoztató és sport alkalmazások amilyen például a Chip-in-the-Ball technológia. Az említett valósidejű helymeghatározási rendszer egy rádiófrekvenciás változatának fejlesztésében vettünk részt a Fraunhofer IIS projektje keretében. A rendszer egy-egy mérőállomása a célobjektum által sugárzott rádiójel beesési szögét továbbá a visszaverődési idejét méri, és a térben különböző pozíciókban elhelyezett mérőállomások mérési adatai alapján határozza meg az objektum pozícióját egy adott pillanatban [8, 9].

A rendszerben az objektumok olyan miniatűr rádióadókkal vannak felszerelve, melyek egy az Intézet által kifejlesztett beltéri működésű valós idejű pozíció-megahtározási rendszer (WISMIT) által feldolgozható periodikusan sugárzott rádiójeleket küldenek. Ezeket a rádiójeleket fogják az

infrastruktúra különböző mérőállomásai, melyek a jelek beesési szögét és a rádiójelet sugárzó objektum távolságát tudják meghatározni minden periódusban (mérési ciklusban). Egy ilyen mérési sorozatot melyen a periodikusan sugárzott rádióimpulzusok egyetlen impulzusát különböző pozíciókban elhelyezett mérőeszközök méréseit értjük, burst-nek nevezünk. A rádió jelek különböző feldolgozható információkat is közvetítenek, egyrészt egy egyedi azonosítót, ami egyértelműen azonosítja azt az objektumot amiből a rádiójel származik, másrészt egy folyamatosan növekvő sorszámot ami a mérési ciklusokat számolja így azonosítva egy-egy mérési ciklust (burst-öt). Ezt a mérési ciklust azonosító sorszámot burst_id-nek nevezzük.



7. ábra. A mérési infrastruktúra sematikus rajza

Az helymeghatározást végző infrastruktúra részeiként működő kü-

lönböző mérést végző eszközök egyrészt az objektum irányát képesek meghatározni a beesési szög mérésével, az ilyen mérést AoA adatnak hívjuk, másrészt a visszaverődési idő méréséből az objektum távolságát, amit RTT értéknek nevezünk. A rendszerben két különböző típusú mérőeszközt lehet installálni. Az egyik a WISMIT eszköz, mely csak a visszaverődési időt méri, majd meghatározza az RTT értékét és továbbítja az adatot. A másik mérőeszköz a Goniométer, mely visszaverődési időt és beesési szöget is mér így mind RRT mind AoA értéket számít és továbbít. A mérési infrastruktúra sematikus rajzát az 5. ábra szemlélteti.

A mérési infrastruktúra pontjain mért adatok a mérőeszközöktől egy TCP/IP hálózaton jutnak el a pozíció számítást végző számítóképhez UDP átviteli protokollon keresztül. Az UDP csomagok a TCP folyammal ellentétben sokkal gyorsabb de kevésbé biztonságos adattovábbítást tesz lehetővé, ugyanis, ha egy mérési eredményt továbbító csomag elveszik vagy sérül nincs idő ennek a detektálására, az esemény visszajelzésére és a csomag újraküldésére. A pozíciót meghatározó algoritmusnak így az adott mérés nélkül kell az objektum pozícióját meghatározni, az adott időpillanatra. További adatvesztés oka lehet, ha az objektum mozgása folyamán olyan pozícióba kerül, hogy egy tereptárgy kitakarja a mérőeszköz elől az objektumot így a mérőeszköz nem tudja mérést végrehajtani, nem kapja meg az objektum által sugárzott rádiójelet, így az adott pillanatban nem is küld a pozíciómeghatározás számára használható mérési eredményt. Tehát több okból nincs garancia arra, hogy minden mérési ciklusban minden mérési eredményt megkapunk. Ha egy mérési ciklusban elegendő mérési adatot összegyűjtünk, a mérési eredményekből a pozíciót kiszámító

algoritmus már ki tudja számítani az objektum pozícióját. Nyilvánvaló, hogy minél több mérési eredményünk van, a pozíció meghatározása annál pontosabb. A nyers mérési eredményektől a következő lépéseken jut el az adat a pozíció kiszámításáig:

1. Nyers adatszűrő
2. Klaszterezés
3. Burst szűrő
4. Pozíciómeghatározás
5. Pozíció szűrő

A lépések többségének az is feladata, hogy a mérések pontatlanságából és egyéb hibákból, fizikai jelenségekből származó zajt és hibát csökkentsék.

Ebben a munkában a második lépésre fókuszálunk, a klaszterezési komponensre. Ennek a lépésnek elsődleges célja, hogy, a burst-höz a lehető legtöbb mérési adatot összegyűjtse, és amilyen hamar csak lehetséges továbbítsa az összegyűjtött adatokat a Burst szűrőhöz majd onnan a pozíció számításhoz. Ehhez on-line módon kell klaszterezni az infrastruktúra mérési pontjaitól kapott adatokat. Itt számolni kell az átviteli hibákkal és zavarokkal, mint a mérési infrastruktúra mérési pontjainak átmeneti takartsága, vagy az egyes hálózati csomagok elvesztése, illetve a változó adattovábbítási és feldolgozási időkkel. Így a klaszterező algoritmus nem várhatja egyszerűen minden mérési eredmény beérkezését, ennél jóval kifinomultabb megoldásra van szükség. A cél az, hogy az algoritmus döntse el mikor érdemes az összegyűjtött adatokat a pozíció számításhoz

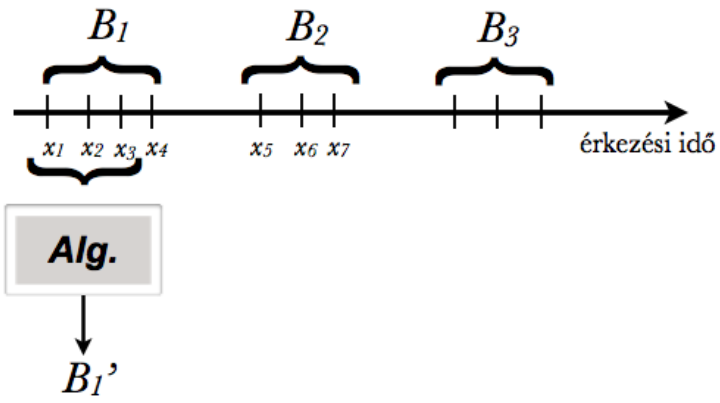
továbbítania. Ehhez két egymásnak ellentmondó célnak kell megfelelnie. Az első, hogy a pozíció számító algoritmusnak az összes rendelkezésre álló mérési adatot meg kell kapnia a mérési pontoktól, a lehető legpontosabb pozíciómeghatározás érdekében. Ugyanakkor a rendszernek nem szabad túl sokáig várnia a mérési adatok begyűjtésére, hiszen minden késedelem az adattovábbításban jelentősen rontja a mozgó objektumra meghatározott pozíció relevanciáját. Egy majdnem pontos pozíció jobb, mint egy teljesen pontos pozíció túl későn. Bemutatunk egy a fenti irányelveket célzó integrált célfüggvényt, és a klaszterezési problémát maximalizációs problémaként definiáljuk. Mivel minden részfolyamat, mint az adatok gyűjtése vagy a pozíció kiszámítása is több objektumra párhuzamosan, egymástól függetlenül végrehajtható, az egyes objektumok pozíciómeghatározását külön problémaként kezeljük, így úgy tekinthetjük, hogy csak egyetlen objektum pozícióját kell meghatározni. Megjegyezzük, hogy a probléma hasonlít az on-line nyugtázási problémához, ahol a cél szintén az, hogy határozzuk meg a nyugtázás elküldésének optimális időpontját. Az egyik bemutatott algoritmus épp az ébresztő algoritmus ötletét használja, amit a nyugtázási probléma megoldására használnak.

Most bemutatjuk a probléma matematikai modelljét, adunk egy célfüggvényt és elemezzük az egyes algoritmusok versenyképességét. Két eset analízisét mutatjuk be, az első akkor használható, ha nincs felső korlát az adatok beérkezési idejére, a második akkor, ha számolhatunk egy felső korláttal, mely idő alatt az összes mért adat beérkezik. Ez a modell jobban leírja a rendszer viselkedését és jobb eredményeket kapunk.

5.2. A probléma matematikai modellje

A klaszterezési probléma inputja egy $X : x_1, \ldots, x_n$ sorozat, a pozíció-meghatározás során mért adatok összessége. A következő jelöléseket fogjuk használni: $Burst(x_i)$ azt a burst_id-t jelöli, mely megadja, hogy x_i mérési adat melyik mérési ciklusból származik. $Node(x_i)$ a valós idejű pozíció-meghatározó rendszer infrastruktúrájának azt a mérőegységét (WISMIT, Goniométer) jelöli, amelyik x_i adatot mérte majd továbbította a szervernek. $Rcpt(x_i)$ az x_i adat beérkezési időpontját jelöli. A mérési ciklus (burst) utolsó és első elemének beérkezési időpontja közötti különbség a burst hossza. Megjegyezzük, hogy a mérési adatokról további információk is rendelkezésre állnak, például az, hogy az adat milyen típusú mérési eredményt tartalmaz (AoA vagy RTT érték), amik a pozíciómeghatározás szempontjtából fontosak, de ezek az információk a klaszterező algoritmus szempontjából lényegtelenek. $B_j = \{x_i | Burst(x_i) = j\}$ halmaz a burst adatcsomag és $B'_j \subseteq B_j$ a burst adatcsomag részhalmaza, mely a pozíció számítás részére elküldött adatokat tartalmazza. Azt az időpontot amikor az algoritmus úgy dönt elküldi B'_j-t a pozíció számításhoz p_j-vel jelöljük, ahol $j = 1, \ldots, b$ és b a burst-ök száma. Ezeket a halmazokat és ezek időbeni elhelyezkedését a 6. ábra mutatja.

Ha a mérési infrastruktúra egyes mérőállásai különböző technikai paraméterekkel rendelkeznek (típus, pozíció) akkor a róluk begyűjtött adatoknak is különböző fontosságuk lehet a pozíció meghatározásában. Ezért minden mérési ponthoz hozzárendelünk egy-egy $w_k > 0$ súlyt, ami a mérési ponton begyűjtött adat fontosságát mutatja. Az általánosság megszorítása



8. ábra. Bemeneti jelek csoportosítása

nélkül feltételezhetjük, hogy $w_1 = \min_{k=1}^{m} w_k$, ahol m a valós idejű pozíciómeghatározási infrastruktúra mérési pontjainak (mérőállomásainak) számát jelöli.

Az algoritmusok értékeléséhez először definiálunk egy célfüggvényt, mely mutatja azok hatékonyságát. Ennek olyannak kell lennie, ami mindkét célt figyelembe veszi: minél kevesebb adat elvesztése mellett minél kevesebb ideig várakozzon, mielőtt a pozíció számítást elkezdi. A célfüggvény meghatározásához a következő jelöléseket használjuk. Minden adattovábbítási p_j időponthoz és k mérési helyhez legyen $e_{jk} = 1$, ha a mérési adatot továbbítottuk a pozíció számításhoz egyébként legyen $e_{jk} = 0$.

Legyen

$$r_j = \max_{i=1}^{n}\{Rcpt(x_i)|x_i \in B'_j\}.$$

A következő maximalizálandó célfüggvényt definiáljuk:

$$f = \sum_{j=1}^{b} \sum_{k=1}^{m} e_{jk} w_k - c \sum_{j=1}^{b} (p_j - r_j).$$

Az első tag azon adatok fontosságát veszi figyelembe a jutalmazáshoz, melyeket elküldünk és így felhasználtunk a pozíció számításhoz, a második tag a küldés késedelmét bünteti egy konstans szorzóval. Ez csak akkor helyes, ha $c > 0$.

Megjegyezzük, hogy lehetséges egy minimalizációs problémát is definiálni a modellhez, amely az elvesztett csomagok súlyozott számának és a késlekedésnek az összege, de ebben az esetben az optimális off-line algoritmus 0 célfüggvényértéket produkálhatna, ami lehetetlenné teszi a versenyképességi elemzést. Ezért döntöttünk úgy, hogy a fejlesztés során a célfüggvény maximalizálandó változatát használjuk.

5.3. Versenyképességi elemzés

5.3.1. Az általános modell

Először azt feltételezzük, hogy nincsenek további megszorítások az inputra vonatkozóan. Meg fogjuk mutatni, hogy erre az esetre nincs az alábbi alsó korlátnál kisebb versenyképességi hányadossal rendelkező algoritmus.

7. Tétel. *Nincs olyan algoritmus melynek a versenyképességi hányadosa kisebb, mint* $\frac{\sum_{k=1}^{m} w_k}{w_1} \geq m$.

Bizonyítás: Tekintsük a következő inputot. Legyen az első adatra a $burst\_id = 1$, a $node\_id = 1$ és az érkezési időpontja 0. Ha az on-line algoritmus $p_1 \geq \frac{w_1}{c}$ időpontot választja az adatok továbbküldésére, akkor a sorozat véget ér és az on-line nyereség nem pozitív, míg az off-line algoritmus használhatja a $p_1^{opt} = 0$ időpontot az adattovábbításhoz, így a nyeresége w_1. Ebben az esetben tehát az algoritmus nem versenyképes. Most tegyük fel, hogy $p_1 < \frac{w_1}{c}$. Ekkor $m-1$ további adat érkezik 1-es burst_id-val és $2, \ldots, m$ node_id-val a $\frac{w_1}{c}$ időpontban. Eben az esetben az on-line algoritmus nyersége legfeljebb w_1, míg az optimális off-line algoritmus a $p_1^{opt} = \frac{w_1}{c}$ időpontban küldi tovább az adatokat, így a nyeresége $\sum_{k=1}^{m} w_k$.■

Az NWT Algoritmus Az előbbi alsó korlát mutatja, hogy a legrosszabb eset, ha a hiányzó adatok épp a már megérkezett adatok elküldése után érkeznek. Ezért az on-line algoritmusnak nincs oka várni további adatokra, miután az első adat megérkezett, érdemes azt azonnal továbbküldenie. Az NWT (No Waiting Time Algorithm) algoritmus ezen az észrevételen alapul, tehát minden burst-höz az első adatot küldi tovább a pozíció kiszámításához, amint az megérkezett. Az algoritmus versenyképességi hányadosa az alábbi.

8. Tétel. *Az NWT algoritmus versenyképességi hányadosa* $\frac{\sum_{k=1}^{m} w_k}{w_1}$.

Bizonyítás: Tekintsük az input egy tetszőleges b számú burst-öt tartalmazó sorozatát. Minden burst-nél a burst első elemét továbbítja az

algoritmus, így a nyereség legalább $b \cdot w_1$. Az optimális off-line nyereség viszont maximum $b \cdot \sum_{k=1}^{m} w_k$. ■

1. Következmény. *NWT a lehető legkisebb versenyképességi aránnyal rendelkező algoritmus az általános modellben.*

5.3.2. A korlátos modell

Ebben a részben egy további megszorítást adunk a bemenet tulajdonságaira, hogy jobban modellezhessük a fizikai rendszer működését. Feltesszük, hogy egy burst nem lehet akármilyen hosszú, azaz ugyanazon burst_id melletti adatcsomagokra az elsőnek és utolsónak beérkezett adat beérkezési időpontjai között eltelt idő nem lehet d-nél nagyobb. Most tegyük fel, hogy $\frac{w_1}{c} > d$, egyébként az általános eset alsó korlátjának bizonyítása alapján itt is beláthatjuk, hogy az NWT algoritmus a lehető legkisebb versenyképességi hányadossal rendelkezik.

9. Tétel. *Nincs olyan algoritmus a korlátos modellben, melynek kisebb a versenyképességi hányadosa mint* $\min\{\frac{w_1}{w_1 - c \cdot d}, \frac{\sum_{k=1}^{n} w_k}{w_1}\}$.

Bizonyítás: Tekintsük a következő inputot. Legyen az első adatra a $burst\_id = 1$, a $node\_id = 1$ és az érkezési időpontja 0. Ha az on-line algoritmus az $p_1 \geq d$ időpontot választja az adatok továbbküldésére, akkor a sorozat véget ér, az on-line nyereség $w_1 - c \cdot p_1 \leq w_1 - c \cdot d$. Az off-line algoritmus a $p_1^{opt} = 0$ időpontot választja az adatok továbbítására, így az off-line nyereség w_1. Ebben az esetben tehát az on-line algoritmus nem lehet jobb, mint $\frac{w_1}{w_1 - c \cdot d}$-versenyképes. Most tegyük fel, hogy $p_1 < d$. Ekkor

$m-1$ további adat érkezik 1-es burst_id-val és $2, \ldots, m$ node_id-val a $\frac{w_1}{c}$ időpontban. Eben az esetben az on-line algoritmus nyeresége legfeljebb w_1, míg az optimális off-line algoritmus a $p_1^{opt} = \frac{w_1}{c}$ időpontban küldi tovább az adatokat, így a nyeresége $\sum_{k=1}^{m} w_k$. ■

A CWT (Constant Waiting Time) algoritmus minden burst-nél az első adatcsomag megérkezése után vár még d ideig, majd elküldi az összes addig megérkezett adatot a pozíció meghatározónak.

10. Tétel. *A CWT algoritmus versenyképességi hányadosa* $\frac{w_1}{w_1 - c \cdot d}$.

Bizonyítás: Tekintsünk egy tetszőleges olyan X bemenetet, mely b burst-öt tartalmaz. A korlátozott várakozási idő miatt az algoritmus minden adatot összegyűjt minden burst_id mellett. Ezért az összes nyereség legalább $G - b \cdot c \cdot d$, ahol G a bejövő adatokon összegyűjthető nyereségek összege. Másfelől az optimális nyereség sem lehet több mint G, és $G \geq b \cdot w_1$. Ezért a CWT algoritmus versenyképességi hányadosa legfeljebb

$$\frac{G}{G - b \cdot c \cdot d} \leq \frac{w_1}{w_1 - c \cdot d}.$$

■

2. Következmény. *A korlátos modellben ha* $\frac{w_1}{w_1 - c \cdot d} \leq \frac{\sum_{k=1}^{m} w_i}{w_1}$, *akkor CWT versenyképességi hányadosa a modellben lehetséges legkisebb versenyképességi hányados, egyébként NWT versenyképességi hányadosa éri el a lehetséges legkisebb értéket.*

5.4. Tapasztalati elemzés

Ahogy az előző részben láttuk, néhány egyszerű algoritmussal elérhetjük az egyes modellekben a lehető legjobb versenyképességi arányt, de a hatékonyság a d paraméter értékének pontos ismeretétől függ. Ebben a részben először bemutatunk egy sokkal kifinomultabb algoritmust, ami a d paraméter rendszerre jellemző értékének ismerete nélkül megpróbálja annak értékét a beérkező adatok alapján folyamatosan megbecsülni, annak időbeni változásával együtt. Egy mérésen alapuló tapasztalati elemzést is bemutatunk, mely az algoritmusok hatékonyságát vizsgálja a tényleges környezet által gyűjtött adatokon, így következtetéseket tudunk levonni azok átlagos viselkedésére vonatkozóan.

5.4.1. A VWT Algoritmus

Ebben az algoritmusban minden j burst_id-hoz tartozik egy $S(j)$ kezdeti időpont, mely az első olyan adat beérkezési időpontja, melynek a burst_id-ja j. Továbbá minden burst-hoz egy adott időpontban egy $DS(j)$ adathalmaz tartozik, mely a burst az adott időpontban már összegyűjtött adatelemeit tartalmazza. (Ezt fogjuk B'_j-vel jelölni, abban az időpontban, amikor továbbítjuk a pozíciószámításhoz.) Az algoritmus egy t változót használ a várakozási idő tárolására, mely azt mutatja, mennyit vár az algoritmus egy burst adatainak begyűjtésére. Az algoritmus ennek a várakozási időnek az optimális értékét próbálja megtanulni. Hasonló paraméter tanuló algoritmusokat láthattunk az előző 2 fejezetben, az

on-line nyugtázási problémára [28], valamint az on-line ütemezésre [31] is. Az algoritmus további két paramétert is használ, a DEC ész az INC paramétereket, az első egy alsó korlátot ad a W változó értékének lehetséges csökkentésére, INC pedig egy biztonsági távolság, melyet a t értékhez adunk abban az esetben, ha W értékét növelni kell. MEM egy paraméter, mely azt határozza meg, az algoritmus milyen hosszan (hány burst-re) tartja nyílván a korábban elküldött burst-ök maximális hosszát.

Az algoritmus a következő szabályokat használja, az adatok elküldési időpontjának meghatározására és az adatok küldésére:

- Ha az aktuális idő $S(j) + W$ és $DS(j)$ még nem lezárt (nincs továbbítva a pozíció számításhoz), akkor az algoritmus lezárja $DS(j)$-t, és elküldi azt a pozíció számító algoritmusnak. Egyébként legyen

$$PD(j) = \max\{0, \min_{a=0}^{MEM-1}\{p_{j-a} - r_{j-a}\} - INC\}$$

  a W változó lehetséges legkisebb csökkentési értéke, mely az utolsó MEM burst minimális hosszát és az INC biztonsági időt veszi figyelembe. Ha $PD(j) \geq DEC$, akkor W-t csökkentjük $PD(j)$-vel.

- Ha az algoritmus a burst összes adatát összegyűjtötte $DS(j)$-be, azaz t várakozási idő elég volt ahhoz, hogy a pozíciómeghatározási infrastruktúra összes mérési adatát összevárjuk, akkor az összegyűjtött adatokat $DS(j)$-t továbbítjuk a pozíció meghatározáshoz. Ez csak úgy történhetett, ha $t \leq S(j) + W$. Ilyenkor az előző pont szerinti a W értékének csökkentése is megtörténhet.

Az algoritmus a következő szabályokat használja az x_i adat megérkezésekor:

- Ha x_i a j burst-höz tartozik, t időpontban érkezik és $DS(j)$ nincs lezárva, akkor bővítjük $DS(j)$-t az új adattal és ellenőrizzük, hogy van-e még a j burst-höz tartozó hiányzó adat az infrastruktúrából.
- Ha $DS(j)$ már lezárt az r_j időpontban W értékét módosítjuk a $t - S(j) + INC$ értékre.

5.4.2. Hatékonysági vizsgálat

Az algoritmus hatékonyságának vizsgálatához az Fraunhofer Intézet szimulációs eszközét használtuk. A szimulációs szoftver egy virtuális objektumot mozgat egy előre megadható útvonalon. Ennek a virtuális objektumnak a pozícióját mérik a virtuális mérési térben elhelyezett különböző mérőeszközök, a mi mérésünkben 3 goniométer (mely RTT és AoA adatokat gyűjt) és 8 távolságmérő (csak RTT adatokat gyűjt). A következő események véletlenszerűen történnek a szimulációs rendszerben a szimuláció futása közben:

- Minden mérési adat, amit a virtuális infrastruktúra mérőeszközei küldenek egy adott valószínűséggel elveszik. (Ez a valószínűség különkülön definiálható a minden mérőeszközre az RTT és AoA adatokra.)
- Az eltelt idő két burst között és a burst-ön belül két adat továbbítása között is normális eloszlású.

A tesztek végrehajtásához a következő 6 bemenetet generáltuk. Az A és B tesztben rendre 68 és 262 burst (mérési sorozat) alatt 952 és

3668 adat érkezett, a burst-ök átlagos hossza 2.04 és 2.6 volt. A C és D tesztekben rövidebb burst-öket generáltunk, az átlagos bort hossz 0.99 és 1.06 volt. A C tesztben 131 burst 1800 adatot míg a D tesztben 138 burst 1932 adatot tartalmazott. Végül az E és F tesztekben sokkal hosszabb burst hosszt alkalmaztunk, az átlagos burst hossz itt rendre 34.07 és 35.9 volt. Az E tesztben 127 burst 1742, míg az F tesztben 879 burst 12079 adatot tartalmazott.

Három különböző CWT algoritmust konstruáltunk. A CWT(1) egy kisebb d konstanst használt, amit a C és D tesztek átlagos burst-hosszaihoz igazítottunk. A CWT(2) egy nagyobb konstanst használt, amit az E és F tesztek átlagos burst hosszaihoz, a CWT (3) pedig ennél még kicsit nagyobb konstanst. A 11. táblázatban az optimális nyerseség és az algoritmus által elért nyereség hányadosát tüntettük fel. Minden i-hez a $w_i = 0.07$ és $c = 0.01$ értéket használtuk a célfüggvényben.

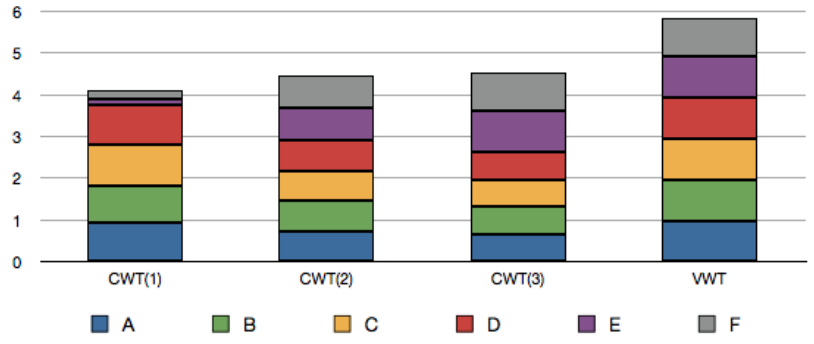
	Teszt A	Teszt B	Teszt C	Teszt D	Teszt E	Teszt F
CWT(1)	0.930	0.883	0.967	0.964	0.142	0.194
CWT(2)	0.730	0.736	0.715	0.721	0.794	0.759
CWT(3)	0.660	0.666	0.643	0.651	0.976	0.936
VWT	0.976	0.985	0.978	0.984	0.985	0.914

11. táblázat. A tesztek eredményei

Az eredmények alapján az alábbi következtetéseket vonhatjuk le:

- Tisztán látszik, hogy a CWT algoritmusok hatékonysága erősen függ a burst-ök átlagos hosszától. A CWT(1) jó eredményeket produkált

az A,B C és D tesztesetben, viszont nagyon rosszakat az E és F tesztesetekben. A CWT(2) és CWT(3) algoritmusok viszont az E és F tesztesetekben voltak hatékonyabbak, a többi tesztesetben viszont gyöngébb teljesítményt nyújtottak. Ez azt jelenti, hogy ezek az algoritmusok csak akkor lennének használhatók valós körülmények között, ha előre több információnk lenne a mérési infrastruktúra felépítéséből adódó jellemzőkről amik a burst hosszát befolyásolják (és ezek a futás során sem változnának). Ebben az esetben ugyanis ezek az algoritmusok is jól működhetnek, a hatékonyságuk minden ilyen tesztesetben 0.9 feletti.



9. ábra. Az egyes algoritmusok hatékonysága

- A VWT algoritmus ezzel ellentétben minden tesztesetben hatékonyan működött, a nélkül, hogy előre ismertük volna az infrastruktúra fizikai jellemzőiből és felépítéséből adódó maximális várakozási időt. Az algoritmus hatékonysága a legrosszabb esetben is 0.914, az átlagos arány pedig 0.97 volt.

A 7. ábrán jól látható, hogy az új paraméter tanuló algoritmus (VWT) lényegesen jobb hatékonysággal működik az átlagos esetben, mint a legjobb versenyképességi aránnyal rendelkező CWT algoritmus, a nélkül is, hogy előzetes információja lenne a mérési infrastruktúra fizikai jellemzőiből adódó maximális burst-hosszról.

5.5. Áttekintés és nyitott kérdések

Ebben a fejezetben egy on-line optimalizációs modellt definiáltunk, az on-line klaszterezés problémájában, mely probléma a valós idejű helymeghatározási rendszerek fejlesztésénél merült fel. Bemutattunk optimális on-line algoritmusokat abban az értelemben, hogy a lehető legjobb versenyképességi arányt érték el, és bemutattunk egy jóval kifinomultabb algoritmust is, mely képes megtanulni a fizikai infrastruktúra felépítéséből adódó maximális várakozási időt is, azaz a burst-ök várható hosszát. Egy szimulált mérés segítségével bemutattuk, hogy ez a tanuló algoritmus jól használható a valós körülmények között, akkor is ha nincs információnk a burst-ök várható hosszáról és lényegesen jobb hatékonysággal működik az átlagos esetben, mint az optimális versenyképességi hányadossal rendelkező algoritmusok. Több további érdekes kérdés is felmerül. Érdekes lehet például más, bonyolultabb célfüggvények vizsgálata is. További érdekes kérdés, mivel ez a modell feltételezi, hogy az adatok tartalmazzák a burst_id-t, de néhány alkalmazásban nem kapjuk meg ezt az információt, hogy milyen algoritmusokat lehet alkalmazni abban az esetben ha a csomagok nem tartalmazzák a burst-id adattagokat.

Összegzés

Disszertációmban az on-line algoritmusok területén, a számítógépes hálózatok és adatfeldolgozás on-line problémáival foglalkozom, így az az elmúlt néhány évben ezen területeken végzett kutatásaim eredményeinek összefoglalását tartalmazza. A téma korszerűségét és fontosságát a területen megjelent nagy számú publikáció is mutatja.

Az első fejezetben az on-line algoritmusok témakörének alapfogalmait és a versenyképességi elemzés alapjait mutatom be.

A 2. fejezetben a nyugtázási problémát ismertetem, majd definiálom a probléma vizsgálatára legáltalánosabban használt matematikai modellt, és bemutatok egy alapvető optimális algoritmust, ami nem csak a 2. fejezetben bemutatott problémák megoldásának alapötletéül szolgál, hanem később a 3. és 5. fejezetben is fontos kiindulópont. Bemutatok a problémára egy új idő szerinti előre néző félig on-line változatot. A problémát vizsgáljuk mindkét a szakirodalomban használt célfüggvény esetén. Az f_{max} célfüggvény esetén egy optimális félig on-line algoritmust adunk meg, abban az értelemben, hogy a lehető legkisebb versenyképességi hányadost éri el. Az f_{sum} célfüggvény esetén egy $1 + O(1/c)$ vesenyképes algoritmust adunk meg, ahol c az előrenéző intervallum hossza. Szintén igazoljuk, hogy nincs olyan félig on-line algoritmus, mely versenyképessége kisebb, mint $1 + \Omega(1/c^2)$.

A 3. fejezetben a legrosszabb eset elemzésen túllépve, a 2. fejezetben már tárgyalt problémákat véletlenül generált és valós adatokon vizsgáljuk.

Ismertetünk egy-egy új, az optimális paraméter tanulásán alapuló algoritmust, és megvizsgáljuk, hogy ezek az algoritmusok milyen hatékonysággal működnek. Megmutatjuk, hogy átlagos esetben ezek az algoritmusok a minimális versenyképességi hányados értékéből adódónál lényegesen jobb célfüggvény értéket is elérhetnek.

A 4. fejezetben egy másik on-line problémának, a visszautasításos on-line ütemezésnek a vizsgálatát végezzük el, az előző fejezetben tárgyalt módszerhez hasonlóan. Bemutatjuk a probléma megoldására már ismert mohó algoritmust, majd egy annál lényegesen jobb, versenyképesség szempontjából optimális algoritmust is. Ebből kiindulva bemutatunk egy, az optimális paraméter-érték tanulásán alapuló algoritmust és megvizsgáljuk, hogy generált és valós adatokon az új algoritmusunk milyen hatékonysággal működik. A tesztek eredményeit értékelve megmutatjuk, hogy az új algoritmus minden inputra az optimálishoz közeli hatékonysággal működik.

Az 5. fejezetben egy új, a valós idejű pozíciómeghatározás fejlesztése közben felmerült problémát, egy on-line klaszterezési problémát mutatunk be. Megadunk két, a probléma kezelésére jól használható matematikai modellt, és ezekre optimális on-line algoritmusokat fejlesztünk ki, abban az értelemben, hogy azok a lehető legkisebb versenyképességi hányadossal rendelkeznek. Továbbá megadunk egy a gyakorlat szempontjából jól használható algoritmust a probléma megoldására és generált és valós inputokon elemezzük az új algoritmusok hatékonyságát. Belátjuk, hogy esetünkben, és az előző két fejezet eredményeivel egybevetve talán általában is, a paraméter tanulásán alapuló algoritmusok jól használhatók

abban az esetben, ha nincs előzetes információnk az input egyes paramétereiről, és a gyakorlatban nem feltétlenül a versenyképességi elemzés szempontjából optimális algoritmus a leghatékonyabb is egyben.

Summary

In this dissertation we deals with some online problems of computer networks and online scheduling with rejection. We summarize the results of our research that has been done over the past few years. The timeliness and relevance of these topics are shown by the large number of publications in the field.

Chapter 1 introduces the basic concepts of the field of online algorithms and the definitions of competitive analysis. Chapter 2 details the data acknowledgement problem and defines the most commonly applied mathematical model for it. We present here a well-known alarming algorithm which has the smallest possible competitive ratio. This algorithm will also play an important role later in Chapters 3 and 5. Then we consider a new semi online version of the problem which is based on the time lookahead property. We study both of the usually objective functions. In the case of objective function f_{max} we present an optimal semi online algorithm in the sense that achieves the smallest possible competitive ratio. In the case of objective f_{sum} we give an algorithm which is $1 + O(1/c)$-competitive where c is the length of the lookahead interval, and we show that no semi online algorithm can have smaller competitive ratio than $1 + \Omega(1/c^2)$.

In Chapter 3 we study the data acknowledgment problem and the data acknowledgment problem with lookahead in the average case. A pair of new algorithms are introduced, which are based on learning some optimal parameters. The efficiency of the algorithms is investigated on randomly generated and also on real data. It is pointed out that on average, these

algorithms can attain a much better objective function value than the algorithms which have the smallest possible competitive ratio.

In Chapter 4 we study an another online problem, the online scheduling with rejection. We use similar methods to the methods discussed in the previous chapter. First we recall a greedy algorithm and a more sophisticated RTP (reject total penalty) algorithm for the solution of the problem. RTP achieves the smallest possible competitive ratio Using the idea of parameter learning we introduced a new online algorithm called PAROLE for the solution of the problem. The efficiency of the algorithms is again investigated on randomly generated and also on real data. Using the results of the tests, it is demonstrated that the new algorithm works with near-optimal efficiency with all inputs.

Chapter 5 introduces an online clustering problem arisen during the development of a real-time position location system. Two mathematical models - which can be efficiently used to tackle the problem are provided. In both models we found optimal online algorithms which achieves the smallest competitive ratio. optimality based on their competitiveness. We also define a further algorithm based on parameter learning. The algorithms are evaluated in the simulation system of the Fraunhofer Institute and again we obtain that the new algorithm has better performance on real data than the ones with the best competitive ratio.

Hivatkozások

[1] S. Albers and H. Bals. Dynamic TCP acknowledgement: Penalizing long delays *SIAM J. Discrete Math.* 19(4) 938–951, 2005.

[2] S. Albers, and B. Schrőder: An Experimental Study of online Scheduling Algorithms. *ACM Journal of Experimental Algorithms*, article 3, 2002.

[3] L. Allulli, G. Ausiello, L. Laura, On the power of lookahead in online vehicle routing problems *Lecture Notes in Computer Science* 3595, Springer, Berlin, 728–736, 2005.

[4] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie: Multiprocessor scheduling with rejection, *SIAM Journal on Discrete Mathematics*, 13:64–78, 2000.

[5] V. Bilo, M. Flammini and R. Giovannelli: Experimental analysis of online algorithms for the bicriteria scheduling problem, *J. Parallel Distrib. Comput.*, 64:1086–1100, 2004.

[6] E. Bittner, Cs. Imreh, J. Nagy-György, The online k-server problem with rejection, submitted to *Discrete Optimization*

[7] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis *Cambridge University Press*, 1998.

[8] M. Brugger, T. Christ, F. Kemeth, S. Nagy, M. Schaefer, M.M. Pietrzyk, The FMCW technology-based indoor localization system, *Proceedings of Ubiquitous Positioning Indoor Navigation and Location Based Service (UPINLBS)*, Helsinki, Finnland, 2010 pp. 1–6.

[9] M. Brugger, F. Kemeth, Locating rate adaptation by evaluating movement specific parameters, *Proceedings of 2010 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Anaheim, USA, 2010 pp. 127–133.

[10] J. R. Correa and M. Wagner: LP-Based online Scheduling: From Single to Parallel Machines *Mathematical Programming*, to appear (prelinimary version in Proceedings of IPCO 2005 LNCS 3509, 186–209).

[11] Gy. Dósa and Y. He: Scheduling with machine cost and rejection, *Journal of Combinatorial Optimization*, 12(4):337–350, 2006.

[12] Gy. Dósa, Y. He, Bin packing problems with rejection penalties and their dual problems, *Information and Computation*, 204, 2006, 795–815

[13] Gy. Dósa, Cs. Imreh, *On-line algoritmusok* Typotex 2011.

[14] D. R. Dooly, S. A. Goldman, S. D. Scott: Online analysis of the TCP acknowledgment delay problem. *J. ACM* 48(2) 243–273, 2001.

[15] D. V. Engels, D. R. Karger, S. G. Kolliopoulos, S. Sengupta, R. N. Uma and J. Wein: Techniques for scheduling with rejection, *Journal of Algorithms*, 49:175–191, 2003.

[16] L. Epstein and J. Sgall: Approximation schemes for scheduling on uniformly related and identical parallel machines, *Algorithmica*, 39(1):43–57, 2004.

[17] L. Epstein, A. Kesselman: On the remote server problem or more about TCP acknowledgments. *Theoretical Computer Science* 369(1-3) 285–299, 2006.

[18] L. Epstein, A. Levin, G. J. Woeginger, Graph coloring with rejection, *Journal of Computer and System Sciences*, 77(2), 2011, 439-447.

[19] L. Epstein, Bin packing with rejection revisited, *Algorithmica*, 56(4), 2010, 505–528.

[20] L. Epstein, Cs. Imreh, A. Levin, J. Nagy-György, Online File Caching with Rejection Penalties, *Algorithmica*, to appear, doi 10.1007/s00453-013-9793-0

[21] A. Fiat, G.J. Woeginger (eds.) Online algorithms: The State of the Art, LNCS 1442, *Springer-Verlag, Berlin*, 1998.

[22] J. S. Frederiksen, K. S. Larsen, J. Noga, P. Uthaisombut: Dynamic TCP acknowledgment in the LogP model. *Journal of Algorithms* 48(2) 407–428, 2003.

[23] R. L. Graham: Bounds for certain multiprocessor anomalies, *Bell System Technical Journal*, 45:1563–1581, 1966.

[24] Cs. Imreh, An online scheduling algorithm for a two-layer multiprocessor architecture, *Acta Cybernetica*, 15, (2001), 163–172

[25] Cs. Imreh: Scheduling problems on two sets of identical machines, *Computing*, 70:277–294, 2003.

[26] Cs. Imreh: Competitive analysis, *Algorithms of Informatics Volume 1*, pages 395–428, Budapest 2007.

[27] Cs. Imreh, T. Németh, On time lookahead algorithms for the online data acknowledgement problem, *32nd International Symposium on Mathematical Foundations of Computer Science, LNCS 4708*, Cesky Krumlov, 2007, pp. 288-297.

[28] Cs. Imreh and T. Németh, Parameter learning algorithm for the online data acknowledgment problem, *Optim. Methods Softw.*, 26, 3 (2011) 397–404.

[29] A. R. Karlin, C. Kenyon and D. Randall, Dynamic TCP acknowledgement and other stories about e/(e-1), *Algorithmica* 36 (2003), pp. 209–224.

[30] J. Nagy-György and Cs. Imreh, Online scheduling with machine cost and rejection, *Discrete Applied Mathematics*, 155: 2546–2554, 2007.

[31] T. Németh, Cs. Imreh, Parameter learning online algorithm for multiprocessor scheduling with rejection, *Acta Cybernetica* 19(1) (2009), pp. 125–133.

[32] T. Németh, B. Gyekiczki, Cs. Imreh, Parameter Learning in Lookahead Online Algorithms for Data Acknowledgment, *IEEE International Symposium on Logistics and Industrial Informatics*, 2011

[33] T. Németh, S. Nagy, Cs. Imreh Online data clustering algorithms in an RTLS system *Acta Universitatis Sapientiae, Informatica*, 5, 1 (2013) 5-15.

[34] J. Noga, S. S. Seiden, G. J. Woeginger, A faster off-line algorithm for the TCP acknowledgement problem. *Information Processing Letters* 81(2) 71–73, 2002

[35] S.S. Seiden, A guessing game and randomized online algorithms, *Proceedings of STOC 2000*, 592–601.

[36] S. S. Seiden: Preemptive Multiprocessor Scheduling with Rejection, *Theoretical Computer Science*, 262:437–458, 2001.

[37] J. Sgall: Online scheduling, *Online algorithms: The State of the Art, LNCS 1442*, pages 196-231, Berlin, 1998, Springer Verlag.

[38] W. R. Stevens, TCP/IP Illustrated, Volume I. The protocols, *Addison Wesley, Reading*, 1994