Kommunikáló rendszerek teljesítménytesztelése és teljesítménynövelése

Erős Levente

MSc., okl. mérnök-informatikus

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Informatikai Tudományok Doktori Iskola
Távközlési és Médiainformatikai Tanszék

Tézisfüzet

Témavezető:

Dr. Csöndes Tibor

címzetes egyetemi docens

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Budapest, Magyarország

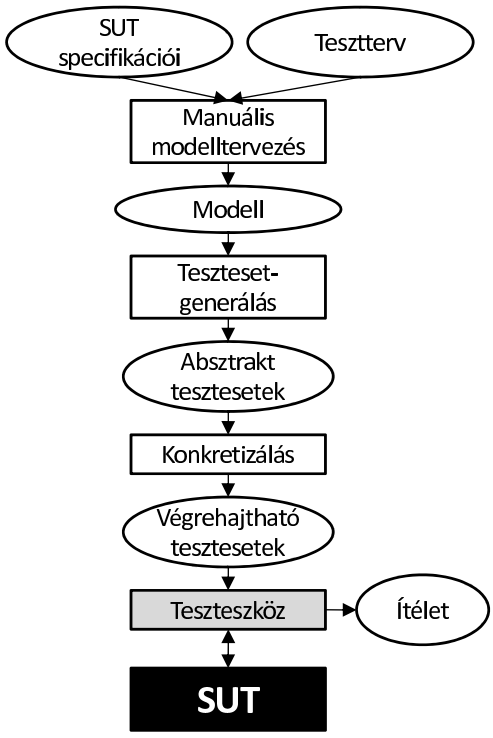
2012.

1. Bevezető

A tesztelés kulcsszerepet játszik különböző kommunikációs protokollokat implementáló kommunikáló rendszerek fejlesztésében. A rendszer implementációját követően a külvilág felé fekete doboznak tekinthetjük, és az implementációt követő tesztek kizárólag a különböző stimulusokra adott válaszok alapján döntik el, hogy a rendszer megfelel-e a vele szemben támasztott követelményeknek.

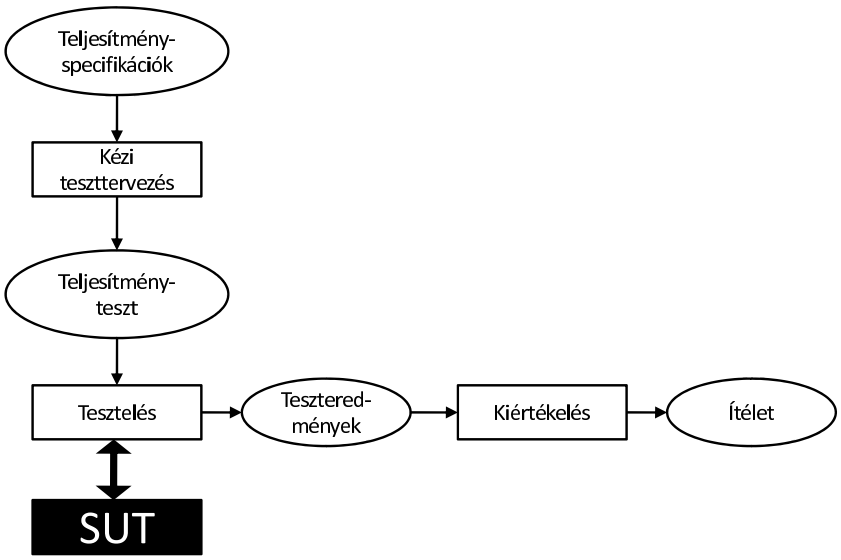
A telekommunikáció területén alkalmazott konformanciatesztek azt vizsgálják, hogy a tesztelt rendszer (angolul system under test, SUT) azt a kommunikációs protokolt valósítja-e meg, amelyet a követelmények szerint meg kell valósítania. Ezzel szemben a teljesítménytesztek a SUT különféle teljesítményjellemzőit mérik.

Az elmúlt évtizedek során a konformanciatesztelés óriási utat tett meg a teljes mértékben manuális, ad-hoc teszteléstől a modell alapú tesztelésig [1,2] (1. ábra), és mára már kiforrott elméleti hátteret tudhat magáénak. Ez magában foglalja a rendszer funkcionális működésének modellezésére szolgáló formális leírótechnikákat [3–5] és félautomata konformanciatesztelési eljárásokat [6–13], csakúgy, mint tesztkészlet-karbantartó eljárásokat [14–16].



1. ábra. Modell alapú tesztelés

A fekete doboz alapú teljesítménytesztelés elméleti háttere korántsem olyan fejlett, mint a konformanciatesztelésé. Míg az idevonatkozó irodalomban fellelhetők teljesítménymodellezésről [20–23] és teljesítményteszt-végrehajtásról szóló cikkek is [24–26], a teljesíménytesztek tervezése általában manuálisan, ad-hoc módon történik, bármilyen elméleti háttér nélkül (2. ábra). A fentiek következtében egyik célom az volt, hogy létrehozzak egy automatikus teljesítménytesztelési eljárást.



2. ábra. Teljesítménytesztelés

A teljesítménytesztelés egy további fontos, nyitott kérdése az, hogy hogyan használjuk ki a tesztkörnyezet kapacitását minél jobban. A kérdés azért merülhet fel, mert egy teljesítményteszt során egy adott célra kifejlesztett tesztelt rendszert leterhelő forgalmat kell generálnunk univerzális hardverkiépítésű, több teljesítményteszt során újra és újra felhasznált tesztkörnyezet segítségével. A probléma megoldására a tesztkörnyezet több tesztelő hosztból áll, amelyek együtt képesek a megfelelő terhelés előállítására. A terhelést ún. terhelésgenerátorok állítják elő, amelyeket a tesztelő hosztok futtatnak. Annak érdekében, hogy a tesztkörnyezet a megfelelő terhelés előállítására legyen képes, olyan eljárásokra van szükség, amelyek a terhelésgenerátorokat úgy rendelik a később őket futtató tesztelő hosztokhoz, hogy azok kapacitását minél jobban kihasználják.

2. Kutatási célkitűzések

Célom az volt, hogy a fekete doboz alapú teljesítménytesztelés fent említett problémáira megoldást nyújtsak.

Értekezésem első részében olyan eljárások létrehozása a célom, amelyek a tesztkörnyezet hosztjai között úgy osztják el a terhelésgenerátorokat, hogy előbbiek kapacitását a lehető legjobban kihasználják.

Az értekezés második részében a cél egy hatékony, automatikus teljesítménytesztelési eljárás kidolgozása, amely el tudja dönteni, hogy a SUT képes-e a tőle megkövetelt számú kérésüzenet másodpercenkénti feldolgozására, miközben adott számú felhasználót szolgál ki párhuzamosan.

Értekezésem harmadik részében olyan eljárások kidolgozása volt a célom, amelyek a teszten megbukott rendszer teljesítményét növelik a teljesítménykövetelményekben meghatározott szintre, minimális költség befektetése mellett.

3. Módszertan

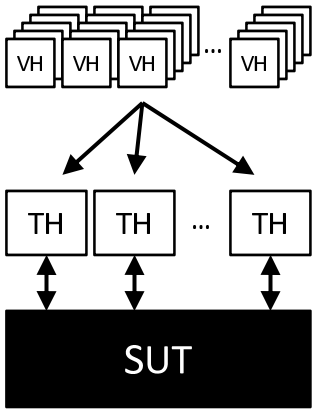
Az első és harmadik téziscsoport problémáinak bonyolultságát analitikusan bizonyítottam. A problémákat ezek után egészértékű lineáris programokként írtam fel, és heurisztikus megoldásokat adtam rájuk. Ezen eljárások teljesítményét szimulációkkal vizsgáltam.

A második téziscsoportban tárgyalt probléma megoldására létrehoztam egy matematikai modellt és két, ezen dolgozó eljárást. Ezen eljárások egyikének helyességét analitikusan, míg a másik eljárás helyességét kísérletekkel igazoltam.

4. Új eredmények

4.1. Terheléselosztás a teljesítményteszt-környezetben

A korábban leírtaknak megfelelően az univerzális hardverkiépítésű teljesítménytesztkörnyezetnek viszonylag nagy terhelést kell generálnia a SUT mint célhardver felé. Ennek érdekében a teszkörnyezet több hosztból (TH) áll, amelyek együtt képesek a kívánt terhelés előállítására.



3. ábra. VH-k és TH-k összerendelése

A terhelésgenerátorok (vagy virtuális hosztok, VH-k) a TH-kon futnak, és ezek száma rendszerint nagyobb, mint a tesztelő hosztoké. Valamennyi tesztelő hosztnak és terhelésgenerátornak adott a kapacitása. Egy terhelésgenerátort akkor rendelhetünk hozzá egy tesztelő hoszthoz, ha a terhelésgenerátor kapacitásánál nagyobb a tesztelő hoszt szabad kapacitása, a terhelésgenerátor teljes futásidejére. Valamennyi VH-t hozzá kell rendelni egy TH-hoz, vagy el kell dobni (3. ábra). A VH-k hozzárendelését úgy kell elvégezni, hogy maximalizáljuk a TH-k átlagos kihasználtságát (tehát hogy maximalizáljuk a tesztkörnyezet által generált terhelést). A fent leírt problémára innentől fogva terheléselosztási problémaként fogunk hivatkozni.

1. tézis. *Bebizonyítottam a terheléselosztási probléma NP-teljességét. Felírtam a terheléselosztási problémát egészértékű lineáris programként, és definiáltam egy olyan eljárást, amely a problémát egészértékű lineáris programok sorozatára transzformálva oldja meg. Adtam egy heurisztikus algoritmust a probléma megoldására.*

A terheléselosztási problémát diszkrét időtengelyen definiáltam, amely elemi egysége az időrés. A problémát a következőképp formalizáltam:

Adott $\mathcal{TH} = \{TH_i\}$, a TH-k halmaza illetve $\mathcal{VH} = \{VH_i\}$, a VH-k halmaza. A TH-knak egyetlen attribútuma van. $TH_i = (TC_i)$, ahol TC_i TH_i teljes kapacitását jelenti. A VH-knak három atribútuma van. $VH_i = (ST_i, RT_i, C_i)$. ST_i VH_i kezdőideje (tehát az az időrés, amelyben VH_i elindul), RT_i VH_i futásideje (ami a VH_i futásához szükséges időrések száma), végül C_i VH_i kapacitását jelenti.

A megoldandó probléma a következő: Valamennyi $VH_i \in \mathcal{VH}$ terhelésgenerátorra adjuk meg a $\sigma \in \mathcal{VH} \rightarrow \mathcal{TH}$ függvény értékét a $\mathcal{D} = \mathcal{TH} \bigcup \{\emptyset\}$ értékkészletből úgy, hogy a lenti 1. és 2. egyenlőtlenségek igazak legyenek. Amennyiben TH_j-t választjuk $\sigma(VH_i)$ értékéül, az annyit jelent, hogy VH_i-t TH_j-hez rendeljük, míg ha $\sigma(VH_i)$ értékét $\emptyset$-nak választjuk, akkor eldobjuk VH_i-t. Az 1. egyenlőtlenségben U a TH-kapacitás alsó korlátja. Ha u a teljes TH-kihasználtság értéke (a felhasznált és a rendelkezésre álló TH-kapacitás hányadosa), akkor $U = u \sum_{i=1}^{|\mathcal{TH}|} TC_i t_{max}$. Az egyenlőtlenségben és a fejezet további részében, t_{max} az utolsó időrést jelöli, azaz $(t_{max} = \max_{k:\, VH_k \in \mathcal{VH}}(ST_k + RT_k - 1))$.

$$\sum_{i=1}^{|\mathcal{TH}|} \sum_{j=1}^{t_{max}} \sum_{\substack{k:\sigma(VH_k)=TH_i \wedge \\ \wedge ST_k \leq j \wedge \\ \wedge j \leq ST_k + RT_k - 1}} C_k \geq U \tag{1}$$

$$\forall (i : TH_i \in \mathcal{TH}) : \forall (j = 1, \ldots, t_{max}) : \sum_{\substack{k:\sigma(VH_k)=TH_i \wedge \\ \wedge ST_k \leq j \wedge \\ \wedge j \leq ST_k + RT_k - 1}} C_k \leq TC_i \tag{2}$$

Az 1. egyenlőtlenség annyit jelent, hogy a TH-k teljes kihasználtságának az U alsó korlát felett kell lennie, míg a 2. egyenlőtlenség szerint az egyes TH_i-k teljes kapacitását egyik időrésben sem haladhatja meg a rajtuk futó VH-k kapacitásainak összege.

1.1. altézis. *[J1, C1] Bebizonyítottam a terheléselosztási probléma NP-teljességét, és felírtam azt egészértékű lineáris programként. Definiáltam egy heurisztikát, a probléma megoldására. A heurisztika a probléma időtengelyét időablakokra osztja fel, majd a problémát az egyes időablakokra felírt egészértékű lineáris programok sorozatának segítségével oldja meg.*

A probléma NP-teljességét a megegyező költség- és súlyfüggvényeket használó NP-teljes hátizsákprobléma egy tetszőleges példányának visszavezetésével igazoltam [27]. A visszavezetés értekezésem 2.2. fejezetében található meg. Mivel a terheléselosztási probléma NP-teljes, optimális megoldásának megtalálásához fel kell írni egészértékű lineáris programként (angolul integer linear program, ILP), amely esetünkben egy bináris lineáris program (angolul binary linear program, BLP).

Mielőtt felírnánk a problémát egészértékű lineáris programként, a formulák könnyebb olvashatósága érdekében vezessük be az a_{kj} bináris változót:

$$a_{kj} = \begin{cases} 0 & \text{, ha } ST_k \leq j \wedge ST_k + RT_k - 1 \geq j \\ 1 & \text{, egyébként} \end{cases} \tag{3}$$

Ezen kívül szükségünk van egy, az előzőeken felüli TH bevezetésére is. Ezt a továbbiakban $TH_{|\mathcal{TH}|+1}$ fogja jelölni. VH_k $TH_{|\mathcal{TH}|+1}$-hez történő hozzárendelése VH_k eldobásának felel meg. Ezen új TH kapacitása végtelen, illetve a gyakorlatban az összes futtatandó VH összkapacitásával egyezik meg (annak érdekében, hogy biztosan valamennyi VH-t hozzá tudjuk rendelni). Az előzőek formálisan leírva:

$$\begin{gathered} \mathcal{TH}' = \mathcal{TH} \bigcup \{ TH_{|\mathcal{TH}|+1} \}, \text{ ahol} \\ TH_{|\mathcal{TH}|+1} = (\, TC_{|\mathcal{TH}|+1}) \text{ és} \\ TC_{|\mathcal{TH}|+1} = \sum_{k:\, VH_k \in \mathcal{VH}} C_k \end{gathered} \tag{4}$$

A terheléselosztási probléma BLP-felírása a következő. Az ismeretlen változók, amelyek értékeit a BLP megoldása során meg kell találni, az s_{ki} változók. Az s_{ki} változó értéke 1, ha VH_k-t TH_i-hez rendeljük, egyébként 0.

Maximalizálandó célfüggvény:

$$\sum_{i=1}^{|\mathcal{TH}|} \sum_{j=1}^{t_{max}} \sum_{k=1}^{|\mathcal{VH}|} a_{kj} s_{ki} C_k \tag{5}$$

Megkötések:

$$\forall (i : TH_i \in \mathcal{TH}') : \forall (l : VH_l \in \mathcal{VH}) : \sum_{k=1}^{|\mathcal{VH}|} a_{kST_l} s_{ki} C_k \leq TC_i \tag{6}$$

$$\forall (k : VH_k \in \mathcal{VH}) : \sum_{i=1}^{|\mathcal{TH}'|} s_{ki} = 1 \tag{7}$$

$$\forall (k : VH_k \in \mathcal{VH}) : \forall (i : TH_i \in \mathcal{TH}') : s_{ki} \in \{0, 1\} \tag{8}$$

Ahogy az értekezésem 2.5. fejezetében bemutatott, különböző paraméterekkel lefuttatott szimulációk igazolták, sok esetben gyakorlatilag nem lehet megoldani a fenti BLP-t, annak nagy futásideje miatt. Ezért létrehoztam egy heurisztikus algoritmust, amely a problémát BLP-k sorozataként oldja meg.

Az algoritmus az időtengelyt W méretű időablakokra osztja, majd valamennyi időablakra felír egy bináris lineáris programot, amely egy részproblémának felel meg.

Az n. időablakra vonatkozó alprobléma felírását a 9-12. formulák adják meg. A felírásban S_k azon TH sorszáma, amelyhez VH_k-t rendeltük az n. időablakot megelőzően. Amennyiben VH_k az n. időablakot megelőző időablak után indul, S_k értéke -1. Ha S_k értéke $|\mathcal{TH}| + 1$, az annyit jelent, hogy VH_k-t eldobtuk.

1. algoritmus: A terheléselosztási probléma ILP alapú megoldása

bemenet: $\mathcal{TH}$, $\mathcal{VH}$, W

kimenet : $\bigcup_{k:\, VH_k \in \mathcal{VH}} \{S_k\}$

```
1 foreach VH_k ∈ VH do
2 |   S_k := −1;
3 n := 1;
4 while (n − 1)W + 1 ≤ t_max do
5 |   BLP_n felírása és megoldása;
6 |   foreach k : VH_k ∈ VH ∧ ST_k ≥ (n − 1)W + 1 ∧ ST_k ≤ nW do
7 |   |   foreach i : TH_i ∈ TH do
8 |   |   |   if s_ki = 1 then
9 |   |   |   |   S_k := i
```

A VH-k hozzárendelését az 1. algoritmus végzi, amely a következő BLP-felírást használja. A 9-12. formulák által az n. időablakra meghatározott BLP-t az algoritmus leírásában BLP_n jelöli.

Maximalizálandó célfüggvény:

$$\sum_{i=1}^{|\mathcal{TH}|} \sum_{j=(n-1)W+1}^{nW} \sum_{\substack{k:\, VH_k \in \mathcal{VH} \wedge \\ \wedge ST_k \geq (n-1)W+1 \wedge \\ \wedge ST_k \leq nW}} a_{kj} s_{ki} C_k \tag{9}$$

Megkötések:

$$\begin{gathered} \forall (i : TH_i \in \mathcal{TH}') : \\ \forall (l : VH_l \in \mathcal{VH} \wedge ST_l \geq (n-1)W + 1 \wedge ST_l \leq nW) : \\ \sum_{\substack{k:\, VH_k \in \mathcal{VH} \wedge \\ \wedge ST_k \geq (n-1)W+1 \wedge \\ \wedge ST_k \leq nW}} a_{kST_l} s_{ki} C_k \leq TC_i - \sum_{\substack{k:\, VH_k \in \mathcal{VH} \wedge \\ \wedge S_k = i}} a_{kST_l} C_k \end{gathered} \tag{10}$$

$$\forall(k : VH_k \in \mathcal{VH} \wedge ST_k \geq (n-1)W + 1 \wedge ST_k \leq nW) : \sum_{i=1}^{|\mathcal{TH}'|} s_{ki} = 1 \tag{11}$$

$$\begin{gathered}\forall(k : VH_k \in \mathcal{VH} \wedge ST_k \geq (n-1)W + 1 \wedge ST_k \leq nW) : \\ \forall(i : TH_i \in \mathcal{TH}') : s_{ki} \in \{0, 1\}\end{gathered} \tag{12}$$

Az 1. algoritmus bemenete a $\mathcal{TH}$ és a $\mathcal{VH}$ halmaz, kimenetei pedig az egyes VH-khoz tartozó S_k értékek.

Az 1. és 2. sorokban az algoritmus valamennyi S_k változót -1 értékkel inicializálja. A 4. sortól kezdve az algoritmus iterációkat futtat, időablakonként egyet. Egy iteráción belül, az 5. sorban felírja, és megoldja az aktuális időablakra vonatkozó ILP-t. A 6-9. sorokban a kiszámított s_{ki} értékek alapján értéket ad azon VH-k S_k változóinak, amelyeket az aktuális időablakban rendeltünk hozzá valamely TH-hoz, vagy dobtunk el. Az algoritmus futásának végére valamennyi S_k változónak van értéke.

Az értekezésem 2.5. fejezetében bemutatott szimulációk alapján vannak olyan helyzetek, amelyekben nem csak az eredeti problémát definiáló ILP, hanem az ILP-sorozattal dolgozó heurisztika sem oldható meg nagy futásideje miatt. Ezen helyzetekre hoztam létre a következőkben bemutatott heurisztikát.

1.2. altézis. *[J1, C1] Létrehoztam egy ládapakolás alapú heurisztikus algoritmust a terheléselosztási probléma megoldására. Az algoritmus adott hosszúságú időablakokra osztja az időtengelyt, majd az egyes időablakokban megoldandó problémákat ládapakolási feladatokként kezeli, és oldja meg.*

A heurisztika alapötlete az, hogy az egymáshoz időben közel induló VH-k befolyásolják egymás TH-khoz való rendelhetőségét hasonlóan ahhoz, ahogy a ládapakolási probléma egy elemének ládákhoz való rendelése is befolyásolja azt, hogy a további elemeket mely ládákhoz lehet hozzárendelni [27]. Az időtengely felosztásával létrejövő W hosszúságú időablakokban megoldandó probléma tehát hasonló egy ládapakolási problémához, így az egy időablakban elvégzendő hozzárendeléseket a heurisztikus algoritmus az ismert ládapakolási heurisztikához, az ún. First Fit Descending (FFD) algoritmushoz hasonló heurisztikával oldja meg. [28]

2. algoritmus: Ládapakolás alapú heurisztika a terheléselosztási probléma megoldására

bemenet: $\mathcal{TH}$, $\mathcal{VH}$, W

kimenet : $\bigcup_{k:\, VH_k \in \mathcal{VH}} \{S_k\}$

```
1  foreach VH_k ∈ VH do
2  |  S_k := −1;
3  n := 1;
4  while (n − 1)W + 1 ≤ t_max do
5  |  VH_n := { VH_k | VH_k ∈ VH ∧ ST_k ≥ (n − 1)W + 1 ∧ ST_k ≤ nW };
6  |  VH_n ← VH_n rendezése C_k szerint, csökkenő sorrendbe; k visszaadása;
7  |  k := 1;
8  |  while k ≤ |VH_n| do
9  |  |  i := 1;
10 |  |  while i ≤ |TH| do
11 |  |  |  if ∀(j : a_(VH_n[k])j = 1) : C_VH_n[k] ≤ TC_i − Σ_{l: VH_l ∈ VH ∧ S_l = i} a_lj C_l then
12 |  |  |  |  S_VH_n[k] = i;
13 |  |  |  |  break;
14 |  |  |  i := i + 1;
15 |  |  if S_k = −1 then
16 |  |  |  S_k := |TH| + 1;
17 |  |  k := k + 1;
18 |  n := n + 1;
```

A 2. algoritmus ábrája a terheléselosztási probléma megoldására kidolgozott heurisztikus algoritmus lépéseit mutatja be. Amennyiben valamely S_k értéke $|\mathcal{TH}|+1$, úgy VH_k-t eldobjuk.

Az S_k változók kezdeti értékének -1-re állítása után (1-2. sor) az algoritmus iterációkat futtat, időablakonként egyet. Az 5. sorban létrehozza azon VH-k $\mathcal{VH}_n$ halmazát, amelyek a jelenlegi időablakban indulnak. Ezután a 6-17. sorban valamennyi $\mathcal{VH}_n$-beli VH-t egy TH'-beli TH-hoz rendeli egy FFD algoritmushoz hasonló heurisztika segítségével. Ennek lépései a következők:

Algoritmus	Rövid leítás	Hatékonyság
Mohó	VH-kat kezdőidejük sorrendjében rendeli a TH-khoz. Egy VH-t az első olyan TH-hoz rendeli, amely a VH végrehajtásának teljes idejére elegendő kapacitással rendelkezik.	Jó futásidő, de alacsonyabb átlagos TH-kihasználtság
BLP alapú	Az időtengelyt időablakokra osztja, majd valamennyi időablakra megold egy BLP-t.	Az átlagos TH-kihasználtság nő az időablak növelésével, de nem skálázható az NP-teljes alproblémák miatt.
Ládapakolás alapú	Az időtengelyt időablakokra osztja, majd valamennyi időablakban egy ládapakoláshoz hasonló problémát old meg.	Jó futásidő, az átlala elért átlagos TH-kihasználtság a mohó és a BLP alapú algoritmusoké között van.

1. táblázat: Terheléselosztási algoritmusok összehasonlítása

A 6. sorban a $\mathcal{VH}_n$ halmaz elemeit kapacitás szerinti csökkenő sorrendbe rendezzük, és sorszámaikat a $\mathbf{VH}_n$ vektorba helyezzük. Ezután a legnagyobb kapacitásútól indulva, valamennyi olyan VH_k-hoz, amely sorszáma $\mathbf{VH}_n$-ben van, megkeressük az első olyan TH-t, amely elegendő szabad kapacitással rendelkezik VH_k futtatásához. Ez azt jelenti, hogy a legelső olyan TH-t keressük, amely szabad kapacitása nagyobb vagy egyenlő VH_k kapacitásánál valamennyi olyan időrésben, amelyben VH_k fut. Ha az algoritmus nem talál ilyen TH-t, eldobja a VH-t, azaz $TH_{|\mathcal{TH}|+1}$-hez rendeli azt.

Értekezésem 2.5. fejezetében megvizsgáltam a téziscsoportban definiált heurisztikák valamint egy mohó algoritmus futásidejét és az általuk elért átlagos TH-kihasználtságot. A szimulációk megmutatták, hogy a kifejlesztett heurisztikák sok esetben hatékonyabbak a mohó algoritmusnál.

Az 1. táblázat az ezen fejezetben bemutatott heurisztikákat hasonlítja össze a

mohó algoritmussal.

4.2. Kommunikáló rendszerek modellvezérelt teljesítménytesztelése

A korábbiakban említetteknek megfelelően a kommunikáló rendszerek fekete doboz alapú teljesítménytesztelése mint tudományterület, még nem rendelkezik kiforrott elméleti háttérrel. Így nem léteznek még olyan automatikus eljárások sem, amelyek azt képesek ellenőrizni, hogy a SUT teljesít-e különböző teljesítménykövetelményeket. A fekete doboz alapú teljesítményteszteket ennek következtében leginkább manuálisan, ad-hoc módon tervezik [29]. Ezen ad-hoc eljárások nagy hátránya az általuk szolgáltatott teljesítménymérési eredmények pontatlansága. A probléma megoldására definiáltam egy automatikus, modellvezérelt teljesítménytesztelési eljárást, amely a tesztelt rendszerrel kommunikálva létrehozza annak formális teljesítménymodelljét, amely teljesítménymodell alapján automatikusan eldönti, hogy a tesztelt rendszer képes-e másodpercenként CR_{usr} kérés feldolgozására, miközben usr darab felhasználót szolgál ki egyszerre.

A másodpercenként feldolgozandó üzenetszám, mint teljesítménykövetelmény, nem egyértelmű. E teljesítménykövetelményt ezért kétféleképpen definiáltam. CR_{usr} egyik értelmezése szerint megegyezik CWR_{usr}-rel, amely azon üzenetek száma, amelyet a SUT-nak másodpercenként a legrosszabb esetben is ki kell tudnia szolgálni. Ez bővebben annyit jelent, hogy a SUT-nak bármilyen bemeneti sorozat esetén képesnek kell lennie CWR_{usr} darab kérésüzenet másodpercenkénti feldolgozására. A másik értelmezés szerint CR_{usr} CER_{usr}-rel egyezik meg, amely a SUT által másodpercenként várhatóan feldolgozott kérésüzenetek száma.

2. tézis. *Definiáltam az időzített kommunikáló véges többállapotú gép nevű teljesítménymodellt (angolul Timed Communicating Finite Multistate Machine, TCFMM a továbbiakban), amely képes a tesztelt rendszer bizonyos teljesítményjellemzőinek reprezentálására. Adtam egy eljárást a SUT által másodpercenként legrosszabb esetben kiszolgált üzenetek számának meghatározására. Továbbá definiáltam egy olyan eljárást, amely kiszámolja a SUT által másodpercenként átlagosan kiszolgált üzenetek számát.*

Az általam kidolgozott eljárás egy modellvezérelt teljesítménytesztelési eljárás, amely bemenetei a tesztelt rendszer funkcionális működését leíró véges állapotau-

tomata (angolul Finite State Machine, FSM), amely helyességét a konformancia-teszt igazolta, valamint a fent említett teljesítménykövetelmények. Az eljárás a funkcionális viselkedést leíró FSM valamint a tesztelt rendszeren végzett mérések alapján felépíti a tesztelt rendszer teljesítménymodelljét, amely alapján analitikus úton eldönti, hogy a tesztelt rendszer megfelel-e a vele szemben támasztott teljesítménykövetelményeknek.

2.1. altézis. *[J2, J5, C2] Definiáltam az időzített kommunikáló véges többállapotú gép nevű teljesítménymodellt (TCFMM). A TCFMM képes modellezni a tesztelt rendszer által adott számú felhasználó párhuzamos kiszolgálása mellett másodpercenként feldolgozott üzenetek számát. Megadtam azon lépéseket, amelyek segítségével a tesztelt rendszer funkcionális működését leíró FSM és a rajta végzett mérések alapján előáll a rendszert reprezentáló TCFMM modell.*

A TCFMM definíciója a következő. A definícióban τ_0 a tesztvégrehajtás kezdőidőpontját jelöli.

1. definíció. *Az időzített kommunikáló véges többállapotú gépet (TCFMM) 10 attribútummal lehet leírni:*

$TCFMM = (I, O, S, s_0, T, U, H, \delta, \chi, \sigma)$, *ahol*

1. $T \subseteq I \times O \times S \times S \times \mathbb{R}^+$
2. $\forall t_i, t_j \in T((t_i = (i_i, o_i, s_{from_i}, s_{to_i}, d_i) \wedge t_j = (i_j, o_j, s_{from_j}, s_{to_j}, d_j) \wedge$ $s_{from_i} = s_{from_j} \wedge i_i = i_j) \Rightarrow t_i = t_j)$
3. $\chi \in H \rightarrow U$, χ *bijektív*
4. $s_0 \in S$
5. $\sigma \in \mathbb{R}^+ \times U \rightarrow S \cup \{\oslash\}$
6. $\forall (u \in U) : \sigma(\tau_0, u) = s_0$
7. $\delta \in \mathbb{R}^+ \times H \times I \rightarrow S \times O \times \mathbb{R}^+$
8. $\forall (t_i \in T, h \in H, \tau \in \mathbb{R}^+) : \sigma(\tau, \chi(h)) = s_{from_i} \Rightarrow \delta(\tau, h, i_i) = (s_{to_i}, o_i, d_i)$ *és az érvénytelen bemenetek eldobásra kerülnek.*

9. $\forall(\tau, h, i, s, o, d : \delta(\tau, h, i) = (s, o, d)) :$
$(\forall(\phi : 0 < \phi < d) : \forall(u \in U - \{\chi(h)\}) :$
$\mathbf{P}(\sigma(\tau + \phi, u) = \oslash) = 1 \Rightarrow (\sigma(\tau + d, \chi(h)) = s \wedge \forall(\epsilon : 0 < \epsilon < d) :$
$\sigma(\tau + \epsilon, \chi(h)) = \oslash)) \wedge (\neg(\forall(\phi : 0 < \phi < d) : \forall(u \in U - \{\chi(h)\}) :$
$\mathbf{P}(\sigma(\tau + \phi, u) = \oslash) = 1) \Rightarrow \exists(d' : d' < d) :$
$(\sigma(\tau + d', \chi(h)) = s \wedge \forall(\epsilon : 0 < \epsilon < d') : \sigma(\tau + \epsilon, \chi(h)) = \oslash)) \wedge$
$\wedge \exists(t_i \in T) : s_{from_i} = \sigma(\tau, \chi(h)) \wedge s_{to_i} = s \wedge i_i = i \wedge o_i = o \wedge d_i = d$

Az általam kidolgozott teljesítménytesztelési eljárás felépíti a tesztelt rendszer teljesítménymodelljét. Ehhez első lépésben a modell funkcionális szerkezetét kell felépíteni, azon M'' FSM segítségével, amelynek a konformanciateszt alapján a tesztelt rendszer megfelel. A [4] cikkben leírt definíciónak megfelelően, M''-et a következőképpen írhatjuk le:

$M'' = (I'', O'', S'', \delta'', \lambda'')$, ahol
$\delta'' : S'' \times I'' \rightarrow S''$ az állapotátmenet-függvény
$\lambda'' : S'' \times I'' \rightarrow O''$ a kimenetfüggvény

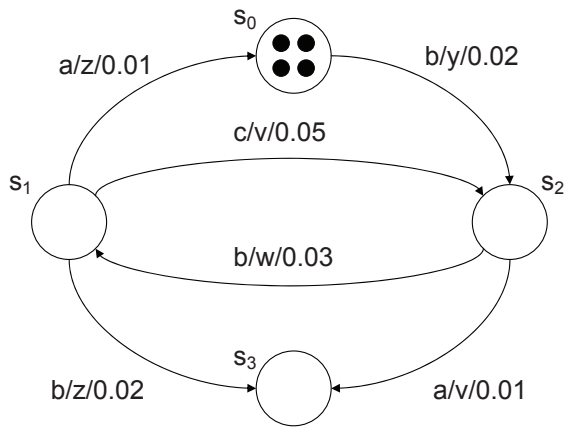
Jelölje $M = (I, O, S, s_0, T, U, H, \delta, \chi, \sigma)$ azt a TCFMM-et, amelyet a teljesítményteszt végrehajtására használunk. M-et M''-ből két lépésben hozzuk létre. Az első lépésben M'' alapján egy $M' = (I', O', S', s'_0, T', U', H', \delta', \chi', \sigma')$ TCFMM-et hozunk létre. M'-t a következőképpen definiáljuk:

- $S' = S''$
- $s'_0 = M''$ kezdőállapota
- $O' = O''$
- $I' = I''$
- $U' = \{u_i | i = 1, ..., usr\}$
- $H' = \{h_i | i = 1, ..., usr\}$
- $\forall(h_i \in H') : \chi'(h_i) = u_i$
- $T' = \{t_i = (i_i, o_i, s_{from_i}, s_{to_i}, \oslash) : \exists(i \in I'', s \in S'') : s_{from_i} = s \wedge s_{to_i} = \delta''(s, i) \wedge$
$\forall t_j = (i_j, o_j, s_{from_j}, s_{to_j}, \oslash) : (s_{from_i} = s_{from_j} \wedge i_i = i_j) \Rightarrow t_i = t_j\}$

A σ' és δ' függvényeket a fenti hozzárendelésekből és az 1. definíció $5 - 9$. megkötéseiből vezetjük le. A második lépésben $M = (I, O, S, s_0, T, U, H, \delta, \chi, \sigma)$ automatát hozzuk létre M' alapján. Ez utóbbi attribútumait a következőek:

- $I = I'$
- $O = O'$
- $s_0 = s'_0$
- $U = U'$
- $H = H'$
- $\forall(h_i \in H) : \chi(h_i) = u_i$
- $T = \{t_i | \exists(t_j \in T') : (\exists(t_k \in T') : s_{from_k} = s_{to_j} \wedge s_{from_j} = s_{from_i} \wedge s_{to_j} = s_{to_i} \wedge$
  $\wedge i_j = i_i \wedge o_j = o_i \wedge d_j = d_i) \vee$
  $\vee(t_{to_i} = s_0 \wedge \exists(t_j \in T') : (\nexists(t_k \in T') : s_{from_k} = s_{to_j}) \wedge$
  $\wedge s_{from_i} = s_{from_j} \wedge i_i = i_j \wedge o_i = o_j \wedge d_i = d_j)\}$
- $S = \{s_i | s_i \in S' \wedge \exists(t_j \in T') : s_{from_j} = s_i\}$

A σ' és δ' függvényeket a fenti hozzárendelések és az 1. definíció 5−9. megkötéseiből vezetjük le.



4. ábra. A TCFMM grafikus ábrázolása

A 4. ábrán a TCFMM grafikus ábrázolását láthatjuk. Az egyes állapotátmeneteken feltüntetett paraméterek rendre *bemenet/kimenet/késleltetés*. Valamennyi token az s_0 állapotban van.

A teljesítményteszt során a tesztkörnyezet emulálja a tesztelt rendszer felhasználóit. Eközben M segítségével leköveti a tesztelt rendszer valamennyi protokollpéldányának (szerverszálának) állapotváltásait. Összesen *usr* darab token elhelyezése M gráfjában annyit jelent, hogy a teljesítményteszt alatt a tesztkörnyezet *usr* fel-

használót fog emulálni. Ha a teszt során a tesztkörnyezet az $u = \chi(h)$ tokent a t_i tranzíció mentén az s_{from_i} állapotból az s_{to_i} állapotba mozgatja, akkor eközben elküldi az i_i input-üzenetet a h felhasználó nevében a tesztelt rendszernek, majd vár arra, hogy a SUT visszaküldje neki az o_i válaszüzenetet. Amikor M' alapján létrehozzuk M-et, valamennyi olyan M'-beli élet, amely nyelőállapotba (kimenő tranzícióval nem rendelkező állapotba) fut, át kell irányítani az s_0 állapotba. Így miután egy teszter által emulált felhasználó elküldi utolsó üzenetét a SUT-nak, új, kiszolgálandó felhasználóként jelenik meg a SUT felé (tehát a hozzátartozó token s_0-ba kerül). Ennek következtében a SUT-nak folyamatosan *usr* felhasználót kell feldolgoznia.

A TCFMM véglegesítéséhez le kell mérni a SUT egyelőre még ismeretlen állapotátmeneti késleltetéseit. Ez a következőképpen történik:

A teszt során a teszter *usr* felhasználót emulál a SUT felé. A teszt közben egyetlen felhasználópéldány sem lehet inaktív, azaz amint megkap egy o_i kimeneti üzenetet a SUT-tól, azonnal visszaküld neki egy i_j bemeneti üzenetet, ahol $s_{to_i} = s_{from_j}$. A t_i állapotátmenet késleltetése megegyezik azzal az idővel, amely aközött telt el, hogy a teszter az i_i üzenetet elküldte a SUT-nak és aközött, hogy az o_i üzenetet megkapta a SUT-tól. Valamennyi állapotátmeneten előre meghatározott számú késleltetésmérést végez a teszter. Ezen mérések átlagolásával kapjuk meg a végső d_i értékeket.

2.2. altézis. *[J2, J5, C2] A SUT-ot leíró TCFMM modellen megadtam annak szükséges és elégséges feltételét, hogy a SUT legrosszabb esetben képes legyen adott számú kérésüzenet másodpercenkénti feldolgozására. A feltétel alapján megadtam egy eljárást, amely kiszámolja, hogy a tesztelt rendszer másodpercenként legrosszabb esetben hány kérésüzenetet képes feldolgozni.*

Ahogy azt értekezésem 3.5.1. fejezetében igazoltam, a tesztelt rendszer akkor és csak akkor képes másodpercenként CWR_{usr} üzenet feldolgozására legrosszabb esetben, ha teljesíti a következő egyenlőtlenséget:

$$\forall(c_i \in C) : \sum_{t_j \in c_i} d_j \leq \frac{|c_i|}{CWR_{usr}} \tag{13}$$

A fenti feltétel alapján a tesztelt rendszer által másodpercenként, legrosszabb esetben feldolgozott üzenetek száma:

$$CW_{usr} = \min_{c_j \in C} \left\{ \frac{|c_i|}{\sum\limits_{t_j \in c_i} d_j} \right\} \tag{14}$$

2.3. altézis. *[J2, J5, C2] Megadtam egy olyan eljárást, amely a SUT TCFMM modellje valamint a rendszer felhasználóinak viselkedését leíró állapotátmeneti valószínűségek alapján képes kiszámolni, hogy a SUT másodpercenként várhatóan hány kérést dolgoz fel.*

Ha a SUT teljesíti a 13. egyenlőtlenség által leírt követelményt, akkor minden esetben képes arra, hogy másodpercenként legalább CWR_{usr} üzenetet dolgozzon fel. A rendszer későbbi felhasználói azonban a végrehajtás egy adott állapotában legális különböző kérésüzeneteket különböző valószínűséggel küldik el a tesztelt rendszernek. Ez a felhasználói viselkedés az egyazon állapotból kiinduló állapotátmenetekhez különböző állapotátmeneti valószínűségeket rendel, és ebből következően a felhaszálók azt tapasztalhatják, hogy a rendszer által másodpercenként feldolgozott üzenetek száma jelentősen meghaladja CW_{usr} értékét. A következőkben bemutatom CE_{usr} kiszámításának lépéseit, amely nem más, mint a rendszer által másodpercenként feldolgozott üzenetek száma, a felhasználók tapasztalata szerint. CE_{usr} mindig nagyobb vagy egyenlő, mint CW_{usr}.

Tegyük fel, hogy egy h felhaszálóhoz tartozó $\chi(h)$ token a s_{from_i} állapotban tartózkodik. Ekkor t_i p_i-vel jelölt valószínűsége annak valószínűsége, hogy h felhasználó i_i bemeneti üzenetet küldi a tesztelt rendszernek (p_i értéke valamennyi felhasználó esetében ugyanaz). Definiáljuk továbbá p_{kl}-t a következőképpen:

$$p_{kl} = \sum_{i:t_i \in T \wedge s_{from_i} = s_k \wedge s_{to_i} = s_l} p_i \tag{15}$$

Tehát amennyiben $\chi(h) = s_{from_i} = s_k$, akkor p_{kl} annak valószínűségét jelöli, hogy a h felhasználó bármely olyan állapotátmenethez tartozó bemenetet küld a rendszernek, amely s_k-ból s_l-be tart. Másképpen fogalmazva p_{kl} annak valószínűsége, hogy a $\chi(h)$ token az s_k állapotból az s_l állapotba megy át (p_{kl} értéke valamennyi felhasználóra és tokenre megegyezik). Jelölje továbbá z_i az s_i állapot *stacionárius állapotvalószínűségét*, azaz annak valószínűségét, hogy az u token éppen s_i állapotba halad át (z_i valamennyi tokenre és felhasználóra megegyezik). CE_{usr} kiszámításához először z_i értékét kell kiszámolni valamennyi s_i állapotra. Ezt a következő mátrixe-

gyenlet megoldásával tehetjük meg. Az egyenletben $n = |S|-1$, azaz M állapotainak száma mínusz egy:

$$\boldsymbol{Fz} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \text{ ahol } \boldsymbol{z} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_n \end{bmatrix} \text{ és } \boldsymbol{F} = \begin{bmatrix} p_{00}-1 & p_{10} & \dots & p_{n0} \\ p_{01} & p_{11}-1 & \dots & p_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{0(n-1)} & p_{1(n-1)} & \dots & p_{n(n-1)} \\ 1 & 1 & \dots & 1 \end{bmatrix} \tag{16}$$

Ha $\det \boldsymbol{F} \neq 0$, a fenti mátrixegyenlet egyértelmű megoldást ad a z_i értékekre [30]. A z_i értékek alapján a CE_{usr} értéket a következőképpen kapjuk meg. A lenti egyenletben z_{from_i} az s_{from_i} állapot stacionárius állapotvalószínűségét jelöli:

$$CE_{usr} = \frac{1}{\sum\limits_{t_i \in T} d_i z_{from_i} p_i} \tag{17}$$

Ahogy értekezésem 3.6. fejezetében megmutattam, a fent ismertetett teljesítménytesztelési eljárás az ad-hoc eljárás jó alternatíváját nyújtja két okból is. Egyrészt az ad-hoc eljárástól eltérően, képes a rendszer által másodpercenként legrosszabb esetben feldolgozott üzenetek számának kiszámítására. Másrészt pedig, ha a két eljárásnak ugyanannyi időt bocsátunk rendelkezésére a teszt lebonyolításához, az általam definiált eljárás pontosabban (alacsonyabb szórással) méri ki a rendszer által másodpercenként várhatóan feldolgozott üzenetek számát.

4.3. Kommunikáló rendszerek legrosszabb esetbeni teljesítményének növelése

Amennyiben a tesztelt rendszeren futtatott teljesítményteszt azt állapítja meg, hogy $CW_{usr} < CWR_{usr}$, azaz a rendszer által másodpercenként legrosszabb esetben feldolgozandó üzenetszámnál a rendszer másodpercenként legrosszabb esetben kevesebb üzenet feldolgozására képes, a tesztelt rendszer teljesítményét addig kell növelni, amíg ezt a teljesítménykövetelményt nem teljesíti. A rendszer által másodpercenként legrosszabb esetben feldolgozott üzenetek számának növelését a rendszer állapotátmeneti késleltetéseinek csökkentésével érjük el. Valamennyi állapotátmeneti késleltetés előre megadott mértékekben csökkenthető, ezek állapotátmenetről ál-

lapotátmenetre változhatnak. Az állapotátmeneteket adott költség befektetésével csökkenthetjük.

A következőkben bemutatott eljárások célja, hogy a rendszer által másodpercenként legrosszabb esetben feldogozott üzenetek számát a kívánt szintre emelje, minimális költség ráfordítása mellett. Mostantól a fent leírt problémát teljesítménykorrekciós problémának fogom nevezni.

3. tézis. *Bebizonyítottam a teljesítménykorrekciós probléma NP-teljességét. Felírtam a problémát egészértékű lineáris programként. Megadtam egy heurisztikát a teljesítménykorrekciós probléma azon esetének megoldására, ahol a költségfüggvény logaritmikus.*

A teljesítménykorrekciós problémát a következőképpen definiáltam:

Adott az állapotátmeneteket tartalmazó $T = \{t_i\}$ halmaz valamint a köröket tartalmazó $C = \{C_i\}$ halmaz, ahol valamennyi $C_i = \{t_j\}$ kör állapotátmenetek halmaza, illetve valamennyi t_j állapotátmenethez tartozik egy d_j késleltetésérték. Adott továbbá egy pozitív CWR_{usr} szám, egy pozitív K szám, és valamennyi t_i állapotátmenethez egy $0 < q_i \leq 1$ változó (ún. korrekciós faktor) és egy $Q_i = \{q_{ij}\}$ halmaz, ahol $q_{ik} < q_{i(k+1)}$ valamennyi $k = 1, \ldots, |Q_i| - 1$ indexre, és $q_{i|Q_i|} = 1$. Továbbá valamennyi állapotátmenethez adott egy monoton csökkenő $Cost_i(x)$ függvény, amelyre $Cost_i : (0, 1] \to \mathbb{R}^+$, $Cost_i(1) = 0$. A probléma megoldása során megválaszolandó kérdés a következőképpen hangzik: Lehetséges-e úgy kiválasztani az egyes q_i értékeket, hogy $\exists(q_{ij} \in Q_i) : q_i = q_{ij}$ igaz legyen, valamint a következő egyenlőtlenségek teljesüljenek?

$$\forall(c_i \in C) : \sum_{j:t_j \in c_i} d_j q_j \leq \frac{|c_i|}{CWR_{usr}} \tag{18}$$

$$\sum_{i:t_i \in T} Cost_i(q_i) \leq K \tag{19}$$

A fenti definícióban q_i a d_i késleltetés csökkentését reprezentáló együttható, azaz t_i csökkentett késleltetése $d_i q_i$. $Cost_i(q_i)$ a t_i késleltetéscsökkentésének költsége. $\forall(i : t_i \in T) : Cost_i(1) = 0$, mivel ha $q_i = 1$, akkor t_i késleltetését nem csökkentettük, így az nem kerül semmibe. Végül pedig K a teljes korrekció költségének felső korlátja. A 18. egyenlőtlenség azt fejezi ki, hogy a késleltetéskorrekciót követően a rendszer ki kell elégítse a 13. egyenlőtlenséget, míg a 19. egyenlőtlenség azt fejezi ki, hogy a teljes korrekció költsége nem haladhatja meg K-t.

3.1. altézis. *[J3, J4] Bebizonyítottam a teljesítménykorrekciós probléma NP-teljességét.*

A probléma NP-teljességét az NP-teljes hátizsákprobléma egy tetszőleges példányának visszavezetésével igazoltam értekezésem 4.2. fejezetében [27]. Mivel a probléma NP-teljes, az optimumát megtudhatjuk, ha felírjuk lineáris egészértékű programként, majd megoldjuk azt.

3.2. altézis. *[J3, J4] A teljesítménykorrekciós probléma optimumának megtalálása érdekében felírtam a problémát egészértékű lineáris programként.*

A probléma ILP felírása a következő. A felírásban $cst_{ij} = Cost_i(q_{ij})$.
Minimalizálandó célfüggvény:

$$\sum_{i:t_i \in T} \sum_{j=1}^{|Q_i|} s_{ij}\, cst_{ij} \tag{20}$$

Megkötések:

$$\forall (i : t_i \in T) : \sum_{j=1}^{|Q_i|} s_{ij} = 1 \tag{21}$$

$$\forall c_i \in C : \sum_{j:t_j \in c_i} d_j \sum_{k=1}^{|Q_i|} s_{jk} q_{jk} \leq \frac{|c_i|}{CWR_{usr}} \tag{22}$$

$$\begin{gathered} \forall (i : t_i \in T) : \forall (j = 1, 2, \ldots, |Q_i|) : \\ s_{ij} \in \{0, 1\} \end{gathered} \tag{23}$$

Azon változók, amelyeknek a program megoldása során értéket kell adni, az s_{ij} változók. Minden s_{ij} értéke 0 vagy 1 kell, hogy legyen (23. egyenlet). Az s_{ij} változókat, ahol $j = 1, \ldots, |Q_i|$ a t_i tranzíció korrekciós faktorának kiválasztására használjuk. A fenti BLP megoldásaként az egyes t_i állapotátmenetekhez tartozó s_{ij} változók közül pontosan egy értéke lesz 1. Az összes többi, ezen állapotátmenethez tartozó s_{ij} változó értéke 0 (a 21. és 23. egyenletek következménye). Ha s_{ij} értéke 1, akkor $q_i = q_{ij}$ és így t_i késleltetéskorrekciójának költsége $Cost_i(q_{ij})$. A 22. egyenlőtlenség bal oldalán $\sum_{k=1}^{|Q_i|} s_{jk} q_{jk}$ a t_i állapotátmenet q_i korrekciós faktorával egyezik meg. Így a 22. egyenlőtlenség azt jelenti, hogy valamennyi c_i állapotátmenetkör korrigált költsége kisebb vagy egyenlő kell legyen $\frac{|c_i|}{CWR_{usr}}$-nél (ez megfelel a 18.

egyenlőtlenségnek). Végül pedig, a célfüggvényben (20. képlet) $\sum_{j=1}^{|Q_i|} s_{ij}\, cst_{ij}$ megegyezik $Cost_i(q_i)$-vel (amely a t_i állapotátmenet késleltetéscsökkentésének költsége). $Cost_i(q_i)$ értékét a $\{cst_{ij}|j = 1, \ldots, |Q_i|\}$ halmazból választhatjuk ki a megfelelő s_{ij} változó 1-be állításával. Tehát a célfüggvény azt fejezi ki, hogy az állapotátmeneti késleltetések csökkentésének teljes költsége minimális kell, hogy legyen.

A BLP megoldásához szükséges idő rendkívül nagy lehet, ezért létrehoztam egy heurisztikus algoritmust a probléma megoldására.

3.3. altézis. *[J3, J4] Létrehoztam egy heurisztikus algoritmust a teljesítménykorrekciós probléma azon esetének megoldására, ahol az állapotátmenetek késleltetéskorrekciójának költségfüggvénye logaritmikus.*

Az algoritmust arra az esetre optimalizáltam, ahol $Cost_i(x) = -\gamma_i \log_a x$, ahol $\gamma_i > 0$ egy t_i állapotátmenethez rendelt konstans. A 3. algoritmus ábrája a heurisztika működését mutatja be. Az algoritmus leírásában r az ún. frissítési granularitást jelöli.

A heurisztika első lépésben valamennyi q_i korrekciós faktort annak minimális q_{i1} értékére állít be, majd iterációkat futtatva az egyes q_i korrekciós faktorokat különböző gyakorisággal növeli következő legális értékükre (értsd q_{ik}-ról $q_{i(k+1)}$-re) addig, amíg már nem növelhetünk egyetlen korrekciós faktort sem a 18. egyenlőtlenség valamely példányának megsértése nélkül. Az algoritmus az első iteráció előtt illetve q_i minden egyes növelése után beállítja α_i értékét, ami nem más, mint azon iterációk száma, amelyeknek még el kell telnie q_i legközelebbi növeléséig. Az algoritmus kulcslépése ez utóbbi, azaz annak meghatározása, hogy az egyes q_i-k következő növeléséig hány iterációnak kell eltelnie annak érdekében, hogy a teljesítménykorrekció költsége minimális legyen.

Értekezésem 4.5. fejezetében megvizsgáltam az általam definiált algoritmus, egy egyszerű round-robin algoritmus valamint az optimumot megtaláló ILP költségét és futásidejét. Utóbbi megoldást sok esetben nem tudtam alkalmazni nagy futásideje miatt, ezért ezekben az esetekben az optimum egy durva alsó becslését adó (relaxált) lineáris programmal helyettesítettem. A szimulációk alapján az általam kidolgozott eljárás költséghatékonyabb a round-robin algoritmusnál, míg futásideje az ILP futásidejével szemben elfogadható.

3. algoritmus: A teljesítménykorrekciós probléma heurisztikus megoldása

bemenet: T, C, CWR_{usr}, r, $\{Q_i|i : t_i \in T\}$, $\{\gamma_i|i : t_i \in T\}$

kimenet : $\bigcup_{i:t_i \in T} \{q_i\}$

foreach $i : t_i \in T$ **do**
  $clist_i := \{j|t_i \in c_j\}$
foreach $i : t_i \in T$ **do**
  $q_i := q_{i1}$
if $\exists(c_i \in C) : \sum_{j:t_j \in c_i} d_j q_j > \frac{|c_i|}{CWR_{usr}}$ **then**
  return "unsolvable";
foreach $i : t_i \in T$ **do**
  $current_i := 1$
foreach $i : t_i \in T$ **do**
  if $|Q_i| > 1$ **then**
    $\alpha_i := \left\lceil \frac{|clist_i| d_i r (q_{i(current_i+1)} - q_{i(current_i)})}{\gamma_i} \right\rceil$
while $\exists i : (current_i < |Q_i|$
$\wedge(\forall j \in clist_i : (\sum_{k:t_k \in c_j \wedge k \neq i} q_k d_k) + q_{i(current_i+1)} d_i \leq \frac{|c_j|}{CWR_{usr}}))$ **do**
  foreach $i : t_i \in T$ **do**
    if $current_i < |Q_i|$ **then**
      $\alpha_i := \alpha_i - 1$
  foreach $i : t_i \in T$ **do**
    if $\alpha_i = 0 \wedge current_i < |Q_i|$
    $\wedge \forall j \in clist_i : (\sum_{k:t_k \in c_j \wedge k \neq i} q_k d_k) + q_{i(current_i+1)} d_i \leq \frac{|c_j|}{CWR_{usr}})$ **then**
      $q_i := q_{i(current_i+1)}$;
      $current_i := current_i + 1$
  if $current_i < |Q_i|$ **then**
    $\alpha_i := \left\lceil \frac{|clist_i| d_i r (q_{i(current_i+1)} - q_{i(current_i)})}{\gamma_i} \right\rceil$
return $\bigcup_{i:t_i \in T} q_i$;

5. Az eredemények alkalmazása

Az első téziscsoport eredményei olyan teljesítménytesztelési környezetek terhelés-elosztására használhatók, amelyben a tesztelt rendszer terhelését megvalósító terhelésgenerátor entitások (VH-k) kezdő- és futásideje kötött. A bemutatott eljárások valós tesztkörnyezetbe való leképezésének egyik lehetséges módja az, amikor a terhelésgenerátorokat a tesztelt rendszerrel kommunikáló felhasználóknak feleltetjük meg. A futtatott szimulációk alapján az általam kidolgozott, ILP alapú eljárás annál jobb TH-kihasználtságot ér el, minél jobban csökken a tesztelő hosztok száma illetve az átlagos TH-kapacitás és VH-kapacitás hányadosa. Az ILP alapú eljárás futásideje azonban elfogadhatatlanul nagy lehet. A bemutatott heurisztikus algoritmus által elért TH-kihasználtság akkor nő, ha az átlagos TH-kapacitás és VH-kapacitás hányadosa csökken, és ha a VH-kapacitások szórása nő, függetlenül a TH-k számától.

A második téziscsoportban bemutatott automatikus teljesítménytesztelési eljárás akkor alkalmazható, ha a SUT specifikációja a rendszer által másodpercenként feldolgozandó üzenetek maximális számára ad megkötést, adott számú felhasználó párhuzamos kiszolgálása mellett. Az eljárás számtalan alkalmazása közül két konkrét példa egy SIP proxy valamint egy webportál tesztelése [31]. A kérések előbbi esetben a felhasználó kattintásai, míg utóbbi esetben a felhasználó által a rendszernek küldött, például hívásfelépítéshez szükséges üzenetek. A bemutatott eljárás hatékonyságát egy, az iparban használt ad-hoc eljárásához hasonlítva kiderült, hogy az előbbi által kiszámolt, a tesztelt rendszer által másodpercenként kiszolgált legrosszabb esetbeni üzenetszám az ad-hoc eljárás valamennyi mérési eredményénél alacsonyabb, míg az általa kiszámolt másodpercenkénti átlagos üzenetszám ugyanannyi, mint amit az ad-hoc eljárás mér ki, azaz ezek átlaga az ad-hoc eljárás által mért eredmények átlagával egyezik meg. A kísérletek ezen felül még igazolták azt is, hogy az általam kidolgozott eljárás által szolgáltatott mérési eredmények szórása szignifikánsan alacsonyabb, mint az ad-hoc eljárás mérési eredményeinek szórása. Ez azt jelenti, hogy az általam kidolgozott eljárás pontosabb az ad-hoc eljárásnál. A kísérletek alapján ez szélsőséges esetekre is igaz, tehát például akkor is, amikor minden egyes állapotátmeneti késleltetés normál vagy egyenletes eloszlású.

A harmadik téziscsoportban bemutatott eljárások egy, a második téziscsoportban bemutatott teljesítményteszten megbukott tesztelt rendszer teljesítménykorrekciójára használhatók. Az itt definiált eljárások feltételezik, hogy a tesztelt rendszer vala-

mennyi állapotátmenetéhez rendelkezésre áll egy költségfüggvény, amely segítségével kiszámolható adott mértékű késleltetéscsökkentés költsége. A téziscsoportban felírt BLP sok esetben nem alkalmazható nagy futásideje miatt, azonban az ennek alternatívájaként definiált heurisztika futásideje minden esetben elfogadható. Szimulációk igazolták, hogy a heurisztikus eljárás teljesítménye a szükséges korrekció mértékének ($\frac{CWR_{usr}}{CW_{usr}}$) növelésével nő.

Köszönetnyilvánítás

Köszönettel tartozom témavezetőimnek, Dr. Csöndes Tibornak mindazért az energiáért és időért, amelyet az elmúlt években rám áldozott, és azért a rengeteg tapasztalatért, amelyet az évek során megosztott velem, illetve Dr. Dibuz Saroltának tanácsaiért és támogatásáért.

Köszönöm Dr. Csopaki Gyulának folyamatos támogatását és iránymutatását, és hogy évekkel ezelőtt bevezetett a tesztelés világába.

Hálás vagyok továbbá kollégáimnak és szobatársaimnak, Dr. Babarczi Péternek, Németh Gábor Árpádnak és Novák Zoltánnak segítségükért és azért a légkörért, amelyben doktoranduszéveim alatt dolgozhattam, valamint Marton József Ernő kollégámnak, technikai tanácsaiért.

Köszönettel tartozom a Nagysebességű Hálózatok Laboratóriumának (HSNLab) kutatómunkám műszaki és anyagi hátterének biztosításáért, Dr. Szabó Róbertnek, Dr. Vidács Attilának és Dr. Molnár Sándornak a publikációimmal kapcsolatos tanácsaikért, valamint Győri Erzsébetnek minden segítségéért.

Nem lenne teljes a lista tanszékvezetőm, Dr. Henk Tamás, Dr. Halász Edit, Prof. Dr. Sallai Gyula, Dr. Adamis Gusztáv és Dr. Kovács Gábor nélkül, akiktől sok értékes tanácsot kaptam az értekezésem véglegesítéséhez.

Végül, de nem utolsó sorban szeretném megköszönni családom minden tagjának, hogy tanulóéveim alatt végig támogattak, hogy kivételes tanulási lehetőségeket biztosítottak számomra, és hogy segítettek leküzdeni minden felmerülő akadályt.

Hivatkozások

[1] **Utting, M and Legeard, B.** Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2007.

[2] **Pretschner, A. and Philipps, J.** Methodological Issues in Model-Based Testing In *Model-Based Testing of Reactive Systems*, pp. 281–291, 2004.

[3] **Dorofeeva, R. and El-Fakih, K. and Maag, S. and Cavalli, A. R. and Yevtushenko, N.** FSM-based conformance testing methods: A survey annotated with experimental evaluation In *Information and Software Technology vol. 52 issue 12.*, pp. 1286–1297, 2010.

[4] **Lee, D. and Yannakakis, M.** Principles and Methods of Testing Finite State Machines – A Survey In *Proceedings of the IEEE vol. 84 issue 8*, pp 1090–1123, 1996.

[5] **Tretmans, J.** Specification Based Testing with Formal Methods: A Theory In *FORTE / PSTV 2000 Tutorial Notes*, 2000.

[6] **Chow, T.** Testing Software Design Modelled by Finite-State Machines In *IEEE Transactions on Software Engineering vol. 4 issue 3.*, pp. 178–187, 1978.

[7] **Fujiwara, S. and Khendek, F. and Amalou, M. and Ghedamsi, A.** Test Selection Based on Finite State Models In *IEEE Transactions on Software Engineering vol. 17 issue 6.*, pp. 591–603, 2002.

[8] **Sun, X. and Shen, Y. and Feng, C. and Lombardi, F.** Advanced Series in Electrical and Computer Engineering - Vol. 12, Protocol Conformance Testing Using Unique Input/Output Sequences, World Scientific Publishing, 1997.

[9] **Sabnani, K. and Dahbura, A.** A Protocol Test Generation Procedure In *Computer Networks and ISDN Systems vol. 15 issue 4.*, pp. 285–297, 1988.

[10] **Naito, S. and Tsunoyama, M.** Fault Detection for Sequential Machines by Transition-Tours In *Proc. 11th Ann. IEEE Internat. Symp. Fault-Tolerant Comput.*, pp. 238–243, 1981.

[11] **Kohavi, Z.** Switching and Finite Automata Theory, McGraw-Hill, 1978.

[12] **Lee, D. and Yannakakis, M.** Testing Finite-State Machines: State Identification and Verification In *IEEE Transactions on Computing vol. 43 issue 3.*, pp. 306–320, 1994.

[13] **Tretmans, G. J.** Test Generation with Inputs, Outputs, and Quiescence In *Proc. Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS*, pp. 127–146, 1996.

[14] **Kovacs, G. and Pap, Z. and Csopaki, G. and Tarnay, K.** Iterative Automatic Test Generation Method for Telecommunication Protocols In *Computer Standards & Interfaces vol. 28 issue 4.*, pp. 412–427, 2006.

[15] **Nemeth, G. A. and Pap, Z. and Kovacs, G. and Subramaniam, M.** A Bounded Incremental Test Generation Algorithm for Finite State Machines In *Proc. 19th IFIP Int. Conf. on Testing of Communicating Systems (TestCom/FATES)*, pp. 244–259, 2007.

[16] **Ghedamsi, A. and Bochmann, G.V.** Test Result Analysis and Diagnostics for Finite State Machines In *Proc. of the 12th International Conference on Distributed Computing Systems*, pp. 244–251, 1992.

[17] **Brinksma, E. and Tretmans, J. and Verhaard, L.** A Framework for Test Selection In *Proc. IFIP WG6.1 11th Int. Symp. on Protocol Specification, Testing, and Verification*, pp. 233–248, 1991.

[18] **El-Fakih, K., Yevtushenko, N., von Bochmann, G.** FSM-based Incremental Conformance Testing Methods In *IEEE Transactions on Software Engineering vol. 30 issue 7.*, pp. 425–436, 2004.

[19] **Ufuk Celikkan and Rance Cleaveland** Computing Diagnostic Tests for Incorrect Processes In *Proc. of the IFIP Symposium on Protocol Specification, Testing, and Verification*, pp. 263–278, 1992.

[20] **Kemper, P. and Kritzinger, P. and Bause, F. and Kabutz, H.** SDL and Petri Net Performance Analysis of Communicating Systems In *Proc. of the 15th International Symposium on Protocol Specification, Testing and Verification*, pp. 269–282, 1995.

[21] **Youness, O. S. and El-Kilani, W. S. and El-Wahed, W. F. A.** A Behavior and Delay Equivalent Petri Net Model for Performance Evaluation of Communication Protocols In *Computer Communications, vol. 31 issue 10.*, pp. 2210–2230, 2008.

[22] **El-Karaksy, M. R. and Nouh, A. S. and Al-Obaidan, A.** Performance Analysis of Timed Petri Net Models for Communication Protocols: A Methodology and Package In *Computer Communications vol. 13 issue 2.*, pp. 73–82, 1990.

[23] **Marsan, M. A. and Chiola, G. and Fumagalli, A.** Timed Petri Net Model for the Accurate Performance Analysis of CSMA/CD Bus LANs In *Computer Communications vol 10. issue 6.*, pp. 304–312, 1987.

[24] **Schieferdecker, I. and Stepien, B. and Rennoch, A.** PerfTTCN, a TTCN Language Extension for Performance Testing In *Proc. of the IFIP TC6 10th International Workshop on Testing of Communicating Systems*, pp. 21–36, 1997.

[25] **Dai, Z. R. and Grabowski, J. and Neukirchen, H.** TimedTTCN-3 - A Real-Time Extension for TTCN-3 In *Testing of Communicating Systems*, pp. 407–424, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[26] **Mingwei, X. and Jianping, W.** A formal approach to protocol performance testing In *Journal of Computer Science and Technology vol. 14 issue 1*, pp. 81–87, 1999.

[27] **Garey, M. R. and Johnson, D. S.** Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., San Francisco, CA, 1990.

[28] **Coffman,Jr., E. G. and Garey, M. R. and Johnson, D. S.** Approximation algorithms for bin packing: a survey In *Approximation algorithms for NP-hard problems*, PWS Publishing Co., pp. 46–93, 1997.

[29] **Szabo, J. Z. and Csondes, T.** TITAN, TTCN-3 test execution environment In *Infocommunications Journal vol. 57. issue 1*, pp. 27–31, 2007.

[30] **Hoffman, K. and Kunze, R** Linear Algebra, Prentice Hall, 2nd Edition, pp. 161–162, 1971.

[31] **RFC 3261** SIP: Session Initiation Protocol, IETF, 2002.

Publikációk

Folyóiratcikkek

[J1] **Erős, L. and Bozóki, F.** Test Component Assignment and Scheduling in a Load Testing Environment In *Periodica Polytechnica vol. 52 issue 3–4*, pp. 145–152, 2008.

[J2] **Erős, L. and Csöndes, T.** Model-Driven Black Box Performance Testing of Communicating Systems In *Journal of Universal Computer Science*, bírálat alatt, 2011.

[J3] **Erős, L. and Csöndes, T.** Model-Driven Performance Correction of Communicating Systems In *Computing and Informatics*, bírálat alatt, 2011.

[J4] **Erős, L. and Csöndes, T.** Model-Driven Diagnostics of Underperforming Communicating Systems In *Acta Cybernetica*, bírálat alatt, 2011.

[J5] **Pernek, Á. and Erős, L. and Csöndes, T.** Kommunikáló rendszerek teljesítménytesztelése *Híradástechnika vol. 65 issue 7–8.*, pp. 28-32, 2010.

[J6] **Erős, L. and Bozóki, F.** Refactorisation Methods for TTCN-3 *Acta Polytechnica, Czech Technical University in Prague vol. 47 issue 4–5.*, pp. 33–37, 2007.

Konferenciacikkek

[C1] **Erős, L. and Csöndes, T.** Test Component Assignment in a Performance Testing Environment In *Proc. IEEE SoftCom*, pp. 399-403, Split-Dubrovnik, Croatia, 2008.

[C2] **Erős, L. and Csöndes, T.** An Automatic Performance Testing Method Based on a Formal Model for Communicating Systems In *Proc. 18th IEEE International Workshop on Quality of Service*, Paper No. 1569284789, Beijing, China, 2010.

[C3] **Bozóki, F. and Erős, L.** Refactoring Test Data Structures In *Proc. IEEE ConTel 2007: 9th International Conference on Telecommunications*, pp. 123-130, Zagreb, Croatia, 2007.

[C4] **Bozóki, F. and Erős, L.** Eliminating the Redundancy of TTCN-3 Sources In *Proc. TRANSCOM 2007: 7th European Conference of Young Research and Science Workers*, pp. 47-50, Zilina, Slovakia, 2007.

[C5] **Wu-Hen-Chang, A. and Adamis, G. and Erős, L. and Kovács, G. and Csöndes, T.** A New Approach in Model-Based Testing: Designing Test Models in TTCN-3 In *Proc. SDL 2011: 15th International Conference on System Design Languages*, pp. 90-105, Toulouse, France, 2011.