Nehéz, kombinatorikus optimalizálási feladatok számítógéppel segített megoldása

doktori (PhD) értekezés tézisei

Készítette: Ábrahám Gyula

Témavezetők: Dr. Dósa György, Starkné dr. Werner Ágnes

Pannon Egyetem
Műszaki Informatikai Kar
Informatikai Tudományok Doktori Iskola

2022

1. A munka háttere, motivációja

Az értekezésben három problémával foglalkoztam az ütemezés és a ládapakolás területekhez kapcsolódóan. Mindhárom területnek számos alkalmazása van a gyakorlatban, többek között az iparban, a gazdasági életben vagy éppen az optimalizálásban. Az ütemezési feladat (*unrelated machine scheduling with precedence constraints*) megoldásában egy, a megerősítéses tanulás területén ismert és népszerű algoritmust vettem alapul, amelyre építve a tevékenységek sorrendjét meghatározó megoldást hoztam létre. A ládapakolási feladatok (*bin packing*) megoldása során ún. előfeldolgozó algoritmusok segítségével igyekeztem megoldani benchmark feladatokat. Továbbá egy viszonylag új területtel is foglalkoztam, amelynek ládafedés szállítással (*Bin Covering with Delivery*, röviden BCD) a neve. Ezen a területen egyrészt természetesen adódó algoritmusokkal oldottam meg a benchmark feladatokat és hasonlítottam össze a megoldásokat az általam kidolgozott új, rugalmas algoritmus eredményeivel. Mindhárom feladat meglehetősen nehéz, bonyolult kombinatorikus optimalizálási feladat. Megoldásuk során számítógépes módszereket alkalmaztam.

1.1. Általánosságban egy ütemezési probléma esetén adottak tevékenységek (munkák) és erőforrások (a mi esetünkben gépek). Az ütemezés során azt határozzuk meg, hogy melyik tevékenységet melyik gép mettől meddig hajtja végre. A tevékenységek vagy munkák az elvégzendő feladatok, ezeknek a száma változó. Az erőforrások pedig olyan egységek, amelyek a tevékenységek végrehajtására szolgálnak. A cél pedig az, hogy ezeket az erőforrásokat a tevékenységekhez rendeljük úgy, hogy valamely célfüggvényt optimalizáljuk. Az általam vizsgált esetben a cél a teljes átfutási idő minimalizálása volt. Az ütemezési feladatok megoldására egy, a Q tanulás által motivált algoritmust fejlesztettem ki, amelyet **Q-tanulás által motivált algoritmusnak**, röviden (az angol elnevezés után) **QLM**-nek neveztem el. Az algoritmus két részből épül fel: egy mohó ütemezőből és a Q tanulást alkalmazó eljárásból, amely a tevékenységek sorrendjét határozza meg.

Az ütemezésről részletes áttekintést például [1]-ben találunk. A könyv igen részletesen tárgyalja az ütemezéshez kapcsolódó elméleti modelleket és a különböző ütemezési problémákat. Az ütemezés területe igen bőséges irodalommal rendelkezik, e helyütt Ronald L. Graham két alapvető munkáját [2, 3] említem meg. Ezekben a cikkekben definiálta a híres LS (*List Scheduling*, lista szerinti ütemezés) algoritmust, valamint ennek rendezett változatát, az LPT (*Longest processing Time*) algoritmust.

A kutatásom során vizsgált feladat a független gépek ütemezése megelőzési relációk figyelembe vételével. A megoldáshoz a megerősítéses tanulás területéről ismert eljárást, a Q-tanulást (*Q-Learning*) alkalmaztam. Az irodalomban tudomásom szerint a fent megnevezett ütemezési problémára ilyen megoldás még nem létezik, azonban az ütemezés egyéb területein van példa a megerősítéses tanulás alkalmazására. Orhean és társai [4] egy, a megerősítéses tanuláson alapuló, elosztott felhő rendszerhez alkalmazható ütemező eljárást mutattak be. A cél egy rendszer teljesítményének az optimalizálása volt az erőforrások ütemezésén keresztül. Aydin és Öztemel [5] egy ágens alapú ütemezési módszert dolgoztak ki, amelyben az ágens különböző feltételek mentén szabályokat választ ki, amelyek alapján az ütemezés végbemegy. Az ágens tanítására a Q-tanulás egy továbbfejlesztett változatát alkal-

mazták. Stefán [6] a Q-tanulás algoritmusát alkalmazta egy permutációs flow shop problémára, ahol a cél a gépek üresjárati idejének a minimalizálása volt. Stefán a disszertációjában [7] bővebb leírást adott az algoritmusról, amely a flow shop típusú probléma megoldására készült. A cikk [6] és a disszertáció [7] az általam bemutatott probléma megerősítéses tanulás oldalról való megközelítésében segített. Gabel és Riedmiller [8] szintén a Q-tanulást alkalmazták, viszont ők egy job shop típusú problémára, amelynél a Q-függvényt neurális háló segítségével közelítették. Shahrabi és társai [9] a megerősítéses tanulást alkalmazták egy job shop típusú problémára kifejlesztett eljárás továbbfejlesztéséhez. További példákat találunk a megerősítéses tanulás alkalmazására az ütemezés területén a [10–12] cikkekben. A [13]-ban például a Q-tanulásnak a neurális hálózatokkal összekapcsolt változatát alkalmazták, amelyet Deep Reinforcement Learning-nek hívnak.

1.2. A ládapakolási feladatok esetében tárgyakat szeretnénk ládákba pakolni úgy, hogy a pakolt ládák száma minimális legyen és az egy ládába pakolt tárgyak mérete ne lépje át a láda kapacitását. A probléma NP-nehéz [14, 15]. A ládapakolási problémát a hetvenes évek elején definiálták és kezdték vizsgálni. Az ún. approximációs algoritmusokat ezen a területen fejlesztették ki. Olyan algoritmust nevezünk approximációs algoritmusnak, amelytől nem várjuk el, hogy feltétlenül optimális megoldást adjon egy feladatra, de egyrészt gyors (polinomiális idejű), másrészt az általa szolgáltatott megoldás garantáltan "nincs túl messze" az optimum értéktől.

D.S. Johnson disszertációja [16] a ládapakolásról és Graham munkája [2] azokhoz a korai munkákhoz tartoznak, amelyek elindították és formálták az approximációs algoritmusok vizsgálatát és megszabták a további kutatások irányát. A ládapakolás területén megkülönböztetünk online és offline eseteket. Online esetben a tárgyak adatai előre nem ismertek, offline esetben viszont igen.

Ezen a területen a ládapakolási feladatok egy új megközelítésével foglalkoztam. Ennek lényege, hogy a ládapakolási problémák egy halmazának megoldása előtt egy ún. előfeldolgozást végzek el. Az új megközelítésben megpróbálom meghatározni az adott feladathalmaz egy elemének az optimális megoldását egy mohó algoritmussal; ha ez sikerült, a feladat megoldásával készen vagyunk. Ha viszont nem sikerült, akkor más, összetettebb algoritmust választunk. A mohó algoritmus alkalmazásának lényege, hogy egyszerű algoritmussal az adott osztályon belül a lehető legtöbb feladatot optimálisan oldjuk meg. Így a megoldott feladatokkal már nem kell foglalkozni, azaz a problémák száma csökken. A feladatok megoldására két algoritmust hoztam létre **REM WS** és **FU** néven, amelyek hatékonyságát ismert és szabadon elérhető benchmark feladatosztályokon teszteltem (Schwerin és Falkenauer_U).

1.3. A ládafedés szállítással egy viszonylag új terület. Hasonlóan a ládapakolási problémához, ebben az esetben is tárgyakat pakolunk ládákba, amelyeket, ha fedetté válnak (vagyis a ládába pakolt tárgyak összmérete legalább akkora, mint a láda kapacitása), lezárunk és elszállítunk. A célfüggvény meghatározása a fedett és elszállított ládák száma alapján történik. Azaz, minden elszállított ládáért pénzt kapunk és a cél az, hogy a profitot maximalizáljuk. A probléma elsőként a [17]-ben lett bemutatva. A probléma offline változatával a [18] foglalkozik, továbbá néhány kapcsolódó probléma a [19]-ben kerül bemutatásra.

A BCD probléma online változatában a tárgyak előre nem ismertek, és egyesével érkeznek egymás után. Az éppen érkező tárgyat azonnal be kell pakolni egy ládába. A célfüggvény a nyitott ládák számának függvényében változik. Minél több láda van nyitva egyszerre, a célfüggvény értéke annál jobban csökken. A cél az, hogy a célfüggvényt, azaz a profitot maximalizáljuk. A ládapakolási és ládafedési probléma offline és online változataival többek között a [14], a [20] és a [21] áttekintő cikkek foglalkoznak.

A kutatás során már az irodalomból ismert természetes algoritmusokat implementáltam (**DNF**, **H(K)** és **SH(K)**), valamint kidolgoztam egy új, rugalmas, paraméteres algoritmust, amelynek az **MMask** nevet adtam. Az algoritmusok tesztelésére a korábban már alkalmazott Schwerin és Falkenauer_U osztályokat, valamint egy általam létrehozott Large Range (röviden: LR) nevű feladatosztályt használtam. A nyereség-függvényekből hármat definiáltam: egy lassan csökkenő, egy négyzetesen csökkenő (emiatt először kicsit, utána gyorsan csökkenő) és egy meredeken csökkenő változatot. A feladatosztályokon módosításokat hajtottam végre: megbontottam a rendezettséget, normalizálást hajtottam végre és összekapcsoltam a feladatosztályokat a nyereségfüggvényekkel.

2. Módszerek és eszközök

2.1. Ütemezés

2.1.1. Az ütemezési probléma

A dolgozatban a független gépek ütemezése megelőzési relációk figyelembe vételével (*unrelated machine scheduling with precedence constraints*) típusú problémával foglalkoztam, amely az alábbiak szerint adható meg:

$$R_m|prec|C_{max} \tag{1}$$

Ahol az R_m a gépek halmaza (m darab független gép), $prec$ jelöli azt, hogy az egyes tevékenységek között megelőzési relációk vannak és $C_{max} = max(C_1, \ldots, C_n)$ jelöli a legkésőbb befejeződő tevékenység befejezési idejét a rendszerben, amit minimalizálunk. A cél pedig a teljes átfutási idő minimalizálása. A feladat egy inputja a

$$\langle \mathcal{T}, \mathcal{M}, G \rangle \tag{2}$$

rendezett hármassal írható le, ahol $\mathcal{T} = \{task_1, \ldots, task_n\}$ az összes tevékenység halmaza, $\mathcal{M} = \{m_1, \ldots, m_m\}$ az összes erőforrás halmaza és $G = (V, E)$ egy gráf, ahol V a csúcsok véges halmaza, $E \subseteq V \times V$ az élek halmaza. A gráf egy élét a $(v_i, v_j) \in E$ alakban írjuk és az alábbiakat követeljük meg:

- irányított gráf, azaz $E \subseteq V \times V$ a csúcsokból alkotott rendezett párok halmaza,
- egyszerű gráf, azaz $(v_i, v_j) \in E$ esetében $i \neq j$ (hurokélmentes) és $(v_j, v_i) \notin E$ (nincs többszörös él) $\forall i, j$-re,

- diszjunkt utak és izolált pontok uniója.

2.1.2. A javasolt módszer

Az eljárás két komponensre bontható fel. Az egyik egy mohó algoritmus, amely az ütemezést végzi a tevékenységek egy megadott sorrendjében. A másik pedig maga a Q-tanulással támogatott komponens, amelynek feladata a tevékenységek egy sorrendjének, más szóval permutációjának az előállítása. A cél az, hogy olyan permutációt találjon az algoritmus, amely szerint mohón ütemezve az átfutási idő minimális lesz. Az algoritmus megpróbálja megkeresni a legjobb sorrendet, de nem garantálja az optimális megoldást. A mohó algoritmus működése a soron következő tevékenységhez mindig azt a gépet rendeli hozzá, amellyel az addig elért átfutási idő a legkisebb mértékben növekszik, természetesen a megelőzési relációkat is figyelembe véve.

A QLM algoritmus az első lépésben egy véletlenszerű sorrendet alakít ki a tevékenységek között. A sorrend ekkor azért véletlen, mert a Q táblázat elemei nullák, emiatt a kiválasztás válószínűsége véletlenszerű. A folyamat során a Q táblázat értékei folyamatosan frissülnek az alábbi (3)-as szabály szerint.

$$Q_{t+1}(S_t, A_t) = (1 - \alpha_t)Q_t(S_t, A_t) + \alpha_t(R_t + \gamma_t \max_a Q_t(S_{t+1}, a)), \tag{3}$$

ahol S_t és A_t az aktuális állapot ás az aktuálisan választott akció. Az α_t a tanulási ráta, amely a tanulás gyorsaságát befolyásolja, a γ_t pedig a diszkontráta, amely azt szabályozza, hogy az algoritmus milyen mértékben veszi figyelembe a jelenbeli döntés alapján a jövőbeni állapotok értékeit. Az R_t az aktuális S_t állapotban, A_t akció végrehajtása után kapott jutalom értéke.

A stratégia kezdetben felfedező, ami azt jelenti, hogy a kiválasztás véletlenszerű, majd ahogy egyre jobban kezd konvergálni a megoldás felé, a stratégia fokozatosan átvált egy olyan állapotra, ahol a meglévő, kiszámított adatokat igyekszik pontosítani, azaz már új, addig ismeretlen lépéseket nem tesz. Az alkalmazott stratégiát Boltzmann felfedezési stratégiának nevezzük, amely a kiszámított Q értékek alapján határozza meg annak a valószínűségét, hogy mi legyen a következő kiválasztott elem, esetünkben tevékenység a permutációban. Az algoritmus a kialakított permutáció alapján elvégzi a mohó ütemezést, és összehasonlítja a korábbi eredményekkel. Az új eredmény minősége alapján meghatároz egy R_t jutalmat, amely a (3)-as frissítő szabály része és befolyásolja a Q értékek változását.

2.1.3. Eredmények

A kifejlesztett algoritmus hatékonyságának tesztelésére a [22, 23] cikkekben foglalt kisméretű feladatot oldottam meg, továbbá ugyanezen cikkekben szereplő feladatok paraméterei alapján generáltam feladatosztályokat.

A kisméretű feladat gépi idő adatai rendelkezésre álltak, a feladat optimális megoldása 13. A [22]-ben közölt algoritmus megoldása 15 lett, az általam fejlesztett

QLM algoritmusé pedig 13. A QLM további teszteléséhez a [22]-ben közölt 33 problémából hármat választottam, a [23]-ban közölt feladatok közül pedig egyet. A feladatok [22]-ben használt számozását megtartva a #1, #2 és #5 került kiválasztásra. Ezek kis méretű feladatok. A [23]-ből pedig egy nagy méretű feladatot választottam, az eredeti számozás szerint a #28-at. Ezen adatok alapján négy osztályt hoztam létre. Az n a tevékenységek, m a gépek és NC a megelőzési relációk száma.

- $Class\ \#1 \rightarrow n = 14$, $m = 8$ és $NC = 5$
- $Class\ \#2 \rightarrow n = 28$, $m = 7$ és $NC = 8$
- $Class\ \#3 \rightarrow n = 27$, $m = 4$ és $NC = 1$
- $Class\ \#4 \rightarrow n = 74$, $m = 19$ és $NC = 10$

Megmutattam, hogy a QLM eljárás alkalmas az ebben a fejezetben bemutatott ütemezési probléma megoldására. Sikerült megmutatni, hogy a kifejlesztett eljárás hatékony a felvetett ütemezési probléma megoldásában. Az algoritmus kifejezetten azokra az ütemezési problémákra lett kifejlesztve, ahol a tevékenységek végrehajtási ideje az erőforrástól függ, a végrehajtás nem megszakítható, továbbá az egyes tevékenységek között megelőzési relációk lehetnek. Az általam megadott probléma megoldásával a megerősítéses tanulás témakörében nem találtam publikációt. Összehasonlítási alap lehetett volna a [22]-ben és a [23]-ban közölt megoldás, de az itt megoldott feladatok részletei nem ismertek. Így alapvetően saját magam által generált feladatokkal teszteltem a QLM algoritmust, továbbá ugyanezeket a feladatokat a CPLEX-szel is megoldottam. Az eredmények alapján látható, hogy a QLM minden esetben, amikor a CPLEX is, megtalálta az optimális megoldást. A többi esetben nem tudjuk biztosan, hogy a QLM optimális megoldást talált, ezt a CPLEX nem tudta megerősíteni. Ez alapján látható, hogy a kidolgozott feladatokra szorítkozva, a QLM algoritmus hatékony és a feladatok megoldásában felveszi a versenyt a CPLEX megoldójával, hiszen a QLM által adott eredmények legalább olyan jók, mint a CPLEX által szolgáltatott eredmények, de a QLM a CPLEX-nél sokkal gyorsabb.

2.2. Ládapakolás

2.2.1. A ládapakolási probléma

A ládapakolási feladatok esetében tárgyakat pakolunk ládákba. Minden tárgy rendelkezik egy w_i mérettel, és minden láda egy C kapacitással. A pakolást úgy végezzük el, hogy a ládákba pakolt tárgyak összmérete ne lépje túl a láda kapacitását, valamint minimális számú ládát használjunk fel. Ez a probléma NP-nehéz.

2.2.2. A javasolt módszer

Az előfeldolgozás célja az, hogy a feladatosztályokban szereplő feladatok minél nagyobb részét egyszerű algoritmussal optimálisan oldjam meg. A Schwerin és a Falkenauer_U benchmark osztályok megoldására két mohó algoritmust hoztam létre

REM SW és FU néven. A két algoritmus kifejezetten erre a két feladatosztályra lett kifejlesztve. A REM SW algoritmus kiválasztja a $0 \leq k \leq 6$ értékét úgy, hogy a k legnagyobb tárgy és a $6 - k$ legkisebb tárgy egy ládába kerüljön. A k legnagyobb tárgy pakolása után a fennmaradó helyet a $6 - k$ tárggyal a lehető legjobban betöltse. Ez utóbbihoz az útkereső segédalgoritmust alkalmazza. Az FU algoritmus négy segédalgoritmussal dolgozik, amelyek párokat, hármasokat és négyeseket pakolnak úgy, hogy figyelembe vesznek egy r értéket, amely a ládában fennmaradó hely (tartalék) alsó korlátja. A fennmaradó tárgyakat az algoritmus az utolsó lépésben az FFD algoritmussal pakolja.

Mindkét feladatosztály esetében megvizsgáltam, hogy milyen tulajdonságokkal rendelkeznek a bennük foglalt feladatok, és a megoldás szempontjából ezekből milyen előnyöket lehet kovácsolni. A felfedezett tulajdonságok alapján a Schwerin osztály minden feladatát, a Falkenauer_U osztály feladatainak pedig 91%-át sikerült optimálisan megoldani. Az eredmények alapján látható, hogy az előfeldolgozás a vizsgált feladatosztályok esetében valóban hatékonyan segítették az optimális megoldás megtalálását.

2.2.3. Eredmények

A Schwerin és a Falkenauer_U osztályokra szorítkozva kijelenthető, hogy a kidolgozott algoritmusok nagyon jó eredményeket értek el. A Schwerin osztály esetében az 200 darab feladatból mindet optimálisan megoldotta a REM SW algoritmus, azaz az optimális megoldások aránya 100%. Az FU algoritmus is nagyon szépen teljesített a Falkenauer_U osztály feladatain, ugyanis a 80 darab feladatból 73 esetében adott optimális megoldást, amely 91%-os arányt jelent az optimális megoldásokra nézve.

A bemutatott mohó eljárások gyorsak, egyszerűek és a legtöbb esetben megtalálják az optimális megoldást. Összetettebb algoritmus alkalmazása csak abban az esetben indokolt, ha az egyszerűbb módszerek nem adnak optimális vagy elfogadhatóan jó megoldást.

Az ismertetett algoritmusok (Rem SW és FU) általános alkalmazhatósága eltérő. A Schwerin algoritmusa kihasználja, hogy a tárgyméretek nagyon közel vannak egymáshoz és minden ládában 5 vagy 6 db tárgy van. Viszont az algoritmus alkalmazható olyan esetben is, amikor továbbra is 150 és 200 között vannak a tárgyméretek, de a ládaméret nem 1000, hanem például 1200. Ekkor minden ládába 6 vagy 7 tárgy fog kerülni. (Csak akkor kerülhet 8 tárgy, ha ezek mindegyikének a mérete 150, ennek azonban nagyon kicsi az esélye.)

A Falkenauer algoritmusa általánosan is jó lehet, ha 0 és C között van a tárgyak mérete. Párokat pakolunk, utána hármasokat, majd négyeseket. Ezeket úgy, hogy jól megtöltik a ládát. A maradékot pedig FFD-vel.

2.3. Ládafedés

2.3.1. A ládafedési probléma

A ládafedési probléma hasonló a ládapakolási problémához, azonban tartalmaz néhány kiegészítést. A tárgyak egyenként érkeznek. Sorrendben az i. tárgy mérete

$w_i > 0$ és feltételezzük, hogy végtelen számú láda áll rendelkezésre ugyanazzal a C kapacitással. Továbbá adott egy $K > 0$ pozitív egész szám, amely megadja, hogy egyszerre hány láda lehet nyitva. Azaz, a pakolást végző algoritmus csak akkor nyithat új ládát, ha a nyitott ládák száma kevesebb, mint K.

Egy ládát fedettnek tekintünk, ha a ládába pakolt tárgyak összmérete legalább a C kapacitással egyenlő. Adott továbbá egy G célfüggvény is, amelyre

$$G : \{1, \ldots, K\} \to R^+. \tag{4}$$

Ha adott időpillanatban $1 \leq k \leq K$ darab láda van nyitva, és egy láda fedetté válik és elszállításra kerül, akkor a realizált profit $G(k)$. Minden fedetté vált és elszállított láda után a k értéke eggyel csökken, de bármikor nyithatunk új ládát is, feltéve, hogy nem lesz több nyitott láda, mint K. A G függvény monoton csökkenő, pozitív értékű függvény. A cél a profit maximalizálása, amelyet a lezárt és elszállított ládák után kapunk.

2.3.2. A javasolt módszer

A vizsgálat során már létező, természetesen adódó algoritmusokat implementáltam (DNF, H(K), SH(K)), továbbá megalkottam egy új, paraméteres algoritmust MMask néven. Az algoritmusok teszteléséhez, hasonlóan a ládapakolási témakörhöz, a Schwerin és a Falkenauer_U feladatosztályokat használtam fel, ezen kívül pedig létrehoztam egy új feladatosztályt is Large Range néven. Első lépésként a feladatokat előkészítettem az alábbiak szerint:

(a) **Rendezettség megbontása.** A tárgyakat méretük alapján véletlenszerűen összekevertem.

(b) **Falkenauer osztály normalizálása.** A láda kapacitását $C = 1000$-nek vettem, emiatt minden tárgy méretét $\frac{1000}{150}$ értékkel kellett megszorozni.

(c) **Új feladatosztály.** Létrehoztam egy új osztályt, ahol a tárgyak méretei az $[1, 1000]$ intervallumból valók. Ebben az osztályban összesen 400 darab feladat található, mindegyik esetében a tárgyak száma legfeljebb 1000.

(d) **Nyereségfüggvények.** Az eredeti benchmarkok esetében nincs megadva nyereségfüggvény. Emiatt három függvényt hoztam létre:

- $G1(k) = 10,1 - 0,1 \times k$
- $G2(k) = 11 - k$
- $G3(k) = 10,05 - 0,05 \times k^2$

(e) **Tárgytípusok és nyereségfüggvények összekapcsolása.** Végezetül, a feladatosztályokat kombináljuk a három nyereségfüggvénnyel az alábbi jelöléseket alkalmazva:

- $SiGu$

- $FjGu$
- $LRjGu$

ahol $i = 1, 2$, $j = 1, 2, 3, 4$ és $u = 1, 2, 3$.

Már ismert, természetesen adódó algoritmusok, amelyeket implementáltam a DNF, H(K), SH(K). Mindegyik algoritmus működése nagyon egyszerű:

- **DNF:** A következő tárgy mindig az aktuálisan nyitott ládába kerül. Ha a láda fedetté válik, akkor az algoritmus lezárja és nyit egy újat a következő tárgyak számára. Ha nincs több tárgy, az algoritmus megáll.
- **H(K):** Azok a tárgyak, amelyeknek a mérete az $I_k = \left(\frac{1}{k+1}, \frac{1}{k}\right]$ intervallumba esik, azokat a k. típusú ládába pakolja az algoritmus, ahol $k = 1, \ldots, K-1$. A legkisebb méretű tárgyak, azaz amelyek mérete az $I_K = \left(0, \frac{1}{K}\right]$ intervallumba esik, azok a K. típusú ládába kerülnek. Ha valamely láda fedetté válik, az algoritmus lezárja. Ha nincs ilyen láda nyitva, akkor nyit egy ilyen típusút. Ha nincs több elem, akkor az algoritmus megáll.
- **SH(K):** Ha a következő tárgy képes lefedni egy vagy több ládát, akkor abba a ládába kerül, amelynek a legalacsonyabb a töltöttsége. Ezután az algoritmus lezárja a ládát. Egyéb esetben az SH(K) algoritmus a H(K) algoritmus szabálya szerint működik.

Az újonnan kifejlesztett algoritmus az MMask, amelynek három paramétere van: K az egy időben nyitott ládák maximális számát meghatározó pozitív egész szám, az α egy K-dimenziós nemnegatív vektor és β egy pozitív egész szám. Az MMask működésének alapja egy elfogadó-elutasító politika. Az algoritmus elfogadja a soron következő tárgy pakolását, ha a pakolás után az adott láda töltöttsége az elfogadó tartományba esik. Ellenkező esetben elutasítja a pakolást.

- Elfogadó tartomány a k. láda esetén ($1 \leq k \leq K$): $[0; C - \alpha_k] \cup [C; C + \beta]$
- Elutasító tartomány a k. láda esetén ($1 \leq k \leq K$): $(C - \alpha_k; C) \cup (C + \beta; \infty)$

MMask: Ha a következő tárgy egy olyan ládába helyezhető, ami ezáltal az elfogadó tartományban fedetté válik, akkor az aktuális tárgyat ebbe a ládába kell pakolni. Ha több ilyen láda is van, akkor ezek közül a legkisebb töltöttségűbe. Ezután az algoritmus lezárja a ládát, majd vagy leáll (ha nincs több tárgy) vagy újraindul. Ha a következő tárgy egy olyan ládába helyezhető el, aminek a töltöttsége az elfogadó tartományba esik a pakolás után, de nem lesz fedett, akkor ebbe a ládába kell pakolni. Ha több ilyen is van, az algoritmus véletlenszerűen választ egyet. Ezután vagy leáll (ha nincs több tárgy) vagy újraindul. Ha $k < K$, akkor az algoritmus nyit egy új ládát. A láda típusa a legkisebb k érték, amelyikre nincs ilyen típusú nyitott láda, és ide pakolja a tárgyat. Ezután vagy leáll (ha nincs több tárgy) vagy újraindul. Ha $k = K$ teljesül, akkor az aktuális tárgy a legkisebb töltöttségű ládába kerül. Ha a láda fedett lesz, az algoritmus lezárja. Ha nincs több tárgy, az algoritmus leáll, egyébként újraindul.

Az MMask algoritmus paramétereinek beállításáért egy lokális keresésen alapuló eljárás felelős. A szomszédsági struktúra a különböző MMask paraméter beállítások között természetes módon definiálható. Egy konkrét paraméter beállítás szomszédját úgy kapjuk meg, hogy a K, α, β paraméterek közül egyet módosítunk. Legyen Δ egy kis pozitív konstans, ekkor minden α_i, β paraméter a Δ értékével lesz növelve vagy csökkentve úgy, hogy az új érték pozitív marad és kisebb lesz, mint a ládaméret. A K értéke is változik 1 egységgel negatív vagy pozitív irányba. A változás iránya véletlenszerű.

2.3.3. Eredmények

A természetesen adódó algoritmusok eredményei alapján az S1 alosztály esetében mindegyik algoritmus (DNF, H(K) és SH(K)) ugyanazokat az eredményeket adta. K értékét hiába növeltem, az elért profitot (160) nem tudtam növelni. Ez azt jelenti, hogy a S1 esetében a $K > 1$ beállítás esetén is az algoritmusok DNF-ként viselkedtek. Az F1 és F4 esetében az SH(K) algoritmus érte el a legmagasabb profitot $K = 2, 3, 4, 5$ értékekre. Az LR1 és LR4 osztályok esetében az SH(K) algoritmus $K = 4$ beállítása mellett adta a legjobb profit értékeket, két esetben $K = 5$ beállítással javított a $K = 4$ beállításhoz képest.

Az MMask algoritmus a paraméterek manuális beállítása mellett az S1 esetében egy feladatnál sem javított a profiton, az F1 és F4 esetében a vizsgált feladatok mindegyikénél javított a profit értékén, az LR1 és LR4 feladatosztály esetében pedig három esetet leszámítva ugyancsak javított az eredményeken.

A lokális kereséssel megválasztott paraméter beállításokat alkalmazva az F1 és F4 osztályok feladatai esetében egy esetet kivéve mindenhol javított a korábbi algoritmusokhoz és a manuális beállításhoz képest is. Az LR1 és LR4 osztályok esetében három esetet leszámítva ugyancsak mindenhol javított.

A vizsgálatok alapján az mondható, hogy a lokális keresés segítségével egy "jó paraméter-beállítás" megtalálható. Természetesen, fejlettebb eljárásokkal valószínűleg még jobb beállításokat lehetne elérni, azonban ez egy jövőbeni továbbfejlesztési lehetőség jelenleg.

A jobb paraméter-beállítások megtalálásához jól alkalmazható a lokális keresés, azaz képes megoldani a paraméterek automatikus keresését úgy, hogy a korábbi algoritmusok és a manuálisan beállított paraméterek eredményeitől jobb eredményeket ér el. Ez nagy könnyebbség, hiszen a paraméterek kézi beállítását kiváltja. Továbbá, az MMask és a lokális keresés, mint paraméter optimalizáló fúziója hatékonynak bizonyult.

3. Új tudományos eredmények

1. Tézis *A független gépek ütemezése megelőzési relációkkal ($R_m|prec|C_{max}$) feladat megoldására létrehoztam egy új algoritmust. A kidolgozott algoritmus a megerősítéses tanulás területéről ismert Q-tanuláson alapuló eljárással meghatározza a tevékenységek sorrendjét, majd egy mohó ütemezővel ebben a sorrendben ütemezi a feladatokat. Az algoritmus átfogó vizsgálatára négy új benchmark feladatosztályt hoztam létre. A feladatosztályok a felépítésük és nehézségük alapján változatosak.*

1.1 A kapott eredményeket összehasonlítottam a CPLEX megoldó által szolgáltatott eredményekkel. A kiértékelés alapján látható, hogy az általam létrehozott QLM algoritmus minden olyan esetben megtalálja az optimális megoldást a vizsgált feladatosztályokban, amikor a CPLEX is. A többi esetben a QLM legalább olyan jó eredményeket produkált mint a CPLEX. A feladatosztályokra szorítkozva kijelenthető, hogy a QLM algoritmus hatékony és a feladatok megoldásában felveszi a versenyt a CPLEX megoldóval, de annál sokkal gyorsabb.

A tézispontban felsorolt eredményeket alátámasztó publikációk: [P1], [P4], [P5], [P6], [P7], [P8], [P9]

2. Tézis *A ládapakolási problémát egy új szemszögből, az előfeldolgozás oldaláról közelítettem meg. A vizsgálathoz a Schwerin és a Falkenauer_U osztályokat alkalmaztam. Létrehoztam két algoritmust REM SW és FU néven. A REM SW algoritmust a Schwerin, az FU algoritmust pedig a Falkenauer_U osztály feladatainak megoldására terveztem.*

2.1 Mindkét feladatosztály esetében megvizsgáltam, hogy milyen tulajdonságokkal rendelkeznek a bennük foglalt feladatok, és a megoldás szempontjából ezekből milyen előnyöket lehet kovácsolni. A felfedezett tulajdonságok alapján a Schwerin osztály minden feladatát, a Falkenauer_U osztály feladatainak pedig 91%-át sikerült optimálisan megoldani.

2.2 Az algoritmusokat nem csak aszerint vizsgáltam meg, hogy az adott feladatosztály hány elemére találnak optimális megoldást, hanem összehasonlítottam őket a HEA algoritmussal. A HEA algoritmus az egyik leghatékonyabb (és az egyik legújabb) általános metaheurisztikus algoritmus ezen és más benchmark feladatok megoldására. A REM SW algoritmust összehasonlítva a HEA [24] algoritmussal, az átlagos futási idők a REM SW esetében sokkal jobbak voltak. A Schwerin 1 esetén 0,0038 (HEA - 0,34 másodperc) másodperc, míg a Schwerin 2 esetén 0,004 (HEA - 0,47 másodperc) másodperc volt. Az FU algoritmus futási ideje a Falkenauer_U osztály 80 feladata esetén is jelentősen kisebb, mint a HEA futási ideje.

A tézispontban felsorolt eredményeket alátámasztó publikáció: [P2]

3. Tézis *Részletes vizsgálatokat folytattam a "ládafedés szállítással" nevű feladattal kapcsolatban, amelynek részeként a Schwerin és a Falkenauer_U osztályokat alkalmaztam, valamint új benchmark osztályokat (LR) is létrehoztam. Megalkottam egy új, paraméteres algoritmust MMask néven, amely a ládák pakolása során egy elfogadó-elutasító politikát folytat, továbbá a paramétereit lokális kereséssel automatikusan beállítja.*

3.1 A benchmark feladatok megoldhatóságát részben az irodalomból ismert, természetesen adódó algoritmusokkal (DNF, H(K), SH(K)) vizsgáltam. Az eredmények kiértékelése alapján megállapítottam, hogy ezen algoritmusok az esetek döntő többségében hatékonyan oldják meg a feladatosztályok elemeit.

3.2 Továbbá, megállapítottam, hogy a paraméterek megfelelő megválasztásával az MMask algoritmus a korábbiaknál jobb eredményeket tudott elérni. A paraméterek optimalizálásához létrehoztam egy lokális keresés alapú heurisztikát, amely a legjobb eredményeket elérő paraméter beállításokat határozta meg. Az optimalizáló még jobb beállításokat tudott megtalálni, mint a manuális beállítások.

A tézispontban felsorolt eredményeket alátámasztó publikációk: [P3]

4. A tézispontokkal kapcsolatos publikációk

Az ismertetett eredményeim folyóiratokban, nemzetközi konferenciákon és konferenciakiadványokban lettek bemutatva. Ezeket az alábbiakban sorolom fel.

[P1] **Gy. Ábrahám**, P. Auer, Gy. Dósa, T. Dulai and Á. Werner-Stark. A reinforcement learning motivated algorithm for process optimization. Periodica Polytechnica Civil Engineering, 63(4):961 970, 2019. (**IF: 1.34**)

[P2] **Gy. Ábrahám**, Gy. Dósa, T. Dulai, Zs. Tuza and Á. Werner-Stark. Efficient Pre-Solve Algorithms for the Schwerin and the Falkenauer_U Bin Packing Benchmark Problems for Getting Optimal Solutions with High Probability. Mathematics, 9(13):1540, 2021. (**IF: 2.592**).

[P3] **Gy. Ábrahám**, P. Auer, Gy. Dósa, T. Dulai, Zs. Tuza and Á. Werner-Stark. The bin covering with delivery problem, extended investigations for the online case. Central European Journal of Operations Research, pages 1-27, 2022. (**IF: 2.345**)

[P4] Á. Werner-Stark, T. Dulai and **Gy. Ábrahám**. Modeling of an agent system to support the management of cooperating and rival resources for business workflows. In 2014 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH), pages 407-412. IEEE, 2014.

[P5] T. Dulai, Á. Werner-Stark and **Gy. Ábrahám**. Support of Efficient resource allocation of technological processes by a heuristic solution and agent technology. Data Envelopment Analysis and its Applications, page 153, 2016.

[P6] T. Dulai, Á. Werner-Stark and **Gy. Ábrahám**. Improvement of resource allocation in workflows by stochastic method. In 10th International Conference on Applied Informatics, 2017.

[P7] **Gy. Ábrahám**, T. Dulai, Á. Werner-Stark and Gy. Dósa. Reinforcement learning in process scheduling. In 13th Miklós Iványi International PhD DLA Symposium - Abstract Book: Architectural, Engineering and Information Sciences, pages 15-15, 2017.

[P8] **Gy. Ábrahám**, T. Dulai, Á. Werner-Stark and Gy. Dósa. Learning the parameters of a reinforcement learning algorithm for process optimization. In Veszprém Optimization Workshop, pages 11-11, 2019.

[P9] **Gy. Ábrahám**, T. Dulai, Á. Werner-Stark and Gy. Dósa. Parameter optimization of q-learning motivated algorithm in process scheduling. In Proceedings of the Pannonian Conference on Advances in Information Technology, pages 17-24, 2020.

A felsorolt publikációk a dolgozat benyújtása napjáig a következő hivatkozásokat kapták: [25], [26]

5. További kutatási és továbbfejlesztési lehetőségek

5.1. Az ütemezési problémára vonatkozóan, a bemutatott megoldásokon túl további vizsgálat tárgya lehet, hogy bizonyos speciális esetekben (például $m = 2$ gép esete, vagy csak kétfajta végrehajtási idő esete) nem kaphatnánk-e jobb alsó korlátokat illetve, hogy működik ezekben az esetekben a QLM algoritmus vagy ennek valamilyen módosított változata. Továbbá meg lehet vizsgálni, hogy némileg más struktúrák esetén pl. fa struktúra, hogyan működik a módszerünk. Természetesen ezekre is új tesztfeladatokat kellene generálnunk.

5.2. A ládapakolásra vonatkozóan néhány további lehetőséget mutatok be, amelyekkel a jövőben továbbfejleszthető az FU algoritmus. A Schwerin osztály esetében, mivel sikerült megoldani az összes feladatot optimálisan, így nincs szükség további kutatásra. Azonban a Falkenauer_U osztály esetében az arány 91% volt, így itt van helye további vizsgálatoknak:

1. Más paraméterek kipróbálása, amelyek meghatározása lehet manuális, tapasztalaton alapuló, vagy automatizált, valamilyen kereső algoritmussal. A különböző beállítások által adott eredmények közül végül kiválasztjuk a legjobbat.

2. További paraméterek bevezetése, pl. nem csak a nagy tárgyak meglétét vizsgáljuk, hanem azok darabszámát is figyelembe vesszük.

Az is megválaszolatlan kérdés jelenleg, hogy mi történik, ha a tárgyak száma növekszik. Tételezzük fel, hogy a láda kapacitása, azaz a C értéke egy rögzített egész, így következésképpen a tárgyak mérete az $[1, C]$ intervallumból kerül ki. Ekkor Lenstra eredménye szerint [27] a feladat polinomiális időben megoldható, ahol n a tárgyak száma és n tetszőlegesen nagy lehet. A polinomiális időben való megoldhatóság ellenére egy ilyen algoritmus lépéseinek száma nagyon nagy lenne, mivel n kitevője nagyon nagy, továbbá az $O(.)$ kifejezés együtthatója is szintén nagy lenne. Ezen okok miatt egy ilyen algoritmus nem biztos, hogy használható lenne a gyakorlatban, esetleg valamilyen előszűrés után.

Továbbá, ez a tétel nem veszi figyelembe azt a tényt, ha a tárgyak mérete véletlenszerűen kerül ki egy adott intervallumból egyenletes eloszlással. Ezek alapján megfogalmazhatjuk a következő sejtést.

Legyenek $1 \leq a < b \leq C$ rögzített egész számok, ahol C a láda mérete. Tételezzük fel, hogy a tárgyak mérete véletlenszerűen, egyenletes eloszlás mellett az $a, a+1, \ldots, b$ egészek közül kerülnek ki. Ekkor létezik egy olyan algoritmus, amelynek futási ideje alacsony rendű polinommal felülről becsülhető és az $O(.)$ kifejezés együtthatója

is megfelelően kicsi, továbbá a feladat optimális megoldásának valószínűsége 1-hez közelít midőn $n \to \infty$.

Például, egy olyan algoritmus, amelynek futási ideje $20n^4$ és 0,9 valószínűséggel találja meg az optimális megoldást $n = 1000$ mellett, már érdekes lehet. Az előbbi várakozásomat azért fogalmaztam meg sejtésként, mert valójában keveset tudunk arról, hogy ilyen esetekben mit lehet csinálni. Ehhez további vizsgálatokra lenne szükség.

5.3. A ládafedés területén néhány, az MMask algoritmus továbbfejleszthetőségére vonatkozó lehetőséget fogalmazok meg.

- A bemutatott paraméter-beállítások egy része manuálisan került meghatározásra. Előfordulhat, hogy más beállításokkal sokkal jobb eredmények is elérhetők.

- Kiderült, hogy az LR osztály esetében az SH(4) algoritmus több esetben is nagyon hatékony. Ez alapján megfontolandó az SH(4) és az MMask algoritmus fúziója (mindkét algoritmust külön-külön futtatjuk és a jobb eredményt választjuk).

- Az MMask algoritmus a 2. lépésében tetszőlegesen választhatott a rendelkezésre álló, megfelelő ládák közül. Lehetséges, ha "okosabban" választ ládát, akkor az MMask teljesítménye növelhető lesz.

- Egy másik módosítási lehetőség az algoritmus 2. lépésére vonatkozóan a következő volt. Legyenek $B_1, B_2, \ldots, B_t$ azok a ládák, amelyekbe az aktuális tárgy pakolható. Ha $t = 1$, akkor egyértelmű, hogy melyik ládába került az aktuális tárgy, mivel csak egy láda van nyitva. Ha viszont $t > 1$, akkor több láda közül választhat az algoritmus. Minden megfelelő ládára kiszámítottam, hogy mennyi a bennük lévő tárgyak méretének átlaga, ezeket jelölje $x_1, x_2, \ldots, x_t$, az aktuális tárgy méretét pedig x. Ekkor minden láda esetében megadtam az $(x - x_i)^2$ kifejezés értékét, és a legkisebbet választottam, ami megadta, hogy a B_i ládába kell pakolni, ahol $1 \leq i \leq t$. Sajnos ez a választási stratégia nem bizonyult hatékonynak. Néhány esetben javított az eredményen, máskor rontott, de összességében nem lett jobb az eljárás.

Hivatkozások

[1] M. L. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Springer Publishing Company, Incorporated, Boston, MA, USA, 4th edition, 2012.

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

[3] R. L. Graham. Bounds on multiprocessing timing anomalies. *Siam Journal on Applied Mathematics*, 17(2):416–429, 1969.

[4] A. I. Orhean, F. Pop, and I. Raicu. New scheduling approach using reinforcement learning for heterogeneous distributed systems. *Journal of Parallel and Distributed Computing*, 117:292–302, 2018.

[5] M. E. Aydin and E. Öztemel. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2):169–178, 2000.

[6] P. Stefan. Flow-shop scheduling based on reinforcement learning algorithm. *Production Systems and Information Engineering*, 1(1):83–90, 2003.

[7] P. Stefan. *Combined Use of Reinforcement Learning And Simulated Annealing: Algorithms and Applications*. PhD thesis, University of Miskolc, Miskolc, 2003.

[8] T. Gabel and M. Riedmiller. Adaptive reactive job-shop scheduling with reinforcement learning agents. *International Journal of Information Technology and Intelligent Computing*, 24(4):14–18, 2008.

[9] J. Shahrabi, M. A. Adibi, and M. Mahootchi. A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Computers & Industrial Engineering*, 110:75–82, 2017.

[10] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pages 1114–1120. Citeseer, 1995.

[11] M. Zweben, E. Davis, B. Daun, and M. J. Deale. Scheduling and rescheduling with iterative repair. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1588–1596, 1993.

[12] Z. Huang, W.M.P. van der Aalst, X. Lu, and H. Duan. Reinforcement learning based resource allocation in business process management. *Data & Knowledge Engineering*, 70(1):127–145, 2011.

[13] Y. Ye, X. Ren, J. Wang, L. Xu, W. Guo, W. Huang, and W. Tian. A new approach for resource scheduling with deep reinforcement learning. *CoRR*, abs/1806.08122, 2018.

[14] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, volume 174. W. H. Freeman and Co., San Francisco, 1979.

[16] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[17] A. Benkő, Gy. Dósa, and Zs. Tuza. Bin packing/covering with delivery, solved with the evolution of algorithms. In *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, pages 298–302. IEEE, 2010.

[18] A. Benkő, Gy. Dósa, and Zs. Tuza. Bin covering with a general profit function: approximability results. *Central European Journal of Operations Research*, 21(4):805–816, 2013.

[19] Gy. Dósa and Zs. Tuza. Bin packing/covering with delivery: Some variations, theoretical results and efficient offline algorithms. *arXiv preprint arXiv:1207.5672*, 2012.

[20] G. Galambos and G. J. Woeginger. On-line bin packing—a restricted survey. *Zeitschrift für Operations Research*, 42(1):25–45, 1995.

[21] J. Csirik and G. J. Woeginger. On-line packing and covering problems. *Online Algorithms*, pages 147–177, 1998.

[22] J. Herrmann, J. Proth, and N. Sauer. Heuristics for unrelated machine scheduling with precedence constraints. *European Journal of Operational Research*, 102(3):528–537, 1997.

[23] C. Liu and S. Yang. A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints. *J. Softw.*, 6:1146–1153, 2011.

[24] I. Borgulya. A hybrid evolutionary algorithm for the offline bin packing problem. *Central European Journal of Operations Research*, 29(2):425–445, 2021.

[25] B. M. Kayhan and G. Yildiz. Reinforcement learning applications to machine scheduling problems: a comprehensive literature review. *Journal of Intelligent Manufacturing*, pages 1–25, 2021.

[26] T. Wagner. *Petri Net-based Combination and Integration of Agents and Workflows*. PhD thesis, Staats-und Universitätsbibliothek Hamburg Carl von Ossietzky, 2017.

[27] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534, 1982.