KAKUKK-ALGORITMUS TANULÁSI STRATÉGIA A MEGBÍZÓ-ÜGYNÖK MODELLBEN[1]

KERÉNYI PÉTER
Budapesti Corvinus Egyetem

A tanulmányban a Kakukk-algoritmust (Cuckoo Search) követő tanulási stratégia teljesítményét vizsgáljuk a megbízó-ügynök modellben. Felállítunk egy standard megbízó-ügynök keretrendszert, ahol az előállított outputból történő részesedéssel (lineáris szerződés) az ügynök ösztönözhető, és elkerülhető az erkölcsi kockázat. A modellben a megbízó nem ismeri sem az ügynök hasznossági függvényét, sem pedig a sztochasztikus környezet tulajdonságait, de a szerződéskötésből és az előállított output megfigyeléséből álló folyamatot iterálva, folyamatosan javítva a szerződés paramétereit kitanulja és optimalizálja a saját célfüggvényét. Monte-Carlo szimulációs módszertant alkalmazva arra a következtetésre jutunk, hogy a megbízó-ügynök problémára jellemző, és a tanulást nehezítő, nem folytonos hasznossági függvény és sztochasztikus környezet ellenére a Kakukk-algoritmus a korábbi tanulási algoritmusoknál pontosabban képes meghatározni a megbízó hasznossági függvényének maximumát.

Kulcsszavak: megbízó-ügynök probléma, lineáris szerződés, ismételt játék, Kakukk-algoritmus. *JEL kódok:* C61, C63, C73, D86

1 Bevezetés

A megbízó-ügynök probléma során az eszköz (vállalat, know-how, információ stb.) tulajdonosa (megbízó) megbízza az ügynököt, hogy az ő nevében eljárva és döntéseket hozva az eszköz használatával minél több outputot állítson elő. A két szereplő azonban különböző szempontok alapján hozza meg a döntéseket, amik érdekkonfliktushoz vezetnek. Fontos kérdés, hogy a megbízó hogyan tudja ösztönözni az ügynököt, hogy a számára optimális döntéseket hozza meg. Egy lehetséges és elterjedt ösztönzési mechanizmus a megbízó és az ügynök közötti lineáris szerződés, ahol a szereplők kifizetése két részből áll, egy fix, illetve egy változó részből, ahol a változó rész az output lineáris függvénye. Stiglitz [1974] a részesaratásnál használt lineáris szerződést vizsgálja ösztönzési szempontból. Ennél a szerződésnél a földbérlő (ügynök) egy fix bérleti díjat fizet a földtulajdonosnak (megbízó) a föld használatáért, majd a földön megtermelt javakból egy előre rögzített arányban részesedik. Holmström és Milgrom [1991] a menedzserszerződések ösztönzési hatásait elemzi.

[1]A szerző a Budapesti Corvinus Egyetem Gazdaságinformatika Doktori Iskola PhD hallgatója, a Pallas Athéné Domus Educationis Alapítvány ösztöndíjasa. E-mail: peter.kerenyi@uni-corvinus.hu. Beérkezett: 2017. október 27.

Ezekben a szerződésekben a vállalat tulajdonosa (megbízó) a menedzsernek (ügynök) egy fix fizetést ad, majd a vállalat által megtermelt profit egy előre meghatározott részét bónuszként fizeti ki. Hazai kutatók közül Csóka et al. [2015] a potenciálisan nem fizető vevővel rendelkező vállalkozások hitelezése során megjelenő morális kockázatot vizsgálja szerződéselméleti és ösztönzési szempontból, Berlinger et al. [2017] az állami támogatás és a morális kockázat viszonyát elemzi, Kóczy és Kiss [2017] pedig a Nobel-emlékdíjas Oliver Hart és Bengt Holmström szerződéselméleti munkásságát mutatja be.

A megbízó-ügynök keretrendszerben a megbízó feladata egy olyan lineáris szerződés felajánlása az ügynök számára, amivel elérhető, hogy az ügynök olyan minőségű munkát végezzen, amivel maximalizálja nem csak saját, hanem a megbízó hasznossági függvényét is. Ha a megbízó ismeri a saját és az ügynöke céljait, azaz a hasznossági függvényeket, valamint a bizonytalan környezet jellemzőit, akkor analitikus úton meghatározhatók a szerződés optimális paraméterei.

A valóságban azonban a megbízó nem ismeri sem az ügynök hasznossági függvényét, sem pedig a zajos környezet pontos jellemzőit. Jelen tanulmányban arra keressük a választ, hogy hogyan és milyen módon érheti el mégis a megbízó, hogy kitalálja, kitanulja az optimális paramétereket. Abból a feltételezésből indulunk ki, hogy a megbízónak lehetősége van a szerződéskötés után megfigyelni az ügynök tevékenységének outputját, ami kapcsolatban van az ügynök erőfeszítésével, majd egy új, finomított szerződést ajánlani az ügynöknek.

Axelrod [1987] a fogoly-dilemmát vizsgálja ilyen iteratív játék segítségével. Szabó és Tőke [1998] evolúciós modellezést használ az iteratív fogoly-dilemma vizsgálatára négyzetes hálókban, Szabó et al. [2005] pedig további kétdimenziós hálókat használ ezen kérdés elemzésére. Roth és Erev [1995] megerősítéses tanulást (reinforcement learning), Rose és Willemain [1996] genetikus algoritmust, Camerer és Hua Ho [1999] pedig tapasztalattal súlyozott vonzerő tanulást (experience weight attraction learning) használ a megbízó-ügynök probléma megoldására. Arifovic és Karaivanov [2010] szimulációs módszertant használva különböző tanulási, evolúciós algoritmusokat hasonlít össze a megbízó-ügynök probléma esetén, és elemezi ezen algoritmusok tulajdonságait, majd arra a következtetésre jut, hogy a társadalmi-evolúciós tanuló algoritmus (tanulás másoktól, learning from others) gyorsabban konvergál az optimális megoldás közelébe, mint az egyéni tanulást végző (cselekvés általi tanulás, learning by doing) algoritmusok.

A különböző tanuló stratégiák teljesítményét, és azt, hogy mennyire alkalmazkodnak egy adott problémához, nagyban befolyásolják a célfüggvény tulajdonságai. A tanuló stratégiáknak problémát jelenthet, ha a célfüggvény nem folytonos, mert ilyenkor a nagy szakadás megnehezíti a paraméterek finomítását. További és az előzőnél nagyobb problémát jelenthet, ha a célfüggvénynek lokális szélsőértékei vannak, mert ilyenkor az iteratív tanuló eljárások könnyedén egy ilyen lokális szélsőértékbe konvergálhatnak. Az optimum megtalálásában további probléma lehet, hogy a zajos környezetben nehezen megfigyelhető a célfüggvény elméleti értéke. Megfelelő iteratív eljárásokkal

azonban a valóságra sok esetben jobban hasonlító, a megbízó számára ismeretlen hasznossági függvények és sztochasztikus környezet is vizsgálhatók.

A tanulmányban Arifovic és Karaivanov [2010] nyomán a megbízó-ügynök mechanizmustervezési feladatot a Yang és Deb [2009] által kifejlesztett, a parazita kakukk madár viselkedését modellező Kakukk-algoritmussal (Cuckoo Search) oldjuk meg. A Kakukk-algoritmust sikeresen alkalmazzák más, feltételes optimalizációs problémáknál, mint a projekt ütemezés és hozzárendelés Tein és Ramli [2010]), a szerkezet optimalizációs problémák (Gandomi et al. [2013]) vagy a hátizsák probléma (Layeb [2011]). Kutatások alapján a Kakukk-algoritmus más optimalizációs-tanulási algoritmusoknál jobb teljesítményt nyújt (Yang et al. [2012], Yang és Deb [2014]). Célunk, hogy a megbízó-ügynök probléma esetén összehasonlítsuk ezen algoritmikus stratégia teljesítményét a korábban megfogalmazott stratégiákkal szemben, és ezáltal egy, akár a gyakorlatban is használható módszert mutassunk be az ügynökök ösztönzésére.

A vizsgálatot Monte-Carlo szimulációs eszközökkel végezzük. A 2. fejezetben bemutatott, Arifovic és Karaivanov [2010] által is használt keretrendszer adja meg a szereplők hasznossági függvényeit és a feltételes szélsőérték feladatot. A 3. fejezetben a Kakukk-algoritmus által kijelölt stratégiát implementáljuk a megbízóra, majd az így meghatározott szabályok és a 4. fejezetben bemutatott paraméterek alapján szimuláljuk a megbízó és az ügynök közötti ismételt játékot. A 5. fejezetben a Kakukk-algoritmus és az Arifovic és Karaivanov [2010] tanulmányban a legjobb teljesítményt nyújtó Social Evolutionary Learning (SEL) stratégiát hasonlítjuk össze különböző szempontok alapján. Arra a következtetésre jutunk, hogy a Kakukk-algoritmus a SEL-hez hasonlóan jól alkalmazkodik a megbízó-ügynök problémához és alkalmas az optimum meghatározására, viszont a SEL-hez képest pontosabb, és gyorsabban konvergál az optimumhoz. A 6. fejezetben a Kakukk-algoritmus érzékenységét vizsgáljuk mind a modell, mind pedig az algoritmus paramétereinek függvényében, és azt tapasztaljuk, hogy ezen paraméterekre az algoritmus robusztus.

Összességében elmondható, hogy a Kakukk-algoritmus jól használható olyan megbízó-ügynök problémák esetén, amikor lehetőség van az előállított output gyakori megfigyelésére és a szerződéses paraméterek ismételt megváltoztatására. Ilyen gyakorlati alkalmazás lehet a időben változó paraméterű lineáris szerződéssel történő optimális ösztönzésre az online tartalmakat, mint például a videómegosztó portálok (Youtube, Twitch stb.), a kérdés-válasz fórumok (Yahoo Answers, StackOverflow stb.) vagy az értékelő oldalak (Epinion, TripAdvisor stb.) tartalmait előállító felhasználók ösztönzése a jobb (több, gyorsabb) tartalmak előállítására és ezáltal a bevételek növelésére. Másik ilyen alkalmazás lehet a taxisofőrök ösztönzése (Uber stb.). Ebben az esetben a taxitársaságnak a sofőrök az online rendszerhez való csatlakozásért napi díjat fizetnek, majd adott arányban részesednek az online platform által szervezett fuvarok díjaiból.

2 Modell keret

A dolgozatban az algoritmusok összehasonlíthatósága miatt Arifovic és Karaivanov [2010] által is használt standard lineáris szerződés modellt használjuk. A célfüggvények ismeretében analitikus módon kiszámíthatók a szerződés optimális paraméterei, amit referenciaként használunk az algoritmus teljesítményének mérésére.

A termelés outputja $y(z) := z + \varepsilon$, ahol $z \in \mathbb{R}$ az ügynök erőfeszítésének mértéke, $\varepsilon \sim N(0, \sigma^2)$ pedig egy véletlen zaj. Az ügynök erőfeszítésének mértékét a megbízó nem képes megfigyelni, pusztán az output alapján következtethet rá. A megbízó kifizetése ekkor $\pi(y) := (1-s)\cdot y + f$, ahol s a megbízó részesedése az outputból ($s \in [0,1]$, változó rész), $f \in \mathbb{R}$ pedig a fix rész. A megbízó költsége egyenlő az ügynök kifizetésével: $c\big(y(z)\big) := s \cdot y(z) - f$. A megbízó kockázatsemleges, ezért ő várható értékben optimalizál. Az ügynök hasznossági függvénye legyen konstans abszolút kockázatelutasítási (CARA) függvény négyzetes erőfeszítés költséggel:

$$u(z) := -e^{-\gamma\cdot(c(y(z))-\frac{1}{2}\cdot z^2)}\,, \tag{1}$$

ahol $\gamma \in \mathbb{R}^+$ a kockázatelutasítási paraméter. Ekkor az ügynök várható hasznossága:

$$E\big(u(z)\big) = -e^{-\gamma\cdot(s\cdot z - f - \frac{1}{2}\cdot z^2 - \frac{\gamma}{2}\cdot s^2\cdot\sigma^2)}\,. \tag{2}$$

A megbízó várható kifizetése:

$$E\big(\pi(y)\big) = E\big((1-s)\cdot y(z)+f\big) = E\big((1-s)\cdot(z+\varepsilon)+f\big) = (1-s)\cdot z + f\,. \tag{3}$$

A megbízó feladatát a következőképpen írhatjuk fel:

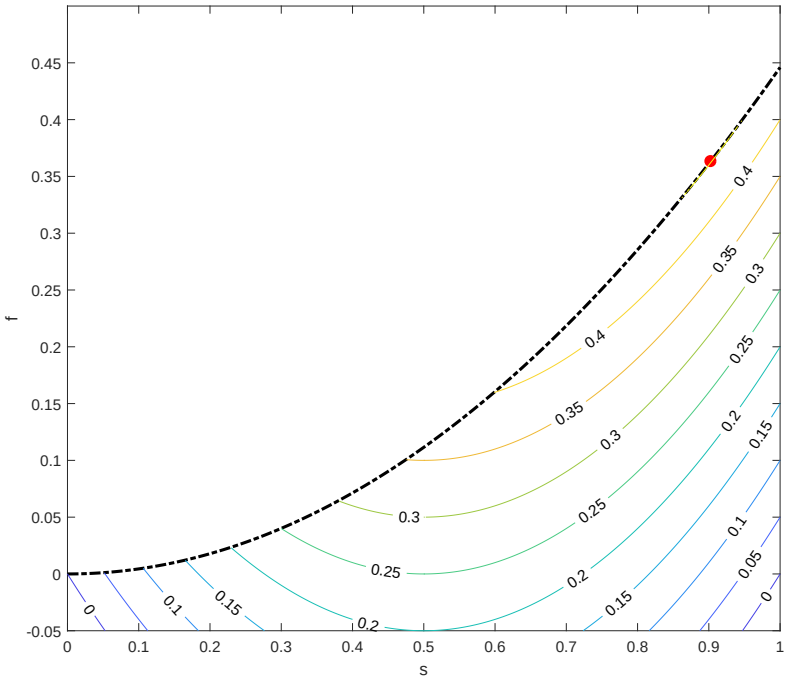
$$\max_{s,f} E\big(\pi(y)\big) \tag{4}$$

s.t.

$$z = \arg\max_{\hat{z}} E\big(u(\hat{z})\big) \tag{ICC}$$

$$E\big(u(z)\big) \geq u(\bar{w}) \tag{PC}$$

A megbízó feladata tehát a szokásos megbízó-ügynök mechanizmustervezési feladat megoldása (pl. Bolton és Dewatripont [2005] [pp. 137-139]). A megbízó maximalizálja a várható profitját (kockázatsemleges) azon korlátok mellett, hogy az ügynök a saját hasznossági függvénye alapján a saját hasznát maximalizálja és ennek függvényében választja meg az erőfeszítés szintjét (ICC, incentive compatibility constraint, ösztönzési korlát), illetve, hogy az ügynök várható hasznossága meghaladja a külső opciójának a hasznosságát (példánkban ez a külső opció kifizetése egyenlő 0-val), azaz az ügynöknek egyáltalán megérje részt venni a játékban, elfogadni a szerződést (PC, participation constraint, részvételi korlát). A feltételes szélsőérték feladatot szemlélteti az *1. ábra.*



1. ábra. A megbízó várható kifizetésének közömbösségi görbéi az $S \times F$ stratégiatérben $\sigma = 0{,}6$ és $\gamma = 0{,}3$ paraméterek mellett. A PC korlát fekete szaggatott vonallal, az elméleti optimum piros ponttal jelölve.

A feladatot visszafelé oldjuk meg (részletes levezetés ld. A. melléklet), azaz adott (s, f) párhoz kiszámítjuk az ügynök optimális z^* erőfeszítés szintjét (megoldjuk az ICC korlátot), majd ezután optimalizáljuk a megbízó célfüggvényét. Ez alapján $z^* = s$.

A feladat végső megoldásaként adódik az optimális szerződés két paramétere:

$$s^* = \frac{1}{1 + \gamma \cdot \sigma^2} \tag{5}$$

$$f^* = \frac{1 - \gamma \cdot \sigma^2}{2 \cdot (1 + \gamma \cdot \sigma^2)^2} \tag{6}$$

Az output szórásából (σ) és az ügynök kockázatelutasításának mértékéből (γ) ismert hasznossági függvények mellett az optimális szerződéses paraméterek kiszámíthatók, és a későbbi fejezetekben az így kapott elméleti optimumokat használjuk referenciaként az algoritmusaink teljesítményének mérésére.

3 Kakukk-algoritmus implementálása

A 2009-ben Xin-She Yang és Suash Debt által kifejlesztett Kakukk-algoritmus a parazita kakukkmadár viselkedését alapul véve próbálja megtalálni a célfüggvény optimális megoldását. A Kakukk-algoritmus jól használható olyan

célfüggvények globális szélsőértékének meghatározására, amely célfüggvények nem folytonosak. A megbízó-ügynök probléma iteratív megoldása során ezt az eljárást implementáljuk.

Az iteratív játék szimulációjában az algoritmus minden fordulóban $m := (s; f), s \in S, f \in F$ paraméterpárokat (ezeket nevezzük a későbbiekben stratégiáknak) ajánl az azonos hasznossági függvénnyel rendelkező ügynököknek. A stratégiákat minden esetben egy G stratégiatérből választja, amely stratégiatérből minden $m \in G$ stratégia rendelkezik egy rátermettségi értékkel. A stratégiák rátermettségi értéke kezdetben 0, majd a fordulók során ezeket az értékeket folyamatosan finomítja az algoritmus. A $\varphi : G \to I\!R$ függvény segítségével az algoritmus megkapja egy adott stratégia aktuális rátermettségi értékét. Egy m stratégiát egy fordulóban J darab ügynöknek ajánl, akik meghatározzák a stratégiához tartozó z^* optimális erőfeszítés szintet, előállítanak egy $y_i = s + \varepsilon_j$ outputot. Ezen outputokat átlagolva az algoritmus kiszámítja a stratégiához tartozó $\hat{\pi} = f + (1 - s) \cdot (s + \frac{1}{J} \sum_{j=1}^{J} \varepsilon_j)$ átlagos megbízó kifizetést. Végül az m stratégiához tartozó rátermettségi értéket ezzel a $\hat{\pi}$ értékkel pontosítja, mégpedig úgy, hogy megnézi, hogy a korábbi fordulókban, amikor ugyanezen stratégiát ajánlotta, milyen $\hat{\pi}$ értékeket tapasztalt, majd ezeket az értékeket átlagolja. Így a stratégia rátermettsége a várható kifizetés torzítatlan becslése lesz.

Az algoritmus minden $t = 1, \ldots, T$ fordulóban egy N elemű $b \in G^N$ stratégiákat tartalmazó vektorral dolgozik. Első lépésként az előző fordulóban meghatározott $b_{i,t-1}$ stratégiákból Lévy-repüléssel $g_{i,t}$ stratégiákat készít. A Lévy-repülés egy olyan véletlen bolyongás, ahol a növekmények vastag szélű eloszlást követnek, így nagy valószínűséggel a $g_{i,t}$ stratégiák a $b_{i,t-1}$ stratégiák közelében maradnak, de normális eloszlásnál nagyobb valószínűséggel távolodnak el az eredeti stratégiától. A Lévy-repülés szimulálására Mantegna [1994] algoritmusát használjuk, ahol két paraméterrel (α és β) definiálható a bolyongás.

$$g_{i,t} := b_{i,t-1} + l_{i,t} \cdot N(0,1)\,, \tag{7}$$

ahol

$$l_{i,t} := \alpha \cdot \left(\frac{u}{|v|}\right)^{\frac{1}{\beta}} \cdot (b_{i,t-1} - \hat{b}_{t-1}) \cdot z \tag{8}$$

$$\hat{b}_{t-1} := \max_i b_{i,t-1} \tag{9}$$

$$u \sim N(0,1) \cdot \left(\frac{\Gamma(1+\beta) \cdot \sin(\beta\pi/2)}{\Gamma\left(\frac{1+\beta}{2}\right) \cdot \beta \cdot 2^{(\beta-1)/2}}\right)^{\frac{1}{\beta}} \tag{10}$$

$$v \sim N(0,1) \tag{11}$$

$$z \sim N(0,1) \tag{12}$$

Egy $g_{i,t}$ stratégia legenerálását tekintjük egy műveletnek. Egy fordulóban a stratégiagenerálás $O(N)$ futási idejű, azaz a stratégiák számának lineáris függvénye.

A $g_{i,t}$ stratégiákat az algoritmus felajánlja az ügynököknek, és kiszámítja, illetve pontosítja a hozzájuk tartozó rátermettségi értékeket. Ezeket a $g_{i,t}$ stratégiákat rátermettségük alapján összehasonlítja az előző fordulóból származó $b_{i,t-1}$ stratégiákkal, és a rátermettebbeket rakja a $h_{i,t}$ gazdastratégiáknak nevezett stratégiákba:

$$h_{i,t} := \begin{cases} g_{i,t}, & \text{ha } \varphi(b_{i,t-1}) < \varphi(g_{i,t}) \\ b_{i,t-1}, & \text{ha } \varphi(b_{i,t-1}) \geq \varphi(g_{i,t}) \,. \end{cases} \tag{13}$$

Egy műveletnek tekintjük egy stratégia egy ügynöknek történő felajánlását és a rátermettségi érték frissítését a megfigyelt outputtal. Ennek a fázisnak a futási ideje $O(N \cdot J)$, azaz a stratégia- és az ügynökszám szorzatának lineáris függvénye.

A forduló második lépésében ezekhez a $h_{i,t}$ gazdastratégiákhoz $c_{i,t}$ kakukkstratégiákat generál az algoritmus. A $c_{i,t}$ kakukkstratégiák p valószínűséggel megegyeznek a $h_{i,t}$ gazdastratégiákkal, ezzel modellezve azt a természetben előforduló eseményt, amikor a gazdamadár felfedezi a fészkébe került kakukktojást és még a kakukkfióka kikelése előtt kilöki a fészkéből. $1-p$ valószínűséggel viszont kikel a kakukkfióka és versenyez a gazdamadár fiókájával a túlélésért. Ezeket a kakukkstratégiákat a gazdastratégiákból egyenletes eloszlású torzított véletlen bolyongással (biased random walk) állítja elő az algoritmus.

$$c_{i,t} := \begin{cases} h_{i,t}, & p \\ h_{i,t} + r_{i,t}, & 1-p, \end{cases} \tag{14}$$

ahol

$$r_{i,t} := u \cdot (h_{j,t} - h_{k,t}) \,, \tag{15}$$

$$u \sim U[0,1] \,, \tag{16}$$

j és k egyenletes eloszlású véletlen egész az $[1; N]$ intervallumból. Ezt a folyamatot (stratégiagenerálás) ismét egy műveletnek tekintve, ennek a fázisnak a futási ideje $O(N)$, azaz a stratégiák számának lineáris függvénye.

Az algoritmus ezeket a $c_{i,t}$ kakukkstratégiákat ajánlja az ügynököknek és az outputok megfigyelése után frissíti a stratégiák rátermettségi értékeit. Végül ezeket a $h_{i,t}$ gazdastratégiákat és $c_{i,t}$ kakukkstratégiákat páronként versenyeztetve meghatározza az adott forduló győztes $b_{i,t}$ stratégiáit:

$$b_{i,t} := \begin{cases} h_{i,t}, & \varphi(c_{i,t}) < \varphi(h_{i,t}) \\ c_{i,t}, & \varphi(c_{i,t}) \geq \varphi(h_{i,t}) \end{cases} \tag{17}$$

Ismét egy műveltnek tekintjük egy stratégia egy ügynöknek történő felajánlását és a rátermettségi érték frissítését a megfigyelt outputtal. Ennek a fázisnak a futási ideje $O(N \cdot J)$, azaz a stratégia- és az ügynökszám szorzatának lineáris függvénye.

A Kakukk-algoritmus tanulási stratégiát követve egy forduló futási ideje $2 \cdot O(N) + 2 \cdot O(N \cdot J) = O(N \cdot J)$, azaz a stratégia- és az ügynökszám szorzatának lineáris függvénye.

A szimuláció egy realizációja során a fenti lépésekből álló folyamatot iteráljuk a T időkorlátig.

4 Szimulációs paraméterek

Az algoritmus teljesítményének tesztelését a 2. fejezetben bemutatott (4) példán végezzük. Vizsgálatunk során először egy referencia szimulációt végzünk, majd komparatív statika segítségével megvizsgáljuk, hogy a különböző paraméterek milyen hatással vannak a teljesítményre. A referencia szimuláció eredményeit összevetjük a SEL algoritmus eredményeivel, ezért kezdetben az Arifovic és Karaivanov [2010] tanulmányban használt és ott a legjobb eredményt elérő paraméterbeállításokat használjuk.

A modellparamétereket a szimulációban minden futás során változtatjuk, hogy kiszűrjük ezek hatását az eredményekből. Az ügynök kockázatelutasítási paraméterét (γ) 0, 2 és 3 értékek között egyenletes lépésközzel választjuk, az outputra rakódó zaj szórását (σ) pedig a 0 (ekkor tökéletesen megfigyelhető az ügynök erőfeszítése) és 0,6 közötti intervallumból választjuk egyenlő lépésközzel.

A 6. fejezetben megvizsgáljuk és elemezzük, hogy az algoritmus mennyire robusztus ezen modellparaméterekre. A szimuláció során törekszünk az Arifovic és Karaivanov [2010] tanulmánnyal vett összehasonlíthatóságra és a reprezentativitásra, ezért az egyes beállításokat 70 különböző véletlen mag mellett futtatjuk le, azaz így összesen $15 \times 7 \times 70 = 7350$ futtatást végzünk. Egy-egy futás során $T = 1200$ fordulót szimulálunk. A (4) feladat diszkretizálásaként a G stratégiateret egy $0,01$ finomságú $[0; 1] \times [-0{,}05; 0{,}5]$ ráccsal közelítjük. A G stratégiatér dimenziónak széleit úgy határozzuk meg, hogy minden modellbeállítás mellett az $(s^*; f^*)$ analitikus optimum a rács szélein belül legyen.

Az algoritmus paraméterei közül a stratégiák számát (N) és az ügynökök számát (J) kezdetben 30-nak, illetve 10-nek választjuk (az Arifovic és Karaivanov [2010] tanulmányhoz hasonlóan), de a 6. fejezetben ezen paraméterekre vett érzékenységet is megvizsgáljuk. A SEL algoritmustól különböző, csak a Kakukk-algoritmushoz szükséges paramétereknél a Lévy-repülés α skála paraméterét a G stratégiatér finomságával megegyező méretűre, azaz $0,01$-re állítjuk, a valószínűség eloszlást meghatározó β paramétert pedig kezdetben 1-nek választjuk, ami pont Cauchy-eloszlást eredményez. A kakukktojás felfedezésének valószínűségét szimbolizáló p paramétert kezdetben $0,25$-re állítjuk. A 6. fejezetben a β és a p paraméterre vett érzékenységet is vizsgáljuk.

A referencia szimulációban használt futtatási paramétereket az *1. táblázat* foglalja össze.[2]

[2] A szimulációkat a MATLAB R2017a programmal futtatjuk. A kódok elérhetőek a MATLAB közösség fájlcserélő oldalán: https://www.mathworks.com/matlabcentral/fileexchange/66251-cuckoo-search-learning-strategy-in-the-principal-agent-model

Paraméter	Érték
Modell	
Kockázatelutasítás paraméter, γ	15 egyenletes távolságú pont a [0,2; 3]-ból
Output szórása, σ	7 egyenletes távolságú pont a [0; 0,6]-ból
Szimuláció	
Véletlen magok száma, *seed*	70 egyenletes eloszlású véletlen a [1; 10000]-ból
Stratégiatér elemszáma, $\lvert G\rvert$	$101 \cdot 56 = 5656$
Rácsfinomság, d	0,1
Fordulók száma, T	1200
Kakukk	
Stratégiák száma, N	30
Ügynökök száma stratégiánkként, J	10
Lévy-repülés skála paramétere, α	0,01
Lévy-repülés eloszlás paramétere, β	1
Kakukktojás felfedezés valószínűsége, p	0,25

1. táblázat. A referencia szimulációban használt paraméterek és azok értékei

5 Elemzés

Elemzésünk során a 3. fejezetben bemutatott Kakukk-algoritmus, illetve az Arifovic és Karaivanov [2010] tanulmányban használt SEL algoritmus (részletes implementációt ld. B. melléklet) által kijelölt stratégiák alapján lefuttatott véletlen szimulációkat hasonlítjuk össze. A két implementáció futási időben nem különbözik egymástól, mindkét esetben egy forduló futási ideje $O(N \cdot J)$, azaz a stratégia- és az ügynökszám szorzatának lineáris függvénye (ld. 3. fejezet és B. Melléklet). Megvizsgáljuk, hogy a szimulációk mekkora hányadában használják az optimális stratégiától adott távolságon belüli stratégiákat, illetve azt, hogy az időben ezek az arányok hogyan változnak. További elemzési szempont az algoritmusok optimális értékhez való konvergenciája és a konvergencia sebessége.

Bár a feltételes szélsőérték feladatnak – (4) feladat – csak egy globális maximuma van és nincsenek lokális maximumai, az optimum megtalálása tanuló algoritmus segítségével mégsem egyszerű feladat. A specifikált feladatban a célfüggvény nem folytonos, a PC korlát mentén szakad és utána 0 értéket vesz fel. Ráadásul a maximum ezen a PC görbén helyezkedik el, ami azt jelenti, hogy közvetlen környezetében már hirtelen 0-ra esik a célfüggvény értéke és így az iterációs tanuló algoritmus nehezen tudja kitapasztalni ezt a maximumot (ld. 1. ábra). További problémát jelenthet az outputra rakódó zaj, ami nehezíti a stratégiák pontos rátermettségének meghatározását.

Az algoritmus teljesítményének teszteléséhez először Arifovic és Karaivanov [2010] által is használt két mértéket definiálunk. Az első egy stratégia távolsága az optimális stratégiától, amit az euklideszi távolsággal mérünk:

$$d_e(m) := |m^* - m| = |(s^*, f^*) - (s, f)| = \sqrt{(s^* - s)^2 + (f^* - f)^2}\,. \quad (18)$$

A teljesítmény mérésére használt második mértékünk az optimális stratégia várható kifizetésének és az adott stratégia adott fordulóban tapasztalt

rátermettségének abszolút eltérése százalékos formában:

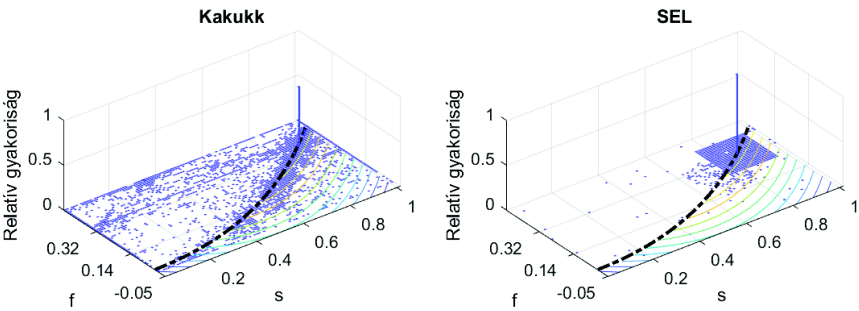
$$d_\pi(m_{i,t}) := \left| \frac{\varphi(m_{i,t})}{E(\pi^*)} - 1 \right|. \tag{19}$$

Először azon stratégiák távolságait vizsgáljuk, melyeket az algoritmus az utolsó fordulókban kap végeredményül (ld. *2. táblázat*). Az esetek 86,44%-ában a Kakukk-algoritmus az optimális stratégiát kapja, de az esetek túlnyomó részében is (94,30%) mindössze az optimum 0,05-os környezetében van az eredmény. A kifizetések tekintetében még jobb teljesítményt nyújt ez az algoritmus, itt az esetek 98,53%-ban az utolsó fordulóban használt stratégiák rátermettségi értéke az optimális várható kifizetés 1%-os környezetén belül található. Az eredmények alapján a kis távolságok esetén (0 vagy 0,01) a Kakukk-algoritmus diktálta stratégia valamivel felülmúlja a SEL stratégia teljesítményét. Ez alapján azt állíthatjuk, hogy a Kakukk-algoritmus képes alkalmazkodni a megbízó-ügynök problémához.

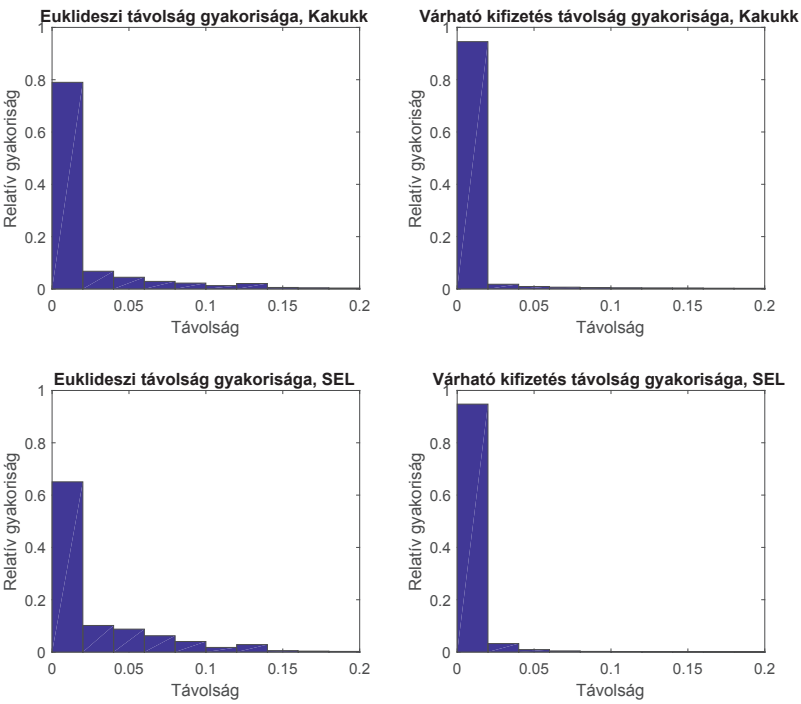
	Utolsó fordulóban az optimumtól x távolságon belül lévő stratégiák aránya, %				Utolsó fordulóban az optimumtól x%-on belül lévő kifizetések aránya, %			
	$x = 0$	$x = 0{,}01$	$x = 0{,}05$	$x = 0{,}1$	$x = 0\%$	$x = 1\%$	$x = 5\%$	$x = 10\%$
Kakukk	86,44	86,60	94,30	97,63	14,16	98,53	98,85	98,89
SEL	29,72	70,34	86,15	97,63	14,00	96,29	99,91	99,99

2. táblázat. Az utolsó fordulóban adott x távolságon belül lévő stratégiák aránya, %

Ezt az állításunkat tovább erősíti, hogy ha nem csak az utolsó fordulóban használt stratégiákat, hanem a teljes futás során használt összes stratégiát vizsgáljuk. Egy adott futás során ($seed = 2017$, $\sigma = 0{,}3$, $\gamma = 0{,}6$) használt stratégiák relatív gyakoriságát mutatja a *2. ábra.* Látható, hogy az adott futás során az algoritmusok a G stratégiatér csak egy kis részét játsszák ki, és ezek közül is messze kiemelkedik egy stratégia a maga 60%-hoz, illetve 80%-hoz közeli relatív gyakoriságával. A teljes szimuláció során használt összes stratégia távolságainak relatív gyakoriságát mutatja a *3. ábra.* Ez jelentősen nem különbözik a SEL eredményektől. A 2. ábra alapján látható, hogy a SEL stratégia során a megbízó a stratégiatér csak egy kisebb részét használja, mint a Kakukk esetben, de az optimális stratégiát nem találja meg pontosan, egy, az optimális kifizetéshez nagyon közeli kifizetést eredményező másik szuboptimális stratégia eltéríti az algoritmust, és az végül ide konvergál. Mindezekből arra következtethetünk, hogy az algoritmus már jóval az utolsó forduló előtt rendszeresen használja az utolsó fordulóban használt stratégiákat, és viszonylag hamar kitanulhatja az optimálishoz közeli paramétereket.



2. ábra. Egy tipikus futás során ($seed = 2017$, $\sigma = 0{,}6$, $\gamma = 0{,}3$) használt stratégiák eloszlása az $S \times F$ stratégiatérben. Az optimális stratégia $s = 0{,}9$ és $f = 0{,}36$. A Kakukk-algoritmus esetén leggyakrabban (65,24%) használt stratégia $(0{,}9; 0{,}36)$, SEL esetén leggyakrabban (89,11%) használt stratégia $(0{,}85; 0{,}32)$



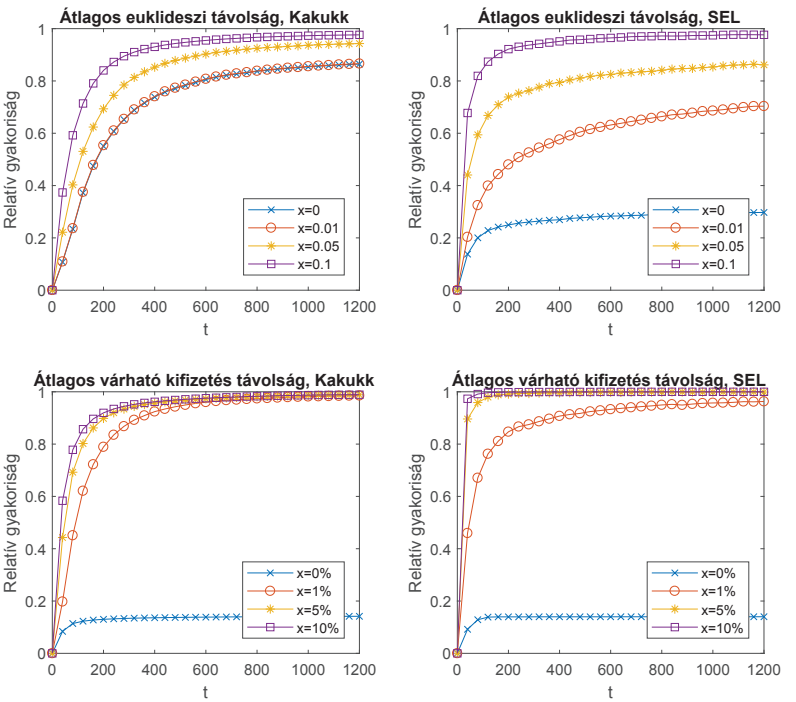
3. ábra. A szimuláció során használt stratégiák és azok rátermettségi értékeinek optimumtól vett távolságainak eloszlása.

Az algoritmus fordulóinak függvényében vizsgálva a távolságarányokat, a *4. ábrán* azt láthatjuk, hogy átlagosan az utolsó fordulóban tapasztalható távolságarányokat az 500. fordulót megelőzően éri el a Kakukk-algoritmus, innentől kezdve jelentős javulás már nem figyelhető meg. Ugyanez a jelenség a SEL esetén már jóval hamarabb bekövetkezik. Ahhoz, hogy ezt a kérdést alaposabban megvizsgálhassuk, Arifovic és Karaivanov [2010] nyomán

definiálunk egy konvergencia kritériumot. Ha létezik $\tau \in [1; T-L]$, hogy

$$\frac{1}{L}\sum_{t=\tau+1}^{\tau+L} 1\left\{\left(\frac{1}{N}\sum_{i=1}^{N} 1\{d(b_{i,t}) < x\}\right) > q\right\} > r \tag{20}$$

feltétel teljesül, akkor az algoritmus az adott paraméterek mellett d mérték szerint τ fordulótól (L, q, r, x, d) konvergens. A legkisebb ilyen τ az algoritmus konvergencia-kezdőpontja. Azaz, ha egy futás során létezik egy τ időpont, amit követő L fordulóból álló részsorozatra igaz, hogy a részsorozaton belül több mint a fordulók q százalékában az adott fordulóban szereplő stratégiák r százaléka a d mérték szerint x távolságon belül van az optimumhoz képest, akkor az algoritmus τ lépéstől (L, q, r, x, d) konvergens, és a legkisebb τ a konvergencia-kezdőpont.



4. ábra. A szimuláció során használt stratégiák és azok rátermettségi értékeinek adott x távolságon belüli átlagos aránya az egyes fordulókban

A $(200, 90\%, 90\%, x, d)$ konvergencia definíciót használva a szimulációs eredményekre azt kapjuk, hogy az algoritmusok a szimulációk jelentős részében konvergensek. A Kakukk-algoritmust használó futások 84,84%-ban van olyan 200 forduló hosszú időszak, amikor több, mint 180 fordulóban (90%) legalább 27 stratégia (90%) pontosan az optimális stratégia. Ilyen esetekben az átlagos konvergencia kezdőpont a 474. fordulóban van, azaz az algoritmus már ebben a fordulóban szinte tökéletesen megtanulja az ügynökök hasznossági függvényét és innentől az optimális szerződést ajánlja nekik. A

SEL algoritmust használó futásoknál ugyanez az eredmény már jóval alacsonyabb teljesítményt tükröz, itt mindössze 30,19% a pontosan az optimumba konvergáló futások aránya. A távolság növelésével a SEL algoritmus konvergencia teljesítménye is javul. A kifizetések szempontjából vizsgálva a kérdést a Kakukk-futások 99,45%-ra igaz, hogy létezik 200 hosszú időszak, amikor legalább 180 fordulóban minimum 27 stratégia átlagos kifizetése az optimális várható kifizetés 1%-os környezetén belül van és az átlagos konvergencia kezdőpont a 311. forduló. A SEL esetében ugyanez az arány 96,99%, viszont az átlagos konvergencia kezdőpont itt már csupán a 104. forduló. A további eredményeket a *3. táblázatban* foglaljuk össze.

Ezek alapján arra a következtetésre jutunk, hogy ha a pontos optimális stratégia használata különösen fontos, akkor a Kakukk-algoritmus által kijelölt stratégia a célravezető, míg ha konvergencia gyorsasága a fontos, és ezért cserébe megelégszünk egy, az optimális kifizetéstől épp hogy elmaradó stratégiával, akkor a SEL algoritmus nyújt gyorsabb teljesítményt. Az optimális stratégia megtalálása kifejezetten fontos olyan problémák esetén, amikor a célfüggvénynek lokális szélsőértékei vannak, és ilyenkor a Kakukk-algoritmus előnyei hatványozottan érvényesülhetnek.

	$(200, 90\%, 90\%, x, d)$ konvergens futások aránya, %				Átlagos konvergencia-kezdőpont			
	$x = 0$	$x = 0{,}01$	$x = 0{,}05$	$x = 0{,}1$	$x = 0$	$x = 0{,}01$	$x = 0{,}05$	$x = 0{,}1$
Stratégiák euklideszi távolsága								
Kakukk	84,84	85,02	93,81	98,29	474,14	473,60	413,71	291,10
SEL	30,19	70,72	89,43	98,35	196,50	217,82	126,12	47,20
Várható kifizetés távolság								
Kakukk	14,29	99,45	99,97	99,99	212,60	311,71	217,74	184,24
SEL	13,99	96,99	100	100	19,07	104,33	14,36	3,49

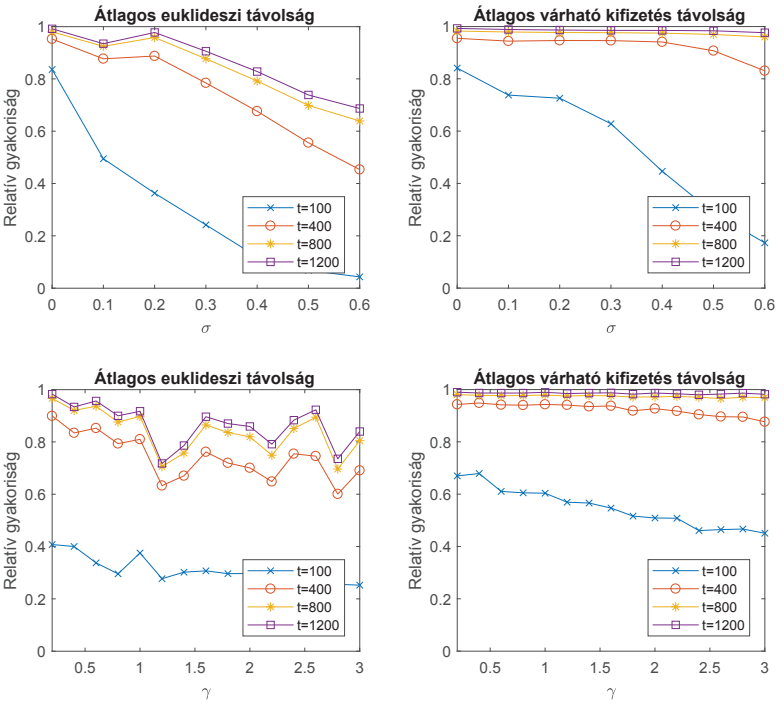
3. táblázat. Konvergens futások aránya (%) és az átlagos konvergencia-kezdőpontok

6 Érzékenységvizsgálat

Ebben a fejezetben az Kakukk-algoritmus robusztusságát vizsgáljuk mind a modellparaméterek (σ, γ), mind pedig az algoritmus paramétereinek (N, J, p, β) függvényében.

Először a modellparaméterekre való robusztusságot vizsgáljuk. Az *5. ábrán* az outputra rakódó zaj szórásának függvényében ábrázoljuk az 0,01 illetve 1% távolságon belüli stratégiák arányát a $t = 100$, $t = 400$, $t = 800$ és $t = 1200$ fordulókban. Az ábrán azt látjuk, hogy az algoritmus az output szórásának növelésével, mind a négy fordulóban egyre kisebb arányban találja meg az adott optimális stratégiától adott távolságon belüli stratégiákat. Ez azt jelenti, hogy a konvergencia sebessége csökken, akár olyan mértékben is, hogy még az 1200. fordulóban sem konvergál. Mivel az algoritmus a magasabb rátermettségi értékkel rendelkező stratégiákat választja, és a rátermettségi értéket a korábban tapasztalt kifizetésekből számítja, így az átlagos várható kifizetés távolság tekintetében a szórás növelésével a 400. fordulótól az arányok nem változnak, csak a 100. fordulóban, azaz a konvergencia sebessége csökken.

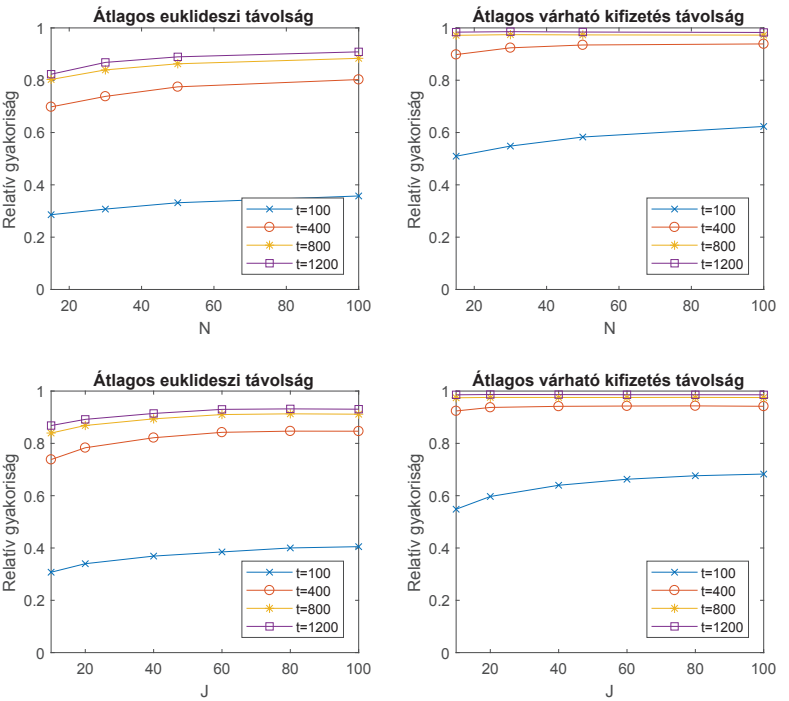
A kockázatelutasítási paraméter (γ) növelésével az arányok nem változnak szignifikánsan az egyes fordulókban, így az algoritmus ezen modellparaméterre robusztus.



5. ábra. A Kakukk-algoritmust implementáló szimulációk során használt stratégiák és azok rátermettségi értékeinek $x = 0{,}01$ illetve $x = 1\%$ távolságon belüli átlagos aránya az egyes fordulókban különböző σ és γ és a többi paraméter (ld. 1. táblázat) változatlansága mellett

Az algoritmus paramétereire való érzékenységet vizsgálva előzetesen arra számítunk, hogy mivel az outputban és ezen keresztül a rátermettségi értékekben megjelenő véletlen zaj szórása rontja az algoritmus teljesítményét, ezért a kiajánlott stratégiák számának, illetve az ügynökök számának növelésével pontosabban becsülhető a kifizetés várható értéke valamint a rátermettségi érték, és így az algoritmus teljesítménye javulhat. Azt tapasztaljuk (ld. *6. ábra*), hogy ha 15-ről 100-ra növeljük a stratégiák számát (N), azaz $2 \times 15 \times 10 \times 1200 = 360000$ szerződéskötés helyett $2 \times 100 \times 10 \times 1200 = 2400000$ szerződéskötés történik futásonként, akkor az optimumhoz képest $0{,}01$-os környezetben lévő stratégiák aránya körülbelül 10%-kal javul az 1200. fordulóban, míg az átlagos várható kifizetés tekintetében nem történik jelentős változás, az arány 98% fölött marad. A 100. fordulóban az arányok több mint 10%-kal javulnak, és ebből arra következtethetünk, hogy a konvergencia sebessége nő a fordulónként használt stratégiák számának növelésével. Ha az ügynökök számát (J) növeljük 10-ről 100-ra, akkor, bár futásonként majdnem kétszer annyi szerződéskötés történik, mint az előző, $J = 10$, $N = 100$ esetben, javulás az algoritmus teljesítményében mégsem figyelhető meg. További

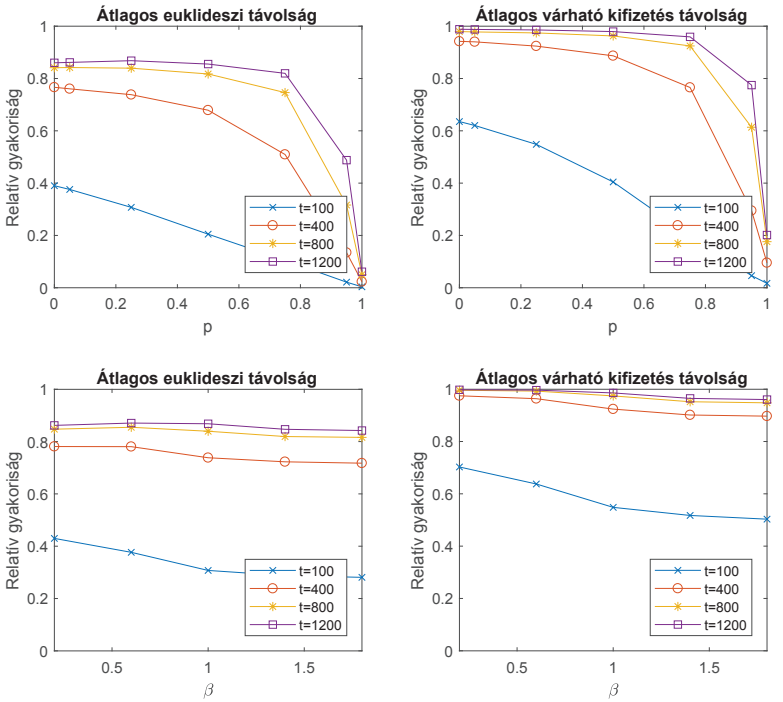
érdekesség, hogy $J = 60$-ról tovább növelve az ügynökök számát, jelentős javulás már nem érhető tetten. Ez alapján azt állíthatjuk, hogy mind a fordulónként használt stratégiák számának (J), mind pedig az ügynökök számának (N) növelésével összességében mérsékelt javulást érhetünk el. Azonban, ha az outputra rakódó zaj szórása (σ) nagy, akkor pontatlanabb lesz az output várható értékének becslése, ami meghatározza a stratégiák rátermettségi értékét, így ebben az esetben N és J növelése nagyobb mértékben javíthatja az algoritmus teljesítményét.



6. ábra. A Kakukk-algoritmust implementáló szimulációk során használt stratégiák és azok rátermettségi értékeinek $x = 0{,}01$ illetve $x = 1\%$ távolságon belüli átlagos aránya az egyes fordulókban különböző N és J és a többi paraméter (ld. 1. táblázat) változatlansága mellett

Az algoritmus paraméterei közül a kakukktojás felfedezésének valószínűségét változtatva azt találjuk, hogy ennek komoly szerepe van az algoritmus működésében. Ha a valószínűséget $p = 1$-re állítjuk, azaz a kakukktojást biztos, hogy felfedezik és a $c_{i,t}$, kakukkstratégiákba a (14) képlet alapján mindig a $h_{i,t}$ gazdastratégiák kerülnek és így lényegében nem készülnek valódi kakukkstratégiák, akkor az algoritmus teljesítménye jelentősen romlik (ld. *7. ábra*), az 1200. forduló után mindössze a stratégiák 6,13%-a van az optimum 0,01-os környezetén belül és a kifizetéseknél csak 20,15% az 1%-os eredmény. A másik szélsőséges esetben, amikor a valószínűség $p = 0$, azaz a kakukktojás biztos, hogy kikel, az eredmények azt mutatják, hogy az algoritmus a leggyorsabban konvergál, de az euklideszi távolságot tekintve nem a legjobb teljesítmény nyújtja a vizsgált hét paraméterbeállítás közül. Az euklideszi

távolság alapján az 1200 fordulót tekintve a $p = 0{,}25$ beállítás a legkedvezőbb. A Mantegna Lévy-repülést előállító algoritmusának β paraméterét változtatva az algoritmus teljesítményében kis mértékű változás figyelhető meg, erre a paraméterre az algoritmus kevésbé érzékeny.



7. ábra. A Kakukk-algoritmust implementáló szimulációk során használt stratégiák és azok rátermettségi értékeinek $x = 0{,}01$ illetve $x = 1\%$ távolságon belüli átlagos aránya az egyes fordulókban különböző p és β és a többi paraméter (ld. 1. táblázat) változatlansága mellett

A Kakukk-algoritmus érzékenységvizsgálatából arra következtethetünk, hogy a modellparaméterek közül az outputra rakódó zaj σ szórásának növekedése, megfelelően az előzetes várakozásoknak (várható érték becslési hibája miatt), rontja az algoritmus teljesítményét, az ügynök kockázatelutasítási γ paraméterének változtatása viszont érdemben nem változtatja meg az eredményeket. Az algoritmus paraméterei közül a fordulónként használt stratégiák N számának és az ügynökök J számának növelésével, bár pontosítható a várható érték becslése, a szimulációkban mégsem figyelhető meg javulás. Ezen paramétereknek zajosabb környezetben lehet jelentősebb szerepük. A kakukktojás felfedezésének valószínűségét szabályozó p paraméter vizsgálatából kitűnik, hogy a $c_{i,t}$ kakukkstratégiáknak lényegi szerepük van. Ha ezt a lépést kihagyjuk a folyamatból, azaz a $p = 1$ paraméterbeállítást választjuk, akkor a teljesítmény drámaian lecsökken. A Lévy-repülést szabályozó β paraméternek mérsékeltebb szerepe van, legfeljebb a konvergencia sebességére van hatással.

7 Összegzés

A dolgozatban a megbízó-ügynök problémát vizsgáljuk egy evolúciós-tanuló algoritmus, a Kakukk-algoritmus segítségével. Az algoritmus teljesítményének mérésére Arifovic és Karaivanov [2010] nyomán felállítunk egy egyszerű lineáris szerződés modellt tartalmazó keretrendszert, ahol a szereplők a szerződés paramétereinek meghatározása és az erőfeszítésszint választásából álló iteratív játékot játsszák. Olyan feltételezésekkel élünk, mint, hogy a megbízó csak lineáris szerződést ajánlhat, kockázatsemleges és várható értékben optimalizál, az ügynökök csak az adott fordulót veszik alapul, a saját, azonos hasznossági függvényük alapján optimalizálnak és nem próbálnak komplexebb stratégiát játszani. Többek között nem vesszük figyelembe például azt sem, hogy a folytonos, fordulóról fordulóra történő szerződésmódosítások befolyásolják-e az ügynökök magatartását, és ennek milyen kihatásai vannak a kifizetésre.

Ezen leegyszerűsítő feltételezések mellett a szimulációk alapján azt találjuk, hogy bár a sztochasztikus környezet és a nem folytonos kifizetésfüggvény miatt nehéz dolga van az adaptív tanuló algoritmusoknak, a Kakukk-algoritmus jól implementálható a felállított megbízó-ügynök keretrendszerben. Az algoritmus a korábbi algoritmusoknál gyorsabban éri el a közel százszázalékos eredményt. A modellparaméterek közül a sztochasztikus környezet zajossága az, ami nagyon megnehezítheti az algoritmus dolgát, minél nagyobb a véletlen zaj szórása, a megbízó annál kisebb arányban képes megtanulni az optimális paramétereket. Az ügynökök és a stratégiák számának növelése, azaz az erőfeszítésszint minél pontosabb becslése, egyaránt segítheti az algoritmus teljesítményének javulását. Az algoritmus érzékenységvizsgálatából az is kiderül, hogy az algoritmus egyik lényegi pontja, a kakukkstratégiák előállítása, ami alapjaiban határozza meg az algoritmus működését.

A Kakukk-algoritmus a felállított keretrendszerben leginkább a pontosság, azaz a optimális stratégia pontos megtalálásában nyújt jobb teljesítményt, mint a korábban használt SEL algoritmus, a kifizetés és a konvergencia tekintetében elmarad tőle. A Kakukk-algoritmus diktálta stratégiát játszva kisebb valószínűséggel ragad a megbízó egy szuboptimális stratégiában. Ezek alapján bízhatunk abban, hogy olyan környezetben, ahol a megbízó célfüggvényének lokális szélsőértékei vannak, a Kakukk-algoritmust használva a megbízó maximalizálhatja a hasznosságát. Ezen hipotézis vizsgálata későbbi tanulmányok témája lehet.

A szimulációk alapján a Kakukk-algoritmus működhet olyan megbízó-ügynök problémánál, ahol a termelőeszköz tulajdonosa viszonylag sok, hasonló hasznossággal rendelkező ügynöknek tudja felajánlani az eszközét, a termelés gyors, szinte azonnal megállapítható az eredmény, és lehetőség van gyakori szerződésmódosításra. Ilyen gyakorlati alkalmazás lehet az online tartalom-előállítás vagy akár a személyfuvar-közvetítés is.

A. Melléklet

A 2. fejezetben bemutatott megbízó-ügynök keretrendszert és az optimális szerződéses paraméterek levezetését Bolton és Dewatripont [2005] [137-139] nyomán ebben a mellékletben közöljük.

$y(z) := z + \varepsilon$, output
$z \in I\!R_0^+$, ügynök erőfeszítés szintje
$\varepsilon \sim N(0, \sigma^2)$, véletlen zaj
$\pi(y(z)) := (1 - s) \cdot y(z) + f$, megbízó kifizetése
$s \in [0; 1]$, változó rész
$f \in I\!R$, fix rész
$c\big(y(z)\big) := s \cdot y(z) - f$, megbízó költsége, ügynök kifizetése
$u\big(z\big) := -e^{-\gamma \cdot \left(c(y(z)) - \frac{1}{2} \cdot z^2\right)}$, ügynök CARA hasznossági függvénye
$\gamma \in I\!R^+$, kockázatelutasítási paraméter

Ekkor a megbízó feladata:

$$\max_{s,f} (1 - s) \cdot E\big(y(z)\big) + f \tag{21}$$

s.t.

$$z = \arg\max_{\hat{z}} E\big(u(\hat{z})\big) \qquad \text{(ösztönzési korlát IC)}$$

$$E\big(u(z)\big) \geq u(\bar{w}) \qquad \text{(részvételi korlát PC)}$$

A feladatot visszafelé oldjuk meg, azaz adott (s, f) párhoz kiszámítjuk az ügynök optimális z^* erőfeszítés szintjét (megoldjuk az IC korlátot), majd ez után optimalizáljuk a megbízó célfüggvényét. Az ügynök várható hasznossága:

$$E\big(u(c(y), z)\big) = E\big(-e^{-\gamma \cdot (s \cdot (z+\varepsilon) - f - \frac{1}{2} z^2)}\big) = -e^{-\gamma \cdot (s \cdot z - f - \frac{1}{2} z^2)} \cdot E(e^{-\gamma \cdot s \cdot \varepsilon}) \tag{22}$$

$$\begin{aligned} E(e^{-\gamma \cdot s \cdot \varepsilon}) &= \frac{1}{\sqrt{2\pi\sigma}} \cdot \int_{-\infty}^{\infty} e^{-\gamma \cdot s \cdot \varepsilon} \cdot e^{-\frac{\varepsilon^2}{2\sigma^2}} \, d\varepsilon = \frac{1}{\sqrt{2\pi\sigma}} \cdot \int_{-\infty}^{\infty} e^{-\frac{\varepsilon^2 + 2 \cdot \gamma \cdot s \cdot \varepsilon \cdot \sigma^2}{2\sigma^2}} d\varepsilon = \\ &= \frac{1}{\sqrt{2\pi\sigma}} \cdot \int_{-\infty}^{\infty} e^{-\frac{(\varepsilon + \gamma \cdot s \cdot \sigma^2)^2 - \gamma^2 s^2 \sigma^4}{2\sigma^2}} \, d\varepsilon = \\ &= e^{\frac{\gamma^2 s^2 \sigma^2}{2}} \cdot \underbrace{\frac{1}{\sqrt{2\pi\sigma}} \cdot \int_{-\infty}^{\infty} e^{-\frac{(\varepsilon - (-\gamma \cdot s \cdot \sigma^2))^2}{2\sigma^2}} \, d\varepsilon}_{=1, \text{ mert } \sim N(-\gamma \cdot s \cdot \sigma^2, \sigma^2) \text{sűrűségfüggvénye}} = e^{\frac{\gamma^2 s^2 \sigma^2}{2}} \end{aligned} \tag{23}$$

A (22) egyenlőséget tovább folytatva:

$$\begin{aligned} E\big(u(c(y), z)\big) &= -e^{-\gamma \cdot (s \cdot z - f - \frac{1}{2} z^2)} \cdot e^{\frac{\gamma^2 s^2 \sigma^2}{2}} = \\ &= -e^{-\gamma \cdot (s \cdot z - f - \frac{1}{2} \cdot z^2 - \frac{\gamma}{2} \cdot s^2 \cdot \sigma^2)} = -e^{-\gamma \hat{w}(z)} \, . \end{aligned} \tag{24}$$

Az ügynök kockázat egyenértékese:

$$\hat{w}(z) = s \cdot z - f - \frac{1}{2} \cdot z^2 - \frac{\gamma}{2} \cdot s^2 \cdot \sigma^2 \,. \tag{25}$$

Mivel a hasznossági függvény exponenciális alakú, így az ügynök ösztönzési korlátja átírható a következő formába:

$$z = \arg\max_{\hat{z}} \hat{w}(\hat{z}) \,. \tag{26}$$

Ezt maximalizáljuk z-ben, azaz az elsőrendű feltétel (z szerinti parciális derivált):

$$\frac{\partial \hat{w}(z)}{\partial z} = s - z \tag{27}$$

$$z^* = s \,. \tag{28}$$

Az optimális z^* értéket behelyettesítjük a PC korlátba. Esetünkben az ügynök külső opciójának értéke 0, azaz $\bar{w} = 0$ és mivel az u hasznossági függvény exponenciális (monoton), ezért a PC korlát a következőképpen alakul:

$$s \cdot z^* - f - \frac{1}{2} {z^*}^2 - \frac{\gamma}{2} s^2 \sigma^2 \geq 0 \tag{29}$$

$$s \cdot s - f - \frac{1}{2} s^2 - \frac{\gamma}{2} s^2 \sigma^2 \geq 0 \tag{30}$$

$$f \leq \frac{1}{2} s^2 - \frac{\gamma}{2} s^2 \sigma^2 \,. \tag{31}$$

A megbízó célfüggvénye, amit maximalizálni szeretne:

$$(1 - s) \cdot s + f = s - s^2 + f \,. \tag{32}$$

Ez az s-ben másodfokú kifejezés akkor lesz maximális, ha f értéke maximális, azaz a (31) kifejezés egyenlőségre teljesül. Ekkor a célfüggvény:

$$(1 - s) \cdot s + f = s - s^2 + \frac{1}{2} s^2 - \frac{\gamma}{2} s^2 \sigma^2 \,. \tag{33}$$

A célfüggvényt deriválva kapjuk az s^* optimális értéket:

$$\frac{\partial \big((1 - s) \cdot s + f\big)}{\partial s} = 1 - 2s + s - \gamma \cdot \sigma^2 \cdot s \tag{34}$$

$$s^* = \frac{1}{1 + \gamma \cdot \sigma^2} \,. \tag{35}$$

Ezt visszahelyettesítve f egyenletébe (31), adódik az f^* optimális érték:

$$f^* = \frac{1 - \gamma \cdot \sigma^2}{2 \cdot (1 + \gamma \cdot \sigma^2)^2} \,. \tag{36}$$

B. Melléklet

Ebben a mellékletben Arifovic és Karaivanov [2010] nyomán a Social Evolutionary Learning (SEL) algoritmus implementációját részletezzük.

$m := (s; f), s \in S, f \in F$, megbízó által játszott stratégiák a G stratégiatérből

$\varphi : G \to I\!R$, meghatározza az adott stratégia rátermettségi értékét az adott fordulóban

$y = s + \varepsilon_j$, a j-edik ügynök által megtermelt output

$\hat{\pi} = f + (1 - s) \cdot (s + \frac{1}{J} \sum_{j=1}^{J} \varepsilon_j)$, adott stratégiához tartozó átlagos kifizetés

Az algoritmus minden $t = 1, \ldots, T$ fordulóban egy N elemű $b \in G^N$ stratégiákat tartalmazó vektorral dolgozik. Első lépésként az előző fordulóban meghatározott $b_{i,t-1}$ stratégiákat $p_{i,t}$ valószínűséggel választja be a $g_{k,t}$ stratégiákba:

$$g_{k,t} := \begin{cases} b_{1,t-1}, & p_{1,t} \\ b_{2,t-1}, & p_{2,t} \\ \vdots & \vdots \\ b_{N,t-1}, & p_{N,t} \end{cases}, \tag{37}$$

ahol adott $b_{i,t-1}$ stratégia beválasztásának valószínűsége a stratégia rátermettségi értékétől függ:

$$p_{i,t} := \frac{e^{\varphi(b_{i,t-1})}}{\sum_{k=1}^{N} e^{\varphi(b_{k,t-1})}}. \tag{38}$$

Az így kiválasztott $g_{i,t}$ stratégiák helyett μ valószínűséggel az adott $g_{i,t}$ stratégia r_m sugarú környezetéből választja a $h_{i,t}$ stratégiát:

$$h_{i,t} := \begin{cases} g_{i,t}, & 1 - \mu \\ g_{i,t} + r_{i,t}, & \mu\,, \end{cases} \tag{39}$$

ahol

$$r_{i,t} \sim U[-r_m; r_m]\,. \tag{40}$$

Végül a megkapott $g_{i,t}$ stratégiákat páronként összehasonlítja a $b_{i,t-1}$ stratégiákkal, és a nagyobbat választja a $b_{i,t}$ stratégiákba:

$$b_{i,t} := \begin{cases} g_{i,t}, & \text{ha } \varphi(b_{i,t-1}) < \varphi(g_{i,t}) \\ b_{i,t-1}, & \text{ha } \varphi(b_{i,t-1}) \geq \varphi(g_{i,t})\,. \end{cases} \tag{41}$$

Egy új $b_{i,t}$ stratégia legenerálását tekintjük egy műveletnek. Egy fordulóban a stratégiagenerálás $O(N)$ futási idejű, azaz a stratégiák számának lineáris függvénye.

Ezek után az ügynökök előállítják az outputokat, majd az algoritmus frissíti a rátermettségi értékeket. Ismét egy műveltnek tekintjük egy stratégia

egy ügynöknek történő felajánlását és a rátermettségi érték frissítését a megfigyelt outputtal. Ennek a fázisnak a futási ideje $O(N \cdot J)$, azaz a stratégia- és az ügynökszám szorzatának lineáris függvénye.

Azért, hogy a SEL összevethető legyen a Kakukk-algoritmussal, és a teljesítménykülönbség ne csak a rátermettségi érték pontosabb becsléséből fakadjon (a Kakukk-algoritmus egy fordulóban kétszer ajánl stratégiákat az ügynököknek, és kétszer frissíti a rátermettségi értékeket), egy fordulóban a fenti folyamatot egymás után kétszer futtatjuk le. A SEL algoritmus tanulási stratégiát követve egy forduló futási ideje $2 \cdot (O(N) + O(N \cdot J)) = O(N \cdot J)$, azaz a stratégia- és az ügynökszám szorzatának lineáris függvénye. A futtatási paramétereknek az Arifovic és Karaivanov [2010] tanulmányban legjobb eredményt elérő paraméterbeállításokat használunk. Ezeket a beállításokat a *4. táblázat* tartalmazza.[3]

Paraméter	Érték
Modell	
Kockázatelutasítás paraméter, γ	15 egyenletes távolságú pont a [0,2; 3]-ból
Output szórása, σ	7 egyenletes távolságú pont a [0; 0,6]-ból
Szimuláció	
Véletlen magok száma, *seed*	70 egyenletes eloszlású véletlen a [1; 10000]-ból
Stratégiatér elemszáma, $\|G\|$	$101 \cdot 56 = 5656$
Rácsfinomság, d	0,1
Fordulók száma, T	1200
SEL	
Stratégiák száma, N	30
Ügynökök száma stratégiánkként, J	10
Mutáció valószínűsége, μ	0,05
Mutáció sugara, r_m	0,1

4. táblázat. A referencia szimulációban használt paraméterek és azok értékei

Irodalom

1. Arifovic, J. és Karaivanov, A. [2010] Learning by doing vs. learning from others in a principal-agent model. *Journal of Economic Dynamics and Control*, 34(10):1967–1992, . URL https://doi.org/10.1016/j.jedc.2010.04.007.
2. Axelrod, R. [1987] The evolution of strategies in the iterated prisoner's dilemma. The dynamics of norms, 1–16.
3. Berlinger, E., Lovas, A., és Juhász, P. [2017] State subsidy and moral hazard in corporate financing. *Central European Journal of Operations Research,* 25(4):743–770, URL https://doi.org/10.1007/s10100-016-0461-8.
4. Bolton, P. és Dewatripont, M. [2005] *Contract theory.* The MIT Press, Cambridge, Massachusetts. ISBN 978-0-262-02576-8.
5. Camerer, C. és Hua Ho, T. [1999] Experience-weighted attraction learning in normal form games. *Econometrica*, 67(4):827–874. URL http://dx.doi.org/10.1111/1468-0262.00054.

[3] A szimulációkat a MATLAB R2017a programmal futtatjuk. A kódok elérhetőek a MATLAB közösség fájlcserélő oldalán: https://www.mathworks.com/matlabcentral/fileexchange/66251-cuckoo-search-learning-strategy-in-the-principal-agent-model

6. Csóka, P., Havran, D., és Szűcs, N. [2015] Corporate financing under moral hazard and the default risk of buyers. *Central European Journal of Operations Research*, 23(4):763–778. URL http://dx.doi.org/10.1007/s10100-013-0319-2.

7. Gandomi, A. H., Yang, X.-S., és Alavi, A. H. [2013] Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with Computers*, 29(1):17–35. URL http://dx.doi.org/10.1007/s00366-012-0308-4.

8. Holmström, B. és Milgrom, P. [1991] Multitask principal-agent analyses: Incentive contracts, asset ownership, and job design. *Journal of Law, Economics, & Organization*, 7:24–52. URL http://www.jstor.org/stable/764957.

9. Kóczy, Á. L. és Kiss, H. J. [2017] Hart és Holmström szerződéselméleti munkássága. *Hitelintézeti Szemle*, 16(1):162–174. URL http://hitelintezetiszemle.hu/letoltes/koczy-a-laszlo-kiss-hubert-janos.pdf.

10. Layeb, A. [2011] A novel quantum inspired cuckoo search for knapsack problems. *International Journal of Bio-inspired Computation*, 3(5):297–305. URL https://doi.org/10.1504/IJBIC.2011.042260.

11. Mantegna, R. N. [1994] Fast, accurate algorithm for numerical simulation of Levy stable stochastic processes. *Physical Review E*, 49(5):4677–4683. URL https://doi.org/10.1103/PhysRevE.49.4677.

12. Rose, D. és Willemain, T. R. [1996] The principal-agent problem with evolutionary learning. *Computational & Mathematical Organization Theory*, 2(2):139–162. URL https://link.springer.com/article/10.1007/BF00240424.

13. Roth, A. E. és Erev, I. [1995] Learning in extensiv e-form games: Experimental data and simple dynamic models in the intermediate term. *Games and Economic Behavior*, 8(1):164–212. URL https://doi.org/10.1016/S0899-8256(05)80020-X.

14. Stiglitz, J. E. [1974] Incentives and risk sharing in sharecropping. *The Review of Economic Studies*, 41(2):219–255. URL http://www.jstor.org/stable/2296714.

15. Szabó, G. és Tőke, C. [1998] Evolutionary prisoner's dilemma game on a square lattice. *Physical Review E,* 58(1):69. URL https://doi.org/10.1103/PhysRevE.58.69.

16. Szabó, G., Vukov, J., és Szolnoki, A. [2005] Phase diagrams for an evolutionary prisoner's dilemma game on two-dimensional lattices. *Physical Review E*, 72(4):047107. URL https://doi.org/10.1103/PhysRevE.72.047107.

17. Tein, L. H. és Ramli, R. [2010] Recent advancements of nurse scheduling models and a potential path. *In Proc. 6th IMT-GT Conference on Mathematics, Statistics and its Applications (ICMSA 2010)*, 395–409.

18. Yang, X.-S. és Deb, S. [2009] Cuckoo search via Lévy flights. In *Nature & Biologically Inspired Computing. NaBIC 2009. World Congress on*, 210–214. IEEE, 2009. URL https://doi.org/10.1109/NABIC.2009.5393690.

19. Yang, X.-S. és Deb, S. [2014] Cuckoo search: recent advances and applications. *Neural Computing and Applications*, 24(1):169–174. URL https://doi.org/10.1007/s00521-013-1367-1.

20. Yang, X.-S., Deb, S., Karamanoglu, M., és He, X. [2012] Cuckoo search for business optimization applications. In *omputing and Communication Systems (NCCCS), National Conference on*, 1–5. IEEE, 2012. URL https://doi.org/10.1109/NCCCS.2012.6412973.

CUCKOO SEARCH LEARNING STRATEGY IN THE PRINCIPAL-AGENT MODEL

In this study we analyze the performance of Cuckoo Search learning algorithm in a principal-agent model. We introduce a standard principal- agent model framework, where the principal can incentivize the agent using output sharing (linear contract) and the moral hazard can be eliminated. In the model the principal knows neither the agent's utility function nor the properties of the stochastic environment, but by iterating the process which consits of contracting, observation of output and update of contract parameters, she can learn and optimize her objective function. We use Monte-Carlo simulations and find that, despite the discontinuous utility function and the stochastic environment, the Cuckoo Search can accommodate to the principal-agent model and find the maximum of principal's utility function more precisely than former algorithms.

Keywords: principal-agent problem, linear contract, repeated game, Cuckoo Search. *JEL codes:* C61, C63, C73, D86