Budapesti Műszaki és Gazdaságtudományi Egyetem
Matematika Intézet - Sztochasztika Tanszék

Nyílt forráskódú szoftver előállítása kooperatív játékok nukleoluszának kiszámítására

DOLGOZAT

Juhász Adél

Témavezető: Dr. Benedek Márton
Tudományos segédmunkatárs
Közgazdaság- és Regionális Tudományi Kutatóközpont

2021

Tartalomjegyzék

1. Bevezetés

Játékelmélettel számos tudományterületen találkozhatunk, például a társadalom- és gazdaságtudományban, pszichológiában, ugyanakkor ez egy olyan ága a matematikának, ami mind a mai napig sokakat megmozgat és rohamosan fejlődik.

Dolgozatomban a kooperatív játékelmélettel foglalkozom, ezen belül pedig a nukleolusz-számítás területét járom körbe.

1.1. Motiváció

A nukleolusz a kooperatív játékelmélet egyik legelterjedtebb megoldáskoncepciója, számos vonzó tulajdonsággal rendelkezik, például (enyhe feltevések mellett) létezik és egyértelmű, a játéknak folytonos függvénye, stb. Ezáltal az egyik legstabilabb, illetve legigazságosabb megoldást biztosítja. Azonban a kiszámítása rendkívül összetett feladat: a nukleolusz a játékosok számában exponenciális méretű, rendezett hiányvektort lexikografikusan minimalizáló (egyénileg racionális) elosztás.

1.2. A feladat leírása

A dolgozatban a legkorszerűbb megoldó algoritmust, a lexikografikus ereszkedés módszerét implementáljuk Python környezetben és végzünk extenzív numerikus összehasonlítást más, a szakirodalom további klasszikus megoldó algoritmusaival. Valamennyi módszerre már létezik C++ nyelven írt kód, azonban a Python környezet akadémiai és ipari területen is gyorsan és széleskörűen terjedő nyelv, amin keresztül nagyon sok potenciális felhasználót tudunk elérni. Ugyanakkor ez a környezet számos kihívást jelent, mivel korábban ilyesmivel még nem foglalkozott senki, így új fejlesztési lehetőségek is felmerülnek.

2. Alapfogalmak

Kooperatív játékkal olyan döntési helyzeteket modellezünk, melyek során a résztvevők együttműködhetnek, amennyiben úgy vélik, hogy az számukra előnyt jelent.

A kooperatív játékok két csoportra oszthatók, az átváltható (TU - transferable utility), illetve nem átváltható hasznossággal (NTU - non-transferable utility) rendelkező játékokra. A továbbiakban mi a TU-játékokat fogjuk tekinteni.

2.1. Definíció (TU-játék). *Egy TU-játék az egy (N, v) pár, ahol N a játékosok halmaza, ami egy nem üres és véges halmaz, $v : 2^N \to \mathbb{R}$ pedig a koalíciós függvény, amelyre teljesül, hogy $v(\emptyset) = 0$.*

- Az (N, v) játék n-szereplős, ha $|N| = n$.
- Ha $S \subseteq N$ a játékosok egy tetszőleges koalíciója, akkor a koalíciós függvény megadja annak $v(S)$ értékét, ami az az érték, amit az S koalíció el tud érni, amennyiben a tagjai együttműködnek, a többi játékostól függetlenül.

Egy játék kimeneteleként megadjuk, hogy egy adott játékosnak mekkora részesedést tulajdonítunk az együttműködés által teremtett értékből. Jelöljük x_i-vel az i játékos kifizetését, a kooperatív döntési helyzet egy lehetséges kimenetelét pedig a játékosok kifizetéseit tartalmazó $\mathbf{x} = (x_i)_{i \in N} \in \mathbb{R}^N$ kifizetés-vektorral. Ennek célja egy igazságos, illetve stabil elosztás megtalálása.

2.2. Definíció. *Azt mondjuk, hogy az (N, v) játékban az $\mathbf{x} = (x_i)_{i \in N} \in \mathbb{R}^N$ kifizetés-vektor*

- *szétosztás, más néven preimputáció, ha $\sum_{i \in N} x_i = v(N)$;*
- *elosztás, más néven imputáció, ha szétosztás és $x_i \geq v(\{i\})$ minden $i \in N$-re.*

A szétosztás és az elosztás természetes követelmények: az előbbi azt írja elő, hogy pontosan a nagykoalíció értékét osszuk szét a játékosok között, utóbbi pedig ezen felül az egyéni racionalitást ragadja meg, azaz minden játékos kapja meg legalább azt az értéket, amit önmaga egyedül is el tudna érni.

Ezentúl jelöljük a preimputációk halmazát **PI**-vel, az imputációkét pedig **I**-vel. Annak ellenére, hogy **I** lehet üres, a **PI** minden játék esetén egy nemüres, (n-1)-dimenziós hipersík.

2.3. Definíció (Hiány). *Minden egyes* $\mathbf{x}$ *kimenetelre az* S *koalícióhoz tartozó hiányt a következőképpen kaphatjuk meg:*

$$d(S, \mathbf{x}) := v(S) - \mathbf{x}(S),$$

továbbá $d(N, \mathbf{x}) = 0$*, ha* $\mathbf{x}$ *szétosztás, illetve* $d(\emptyset, \mathbf{x}) = 0$*.*

Mindezek alapján a hiányt tekinthetjük úgy, hogy minél nagyobb a hiány értéke az S koalícióra nézve, az annál elégedetlenebb az $\mathbf{x}$ kimenetellel.

2.4. Definíció (Hiányvektor). *Az* $\mathbf{x}$ *kimenetelhez tartozó hiányvektor:*

$$E(\mathbf{x}) = [d(S_1, \mathbf{x}), d(S_2, \mathbf{x}), \ldots, d(S_{2^n-2}, \mathbf{x})] \in \mathbb{R}^{2^n-2}$$

Mivel a hiány definíció szerint 0, az üres halmaz esetén, illetve amennyiben $\mathbf{x}$ szétosztás, úgy a nagykoalíció esetében is, ezért számunkra csak a fennmaradó $2^n - 2$ db hiány érdekel.

A hiányfogalom segítségével meghatározhatjuk a kifizetésvektorok egyes halmazait.

2.5. Definíció (Mag). *A koalíciókra nézve racionális kifizetésvektorok alkotják a játék magját [1, 2]:*

$$c(v) = \{\mathbf{x} \in \boldsymbol{I} : d(\mathbf{x}, S) \leq 0 \ \forall S \subsetneq N\}.$$

A mag azonban lehet üres, illetve lehet egynél több eleme is.

Egy mag-elosztás stabil, mert minden koalíciónak megéri, vegyük észre azonban, hogy exponenciálisan sok egyenlőtlenség határozza meg. Nem minden játékban létezik mag-elosztás, ugyanakkor ennek eldöntése rendkívül nehéz feladat. Ha pedig mégis létezik, akkor sem feltétlenül egyértelmű.

2.6. Definíció (ε-mag). *A relaxációja vagy éppen megszorítása (ε előjelétől függően) a magnak:*

$$c_\varepsilon(v) = \{\mathbf{x} \in \boldsymbol{I} : d(\mathbf{x}, S) \leq \varepsilon \ \forall S \subsetneq N\}.$$

Minden játék esetén van olyan kellően nagy ε, amire ez már nemüres. Innen már értelemszerűen adódik a kérdés, hogy melyik a legkisebb ilyen ε, amire ez teljesül.

2.7. Definíció (Szűkmag). *A szűkmag az*

$$lc(v) = c_{\varepsilon^*}(v)$$

halmaz, ahol ε^* a legkisebb olyan érték, melyre a halmaz nem üres, azaz $\varepsilon^* = \min\{\varepsilon : c_\varepsilon(v) \neq \emptyset\}$.

A szűkmag esetében a nemürességet garantált, azonban az egyértelműség még mindig nem.

Tehát a cél, hogy tovább szűkítsük a kifizetésvektorok halmazát mindaddig, amíg egyértelműségre nem jutunk. Ezt pedig úgy tehetjük meg, ha kiterjesztjük a szűkmag keresésének koncepcióját további szintekre a következőképpen: a szűkmagot megkaphatjuk, ha minimalizáljuk a legnagyobb hiányt a koalíciók körében, majd pedig a lehető legkisebbre csökkentjük azon koalícióknak a számát, amelyek ezzel a legnagyobb hiánnyal rendelkeznek. Ezt követően minimalizáljuk a második legnagyobb hiányt, illetve azon koalíciók számát, amelyekre ez vonatkozik, és ezt így folytatjuk, míg meg nem kapjuk a nukleoluszt.

Mindehhez azonban szükségünk van egy rendezésre, melynek segítségével összehasonlíthatóvá válnak az egyes hiányvektorok, illetve értelmezhető a minimalizálás fogalma is.

2.8. Definíció (Lexikografikus rendezés). *Egy adott* $\mathbf{x}$ *szétosztás esetén tekintsük a* $\Theta(\mathbf{x}) = (\Theta_1(\mathbf{x}), \Theta_2(\mathbf{x}), \ldots, \Theta_{2^n-2}(\mathbf{x}))$ *vektort, ahol az egyes értékek az* $\mathbf{x}$ *vektorhoz tartozó* $2^n - 2$ *db (nemtriviális) hiányértéket jelölik nem növekvő sorrendben, azaz* $\Theta_i(\mathbf{x}) \geq \Theta_{i+1}(\mathbf{x})$ *minden* $1 \leq i < 2^n$*-re.*

Ekkor az $\mathbf{x}$ *vektor lexikografikusan kisebb az* $\mathbf{y}$*-nál* ($\Theta(\mathbf{x}) <_L \Theta(\mathbf{y})$)*, ha létezik olyan* $r \leq 2^n$*, hogy* $\Theta_i(\mathbf{x}) = \Theta_i(\mathbf{y})$ *minden* $1 \leq i < r$ *és* $\Theta_r(\mathbf{x}) < \Theta_r(\mathbf{y})$*.*

2.9. Definíció (Nukleolusz). *Ekkor* $\nu = \nu(N, v) \in \boldsymbol{I}$ *a nukleolusz, ha* $\Theta(\nu) <_L \Theta(\mathbf{x})$ *minden* $\mathbf{x} \in \boldsymbol{I}$*,* $\mathbf{x} \neq \nu$*.*

Schmeidler [3] megmutatta, hogy a nukleolusz több vonzó tulajdonsággal is rendelkezik, amennyiben $\mathbf{I} \neq \emptyset$:

- mindig létezik,
- egyértelmű,
- minden $\varepsilon \geq \varepsilon^*$-ra az ε-mag tartalmazza,
- következményképpen a szűkmag is tartalmazza, illetve
- folytonos függvénye v-nek.

A definícióból adódóan innentől feltesszük, hogy olyan játékkal van dolgunk amelyben az I nem üres. Azonban minden ami bemutatásra kerül, az a megfelelő egyszerűsítésekkel (imputációs korlátok és következményeik elhagyása) az üres imputációs halmazzal rendelkező játékokra is adaptálható: ilyenkor a prenukleoluszt tudjuk definiálni, ami a nukleolusz definícója $\mathbf{I}$ helyett **PI**-vel (amiről már kiemeltük, hogy sosem üres), és a fentebb felsorolt tulajdonsággokal továbbra is rendelkezik.

Mindezekből kifolyólag innentől kezdve a fő célunk a nukleolusz kiszámítása, melyre a következőkben bemutatjuk a rendelkezésre álló elméleti módszereket.

3. A nukleolusz kiszámítása

3.1. Primál LP-sorozat

Egy kooperatív játék nukleoluszának kiszámítása során az összes koalíció hiány értékének (együttes) felső korlátját minimalizáljuk egy lineáris program megoldásával.

Az LP sorozat első tagja a következőképpen írható fel:

$$\begin{array}{lll} \min\limits_{x^{(1)},\varepsilon^{(1)}} & \varepsilon^{(1)} & \\ \text{f.h.} \quad \varepsilon^{(1)} + x^{(1)}(S) & \geq v(S) & \forall S \subsetneq N, S \neq \emptyset \\ x^{(1)}(\{i\}) & \geq v(\{i\}) & \forall i \in N \\ x^{(1)}(N) & = v(N) & \end{array} \qquad (\mathbf{P^{(1)}})$$

Jelöljük $\mathcal{P}$-vel az LP megengedett megoldáshalmazát és $\mathcal{P}^*$-gal az optimális megoldáshalmazt. Az optimális (x^*, ε^*) párokból az x^* pontok alkotják a szűkmagot, illetve ε^* a szűkmaghoz tartozó ε-érték. Ahogy azt korábban megállapítottuk, ennek az LP-nek minden játék esetén van optimális megoldása, tehát $\mathcal{P}^*$ sosem üres. Ugyanakkor ε^* egyértelmű, de x^* már nem feltétlenül.

3.1. Definíció (Aktív halmaz). *Legyen $(x^{(1)}, \varepsilon^{(1)}) \in \mathcal{P}^{(1)}$ primál megengedett megoldás, ekkor $\mathcal{T}^{(1)}(x^{(1)}, \varepsilon^{(1)})$ és $\widetilde{\mathcal{T}}^{(1)}(x^{(1)})$ az aktív korlátok halmaza:*

$$\begin{array}{rll} \epsilon^{(1)} + x^{(1)}(S) & = v(S) & \forall S \in \mathcal{T}^{(1)}(x^{(1)}, \varepsilon^{(1)}) \\ x_i^{(1)} & = v(\{i\}) & \forall \{i\} \in \widetilde{\mathcal{T}}^{(1)}(x^{(1)}) \end{array}$$

Az aktív halmazok számosságának igen nagy szerepe van a nukleolusz megtalálásában, ugyanis, ha $x, y \in \mathcal{P}^{(1)*}$ melyekre $j = |\mathcal{T}^{(1)}(x)| < |\mathcal{T}^{(1)}(y)|$, akkor a rendezett hiányvektorok első j elemére igaz a következő: $\Theta(E(x))_i = \Theta(E(y))_i = \varepsilon^{(1)*}$ ha $i \leq j$, azonban $\Theta(E(x))_{j+1} < \varepsilon^{(1)*} = \Theta(E(y))_{j+1}$. Így $\Theta(E(y)) \not\leq_L \Theta(E(x))$, tehát y nem lehet a nukleolusz. Mindezek alapján a ν nukleoluszra teljesül, hogy $|\mathcal{T}^{(1)}(\nu)| \leq |\mathcal{T}^{(1)}(x)|$ bármely $x \in \mathcal{P}^{(1)*}$ esetén.

Optimális x^* esetén a $\mathcal{T}(x^*)$ aktív halmaz esetén a jelülés egyszerűsítése végett elhagyjuk ε^*-ot, ami úgyis egyértelmű.

3.1. Lemma. *Egyértelműen létezik egy aktív halmaz pár* $\mathcal{T}^{(1)*}$ *és* $\widetilde{\mathcal{T}}^{(1)*}$ *melyre* $|\mathcal{T}^{(1)*}| \leq |\mathcal{T}^{(1)}(x)|$ *és* $|\widetilde{\mathcal{T}}^{(1)*}| \leq |\widetilde{\mathcal{T}}^{(1)}(x)|$ *minden* $x \in \mathcal{P}^{(1)*}$. *Sőt,* $\mathcal{T}^{(1)*} \subseteq \mathcal{T}^{(1)}(x)$ *és* $\widetilde{\mathcal{T}}^{(1)*} \subseteq \widetilde{\mathcal{T}}^{(1)}(x)$ *minden optimális* x^* *esetén.*

A $\mathcal{T}^{(1)*} = \mathcal{T}^{(1)}(\nu)$, $\widetilde{\mathcal{T}}^{(1)*} = \widetilde{\mathcal{T}}^{(1)}(\nu)$ párt nevezzük a legkisebb aktív halmaznak.

Ezek után rögzítjük az aktív halmazban lévő koalíciók hiány értékét, avagy a nekik már biztosított kifizetés $x(S)$ értékét. Egyúttal ezek a koalíciók kikerülnek az egyenlőtlenség korlát alól. Innentől kezdve a cél a még nem rögzített koalíciók hiány értékének minimalizálása.

$$
\begin{array}{lrll}
\min\limits_{x^{(2)},\varepsilon^{(2)}} & \varepsilon^{(2)} & & \\
\text{f.h.} & \varepsilon^{(2)} + x^{(2)}(S) & \geq v(S) & \forall S \subsetneq N, S \neq \emptyset, S \notin \mathcal{T}^{(1)*} \\
& x^{(2)}(\{i\}) & \geq v(\{i\}) & \forall i \in N, \{i\} \notin \widetilde{\mathcal{T}}^{(1)*} \\
& x^{(2)}(S) & = v(S) - w_S^* & \forall S \in \mathcal{H}^{(1)}
\end{array}
\qquad (\mathbf{P^{(2)}})
$$

ahol $\mathcal{H}^{(1)} = N \cup \mathcal{T}^{(1)*} \cup \widetilde{\mathcal{T}}^{(1)*}, w_S^* = \varepsilon^{(1)*}$, ha $S \in \mathcal{T}^{(1)*}$, $w_{\{i\}}^* = 0$, ha $\{i\} \in \widetilde{\mathcal{T}}^{(1)*}$ $w_N^* = 0$.

Ahhoz, hogy az egyenlőséggel teljesülő korlátok számát csökkenthessük, bevezetjük az egyes koalíciókhoz tartozó karakterisztikus vektorok, illetve a koalíciós altér fogalmát.

3.2. Definíció (Karakterisztikus vektor). *Az* S *koalícióhoz tartozó karakterisztikus vektor:*

$$e(S) \in \{0,1\}^n : e(S)_j = 1 \iff j \in S.$$

3.3. Definíció (Koalíciós altér). *Legyen* $\mathcal{B}$ *koalíciók egy kollekciója. Ekkor az általa kifeszített altér tartalmazza* S*-t, tehát* $S \in \operatorname{span}(\mathcal{B})$, *ha* $\exists \beta \in \mathbb{R}^{|\mathcal{B}|}$*:*

$$e(S) = \sum_{Q \in \mathcal{B}} \beta_Q e(Q)$$

3.4. Definíció (Koalícióhalmazok rangja). $\operatorname{rank}(\mathcal{B}) = r$, *ha* $\exists \mathcal{R} \subseteq \mathcal{B}$ *legszűkebb részhalmaz, melyre* $|\mathcal{R}| = r$ *és* $S \in \operatorname{span}(\mathcal{B}) \iff S \in \operatorname{span}(\mathcal{R})$

Ezek után az LP a következő formába írható át:

$$\begin{array}{lrll} \min\limits_{x^{(2)},\varepsilon^{(2)}} & \varepsilon^{(2)} & & \\ \text{f.h.} & \varepsilon^{(2)} + x^{(2)}(S) & \geq v(S) & \forall S \notin \operatorname{span}(\mathcal{R}^{(1)}) \\ & x^{(2)}(\{i\}) & \geq v(\{i\}) & \forall \{i\} \notin \operatorname{span}\mathcal{R}^{(1)} \\ & x^{(2)}(S) & = v(S) - w^*_S & \forall S \in \mathcal{R}^{(1)} \end{array} \qquad (\mathbf{P^{(2)}})$$

ahol $\operatorname{span}(\mathcal{R}^{(1)}) = \operatorname{span}(\mathcal{H}^{(1)}), |\mathcal{R}^{(1)}| = \operatorname{rank}(\mathcal{R}^{(1)})$.

Ezt az LP-t már általánosíthatjuk tetszőleges k-ra:

$$\begin{array}{lrll} \min\limits_{x^{(k)},\varepsilon^{(k)}} & \varepsilon^{(k)} & & \\ \text{f.h.} & \varepsilon^{(k)} + x^{(k)}(S) & \geq v(S) & \forall S \notin \operatorname{span}(\mathcal{R}^{(k-1)}) \\ & x^{(k)}(\{i\}) & \geq v(\{i\}) & \forall \{i\} \notin \operatorname{span}\mathcal{R}^{(k-1)} \\ & x^{(k)}(S) & = v(S) - w^*_S & \forall S \in \mathcal{R}^{(k-1)} \end{array} \qquad (\mathbf{P^{(k)}})$$

ahol

- $\mathcal{T}^{(0)*} = N, \widetilde{\mathcal{T}}^{(0)*} = \emptyset, \mathcal{H}^{(k)} = \bigcup\limits_{i=0}^{k} (\mathcal{T}^{(i)*} \cup \widetilde{\mathcal{T}}^{(i)*})$,
- $\operatorname{span}(\mathcal{R}^{(k)}) = \operatorname{span}(\mathcal{H}^{(k)}), |\mathcal{R}^{(k)}| = \operatorname{rank}(\mathcal{R}^{(k)})$,
- $\forall k \geq 1 : w^*_S = \varepsilon^{(k)*}$ minden $S \in \mathcal{T}^{(k)*}$, $w^*_{\{i\}} = 0$ minden $\{i\} \in \widetilde{\mathcal{T}}^{(k)*}$ és $w^*_N = 0$.

Fontos megjegyezni, hogy legfeljebb $(n-1)$-szer kell megoldanunk $(\mathbf{P^{(k)}})$-t, ugyanis

$$n \geq \operatorname{rank}(\mathcal{R}^{(k)}) \geq \operatorname{rank}(\mathcal{R}^{(k-1)}) + 1 \geq k + \operatorname{rank}(\mathcal{R}^{(0)}) = k + 1$$

3.2. Duál LP-sorozat

A primál LP-sorozathoz a következő duál LP-sorozatot írhatjuk fel:

$$\begin{aligned}
\max_{u^{(k)},w^{(k)},z^{(k)}} \quad & \sum_{S\notin \mathrm{span}(\mathcal{R}^{(k-1)})} u_S^{(k)} v(S) + \sum_{\{i\}\notin \mathrm{span}(\mathcal{R}^{(k-1)})} w_{\{i\}}^{(k)} v(\{i\}) + \sum_{S\in\mathcal{R}^{(k-1)}} z_S^{(k)}(v(S)-w_S^*) \\
\text{f.h.} \quad & \sum_{S\notin \mathrm{span}(\mathcal{R}^{(k-1)})} u_S^{(k)} e(S)_i + \sum_{\{i\}\notin \mathrm{span}(\mathcal{R}^{(k-1)})} w_{\{i\}}^{(k)} e(\{i\})_i + \sum_{S\in\mathcal{R}^{(k-1)}} z_S^{(k)} e(S)_i = 0 \quad \forall i \in N \\
& \sum_{S\notin \mathrm{span}(\mathcal{R}^{(k-1)})} u_S^{(k)} = 1 \\
& u^{(k)}, w^{(k)} \geq 0
\end{aligned} \tag{$\mathbf{D^{(k)}}$}$$

A nukleolusz kiszámításához dönthetünk úgy is, hogy a primál LP-sorozat helyett a duál LP-sorozattal dolgozunk, mivel utóbbi segítségével is előállíthatjuk a korábban definiált legkisebb aktív halmazt.

A primál LP-hez hasonlóan itt is bevezetjük a következő jelöléseket: legyen $\mathcal{D}$ a duál megengedett megoldások halmaza és $\mathcal{D}^*$ a duál optimális megoldások halmaza.

3.5. Definíció (Duál inaktív halmaz). *Duál megengedett megoldás $(u^{(k)}, w^{(k)}) \in \mathcal{D}^{(k)}$ esetén a duál inaktív halmazok:*

$$\mathcal{DNT}^{(k)}(u^{(k)}) = \{S \notin \mathrm{span}(\mathcal{R}^{(k-1)}) : u_S^{(k)} > 0\}$$

$$\widetilde{\mathcal{DNT}}^{(k)}(w^{(k)}) = \{\{i\} \notin \mathrm{span}(\mathcal{R}^{(k-1)}) : w_{\{i\}}^{(k)} > 0\}$$

A primálhoz teljesen analóg módon a duál esetében létezik egy legnagyobb inaktív halmaz, mely minden duál inaktív halmazt tartalmaz. Ez a legnagyobb inaktív halmaz pedig megegyezik a legkisebb aktív halmazzal a primál LP esetében:

3.1. Tétel. *Tetszőleges $x^{(k)} \in \mathcal{P}^{(k)*}$, $(u^{(k)}, w^{(k)}) \in \mathcal{D}^{(k)*}$ primál-duál optimális pár esetén*

$$\mathcal{DNT}^{(k)}(u^{(k)}) \subseteq \mathcal{DNT}^{(k)*} = \mathcal{T}^{(k)*} \subseteq \mathcal{T}^{(k)}(x^{(k)})$$

$$\widetilde{\mathcal{DNT}}^{(k)}(w^{(k)}) \subseteq \widetilde{\mathcal{DNT}}^{(k)*} = \widetilde{\mathcal{T}}^{(k)*} \subseteq \widetilde{\mathcal{T}}^{(k)}(x^{(k)})$$

Ez a tétel biztosítja, hogy a duál LP-sorozat megoldásával is előállíthatjuk a nukleoluszt.

Mindezek tudatában a ν nukleoluszt LP-sorozat segítségével kiszámító algoritmust a következőképpen írhatjuk fel:

Algorithm 1: v kooperatív játék ν nukleoluszának kiszámítása

Input: (N, v);

1. Inicializáció: $k = 1, \mathcal{R}^{(0)} = N, w_N^* = 0$;

while $\underline{\text{rank}(\mathcal{R}^{(k-1)}) < n}$ **do**

2. Megoldjuk $(\mathbf{P^{(k)}})$-t (vagy $(\mathbf{D^{(k)}})$-t): meghatározzuk $(x^{(k)}, \epsilon^{(k)*}) \in \mathcal{P}^{(k)*}$ (vagy $(u^{(k)}, z^{(k)}) \in \mathcal{D}^{(k)*}$);

3. Meghatározzuk a legkisebb aktív halmaz egy részhalmazát: $\mathcal{T} \subseteq \mathcal{T}^{(k)*}$, $\widetilde{\mathcal{T}} \subseteq \widetilde{\mathcal{T}}^{(k)*}$;

4. Legyen $\mathcal{R}^{(k-1)}$: $\text{span}\left(\mathcal{R}^{(k)}\right) = \text{span}\left(\mathcal{R}^{(k-1)} \cup \mathcal{T} \cup \widetilde{\mathcal{T}}\right)$, $\text{rank}\left(\mathcal{R}^{(k)}\right) = |\mathcal{R}^{(k)}|$, $k = k + 1$;

5. Meghatározzuk minden $S \notin \text{span}\left(\mathcal{R}^{(k-1)}\right)$ koalíciót $(\mathbf{P^{(k)}})$ (vagy $(\mathbf{D^{(k)}})$) LP-hez;

end

Output: ν az $x(S) = v(S) - w_S^*$ $\forall S \in \mathcal{R}^{(k-1)}$ egyértelmű megoldása;

Ez az algoritmus azonban számos nehézséget rejt magában: egyrészt felmerül a kérdés, hogy pontosan hogyan is oldjuk meg ezt az LP-t, ugyanis az exponenciálisan sok korlát miatt a feladat igen nagy méreteket ölt, másrészt hogyan találjuk meg a legkisebb aktív halmazt az LP megoldása után, harmadrészt pedig hogy hogyan kezeljük azt a feltételt, hogy $S \notin \text{span}(\mathcal{R}^{(k-1)})$.

Ezeket a nehézségeket kiküszöbölve mutat be témavezetőm, Dr. Benedek Márton többedmagával egy olyan hatékony módszert, mely könnyen kezeli ezeket a kihívásokat egy lexikografikus ereszkedő algoritmus segítségével [4].

4. Az algoritmus bemutatása

4.1. Kohlberg kritérium

A lexikografikus ereszkedő algoritmushoz Benedek et al [4] felhasználja a Kohlberg kritériumot [5], melynek segítségével eldönthetjük egy adott x kimenetelről, hogy az a nukleolusz-e vagy sem.

4.1. Definíció (Kiegyensúlyozottság). *Koalíciók egy $Q \subset 2^N$ kollekciója kiegyensúlyozott, ha létezik egy súlyvektor $\omega \in \mathbb{R}_{>0}^{|Q|}$ úgy, hogy $\mathbf{e}(N) = \sum_{S \in Q} \omega_S \mathbf{e}(S)$.*

Egy adott $T_0 \subset 2^N$ esetén a $Q \subset 2^N$ kollekció T_0-kiegyensúlyozott, ha létezik $\gamma \in \mathbb{R}_{\geq 0}^{|T_0|}$ és $\omega \in \mathbb{R}_{>0}^{|Q|}$ súlyvektorok úgy, hogy $\mathbf{e}(N) = \sum_{S \in T_0} \gamma_S \mathbf{e}(S) + \sum_{S \in Q} \omega_S \mathbf{e}(S)$.

Kohlberg először meghatározza a következő koalíciós halmazokat [5]: $T_0(\mathbf{x}) = \{\{i\}, i = 1, ..., n : x_i = v(\{i\})\}$, $H_0(\mathbf{x}) = \{N\}$ és $H_k(\mathbf{x}) = H_{k-1}(\mathbf{x}) \cup T_k(\mathbf{x})$, $k = 1, 2, ...$, ahol minden $k \geq 1$-re

$$T_k(\mathbf{x}) = \underset{S \notin H_{k-1}(\mathbf{x})}{\arg\max} \{v(S) - x(S)\}$$

$$\epsilon_k(\mathbf{x}) = \max_{S \notin H_{k-1}(\mathbf{x})} \{v(S) - x(S)\}$$

Ekkor $T_k(\mathbf{x})$ tartalmazza az összes olyan koalíciót, amelynek a hiányértéke pontosan $\epsilon_k(\mathbf{x})$, ugyanakkor $\epsilon_1(\mathbf{x}) > \epsilon_2(\mathbf{x}) > ...$ és $T_0(\mathbf{x})$ azon játékosok halmaza, amelyek esetén $\mathbf{x}$ az egyéni racionalitás határán van.

Mindezek alapján a Kohlberg kritérium algoritmikus megközelítésből a következőképp írható fel:

Algorithm 2: (Eredeti) Kohlberg algoritmus

Input: (N, v) játék, $\mathbf{x} \in \mathbf{I}$ imputáció;

Output: $\mathbf{x}$ a nukleolusz vagy sem;

1. Inicializáció: Legyen

$H_0(\mathbf{x}) = \{N\}, T_0(\mathbf{x}) = \{\{i\}, i = 1, ..., n : x_i = v(\{i\})\}$ és $k = 1$;

while $H_{k-1} \neq 2^N \backslash \{\emptyset\}$ **do**

2. Legyen $T_k(\mathbf{x}) = \arg\max_{S \notin H_{k-1}(\mathbf{x})} \{v(S) - x(S)\}$;

if $\cup_{j=1}^{k} T_j$ T_0-kiegyensúlyozott **then**

3. Legyen $H_k = H_{k-1} \cup T_k, k = k + 1$ és folytassuk

else

4. Stop és $\mathbf{x}$ nem a nukleolusz

end

end

5. $\mathbf{x}$ a nukleolusz.

Ebben az algoritmusban iteratíven képezzük a T_j halmazokat, mindaddig, amíg az összes koalíciót már bevettük és megállapítottuk, hogy az adott kimenetel a nukleolusz, vagy megállunk egy ponton, ahol a kollekciók uniója már nem T_0-kiegyensúlyozott, ahol pedig megállapítjuk, hogy a kimenetel nem a nukleolusz.

4.1.1. Egy továbbfejlesztett Kohlberg kritérium

A Kohlberg kritérium egy jó módszer annak megállapítására, hogy egy adott kimenetel a nukleolusz-e vagy sem szükséges és elégséges feltételek biztosításával. Azonban a feltételek ellenőrzése során exponenciálisan sok részhalmaz kiegyenesúlyozottságát kell megvizsgálni, amely részhalmazok számossága exponenciális is lehet. Ez viszonylag kisebb játékok esetén még elfogadható, azonban nagyobb játékok során már kezelhetetlenné válik.

Ezen probléma megoldása egy továbbfejlesztett Kohlberg kritérium, ahol a kulcsfontosságú gondolat az, hogy amennyiben megállapítottuk, hogy $\cup_{j=1}^{k} T_j$ T_0-kiegyensúlyozott, utána már felesleges foglalkozni azon koalíciókkal, melyek $\text{span}(\cup_{j=1}^{k} T_j)$-ban vannak, ugyanis amennyiben egy koalíció T_0-kiegyensúlyozott, úgy az általa kifeszített koalíciós altér szintén T_0-kiegyensúlyozott.

Mindezek alapján a Kohlberg algoritmust a következőképpen írhatjuk fel:

Algorithm 3: Továbbfejlesztett Kohlberg algoritmus

Input: (N, v) játék, $\mathbf{x} \in \mathbf{I}$ imputáció;

Output: $\mathbf{x}$ a nukleolusz vagy sem;

1. Inicializáció: Legyen

$H_0(\mathbf{x}) = \{N\}, T_0(\mathbf{x}) = \{\{i\}, i = 1, ..., n : x_i = v(\{i\})\}$ és $k = 1$;

while $\underline{\text{rank}(H_{k-1}) < n}$ **do**

2. Legyen $T_k(\mathbf{x}) = \arg\max_{S \notin \text{span}(H_{k-1}(\mathbf{x}))}\{v(S) - x(S)\}$;

if $\underline{\cup_{j=1}^{k} T_j}$ T_0-kiegyensúlyozott **then**

3. Legyen $H_k = H_{k-1} \cup T_k, k = k + 1$ és folytassuk

else

4. Stop és $\mathbf{x}$ nem a nukleolusz

end

end

5. $\mathbf{x}$ a nukleolusz.

Ezen módosítás eredményeképp azt kapjuk, hogy a továbbfejlesztett Kohlberg algoritmus *while*-ciklusa legfeljebb $(n - 1)$ iteráció után leáll.

4.2. A kiegyensúlyozottság ellenőrzése

A Kohlberg kritérium szerint ahhoz, hogy ellenőrizzük a T kollekció T_0-kiegyensúlyozottságát, meg kell vizsgálnunk, hogy léteznek-e $\gamma \in \mathbb{R}_{\geq 0}^{|T_0|}$ és $\omega \in \mathbb{R}_{>0}^{|T|}$ súlyvektorok úgy, hogy

$$\mathbf{e}(N) = \sum_{S \in T_0} \gamma_S \mathbf{e}(S) + \sum_{S \in Q} \omega_S \mathbf{e}(S).$$

Solymosi és Sziklai [6] megközelítése alapján meg kell oldanunk $|T|$ darab lineáris programot a következőképpen megadva. Minden $\mathcal{C} \in T$-re legyen

$$q_{\mathcal{C}}^* = \left\{ \max \omega_{\mathcal{C}} : \sum_{S \in T_0} \gamma_S \mathbf{e}(S) + \sum_{S \in T} \omega_S \mathbf{e}(S) = \mathbf{e}(N), (\gamma, \omega) \in \mathbb{R}_{\geq 0}^{|T_0|+|T|} \right\}.$$

Ekkor T T_0-kiegyensúlyozott pontosan akkor, ha $q_{\mathcal{C}}^* > 0$ minden $\mathcal{C} \in T$-re.

Ugyanakkor a Kohlberg kritériumban szereplő T kollekciók exponenciálisan sok koalícióból is állhatnak, így nagyobb játékokra nem praktikus megoldani

a $|T|$ darab lineáris programot. Ezen javítva Solymosi [7] bemutatott egy gyorsabb megközelítést, amely során legfeljebb $\text{rank}(T)$ darab lineáris program megoldása szükséges. Mindezeket felhasználva Benedek [4] egy olyan algoritmust mutat be, amely nem csupán a kiegyensúlyozottság tényét állapítja meg a T halmazra nézve, hanem megtalálja a legnagyobb T_0-kiegyensúlyozott részhalmazát T-nek. Az algoritmus a következőképpen írható fel:

Algorithm 4: Algoritmus a legnagyobb T_0-kiegyensúlyozott részkollekció megtalálására

Input: T kollekció, T_0-kiegyensúlyozott $Q \subsetneq T$ részkollekcióval;

Output: $U \subset T$ legnagyobb T_0-kiegyensúlyozott részkollekció;

1. Inicializáció: Legyen $U = \text{span}(Q) \cap T$;

while $\text{rank}(U) < \text{rank}(T)$ **do**

2. Keressünk olyan $\gamma^* \in \mathbb{R}_{\geq 0}^{|T_0|}, \omega^* \in \mathbb{R}_{\geq 0}^{|T\setminus U|}, \mu^* \in \mathbb{R}^{|U|}$, melyek megoldják

$$\arg\max_{\gamma,\omega,\mu} \left\{ \sum_{S\in T\setminus U} \omega_S : \sum_{S\in T_0} \gamma_S \mathbf{e}(S) + \sum_{S\in T\setminus U} \omega_S \mathbf{e}(S) + \sum_{S\in U} \mu_S \mathbf{e}(S) = \mathbf{e}(N) \right\}$$

if $\omega^* = \mathbf{0}$ vagy az LP infízibilis **then**

3. Stop és output $U \subsetneq T$;

else

4. Legyen $U = \text{span}(U \cup \{S : \omega_S^* > 0\}) \cap T$;

end

end

5. Output $U = T$.

Ez az algoritmus legfeljebb $\text{rank}(T) - \text{rank}(Q)$ iteráció után leáll.

4.3. Az algoritmus

Benedek [4] lexikografikus ereszkedő algoritmusa felhasználja a továbbfejlesztett Kohlberg kritériumot, azonban nem csupán megerősíti, hogy az adott kifizetésvektor a nukleolusz-e, hanem negatív válasz esetén egy hatékony módszert is ad annak megtalálására.

Ez az új algoritmus a következő lépésekből áll:

- Bármely $\mathbf{x} \in \mathbf{I}$ imputációval kezdve végrehajt egy (lokális) optimalitási tesztet.
- Ha ezt a tesztet $\mathbf{x}$ elbukja, generál egy $\mathbf{y}$ javító irányt és egy α lépéshosszt.
- Frissítjük $\mathbf{x}$-et: $\mathbf{x} = \mathbf{x} + \alpha\mathbf{y}$ és ezeket a lépéseket ismételjük, amíg már nem találunk javító irányt.

4.3.1. Javító irányok generálása

Ha a Kohlberg algoritmus során egy olyan T halmazt találunk, amely nem T_0-kiegyensúlyozott, úgy lehetséges egy $\mathbf{y}$ javító irány generálása, melyre teljesül, ha ellépünk $\mathbf{x}$-ből $\mathbf{x} + \alpha\mathbf{y}$-ba, akkor ez az új kifizetésvektor növeli az elégedettségét a lehető legtöbb, elégedetlen, kiegyensúlyozatlan koalíciónak, miközben a már rögzített, kiegyensúlyozott koalíciók hiányértékét nem változtatja.

Amikor az Algortimus 5 kilép $\operatorname{rank}(U) < \operatorname{rank}(T)$-vel, az egyenletrendszer

$$\sum_{S \in T_0} \gamma_S \mathbf{e}(S) + \sum_{S \in T \setminus U} \omega_S \mathbf{e}(S) + \sum_{S \in U} \mu_S \mathbf{e}(S) = \mathbf{e}(N)$$

$$\begin{aligned} \omega_Q &> 0 \\ \gamma_S, \omega_P &\geq 0 \quad \forall S \in T_0, P \in T \setminus U \\ \mu_S &\in \mathbb{R} \quad \forall S \in U \end{aligned}$$

infízibilis minden $Q \in T \setminus U$ esetén, tehát a Farkas-lemma alapján

$$\{\mathbf{y} \in \mathbb{R}^n : \mathbf{y}(Q) > 0, \mathbf{y}(P) \geq 0, \forall P \in T_0 \cup (T \setminus U), \mathbf{y}(S) = 0, \forall S \in U \cup \{N\}\} \neq \emptyset.$$

Az egyes $\mathbf{y}$ irányok eltérőek lehetnek különböző Q esetén, így vehetjük az átlagukat (vagy összegüket), hogy megkapjunk egy normalizált $\mathbf{y}$-t a következő halmazból:

$$\{\mathbf{y} \in \mathbb{R}^n : \mathbf{y}(Q) \geq 1, \forall Q \in T \backslash U, \mathbf{y}(P) \geq 0, \forall P \in T_0, \mathbf{y}(S) = 0, \forall S \in U \cup \{N\}\}.$$

Ugyanakkor igaz az, hogy amikor az összes k-ra teljesülő T_0-kiegyensúlyozottságát ellenőrizzük az $\cup_{j=1}^{k} T_j$ kollekciónak, elég, ha csupán az aktuális T_k kollekcióhoz tartozó súlyok esetén kívánjuk meg a szigorú pozitivitást, amennyiben azt találtuk, hogy a kollekció $(k-1)$-ig T_0-kiegyensúlyozott.

Ha T nem T_0-kiegyensúlyozott a k. iterációban, akkor még mindig fízibilis megoldást kapunk, ha megkívánjuk $\mathbf{y}(Q) = 0$-t minden $Q \in \cup_{j=1}^{k-1} T_j \cup \{N\}$ koalícióra.

Ráadásul, mivel minden $S \in \cup_{j=1}^{k} T_j \cup \{N\}$ koalícióhoz létezik $\lambda \in \mathbb{R}^{|R_{k-1}|}$ úgy, hogy $\mathbf{y}(S) = \sum_{Q \in R_{k-1}} \lambda_Q \mathbf{y}(S)$, a következő halmaz sem üres:

$$\{\mathbf{y} \in \mathbb{R}^n : \mathbf{y}(Q) \geq 1, \forall Q \in T_k \backslash U, \mathbf{y}(P) \geq 0, \forall P \in T_0, \mathbf{y}(S) = 0, \forall S \in R_{k-1} \cup (U \cap T_k)\}. \tag{1}$$

Az (1)-beli $\mathbf{y}$ vektorokat *javító irány*nak nevezzük.

4.3.2. Lépéshossz

Egy $\mathbf{y}$ fízibilis pont (1)-ben egy javító irány abban az értelemben, hogy ha az aktuális $\mathbf{x}$ pontból $\mathbf{y}$ mentén mozgunk, akkor javítunk azon koalíciók elégedettségén, amelyek jelenleg a legelégedetlenebbek és a kiegyensúlyozatlanságot okozzák, miközben a korábban már ellenőrzött, kiegyensúlyozott részkollekciók elégedettségén nem változtatunk, ugyanakkor biztosítja, hogy megfelelő lépéshossz esetén az imputációs halmazban maradunk.

Egy adott $\mathbf{y}$ javító irányhoz tartozó megfelelő $\alpha > 0$ lépéshossz meghatározásakor elkerülendőek a túl kicsi lépések, melyek nem változtatnak T-n, ugyanakkor szeretnénk növelni a lépéshosszt mindaddig, amíg változást nem észlelünk T-ben (vagy T_0-ban), annak reményében, hogy az új kollekció is T_0-kiegyensúlyozott.

Tegyük fel, hogy a k. iterációban az $\mathbf{x}$ imputációnál vagyunk. Minden S koalícióra a hiány változása a következőképpen alakul α lépéshossz és $\mathbf{y}$

javító irány esetén:

$$d(S, \mathbf{x} + \alpha\mathbf{y}) - d(S, \mathbf{x}) = v(S) - (\mathbf{x}(S) + \alpha\mathbf{y}(S)) - (v(S) - \mathbf{x}(S)) = -\alpha\mathbf{y}(S).$$

Jelenleg a legnagyobb elégedetlenség $\epsilon_k(\mathbf{x}) = d(S, \mathbf{x})$ minden $S \in T_k(\mathbf{x})$ esetén. Tehát elég kicsi $\alpha > 0$-ra az új maximális elégedetlenség $\epsilon_k(\mathbf{x}+\alpha\mathbf{y}) = d(S, \mathbf{x} + \alpha\mathbf{y})$ valamely $S \in T_k(\mathbf{x})$-re. Rögzítsünk egy ilyen $\tilde{S}$ koalíciót, ekkor a maximális elégedetlenség a következőképp változik: $\epsilon_k(\mathbf{x} + \alpha\mathbf{y}) - \epsilon_k(\mathbf{x}) = -\alpha\mathbf{y}(\tilde{S})$.

α megállapítása során alapvetően az érdekel bennünket, hogy az egyes koalíciók hiánya milyen messzire van a maximális hiánytól:

$$(d(S, \mathbf{x}+\alpha\mathbf{y})-\epsilon_k(\mathbf{x}+\alpha\mathbf{y}))-(d(S, \mathbf{x})-\epsilon_k(\mathbf{x})) = \alpha(\mathbf{y}(\tilde{S})-\mathbf{y}(S)) \geq \alpha(1-\mathbf{y}(S)), \tag{2}$$

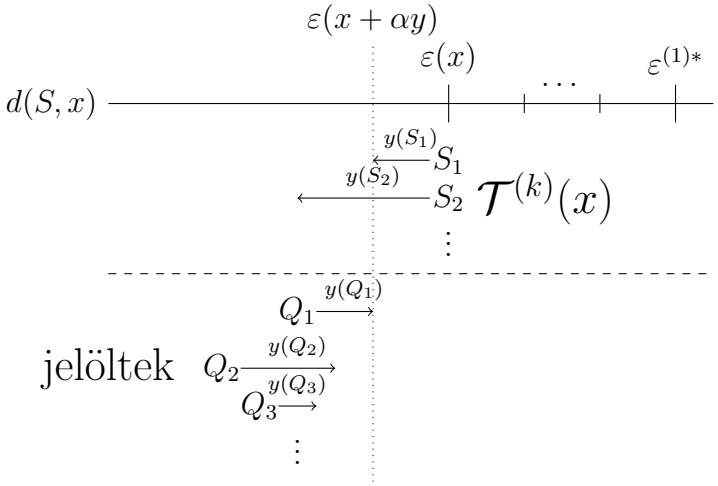
ahol az utolsó egyenlőtlenségben felhasználtuk, hogy $\mathbf{y}(\tilde{S}) \geq 1$.

Ezáltal vissza kell térnünk ahhoz a kérdéshez, hogy pontosan hogyan is válasszuk meg a javító irányt. Mivel minden fízibilis pontja (1)-nek egy alkalmazható javító irány, ezért szabadon megválaszthatjuk az optimalizálni kívánt célfüggvényt. Ahhoz, hogy a (2)-ben használt becslés egyenlőséggel teljesüljön, a célfüggvényt a következőnek választjuk:

$$\begin{array}{lll} \min\limits_{\mathbf{y}} & \sum_{Q \in T_k \backslash U} \mathbf{y}(Q) & \\ \text{f.h.} & \mathbf{y}(Q) \geq 1 & \forall Q \in T_k \backslash U \\ & \mathbf{y}(P) \geq 0 & \forall P \in T_0 \\ & \mathbf{y}(S) = 0 & \forall S \in U \backslash T_k \end{array} \qquad (ID(T_0; T_k; U))$$

$ID(T_0; T_k; U)$ minden optimális $\mathbf{y}$ megoldására teljesül, hogy $\mathbf{y}(\tilde{S}) = 1$. Ahogy 0-ról elkezdjük növelni α-t, a következőket tapasztaljuk: az S koalícióhoz tartozó, (2)-beli aktivitás

- csökken, ha $\mathbf{y}(S) > 1$;
- nem változik, ha $\mathbf{y}(S) = 1$;
- nő, ha $\mathbf{y}(S) < 1$.



1. ábra. Az optimális lépéshossz

Az aktivitás növekedésén az $\epsilon_k(\mathbf{x}+\alpha\mathbf{y})-d(S,\mathbf{x}+\alpha\mathbf{y})$ különbség csökkenését értjük.

Legyen $\mathcal{I}$ azon koalíciók egy kollekciója, amelyeknek növekszik az aktivitása, azaz $\mathcal{I}=\{S\notin \text{span}(R_{k-1})\cup T_k : \mathbf{y}(S)<1\}$. Ezek azok a koalíciók, amelyek beléphetnek az aktív halmazba, ahogy teszünk egy lépést.

Ugyanakkor meg kell szabnunk egy korlátot is α-ra annak érdekében, hogy az imputációs halmazban maradjunk.

Mindkét feltételt figyelembe véve és bevezetve az $\mathcal{N}_0=\{j\in N\backslash T_0 : y_j<0\}$ jelölést, az optimális lépéshossz:

$$\alpha=\min\left(\left\{\frac{\epsilon_k(\mathbf{x})-d(S,\mathbf{x})}{1-\mathbf{y}(S)} : S\in\mathcal{I}\right\}\cup\left\{\frac{x_j-v(\{j\})}{-y_j} : j\in\mathcal{N}_{\prime}\right\}\right). \quad (3)$$

Mindezek után a lexikografikus ereszkedő algoritmus a következőképpen írható fel:

Algorithm 5: Algoritmus egy kooperatív játék nukleoluszának kiszámítására

Input: (N, v) játék, $\mathbf{I} \neq \emptyset$ imputációs halmazzal;

Output: ν nukleolusza az (N, v) játéknak;

1. Inicializáció: Legyen $\mathbf{x} \in \mathbf{I}$ tetszőleges, $R_0 = \{N\}$ és $k = 1$;

while $|R_{k-1}| < n$ **do**

2. Keressük meg

$\epsilon_k(\mathbf{x}) = \max_{S \notin \text{span}(R_{k-1})} d(S, \mathbf{x}), T_k(\mathbf{x}) = \{S \notin \text{span}(R_{k-1}) : d(S, \mathbf{x}) = \epsilon_k\}$,$T_0(\mathbf{x}) = \{\{i\}, i = 1, ..., n : x_i = v(\{i\})\}$ és $U \subset (R_{k-1} \cup T)$, melyeket az Algoritmus 5 által kaptunk;

if $T_k \backslash U \neq \emptyset$ **then**

3. Keressük meg $\mathbf{y}$-t $ID(T_0; T_k; U)$-t megoldva és α-t 3-at használva. Frissítsük $\mathbf{x}$-et: $\mathbf{x} = \mathbf{x} + \alpha\mathbf{y}$ és térjünk vissza a **2.** lépéshez;

else

4. Legyen $\mathcal{R}_k$ olyan, hogy $\text{span}(\mathcal{R}_k) = \text{span}(\mathcal{R}_{k-1} \cup \mathcal{T})$, $\text{rank}(\mathcal{R}_k) = |\mathcal{R}_k|$, $k = k + 1$;

end

end

5. $\mathbf{x} = \nu$ a nukleolusz.

4.4. A szimplex módszer egy implementálása

Annak érdekében, hogy az algoritmus futási idejét csökkentsük, a lépéshosszszámítás során Derks és Kuipers [9] módszerét alkalmaztuk, ugyanakkor az összehasonlítandó algoritmusok közé bevettük az általuk a prenukleoluszra kidolgozott módszer nukleoluszra kibővített változatát. Az alábbiakban leírjuk az eredeti megvalósítást.

Tegyük fel, hogy a következő problémát szeretnénk megoldani:

$$\mathcal{P} := \min\{t | Fx = c, Ax + \mathbb{1}t \geq b\},$$

ahol A egy $m \times n$-es mátrix, F egy $p \times n$-es, p-rangú mátrix, $\mathbb{1}$ a csupa egyest tartalmazó, m-hosszú vektor és b egy m-hosszú oszlopvektor. Fel-

tesszük, hogy az $Fx = c$ rendszernek van legalább egy megoldása, különben a probléma infízibilis.

Válasszunk egy fízibilis (x^0, t^0) megoldást és egy $A^0x + \mathbb{1}t \geq b^0$ alrendszert úgy, hogy $A^0x^0 + \mathbb{1}t^0 = b^0$ és $\begin{pmatrix} F & 0 \\ A^0 & 1 \end{pmatrix}$ sorai lineárisan függetlenek legyenek. Ez lehetséges, például a következőképpen: legyen x^0 egy megoldása $Fx = c$-nek és definiáljuk $t^0 = \max\{b_i - a_ix^0 | i = 1, ...m\}$. Ekkor (t^0, x^0) fízibilis megoldás, ráadásul A-nak létezik egy sora, a_i, amelyre teljesül, hogy $a_ix^0 + t^0 = b_i$. Tehát ha A^0-t úgy definiáljuk, mint az a_i-t tartalmazó, egyetlen sorból álló mátrixot, akkor x^0, t^0 és A^0 teljesítik a korábban említett követelményeket.

Figyeljük meg, hogy a

$$zF + uA^0 = 0, u\mathbb{1} = 1$$

rendszernek legfeljebb egy megoldása van. Vegyük a következő három esetet.

1. eset. A $zF + uA^0 = 0, u\mathbb{1} = 1$ rendszernek létezik egy (egyértelmű) megoldása: $(z, u) = (\nu, \mu)$ úgy, hogy $\mu \geq 0$. Ekkor (x^0, t^0) egy optimális megoldása $\mathcal{P}$-nek, hiszen

$$\begin{aligned} t^0 &= (\nu F + \mu A^0)x^0 + \mu\mathbb{1}t^0 \\ &= \nu Fx^0 + \mu(A^0x^0 + \mathbb{1}t^0) \\ &= \nu c + \mu b \\ &\geq \max\{zc + ub | zF + uA = 0, u\mathbb{1} = 1, u \geq 0\} \\ &= \min\{t | Fx = c, Ax + \mathbb{1}t \geq b\}. \end{aligned}$$

2. eset. A $zF+uA^0 = 0, u\mathbb{1} = 1$ rendszernek van egy $(z, u) = (\nu, \mu)$ megoldása úgy, hogy μ legalább egy koordinátája negatív. Legyen $\hat{i}$ a legkisebb index, melyre $\mu_{\hat{i}} < 0$ és legyen B^0 A^0-nak az a részmátrixa, melyet úgy kapunk, hogy A^0-ból kitöröljük az $\hat{i}$-hez tartozó sort, $\overline{\mu}$ pedig az a vektor, melyet μ $\hat{i}$-edik koordinátájának törlésével kapunk. Mivel a $zF + uA^0 = 0, u\mathbb{1} = 1$ rendszer megoldása egyértelmű, így nincs olyan megoldás, ahol $u_{\hat{i}} = 0$ lenne.

Ezzel ekvivalensen, a $zF + uB^0 = 0, u\mathbb{1} = 1$ rendszernek nincs megoldása. Ekkor a Farkas lemma alapján az $Fy = 0, B^0y = 0$ rendszernek van megoldása, legyen egy megoldás y^0 és jelölje A^0 törölt sorát $a_{\hat{i}}$. Ekkor $a_{\hat{i}}y^0 > 0$, mivel $\mu_{\hat{i}}a_{\hat{i}}y^0 = -(\nu F + \overline{\mu}B^0)y^0 = -\overline{\mu}\mathbb{1} = \mu_{\hat{i}} - 1$.

3. eset. A $zF + uA^0 = 0, u\mathbb{1} = 1$ nincs megoldása. Hasonlóan a 2. esethez, a Farkas lemma alapján $Fy = 0, A^0y = 0$-nak van megoldása, legyen egy megoldás y^0 és legyen $B^0 = A^0$.

Az első esetben kész vagyunk. A 2. és 3. esetben találtunk egy y^0 vektort úgy, hogy $Fy^0 = 0$ és $A^0y^0 \geq 1$. Ha $Ay^0 \geq 1$, akkor az $(x^0 + \lambda y^0, t^0 - \lambda)$ vektor is fízibilis minden $\lambda \geq 0$-ra, tehát a probléma nem korlátos, úgyhogy ekkor szintén kész vagyunk. Végül nézzük azt az esetet, amikor $ay^0 < 1$ A-nak valamely a sorára. Ekkor legyen λ^0 a legnagyobb olyan λ, melyre $(x^0 + \lambda y^0, t^0 - \lambda)$ még fízibilis, azaz

$$\lambda^0 = \min\left\{\frac{a_jx^0 + t^0 - b_j}{1 - a_jy^0} : a_jy^0 < 1\right\},$$

és legyen $\hat{j}$ a legkisebb index, ahol felvevődik ez a minimum.

Ekkor legyen $(x^1, t^1) = (x^0 + \lambda y^0, t^0 - \lambda)$ és A^1 legyen az a mátrix, melyet B^0-ból kapunk, ha hozzáadjuk az $a_{\hat{j}}$ sort. Figyeljük meg, hogy az $\begin{pmatrix} F & 0 \\ A^1 & 1 \end{pmatrix}$ mátrix sorai lineárisan függetlenek. Sőt, $A^1x^1 + \mathbb{1}t^1 = b^1$, ahol b^1 a b A^1-hez tartozó része. Tehát megismételhetjük ugyanezeket a lépéseket A^1, x^1-gyel, stb.

Az A^i mátrix lépésenként történő frissítése Bland pivotáló szabályaként ismert. Bland [8] megmutatta, hogy a folyamat véges lépésben leáll, ha ezt a pivotáló szabályt alkalmazzuk, és a végén vagy találtunk egy optimális megoldást, vagy pedig megállapítottuk, hogy a probláma nem korlátos.

Ezek után a következő - korábbihoz hasonló - keretrendszerbe foglalhatjuk Derks és Kuipers algoritmusát:

- Megoldjuk a $zF + uA^i = 0, u\mathbb{1} = 1$ egyenletrendszert.
- Amennyiben szükséges, megoldjuk az $Fy = 0, B^iy = 1$ egyenletrendszert.

- Kiszámoljuk az optimális λ^i lépéshosszt.

Azonban egyetlen pivotáló lépés megkíván $\mathcal{O}(n2^n)$ lépést, melyet Derks és Kuipers [9] lecsökkent $\mathcal{O}(2^n)$-re.

Ehhez rendezzük az összes n-hosszú $(0,1)$-vektort (kivéve a nullvektort) lexikografikusan növekvő sörrendbe. Jelöljük az i-edik vektort a_i-vel és legyen f az a 2^n-1-hosszú vektor, melyre teljesül, hogy $f_i = 1$, ha a_i az A-nak egy sora, 0 különben és tegyük fel, hogy az f vektor adott.

Legyen a és b két, sorrendben egymást követő $(0,1)$-sor és legyen j az a legjobboldali 0-jának pozíciója. Jelöljük u^j-vel azt a vektort, melynek elemei 0-k az 1-es pozíciótól kezdve $j-1$-ig, a j pozícióban 1, és a $j+1$-es indextől n-ig -1. Vegyük észre, hogy $b = a + u^j$.

Ekkor tekintsük azt az algoritmust, amely végigmegy az összes n-hosszú $(0,1)$-sorvektoron, a lexikografikusan legkisebb e^n vektorral kezdve és meghatározza a λ lépéshosszt:

Algorithm 6: Az optimális lépéshossz kiszámítása

Inicializáció: $a := e^n, ax = x_n, ay = y_n$;
if $ay < 1$ és $f_1 = 1$ **then**
  $\lambda := \frac{ax+t-b_1}{1-ay}$;
else
  $\lambda := \infty$;
end
for $i := 2$ to 2^n **do**
  $j := n$;
  while $j > 0$ és $a_j = 1$ **do**
    $j = j - 1$
  end
  $a := a + u^j$;
  $ax := ax + u^j x$;
  $ay := ay + u^j$;
  if $ay < 1$ és $f_i = 1$ **then**
    $\lambda = \min(\lambda, \frac{ax+t-b_i}{1-ay})$
  end
end
Output: λ optimális lépéshossz

Derks és Kuipers [9] megmutatta, hogy ennek az algoritmusnak a lépésszáma $\mathcal{O}(2^n)$: mivel u^j első $(j-1)$ koordinátája 0, ezért az exponenciális hosszú ciklusban minden lépés műveletigénye az $(n-j+1)$ konstansszorosa. 2^{j-1} db azon koalíciók száma, melyben a j. helyen van a legjobboldalibb 0, tehát a módszer teljes műveletigénye

$$\mathcal{O}(\sum_{j=1}^{n} 2^{j-1}(n-j+1)) = \mathcal{O}(2^{n+1} - n - 2) = \mathcal{O}(2^n).$$

Ezt a vektorok struktúrájából adódó gyorsítást Benedek [4] algoritmusának implementálása során is alkalmaztuk.

5. A projekt leírása

Munkám során a feladatom a korábban bemutatott nukleolusz-számító algoritmusok alapján Python kód készítése és implementálása volt, majd pedig az eredmények összehasonlítása. Ezek mind korszerű megoldási módszerek, melyekből már készültek C++ nyelven írt kódok, azonban a Python környezet akadémiai és ipari területen is gyorsan és széleskörűen terjedő nyelv, amin keresztül nagyon sok potenciális felhasználót tudunk elérni. Ugyanakkor ez az új környezet számos kihívást jelent, mivel korábban ilyesmivel még nem foglalkozott senki, így új fejlesztési lehetőségek is felmerülnek.

Kezdetben `PuLP` programcsomagot és GLPK megoldót használtam, azonban később váltottam a `CyLP` programcsomagra, mely egy Pythonban megírt LP modellező program. Ennek okát később tárgyalom.

A program tesztelését kezdetben 3 játékosra kezdtem, majd idővel növeltem a játékosszámot egészen 20-ig.

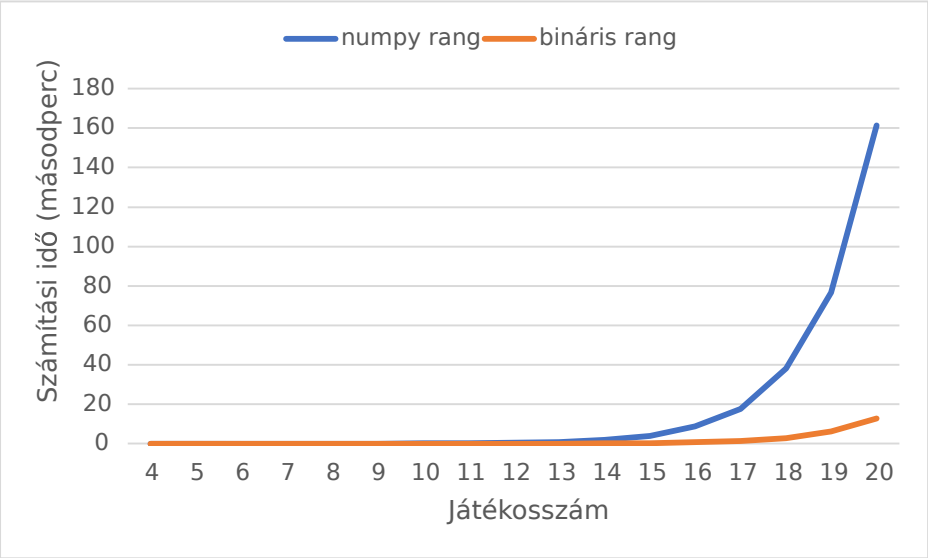
5.1. Észrevételek a munka során

Miután kisebb játékosszámra sikeresen lefutottak a kódok és megkaptuk a nukleoluszt, nagyobb játékosszám esetén egyes játékokban problémákba ütköztünk. Ezen problémák megoldása közben olyan érdekes észrevételeket tettünk, melyek úgy vélem, említésre méltóak, illetve a továbbiakban tanulhatunk belőlük.

5.1.1. Rangszámítás

Annak érdekében, hogy csökkenteni tudjuk a futási időt, meg kellett vizsgálni a kód egyes pontjaiban használt függvényeket. Korábbi tanulmányaim során azt az információt kaptam, hogy bármilyen függvényt megírhatunk magunk, azonban amire létezik beépített függvény, azzal nem érdemes plusz időt eltöltenünk, ugyanis nem lesz gyorsabb a saját kódunk az előre megírtnál. Azonban a projekt során kiderült, hogy erre is vannak kivételek.

Benedek [4] és Derks-Kuipers [9] nukleolusz-számító algoritmusaik számos pontján alkalmazunk mátrixrang-számítást, melyre kezdetben a `numpy` prog-



2. ábra. Bináris vektorok rangnövelő hatásának számítási ideje (másodpercben).

Módszer / játékosszám	4	6	8	10	12	14	16	18	20
numpy rang	0.007	0.008	0.028	0.113	0.45	1.92	8.92	38.2	161
bináris rang	0.0002	0.0004	0.002	0.009	0.03	0.15	0.69	2.93	12.8
bináris (numpy=100%)	2.2%	5.4%	7.5%	7.6%	8.2%	8%	7.8%	7.7%	7.9%

1. táblázat. Bináris vektorok rangnövelő hatásának számítási ideje (másodpercben).

ramcsomag `numpy.linalg.matrix_rank` függvényt használtuk, azonban ez nagyságrendekkel lassabbnak bizonyult a saját magunk által készített rangszámító függvénynél, főleg nagyobb játékosszámú játékok esetén.

Ez a jelenség főként annak köszönhető, hogy míg a `numpy` beépített függvénye bármilyen mátrixra működik, illetve a rang mellett számos egyéb információt is kiszámol, addig az általunk készített függvény kihasználja a mátrix speciális szerkezetét, elemeinek bináris voltát, kizárólag a rangszámításra fókuszálva.

Az 1. táblázatban látható, hogy a játékosszám növelésével milyen ütemben és mértékben növekszik a kétféle rangszámító módszer időigénye, mely a 2. ábrán szemléltetve is megfigyelhető.

5.1.2. Kerekítési problémák

A program tesztelése során az összes játék esetén megvizsgáltuk az általunk számított nukleolusz abszolút eltérését a játékhoz tartózó, valódi nukleolusszal. Erre az esetek nagyobb részében, kisebb játékok esetén 1e-14 nagyságrendű, vagy annál kisebb eltéréseket kaptunk, azonban ritkán előfordult 1e-6 nagyságrendű hiba, illetve néhány játék nagyon sokáig futott eredmény nélkül. Ezekben az esetekben arra lettünk figyelmesek, hogy a köztes lépésekben számolt LP-k megoldása során a GLPK megoldó használata esetén kerekítési hiba miatt nem tudott továbblépni az algoritmus. Ez annak volt betudható, hogy a GLPK megoldó az LP-re talált optimális megoldást 6 tizedesjegyre kerekíti, mely esetünkben egy eléggé pontatlan megoldást eredményez, ugyanis a köztes számításoknál a további tizedesjegyek is döntő szerepet játszhatnak a nukleolusz megtalálásában.

Ennek kiküszöbölésére ideiglesen elkezdtük a Gurobi megoldót alkalmazni, azonban mivel ez nem egy nyíltan hozzáférhető megoldó, ezért a továbbiakban orvosolnunk kellett a kerekítési problémát egy ingyenes megoldó esetében is.

Megoldásként a `CyLP` programcsomagra esett a választásunk, mely az ingyenes LP-megoldó szoftverek közül a leghatékonyabbnak bizonyult[1] mind pontosság, mind pedig futásidő szempontjából, ezáltal megkerülve mind a GLPK, mind pedig a `PuLP` hátrányait.

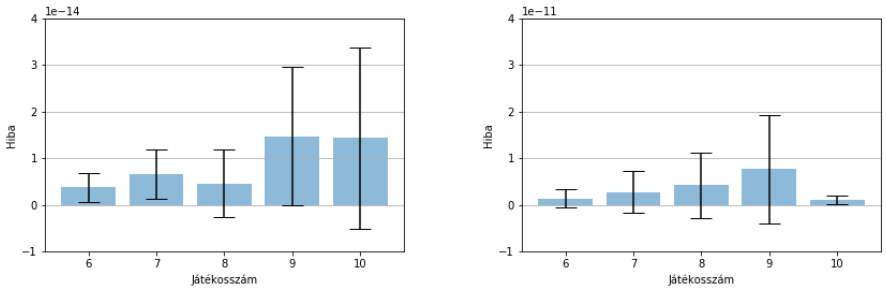
[1]Forrás: http://plato.asu.edu/bench.html

6. Eredmények

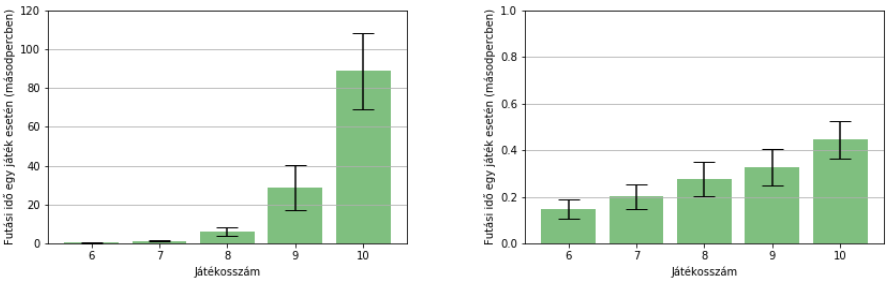
Jelenleg négy jól működő Python kóddal rendelkezünk, melyet 20 játékosig teszteltünk mindezidáig illetve egy-két esetben 25 játékosra. A próbák során az egyes játékoknál abszolút eltérést, futási időt és különböző iterációszámokat számoltunk és mentettünk külön fájlokba.

6.1. A programcsomag-váltás hatása

A 3. ábrán láthatóak a játékosszámok függvényében az abszolút eltérése a Benedek [4] algoritmusához tartozó program által számolt nukleolusznak az eredeti nukleolusztól. Az ábra bal oldalán a kezdetleges, `PuLP`-ot, GLPK-t és `numpy` rangszámító függvényt használó algoritmus, a jobb oldalán pedig a már `CyLP`-t és bináris rangszámító függvényt alkalmazó kód teljesítménye látható. Ugyanakkor a 4. ábrán pedig megfigyelhetjük, hogy a program futásideje a játékosszám függvényében hogyan csökkent, ahogy áttértünk a hatékonyabb programcsomagra és rangszámító függvényre. Megfigyelhető, hogy az említett csökkenés több, mint százszoros.



3. ábra. Abszolút eltérés Benedek algoritmusával [4] számolt és az eredeti nukleolusz között. A bal oldali grafikonon a `PuLP`-ot, GLPK-t és `numpy` rangszámító függvényt használó kód teljesítménye látható, míg a jobb oldalon a már `CyLP`-t és bináris rangszámító függvényt alkalmazóé.



4. ábra. Szükséges futási idő játékonként (másodpercben) Benedek algoritmusával [4] számolt és az eredeti nukleolusz között. A bal oldali grafikonon a `PuLP`-ot, GLPK-t és `numpy` rangszámító függvényt használó kód teljesítménye látható, míg a jobb oldalon a már `CyLP`-t és bináris rangszámító függvényt alkalmazóé.

6.2. Az egyes algoritmusok összehasonlítása

Az algoritmusok összehasonlítása során a tesztelést a kooperatív játékok négy típusán végeztük, mely típusok a következők:

I-es típusú játékok: Az I-es típusú játékok karakterisztikus függvénye a következő: $v(\{i\}) = 0$ minden $i \in N$, $v(N)$ egy random egész szám $100(n-2)$ és $100n$ között, illetve $v(S)$ egy random egész szám 1 és $100|S|$ között minden további nemüres S koalícióra.

II-es típusú játékok: A II-es típusú játékok karakterisztikus függvénye a következő: $v(\{i\}) = 0$ minden $i \in N$ és $v(S)$ egy random egész szám 1 és $50n$ között minden további nemüres S koalícióra.

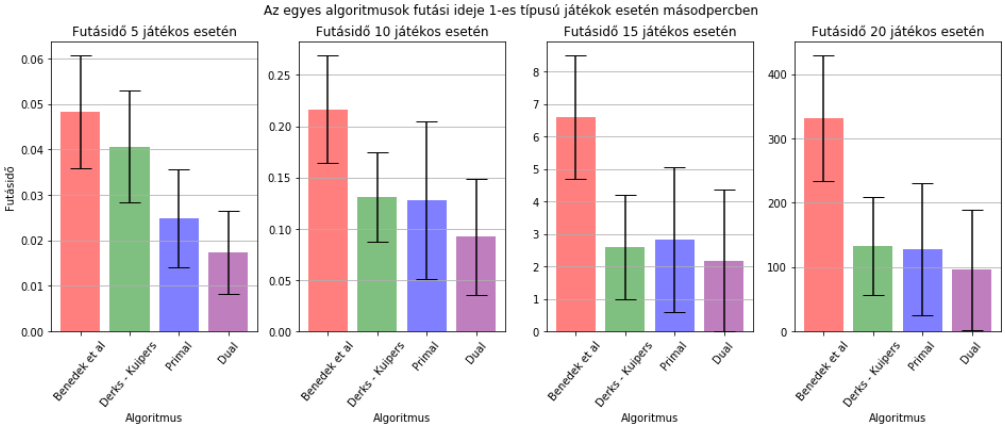
III-as típusú játékok: A III-as típusú játékok karakterisztikus függvénye a következő: $v(S) = 0$ minden $|S| < n - 2$, $v(S) = 1$ 0,9 valószínűséggel $n - 2 \leq |S| < n$ esetén és $v(N) = 1$.

IV-es típusú játékok: A IV-es típusú játékok karakterisztikus függvénye a következő: $v(\{i\}) = 0$ minden $i \in N$, és $v(S)$ egy random egész szám 1 és n között minden további nemüres koalícióra.

Mindegyik algoritmus az összes tesztelt játékon helyesen kiszámolta a nukleoluszt.

6.2.1. Összehasonlítás I-es típusú játékok esetén

Az 5. ábrán láthatóak a játékosszámok és az egyes algoritmusok függvényében a programok futási ideje I-es típusú játékok esetén. Azt vehetjük észre, hogy Benedek algoritmusához tartozó kód számít a leghosszabb ideig, azonban ez szemben áll [4] eredményeivel, illetve a meglévő C++ kódok sem ezt tükrözik. Arra a következtetésre juthatunk, hogy az új implementációt további felülvizsgálatnak kell kitennünk hatékonyság szempontjából.

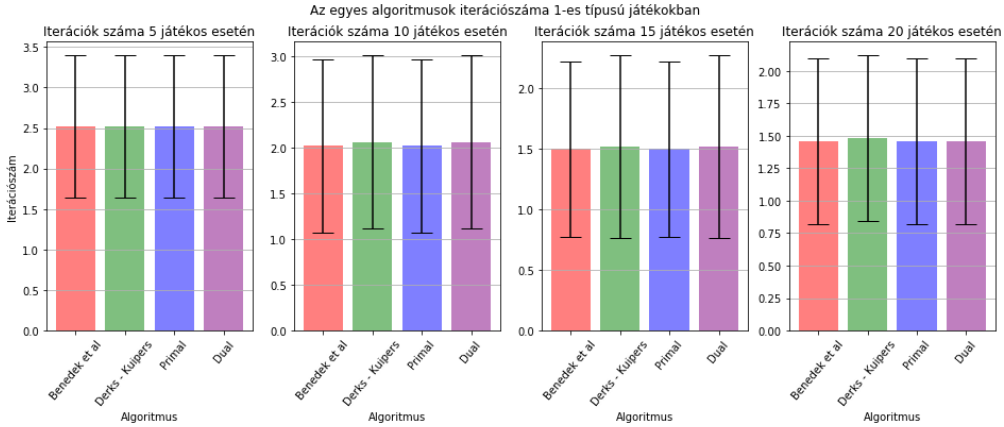


5. ábra. Szükséges futási idő I-es típusú játékok esetén játékonként (másodpercben)

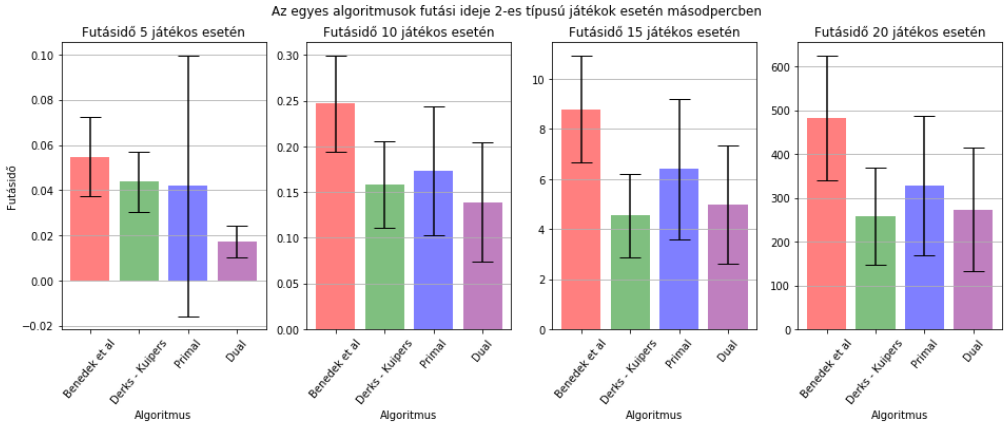
A 6. ábrán láthatóak a játékosszámok és az egyes algoritmusok függvényében a programok által végrehajtott iterációk száma I-es típusú játékok esetén. A kooperatív játékok ezen típusánál nagy különbséget nem észlelünk az iterációk számában, nagyobb eltérések a III-as, illetve IV-es típusnál tűnhetnek fel.

6.2.2. Összehasonlítás II-es típusú játékok esetén

A 7. ábrán láthatóak a játékosszámok és az egyes algoritmusok függvényében a programok futási ideje II-es típusú játékok esetén. Hasonlóan az I-es típushoz, ugyanazt a következtetést vonhatjuk le, miszerint felülvizsgálandó az első algoritmushoz tartozó kód.



6. ábra. Végrehajtott iterációk száma I-es típusú játékok esetén játékonként

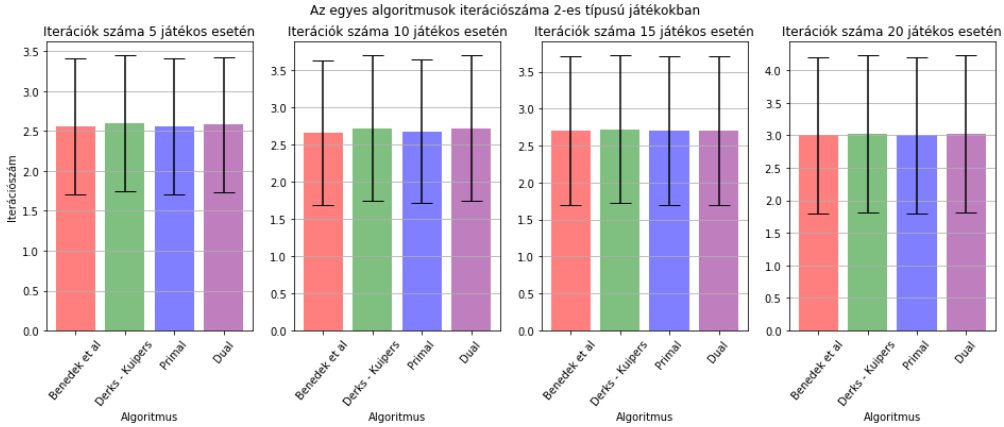


7. ábra. Szükséges futási idő II-es típusú játékok esetén játékonként (másodpercben)

A 8. ábrán láthatóak a játékosszámok és az egyes algoritmusok függvényében a programok által végrehajtott iterációk száma II-es típusú játékok esetén.

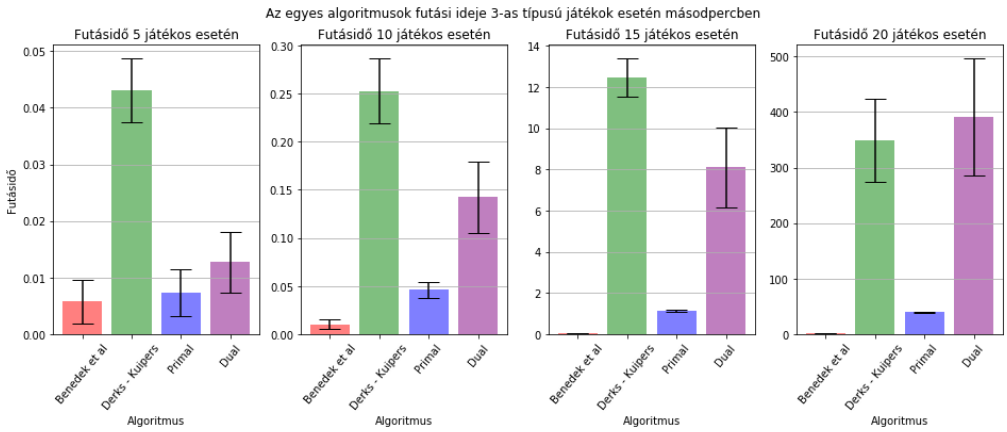
6.2.3. Összehasonlítás III-as típusú játékok esetén

A 9. ábrán láthatóak a játékosszámok és az egyes algoritmusok függvényében a programok futási ideje III-as típusú játékok esetén. Ebben az esetben, a korábbiaktól eltérően, Benedek algoritmusa teljesít a legjobban, szinte 0-hoz



8. ábra. Végrehajtott iterációk száma II-es típusú játékok esetén játékonként

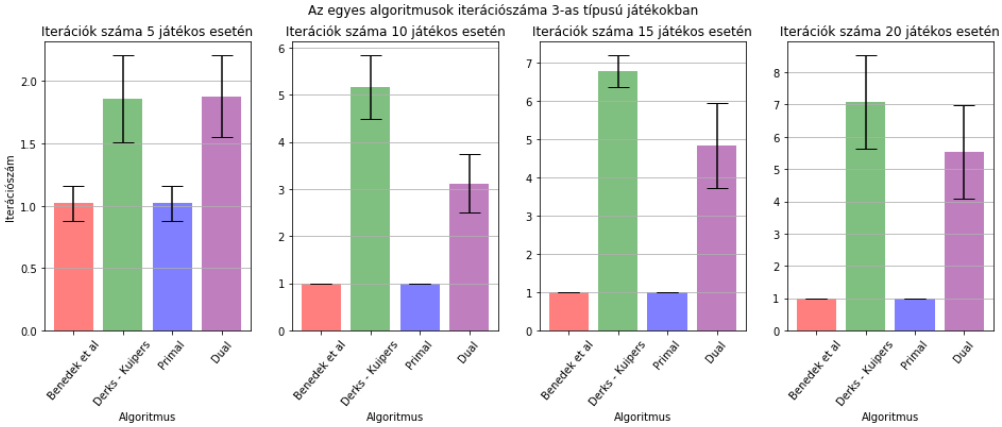
közeli futásidőben minden vizsgált játékosszám esetén. Ez annak tudható be, hogy a III-as típusú játékok a legkedvezőbb struktúrájúak ezen algoritmusnak, míg azt is megfigyelhetjük, hogy Derks és Kuipers algoritmusa számára ez a legkedvezőtlenebb struktúra.



9. ábra. Szükséges futási idő III-as típusú játékok esetén játékonként (másodpercben)

A 10. ábrán láthatóak a játékosszámok és az egyes algoritmusok függvényében a programok által végrehajtott iterációk száma III-as típusú játékok esetén.

Ebben az esetben már az iterációk száma is jobban elkülönül az egyes algoritmusok esetében.

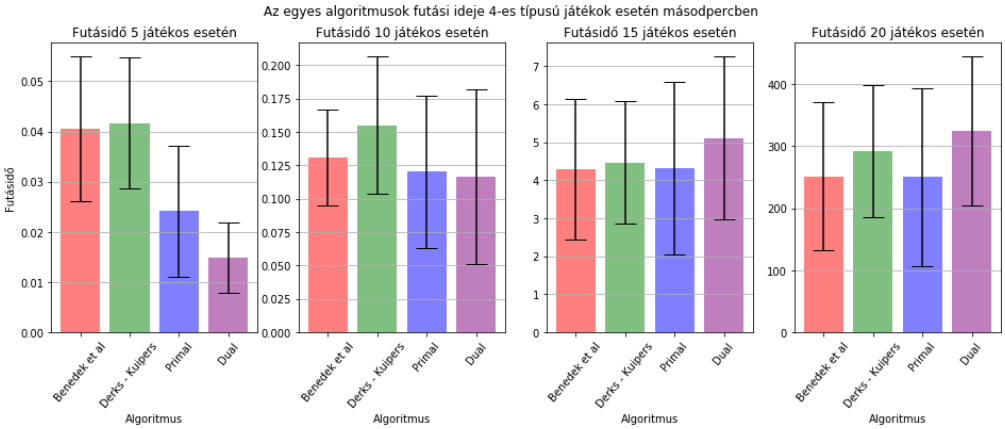


10. ábra. Végrehajtott iterációk száma III-as típusú játékok esetén játékonként

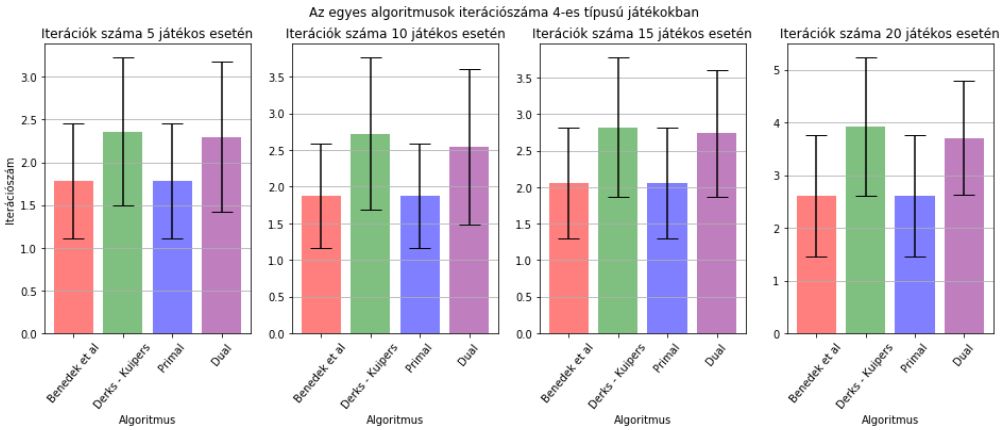
6.2.4. Összehasonlítás IV-es típusú játékok esetén

A 11. ábrán láthatóak a játékosszámok és az egyes algoritmusok függvényében a programok futási ideje IV-es típusú játékok esetén.

A 12. ábrán láthatóak a játékosszámok és az egyes algoritmusok függvényében a programok által végrehajtott iterációk száma IV-es típusú játékok esetén.



11. ábra. Szükséges futási idő IV-es típusú játékok esetén játékonként (másodpercben)



12. ábra. Végrehajtott iterációk száma IV-es típusú játékok esetén játékonként

7. További célok

7.1. Rövidtávú célok

A továbbiakban az első feladatunk Benedek algoritmusához tartozó kód felülvizsgálata a futási idő csökkenésének érdekében.

Ugyanakkor szeretnénk még nagyobb játékosszámú játékokra is alkalmazni a megírt programokat, így a futásidő csökkentése az összes algoritmushoz tartozó kódnál kulcsfontosságú. Jelenleg tesztelés alatt áll a 25 játékosszámú játékokon való teljesítmény, azonban az eredmények még további gyorsítási kísérletekre adnak okot.

7.2. Hosszútávú célok

Hosszabb távon a lépéshossz-számítással szeretnénk nagyobb mértékben foglalkozni, a meghatározására újabb, illetve más módszereket kipróbálni. Ilyen módszer lenne például, ha előre meghatározott lépéshosszt alkalmaznánk, majd pedig ezt csökkentve közelednénk a nukleoluszhoz. Szeretnénk párhuzamot vonni a továbbiakban a több területen is ismeretes legmeredekebb ereszkedés módszerével, melyben sok lehetőség rejtőzhet a további fejlesztések szempontjából.

8. Kitekintés

A nukleolusz-számítás nem csak szigorúan kutatási környezetben alkalmazható, ugyanis az elmúlt években elkezdett betörni a sokak által kedvelt videójátékok világába is.

Az Avian[2] videójáték, egy érdekes ismeretterjesztő és oktatási játékszoftver, ahol a felhasználónak egy falu vízellátását kell megoldania, komoly játékelméleti alapokon nyugszik, azon belül pedig főként nukleolusz-számítás az alapja.

[2]ld. https://playavian.com/we-are-nucleolus

Ábrák jegyzéke

Táblázatok jegyzéke

Hivatkozások

[1] Shapley, Lloyd S. *Markets as cooperative games.* (1955).

[2] Gillies, Donald Bruce. *Solutions to general non-zero-sum games.* Princeton University Press. (1959).

[3] Schmeidler, David. The nucleolus of a characteristic function game. *SIAM Journal of Applied Mathematics.* 17(6): 1163-1170 (1969).

[4] Benedek, M., Fliege, J., Nguyen, TD. Finding and verifying the nucleolus of cooperative games. *Math. Program.* (2020).

[5] Kohlberg, E. On the nucleolus of a characteristic function game. *SIAM Journal of Applied Mathematics.* 1(20): 62-66 (1971).

[6] Solymosi, T., Sziklai, B. Characterization sets for the nucleolus in balanced games. *Oper. Res. Lett.* 44(4): 520-524 (2016).

[7] Solymosi, T. *On computing the nucleolus of cooperative games. Ph.D. Thesis.* University of Illinois. (1993).

[8] Bland, RG. New finite pivoting rules for the simplex method. *Mathematics of Operations Research.* 2: 103-107 (1977).

[9] Derks, J., Kuipers, J. *Implementing the simplex method for computing the prenucleolus of transferable utility games.* (1997).