Nehéz, kombinatorikus optimalizálási feladatok számítógéppel segített megoldása

DOI:10.18136/PE.2022.829

Doktori (PhD) értekezés

Készítette: Ábrahám Gyula

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

Pannon Egyetem
Műszaki Informatikai Kar
Informatikai Tudományok Doktori Iskola

2022

Nehéz, kombinatorikus optimalizálási feladatok számítógéppel segített megoldása

Az értekezés doktori (PhD) fokozat elnyerése érdekében készült a Pannon Egyetem Informatikai Tudományok Doktori Iskolája keretében

informatikai tudományok tudományágban

Írta: Ábrahám Gyula

Témavezető/i: Dr. Dósa György, Starkné dr. Werner Ágnes

Elfogadásra javaslom: (igen / nem)

.....
Dr. Dósa György

Elfogadásra javaslom: (igen / nem)

.....
Starkné dr. Werner Ágnes

Az értekezést bírálóként elfogadásra javaslom:

Bíráló neve: ..... igen / nem

.....
bíráló

Bíráló neve: ..... igen / nem

.....
bíráló

A jelölt az értekezés nyilvános vitáján .....%-ot ért el.

Veszprém, ...
a Bíráló Bizottság elnöke

A doktori (PhD) oklevél minősítése .....

Veszprém, ...
az EDHT elnöke

Tartalmi kivonat

Az értekezésben három olyan problémával foglalkoztam, amelyek az ütemezés és a ládapakolás területéhez tartoznak. Mindhárom területnek számos alkalmazása van a gyakorlatban, többek között az iparban, gazdasági életben vagy éppen optimalizálásban. Az ütemezési feladat megoldásában egy, a megerősítéses tanulás területén ismert és népszerű algoritmust vettem alapul. A ládapakolási feladatok megoldása során ún. előfeldolgozó algoritmusok segítségével igyekeztem megoldani benchmark feladatokat. Továbbá egy viszonylag új területtel is foglalkoztam, amelynek ládafedés szállítással (angolul: *Bin Covering with Delivery*, röviden *BCD*) a neve. Ezen a területen természetesen adódó algoritmusokkal oldottam meg a benchmark feladatokat, valamint bemutattam egy új, rugalmas algoritmust is. Mindhárom feladat meglehetősen nehéz, bonyolult kombinatorikus optimalizálási feladat.

A 2. fejezetben egy nehéz ütemezési feladattal foglalkoztam, amely a független gépek ütemezése megelőzési relációkkal. Ez egy klasszikus ütemezési probléma, ahol az egyes feladatok között megelőzési relációk vannak és a feladatok végrehajtási ideje a hozzárendelt erőforrástól függ. (Megelőzési reláción a következőt értjük: ha az i. munka megelőzi a j. munkát, akkor a j. munka végrehajtása csak akkor kezdődhet el, ha az i. munka végrehajtása már befejeződött. Az i. és a j. munkák akár különböző gépeken is végrehajthatóak.) Az ütemezési feladat megoldásában a megerősítéses tanulás területéről ismert Q-tanulást alkalmaztam. Az algoritmus célja egy olyan feladatsorrend kialakítása a megelőzési relációkat figyelembe véve, hogy ebben a sorrendben ütemezve a feladatokat az LS algoritmus által, a teljes átfutási idő minél kisebb legyen. Az eredmények alapján sikerült hatékony algoritmust előállítani.

A 3. fejezetben bizonyos típusú ládapakolási feladatok mohó módszerekkel történő megoldásával foglalkoztam. Előbb bizonyos fajta előfeldolgozást hajtunk végre, amelyek a probléma egyes tulajdonságait kihasználva egyszerűsítik a megoldást úgy, hogy az optimalitás nem sérül, vagyis továbbra is lehetséges optimális megoldást kapnunk. Azaz, ha például van 120 pakolandó tárgy, amelyből valamilyen tulajdonságot felhasználva rögtön 60 tárgy pakolható "gondolkodás" és összetett eljárások nélkül, akkor máris felére csökkent azon tárgyak száma, amelyeket ügyesen, óvatosan kell pakolni további ládákba. A vizsgált feladatosztályok a Schwerin és a Falkenauer voltak. A megmaradt tárgyak pakolására mohó algoritmusokat fejlesztettem ki. Összefoglalva, mindkét feladatosztályra mohó algoritmusokat fejlesztettem, amelyek a Schwerin osztály esetén minden feladat, a Falkenauer osztály esetében pedig a feladatok 91%-a esetén találtak optimális megoldást, és gyorsan.

A 4. fejezetben egy új feladat került definiálásra, amely egy bonyolult ládapakolási feladat egy bizonyos célfüggvénnyel kombinálva. A megoldás során korábbi, természetesen adódó algoritmusok kerültek kifejlesztésre és vizsgálatra a megadott

benchmark példákon. Megjegyezzük, hogy új benchmark osztályok (LR) is definiálásra kerültek. Továbbá, bemutattam egy új, flexibilis algoritmuscsaládot. Az algoritmus paramétereinek automatikus beállítása paraméter optimalizálással történt. Az új algoritmus a vizsgált feladatosztályokon jó eredményeket ért el.

Abstract

In the thesis three problems were considered belong to the area of scheduling and bin packing. All three areas have many applications in practice, including industry, economics or optimization. To solve the scheduling problem, I used an algorithm known and popular in the field of reinforced learning. During the solution of the bin packing problems, I used the so-called preprocessing algorithms to solve the benchmarks. I also dealt with a relatively new area called Bin Covering with Delivery (BCD for short). In this field, I solved the benchmark problems with already known algorithms, and I also presented a new, flexible algorithm. All three problems belong to the field of combinatorial optimization and finding their optimal solution is difficult.

In Chapter 2, I dealt with a difficult scheduling task, which is called unrelated machine scheduling with precedence constraint. This is a classic scheduling problem where there are precedence constraints between tasks and the execution time of the tasks depends on the assigned resource. (Precedence constraint means the following: if task i preceeds task j then the execution of task j can only be started the task i is already finished. Task i and j can be assigned to different machines.) In the solution of the scheduling problems, I used Q-learning known from the field of reinforced learning. The goal of the algorithm is to create a sequence of tasks, taking into account the precedence constraints, so that by scheduling the tasks in this order, the makespan is kept to minimum. Based on the results, it was possible to achieve an efficient algorithm.

In Chapter 3, I dealt with certain preprocessing algorithms for bin packing problems. Preprocessing algorithms simplify the solution by exploiting some properties of the problem so that optimality is not compromised. That is, if, for example, there are 120 items to be packed, of which 60 items can be packed immediately without any "thinking" and complex procedures using some property, the number of items to be examined has already been halved. The examined problem classes were Schwerin and Falkenauer. The remaining items are packed with FFD. I developed greedy algorithms for both problem classes, which solved all problems optimally and quickly for the Schwerin class and 91% of the problems for the Falkenauer class.

In Chapter 4, a new problem was defined, which is a complex bin packing problem combined with a certain objective function. The benchmark instances belong to the Schwerin, Falkenauer, and LR classes (LR was created by me). During the solution, previously known algorithms were implemented, and a new, flexible family of algorithms was introduced. The automatic setting of the parameters of the algorithm was optimized by parameter optimization technique called local search. The new algorithm achieved good results on the examined problem classes.

Auszug

In der Doktorarbeit wurden drei Probleme betrachtet, die zum Bereich Terminplanung und Bin Packing gehören. Alle drei Bereiche haben viele Anwendungen in der Praxis, sei es in der Industrie, in der Wirtschaft oder in der Optimierung. Um das Scheduling-Problem zu lösen, habe ich einen Algorithmus verwendet, der im Bereich des verstärkten Lernens bekannt und beliebt ist. Bei der Lösung der Bin-Packing-Probleme habe ich die sogenannten Preprocessing-Algorithmen zur Lösung der Benchmarks eingesetzt. Außerdem beschäftigte ich mich mit einem relativ neuen Bereich namens Bin Covering with Delivery (kurz BCD). In diesem Bereich habe ich die Benchmark-Probleme mit bereits bekannten Algorithmen gelöst und auch einen neuen, flexiblen Algorithmus vorgestellt. Alle drei Probleme gehören zum Gebiet der kombinatorischen Optimierung.

In Kapitel 2 habe ich mich mit einer schwierigen Scheduling-Aufgabe befasst, die als unabhängiges Maschinen-Scheduling mit Präzedenzbeschränkung bezeichnet wird. Dies ist ein klassisches Planungsproblem, bei dem Prioritätsbeschränkungen zwischen Aufgaben bestehen und die Ausführungszeit der Aufgaben von der zugewiesenen Ressource abhängt. Bei der Lösung der Scheduling-Probleme habe ich das aus dem Bereich des Reinforced Learning bekannte Q-Learning eingesetzt. Das Ziel des Algorithmus ist es, unter Berücksichtigung der Vorrangbeschränkungen eine Abfolge von Aufgaben zu erstellen, so dass durch die Planung der Aufgaben in dieser Reihenfolge die Makespan auf ein Minimum reduziert wird. Basierend auf den Ergebnissen war es möglich, einen effizienten Algorithmus zu entwickeln.

In Kapitel 3 habe ich mich mit bestimmten Vorverarbeitungsalgorithmen für Bin-Packing-Probleme beschäftigt. Vorverarbeitungsalgorithmen vereinfachen die Lösung, indem sie einige Eigenschaften des Problems ausnutzen, sodass die Optimalität nicht beeinträchtigt wird. Das heißt, wenn beispielsweise 120 Artikel zu verpacken sind, von denen 60 Artikel ohne Nachdenken und aufwändige Proze- duren unter Verwendung einiger Eigenschaften sofort verpackt werden können, hat sich die Anzahl der zu untersuchenden Artikel bereits halbiert. Die untersuchten Problemklassen waren Schwerin und Falkenauer. Für beide Problemklassen habe ich Greedy-Algorithmen entwickelt, die alle Probleme optimal und schnell lösen konnten.

In Kapitel 4 haben wir ein neues Problem definiert, nämlich ein komplexes Bin-Packing-Problem in Kombination mit einer bestimmten Zielfunktion. Die Benchmark-Instanzen gehören zu den Klassen Schwerin, Falkenauer und LR (LR wurde von mir erstellt). Bei der Lösung wurden bereits bekannte Algorithmen implementiert und eine neue, flexible Familie von Algorithmen eingeführt. Die automatische Einstellung der Parameter des Algorithmus wurde durch eine als lokale Suche bezeichne-

te Parameteroptimierungstechnik optimiert. Der neue Algorithmus erzielte bei den untersuchten Problemklassen gute Ergebnisse.

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőimnek, Starkné dr. Werner Ágnesnek és dr. Dósa Györgynek támogatásukért, szakmai segítségükért, amely végigkísérte a felkészülésemet, a PhD tanulmányaimat és a disszertáció elkészülését.

Köszönettel tartozom kollégáimnak, hallgatótársaimnak, a Villamosmérnöki és Információs Rendszerek Tanszéknek és a Műszaki Informatikai Karnak a rengeteg támogatásért és segítségért, a barátságos és kellemes környezet biztosításáért. Köszönöm a doktori iskola titkárainak, dr. Dulai Tibornak és dr. Görbe Péternek a segítségét. Végül, de nem utolsó sorban köszönöm norvég jó barátom és hallgatótársam, Tomas Attila Olaj támogatását és biztatását a dolgozat készítése alatt. Tusen takk Tomas!

Szívből hálás vagyok és köszönettel tartozom családomnak biztatásukért, támogatásukért és azért, hogy eljuthattam idáig. A disszertációt édesapám emlékének kívánom szentelni, aki már nem láthatta a dolgozat elkészültét.

Tartalomjegyzék

Ábrák jegyzéke

Táblázatok jegyzéke

Jelölések

Megerősítéses tanulás (2. fejezet)	
s	állapot
a	akció
s'	következő állapot
a'	következő akció
$\mathcal{S}$	összes nem végállapot halmaza
$\mathcal{S}^+$	összes állapot halmaza (végállapotok is)
$\mathcal{A}(s)$	s állapotban elérhető akciók halmaza
$\mathcal{R}$	lehetséges jutalmak halmaza
t	diszkrét időpillanat/epizód
T	egy epizód utolsó időpillanata
S_t	állapot a t pillanatban
A_t	akció a t pillanatban
R_t	jutalom a t pillanatban
G_t	kumulatív jutalom a t pillanat után
π	stratégia
π_*	optimális stratégia
$\pi(a\|s)$	a akció kiválasztásának valószínűsége s állapotban π szerint (sztochasztikus)
$v(s)$	állapotértékelő függvény s állapotra
$v_\pi(s)$	s állapot értéke a π stratégia mellett
$v_*(s)$	s állapot értéke a π optimális stratégia mellett
$q_\pi(s,a)$	a akció kiválasztásának értéke s állapotban a π stratégia mellett
$q_*(s,q)$	a akció kiválasztásának értéke s állapotban a π optimális stratégia mellett
$Q_t(s,a)$	$q_\pi(s,a)$ vagy $q_*(s,a)$ becsült értéke
Q	mátrix a Q-értékek tárolására
γ	diszkontálási paraméter
α	tanulási paraméter
ϵ_t	valószínűségi változó a t. időpillanatban
$\mathbb{E}$	várható érték
$\mathbb{P}$	valószínűségi érték
$\mathcal{P}$	állapotátmenet mátrix
τ	hőmérséklet
p_i	Boltzmann valószínűség az i iterációban
n	tevékenységek száma
m	erőforrások száma

m_i	az i. erőforrás
$task_i$	az i. tevékenység
t_{sum}	a végrehajtási idők összege
$selectedRes$	kiválasztott erőforrás indexe
$\mathcal{M}$	erőforrások (gépek) halmaza
$\mathcal{T}$	tevékenységek halmaza
$\mathcal{I}$	egységmátrix
G	irányított gráf diszjunkt utakkal és izolált pontokkal
L_t	tevékenységek egy permutációjának listája a t. epizód után
H	megelőzési relációban résztvevő tevékenységek indexhalmaza
B	választható tevékenységek indexhalmaza
D	permutációba már beválasztott tevékenységek indexhalmaza
z_t	ütemezés eredményeként kapott átfutási idő a t epizódban
Z	eddigi legjobb ütemezés
R	approximációs arány (legrosszabb eset)
R_m	erőforrások halmaza
C_m	legkésőbb befejeződő tevékenység befejezési ideje
LB_1, LB_2	alsó korlátok
$CPLEXLB$	a CPLEX által kiszámolt alsó korlát
$CPLEXUB$	a CPLEX által kiszámolt felső korlát
NC	megelőzési relációk száma
QLM	a QLM algoritmus által kiszámított megoldás
$QLM - freq$	tíz futásból hányszor találta meg a QLM az optimumot

Ládapakolás (3. fejezet)

n	tárgyak száma
z	célfüggvény
OPT	egy optimális offline algoritmus
A	egy tetszőleges ládapakolási algoritmus
$OPT(L)$	OPT által generált ládák száma
$A(L)$	A által generált ládák száma
LB_1, LB_2, LB_3	alsó korlátok
R_{abs}	abszolút közelítési arány
R	aszimptotikus közelítési arány
L	pakolandó tárgyak listája
C	láda, hátizsák kapacitása
L_b	láda, hátizsák töltöttsége
K	tárgyhalmaz összmérete
k	aktuálisan nyitott ládák száma
n_i	i. csomópont a gráfban
l_i	i. csomópont címkéje
w_i	az i. tárgy súlya (mérete)
g_i	az i. tárgy haszna (nyereség)
x_{ij}	az i. tárgy a j. ládában van-e
y_j	a j. láda használatban van-e
r	tartalékra vonatkozó alsó korlát

res_0	kezdeti tartalék (kihasználatlan helyek a ládákban)
res	a teljes feladatra vonatkozó tartalék

Ládafedés (4. fejezet)	
n	tárgyak száma
C	láda, hátizsák kapacitása
K	nyitott ládák megengedett maximális száma
k	aktuálisan nyitott ládák száma
w_i	az i. tárgy súlya (mérete)
G	célfüggvény
I	bemenet (input)
C_A	az A offline algoritmus eredménye
C^*	offline optimum
ρ	versenyképességi arány
μ	a probléma felső korlátja
S	Schwerin bemenet típus
F	Falkenauer bemenet típus
LR	Large Range bemenet típus
S_iG_u	Schwerin osztály összekapcsolása a G célfüggvénnyel
F_jG_u	Falkenauer osztály összekapcsolása a G célfüggvénnyel
LR_jG_u	Large Range osztály összekapcsolása a G célfüggvénnyel
α	K-dimenziós nemnegatív vektor
β	egy pozitív egész szám
Δ	kis pozitív konstans, amelynek mértékével az α és β paraméterek változnak

1. táblázat: Jelölések jegyzéke

1. fejezet

Bevezetés

Dolgozatomban három olyan feladattal foglalkoztam, amelyek mindegyike az ütemezés elmélet vagy a ládapakolás területéhez tartozik. Ezeken a területeken sok és jelentős alkalmazás van, többek között az iparban, gazdasági folyamatok elemzésében, optimalizálásában és egyéb területeken. Az ütemezési feladatot megerősítéses tanulás alapú algoritmussal oldottam meg. Emiatt az ütemezési feladatokról és a megerősítéses tanulásról adunk az alábbiakban egy bevezető áttekintést. Utána következik majd a bevezetésben egy általános ismertető a ládapakolási feladatokkal kapcsolatban.

1.1. Ütemezés

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 általunk vizsgált esetben a cél a teljes átfutási idő minimalizálása.

Az erőforrásoknak különböző típusai lehetségesek. Egy erőforrás lehet valamilyen gép, feldolgozó egység, ember, valamilyen szoftver, megújuló és nem megújuló erőforrás. A gépek típus szerint lehetnek identikusak (*identical machines*), az ilyen típusú gépek működésükben azonosak, egymás másolatainak is tekinthetők. Továbbá, lehetnek hasonló gépek (*uniform machines*), amelyek ugyanazt a munkát tudják elvégezni, csak a sebességük eltérő. Végül pedig beszélhetünk független gépekről (*unrelated machines*), ahol egy munka elvégzési ideje attól függ, hogy melyik gép fogja elvégezni. Az általunk tárgyalt esetben összesen m darab független gép áll rendelkezésre. Egy gép egyszerre csak egy munkát végezhet, minden munkát el kell végezni és a végrehajtás során a megszakítás nem lehetséges.

A tevékenységek atomi műveletek (amelyek további altevékenységekre nem bonthatók), amelynek végrehajtása az erőforrások segítségével történik. Az egyes tevékenységek általában különböző paraméterekkel rendelkezhetnek, pl. prioritás, a végrehajtás legkorábbi időpontja (*release time*), a végrehajtás befejezésének lehetséges

legkésőbbi időpontja (*due date* vagy *deadline*, a kettő között az a különbség, hogy az első esetben szeretnénk, hogy addig befejeződjön a munka, ha lehet, a másik esetben eddig az időpontig mindenképpen be kell fejezni a munkát). A munkák között lehetnek megelőzési relációk (*precedence constraints*). Továbbá, minden munkának adott a végrehajtási ideje (*processing time*). Az általunk vizsgált esetben tehát m független gép van és bizonyos munkák között vannak megelőzési relációk. Emlékeztetünk arra, hogy ha elő van írva, hogy az i. munka megelőzi a j. munkát, azon azt értjük, hogy a j. munkát csak akkor szabad elkezdeni (valamely gépen), ha az i. munka végrehajtása már befejeződött.

A célfüggvény általában többféle lehet, az egyik leggyakrabban vizsgált célfüggvény a teljes átfutási idő (*makespan*), amit minimalizálunk. A 2. fejezetben tárgyalt ütemezési probléma esetén is a teljes átfutási idő minimalizálása a cél. A teljes átfutási időn azt az időintervallumot értjük, ami az első tevékenység végrehajtásának kezdetétől az utolsó tevékenység befejezési időpontjáig tart.

Az ütemezésről részletes áttekintést például [1]-ben találunk. A könyv az ütemezéshez kapcsolódó elméleti modelleket és a különböző ütemezési problémákat tárgyalja igen részletesen. Az igen bőséges irodalomból e helyütt még megemlítjük Ronald L. Graham két alapvető munkáját [2, 3]. Mindkét munkában a többprocesszoros rendszerekben előforduló, az ütemezésekhez kapcsolódó anomáliákkal foglalkozott. Cikkeiben többek között azt vizsgálta, hogy melyek azok az anomáliák, amelyek befolyásolhatják a teljes átfutási időt. Ezekben a cikkeiben definiálta a híres LS (*List Scheduling*, vagyis lista szerinti ütemezés) algoritmust, amely az első online ütemezési algoritmusnak tekinthető; valamint ennek a rendezett változatát, az LPT (*Longest Processing Time*) algoritmust. Az LS algoritmus valamilyen sorrendben ütemezi a munkákat, a következő munkát arra a gépre teszi, amelyik azt a legkorábban képes befejezni. Az LPT esetén a munkák a hosszúságaik szerinti monoton csökkenő sorrendbe vannak rendezve.

Megjegyezzük, hogy a Graham által vizsgált $P_m||C_{max}$ feladat esetén a munkáknak nincs kibocsájtási ideje sem és határideje sem, nem megszakíthatóak a munkák és amennyiben megelőzési reláció is van, az csak időbelit jelent, attól lehetnek a munkák külön gépeken.

Valamely ütemezési algoritmus approximációs aránya (*approximation ratio*, legrosszabb eset aránya) az R szám, ha az algoritmus által kapott célfüggvényérték legfeljebb R-szerese az optimális megoldás értékének, tetszőleges input esetén. Köztudott, hogy az LS algoritmus approximációs aránya $2 - \frac{1}{m}$ m gép esetén, míg az LPT algoritmusnak az approximációs aránya $\frac{4}{3} - \frac{1}{3m}$.

Mivel a 2. fejezetben egy ütemezési feladatot gépi tanulási módszerrel oldok meg, a gépi tanulásról is megadok egy rövid általános ismertetőt az alábbiakban.

1.1.1. A mesterséges intelligencia

A mai értelemben mesterséges intelligenciának nevezett tudományterület nagyon fiatal, keletkezése formálisan az 1956-os évre datálható; ekkor alkották meg a tudományterület nevét. Azonban azon tudományok fejlődése, amelyek ezt a területet megalapozták, már időszámításunk előtt elkezdődött.

A mesterséges intelligencia a számítógép-tudomány azon részterülete, amely az-

zal foglalkozik, hogyan lehetne hardver és szoftver rendszerekkel a lehető legjobban lemásolni az ember kognitív képességeit. Egy nagyon általános területről van szó, amely alapvetően az intelligencia megértéséhez és mesterséges lemásolásához szükséges eszközöket kutatja azzal a céllal, hogy ezt a képességet a szoftverből és hardverből felépülő gépeknek átadja.

A mesterséges intelligencia leírása alapvetően négy oldalról közelíthető meg:

- emberi módon cselekvő rendszerek,
- emberi módon gondolkodó rendszerek,
- racionálisan gondolkodó rendszerek,
- racionálisan cselekvő rendszerek.

Az emberi módon cselekvő rendszereknek képesnek kell lenniük a természetes nyelvek feldolgozására, a tudásreprezentációra, az önálló következtetésre, a gépi tanulásra, a gépi látásra és a robotikára. Azok a gépek, amelyek ezekkel a képességekkel rendelkeznek már intelligens viselkedést mutathatnak. Ennek mérésére 1950-ben Alan Turing javasolta a Turing-tesztet [4]. A teszt lényege, hogy a gép tud-e olyan intelligens viselkedésmintákat mutatni, amely alapján az ember elhiszi, hogy egy másik emberrel kommunikál. Ha igen, a teszt sikeres, ha nem, akkor a teszt nem sikeres.

Az emberi módon való gondolkodás területével a kognitív tudományok foglalkoznak, azonban még ma sem tudjuk pontosan, hogy hogyan működnek a kognitív funkciók. A mesterséges intelligencia és a kognitív tudományok összefonódnak és egymás fejlődését serkentik, jelenleg leginkább a látás, a természetes nyelvek feldolgozása és a tanulás területén.

A racionális gondolkodás fontos alapja a formális logika, amely logikai kifejezésekkel leírt problémák megoldását igyekszik megadni következtetések útján. Olyan programok, amelyek egy logikai kifejezésekkel reprezentált problémát megoldottak, már 1965-ben léteztek.

A racionális cselekvés nem más, mint egy meghatározott cél elérése érdekében megtett cselekvés. A mesterséges intelligenciában a racionálisan cselekvő entitásokat ágenseknek nevezzük. Fontos, hogy a racionális cselekvéshez szorosan hozzátartozik a racionális gondolkodás is. Azaz a legtöbb esetben a racionális következtetés eredménye a racionális cselekvés.

Ez tehát az a négy szempont, amelyek mentén könnyebben és érthetőbben lehet megfogalmazni, hogy mi is az a mesterséges intelligencia. Ez egy nagyon általános és szerteágazó terület, amely több más tudományterülettel összefonódva fejlődik. Az alapvető cél az intelligensen gondolkodó és viselkedő hardver- és szoftverrendszerek létrehozása, amelyek lényegében az ember kognitív képességeit próbálják másolni.

1.1.2. Gépi tanulás

A gépi tanulás fogalmát Alan Turing veszette be az 1950-ben megjelent cikkében [4]. A gépi tanulás a mesterséges intelligenciának egy, ma nagyon népszerű részhalmaza, amely olyan algoritmusokkal foglalkozik, amelyek képesek az általánosítás

elvével tanulni anélkül, hogy explicit módon programozva lennének konkrét feladat megoldására.

A mesterséges intelligencia az összes olyan algoritmus gyűjtőhelye, amelyek képesek a tanulásra és a következtetésre, hasonlóan az emberekhez. Ezen algoritmusok részhalmaza a gépi tanulás, amelyről már volt szó az előzőekben, majd ennek szűkebb részhalmaza a mély tanulás. A gépi tanulás módszereit három csoportba sorolhatjuk alapvetően: felügyelt tanulás, nem felügyelt tanulás és a megerősítéses tanulás.

A felügyelt tanulás esetében rendelkezésre állnak a bemenetek és a hozzájuk tartozó elvárt kimenetek, vagy más néven a tanítóminták. Ezek alapján a cél egy leképezés, azaz egy függvény megtanulása a bemeneti és a kimeneti halmaz között. A bemeneti és kimeneti attribútumok lehetnek diszkrétek vagy folytonosak. Ha a kimeneti attribútumok diszkrétek, akkor osztályozásról, ha folytonosak, akkor regresszióról van szó. Az osztályozás esetében gyakorlati alkalmazásként említhető a képek osztályozása, diagnosztika vagy a detektálás (pl. csalás detektálása) . A regressziónál a jellemzően felmerülő feladatok közé tartozik például az időjárás előrejelzés, az árak és árfolyamok meghatározása, különböző becslések elvégzése.

A felügyelet nélküli tanulásnál a bemeneti minták tanulása történik úgy, hogy nincs tanítóhalmaz, azaz nem áll rendelkezésre címkézett kimenet. Ez azt jelenti, hogy az algoritmus nem tudja eldönteni a kimenetről, hogy az jó, vagy nem jó. A nem felügyelt tanulás esetében az algoritmusok a bemeneti adathalmazon próbálnak szabályokat és mintákat felismerni. A két nagy algoritmuscsoport a klaszterező és az összefüggéseket kereső algoritmusok. A klaszterező algoritmusok célja, hogy a bemeneti adatok alapján a kimeneteket csoportokba (klaszterekbe) sorolják hasonlósági minták alapján. Másképpen fogalmazva, az egymáshoz hasonló kimenetek azonos klaszterbe kerülnek. Az összefüggési szabályokat kereső algoritmusok célja pedig, hogy olyan szabályokat keressenek, amik egy nagy adathalmazt leírnak. Például, ha az emberek megvásárolják az **A** terméket, akkor valószínűleg a **B** terméket is megveszik, viszont aki a **C** terméket választja, az szinte biztos nem fogja megvenni az **A**-t.

A **megerősítéses tanulás** a gépi tanulás harmadik csoportja. A megerősítéses tanulás feladata egy ágens dinamikus környezetben történő döntéshozatalának az optimalizálása úgy, hogy a döntések után kapott jutalmak összege maximális legyen. A környezetben végrehajtott döntések után, mint visszajelzés vagy megerősítés, az ágens egy jutalomnak nevezett értéket kap, ami lehet negatív vagy pozitív. A jutalom értéke, valamint az állapotokat és az akciókat értékelő függvények kimenetei alapján az ágens iterációk alatt képes megtanulni egy stratégiát arra vonatkozóan, hogy a dinamikus környezetben hogyan kell viselkednie a jutalom maximalizálása és ezáltal egy cél elérése érdekében. A megerősítéses tanulást tipikusan olyan problémáknál alkalmazzák, ahol valós időben, azonnal döntéseket kell hozni. Ilyen lehet például egy játékot játszó ágens, egy robotot vagy járművet irányító ágens vagy valamilyen logikai rejtvényt (pl. labirintus) megfejtő ágens.

Az általam tárgyalt ütemezési feladatra a megerősítéses tanulásból ismert Q-tanulás (*Q-Learning*) algoritmusát alkalmazva egy olyan eljárást dolgoztam ki, amely a tevékenységeknek egy olyan sorrendjét próbálja meghatározni, amely sorrendben a tevékenységeket ütemezve minimális átfutási időt kapunk. A tevékenységek kö-

zött megelőzési relációk adottak. Az eljárás kifejlesztésekor a Q-Learning módszerét alkalmaztam (viszonylag szabadon), emiatt az algoritmust "*Q-Learning Motivated Algorithm*" vagy röviden QLM-nek neveztem. A Q-Learning és az algoritmus részletes ismertetése az 2. fejezetben található.

1.2. Ládapakolás

A dolgozat 3. fejezetében ládapakolási feladatokkal foglalkozom, emiatt itt röviden, általánosan ismertetem a ládapakolási (*bin packing*) feladatkört.

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 [5, 6]. 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. Az approximációs arányt a következő alfejezetben pontosan definiáljuk.

D.S. Johnson disszertációja [7] 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.

A ládapakolási probléma témakörében már ismert benchmark feladatokat használtam az algoritmusok tesztelésére. Ebben a kutatási témában a cél az volt, hogy olyan, a ládapakoláshoz kapcsolódó előfeldolgozó algoritmusokat adjak meg, amelyek egyszerűsítik az egyes feladattípusok megoldását. Az általam kidolgozott előfeldolgozó algoritmusok esetén a következő történik. A feladatosztály elemeire bizonyos, a 3. fejezetben részletesen ismertetendő mohó módszerrel pakoljuk a tárgyakat. Kiderül, hogy bizonyos esetekben sikerül optimális pakolást készíteni (annak ellenére, hogy a ládapakolási feladat NP-nehéz). Természetesen nincs arra garancia, hogy minden inputra működik a módszer, de ez nem is cél. Azt fogjuk látni, hogy ha a benchmark feladatosztály 100 inputot tartalmaz, akkor a 100-ból 80 vagy akár több esetben optimális megoldást tudunk kapni mohó algoritmusok segítségével. Emiatt elegendő csak a maradék inputra alkalmazni valamilyen összetettebb algoritmust.

1.2.1. Approximációs eljárások

Johnson a disszertációjában számos "Fit-típusú" algoritmust vizsgált, mint például a First Fit (FF) vagy a Best Fit (BF). Ebben a munkámban az ismert algoritmusok közül a First Fit algoritmust használtam, amely feltehetőleg elsőként 1971-ben jelent meg Ullman munkájában [8].

A First Fit algoritmus a tárgyakat valamilyen adott sorrendben pakolja. A soron következő tárgy mindig az első olyan ládába kerül, amelybe a láda kapacitását figyelembe véve belefér. Amennyiben egyik nyitott ládába sem pakolható, úgy új ládát nyit és a tárgyat ebben helyezi el. Az algoritmust First Fit Decreasing-nek

nevezzük abban az esetben, ha a tárgyak a méretük alapján csökkenő sorrendben következnek egymás után.

Egy ládapakolási algoritmus hatékonyságát általában az approximációs aránnyal jellemzik, amelynek két fő fajtája van: aszimptotikus (*asymptotic approximation ratio*) és abszolút approximációs arány (*absolute approximation ratio*). Legyen L a pakolandó tárgyak halmaza. Legyen OPT egy optimális offline algoritmus és A egy tetszőleges ládapakolási algoritmus. $OPT(L)$ és $A(L)$ jelölje a ládák számát, amelyeket a két fenti algoritmus generál. Az abszolút és aszimptotikus közelítési arány a következőképpen definiálható:

$$R_{abs}(A) = \sup_L \left\{ \frac{A(L)}{OPT(L)} \right\}, \tag{1.1}$$

és

$$R(A) = \limsup_{n\to\infty} \left\{ \sup_L \left\{ \frac{A(L)}{OPT(L)} \mid OPT(L) = n \right\} \right\}. \tag{1.2}$$

Ullman munkájában már bebizonyította, hogy az $R(FF)$ értéke legfeljebb 1,7. Garey és társai [9] valamint Johnson és társai [10] pedig erre vonatkozó alsó korlátot is megadtak. Ezek az eredmények az approximációs algoritmusokkal kapcsolatos első eredményekhez tartoznak. Simchi-Levi 1994-ben megjelent munkájában [11] bebizonyította, hogy $R_{abs}(FF) \leq 1,75$. Az FF esetében az éles korlát $R_{abs}(FF) = 1,7$, amelynek bizonyítása a [12] és a [13] munkákban található. Az FFD esetében az éles korlát a [14] és [15] alapján az $FFD(L) \leq \frac{11}{9} \cdot OPT(L) + \frac{6}{9}$ formában adható meg. Az éles jelző azt jelenti, hogy a $\frac{6}{9}$ érték nem csökkenthető.

Az FF és FFD algoritmusok a mai napig nagyon népszerűek ládapakolási feladatok megoldásához, mert egyszerűek és sok esetben hatékonyak. Természetesen ezenkívül más algoritmusok is léteznek, mint pl. aszimptotikus polinomiális idejű approximációs sémák [16, 17]. Az approximációs séma azt jelenti, hogy bármilyen $\varepsilon > 0$ esetén létezik olyan A_ε algoritmus, amelyre $A_\varepsilon(L) \leq (1+\varepsilon) \cdot OPT(L) + C$ teljesül, ahol a C az inputtól független, univerzális konstans, és az algoritmus futási ideje polinomiális ε-ban. Ezek az algoritmusok inkább elméleti jelentőségűek.

1.2.2. Egzakt eljárások

Számos egzakt algoritmust publikáltak a ládapakolási probléma megoldására, többek között dinamikus programozáson alapulót, LP-relaxáción alapulót (utófeldolgozással), branch-and-bound, branch-and-price vagy constraint programming alapú módszereket. Egzakt eljárásnak olyan módszert nevezünk, amelyik minden esetben megtalálja az optimális megoldást, azonban hátrányuk ezeknek, hogy relatíve kis méretű (ládapakolási) feladatokat tudnak csak megoldani elfogadható időn belül. Alább néhány kapcsolódó publikációt adok meg.

Delorme és társai [18] górcső alá vették a legfontosabb matematikai modelleket és algoritmusokat, amelyek a ládapakolási probléma megoldását egzakt módon állítják elő, majd tesztelték a legjobbnak vélt szoftverek teljesítményét. Carvalho [19] lineáris programozási modellekkel foglalkozott a ládapakolási probléma megoldására. Wei és társai [20] egy új branch-and-price-and-cut algoritmussal oldották meg a

ládapakolási problémát. Ez az egzakt eljárás a klasszikus osztályozási modellen és egyéb módszereken alapult.

A ládapakolási probléma MILP (*mixed-integer linear programming*, kevert egészértékű lineáris programozási feladat) modellje meglehetősen egyszerű. A ládapakolási feladathoz egy ilyen MILP modellt adok meg.

Legyen n db tárgy, amelyek $w_1, w_2, \ldots, w_n$ mérettel rendelkeznek. A tárgyak méretei a racionális számok halmazából kerülnek ki, továbbá 0 és C között helyezkednek el. A ládáknak C a kapacitása és feltesszük, hogy n db ládánk van. Továbbá legyenek x_{ij} és y_j változók az alábbiak szerint.

$$x_{ij} = \begin{cases} 1 & \text{ha az i. tárgy a j. ládában van} \\ 0 & \text{különben} \end{cases} \tag{1.3}$$

$$y_j = \begin{cases} 1 & \text{ha a j. láda használatban van} \\ 0 & \text{különben} \end{cases} \tag{1.4}$$

Az y_i változó megadja, hogy a j. láda használatban van-e vagy sem. Az x_{ij} változó pedig azt írja le, hogy az i. tárgy a j. ládában van-e vagy sem. A cél nem más, mint a felhasznált ládák számának a minimalizálása néhány feltételt figyelembe véve.

A probléma formális leírása a következőképpen adható meg.

$$z = min \sum_{j=1}^{n} y_j \tag{1.5}$$

s. t.

$$\sum_{i=1}^{n} w_i x_{ij} \leq y_j C,\ j = 1, \ldots, m \tag{1.6}$$

$$\sum_{j=1}^{n} x_{ij} = 1,\ i = 1, \ldots, n \tag{1.7}$$

$$y_i, x_{ij} \in \{0, 1\},\ i = 1, \ldots, n,\ j = 1, \ldots, m \tag{1.8}$$

Az (1.6) feltétel megköveteli, hogy a j. láda töltöttsége nem lehet nagyobb C kapacitásnál, továbbá rákényszeríti az y_j változót, hogy 1 legyen ha van tárgy a ládába pakolva. Az (1.7) pedig azt írja le, hogy az i. tárgy pontosan egy ládába van bepakolva.

1.2.3. Metaheurisztikák, a fő változatok

A ládapakolási probléma NP-nehéz és emiatt az egzakt algoritmusoknál problémák adódhatnak, mint pl., hogy nem adnak optimális eredményt elfogadható időn belül. Emiatt a problémát a metaheurisztikák oldaláról is indokolt volt megközelíteni és számos eljárás látott napvilágot, úgy mint genetikus algoritmusok (*genetic algorithms*), részecske raj optimalizálás (*particle swarm optimization*) vagy tabu keresés (*tabu search*). Ezek az eljárások közel optimális megoldásokat találnak, azonban

néhány esetben képesek az optimális megoldás felderítésére is. Dokeroglu és Cosar munkájukban [21] összesen 1318 benchmark feladatot vizsgáltak meg a cikkükben bemutatott genetikus algoritmusokkal. A megoldott feladatok 88,5%-ban sikerült optimális megoldást találni.

Loh és társai [22] egy másfajta, egyszerű és gyors heurisztikus megoldást fejlesztettek. Az algoritmus teszteléséhez egy 1584 feladatból álló benchmarkot alkalmaztak. A feladatok kapcsán ismert legjobb illetve optimális megoldásokat az algoritmus megtalálta, valamint három további feladat esetén talált optimális megoldást (amely feladatokra nem tudták korábban, hogy mi az optimális megoldás).

Kucukyilmaz és társai [23] szintén egy genetikus algoritmust mutattak be. 1318 benchmark feladatot vizsgáltak és 99,6%-ban optimális megoldást találtak. A Hard28 teszthalmaz esetében 28 feladatból 23 alkalommal találták meg az optimális megoldást. Azonban a szép eredmények ellenére az algoritmus nagyon magas futási idővel dolgozik.

Borgulya [24] egy hibrid evolúciós algoritmust (*evolutionary algorithm*) fejlesztett, ahol egy egyed a megengedett megoldás és tartalmazza a ládák leírását is. Az algoritmus két új mutációs operátorral dolgozik és a megoldás minőségét lokális keresési eljárással javítja. A szerző 1615 benchmark feladatot oldott meg és 99,7%-ban optimális megoldást kapott. A Hard28-as teszthalmaz mind a 28 feladatára sikerült optimális megoldást találni.

1.2.4. Állatok viselkedésén alapuló, különböző metaheurisztikus változatok

Érdekességként megemlítem, hogy számos olyan metaheurisztika létezik, amely az állatok viselkedését próbálja lemásolni. Az első ilyen volt a hangya kolónia algoritmus (*ant colony algorithm*), ami számos optimalizálási probléma megoldásában [25] hatékonynak bizonyult. Hasonló eljárás a részecske raj optimalizálás is [26]. Számos egyéb, az állatok viselkedését utánzó algoritmus létezik ma már. Ilyen például a bálna optimalizálás (*whale optimization algorithm*), amelyet a ládapakolási problémára is alkalmaztak [27]. További hasonló algoritmusok, amelyek működésének ötlete a természetből való, a kakukk (*cuckoo search*) [28] és mókus keresés (*squirrel search*) [29]. A szentjánosbogár (*firefly search*) és a kakukk kereséssel, valamint a mesterséges méh kolónia algoritmussal (*artificial bee colony algorithm*) a [30]-ban foglalkoztak. Létezik még a denevér optimalizálás (*bat optimization*) [31], az afrikai bivaly optimalizálás (*african buffalo optimization*) [32, 33] vagy a kenguru keresés (*kangaroo search*) [34] és a szöcske algoritmus (*grasshopper algorithm*) [35].

1.2.5. Kapcsolódó, további releváns publikációk

Ebben az alfejezetben a ládapakolási probléma területéhez kapcsolódó, releváns publikációkat mutatom be. A ládapakolási feladatokkal kapcsolatos áttekintő cikkek például a következők [5, 18, 36]. Az irodalmi áttekintésből látható lesz, hogy a ládapakolás, mint kutatási terület jelenleg is aktív, továbbá számos, egymással versengő kutatási irány létezik. Emellett az elméleti eredményeket számos ipari területen alkalmazzák.

A [37]-ben a szerzők egy ládapakolási keretrendszert mutatnak be.

A [38]-ban a szerzők két új ládapakolási problémát mutatnak be néhány hasznos alkalmazási lehetőséggel, elsősorban logisztikai területen. Mindkét probléma esetében a hagyományos láda kiválasztási költség mellett számos más költség is megjelenik. Újfajta heurisztikákat vezettek be, amelyeket számos feladatra teszteltek. Az eredmények alapján a bemutatott heurisztikák teljesítménye jó, továbbá az új modellek gyakorlati alkalmazásának lehetnek előnyei.

A [39]-es publikációban a szerzők repülőgépek karbantartási feladatainak ütemezésével foglalkoznak, amelyek szükségesek ahhoz, hogy a gépek biztonságosak legyenek minden repülés során. Ez egy összetett kombinatorikus feladat, amelyet minden nap el kell végezni. A probléma egy időfüggő, változó méretű ládapakolási feladatként adható meg. Az új megközelítés képes hatékonyan megoldani a többéves feladatkiosztási problémát néhány perc alatt. A probléma megoldásához egy, a Worst Fit Decreasing-en alapuló heurisztikus módszert alkalmaznak. A heurisztikát egy európai légitársaságtól származó adatokkal tesztelték és validálták. A korábban használt módszerhez képest az új algoritmus 30%-al gyorsabb volt minden elvégzett tesztesetre és a legtöbb esetben az optimális eredménytől való eltérés 3% alatt maradt.

A [40]-es publikációban bemutatják az egydimenziós ládapakolási probléma megoldó algoritmusainak legújabb implementációit, különös tekintettel a populációalapú metaheurisztikus algoritmusokra.

A [30]-ban a szerzők egy szisztematikusan elvégzett teljesítmény kiértékelést mutatnak be néhány reprezentatív algoritmuson. A teszthez három standardnak számító ládapakolási adathalmazt használtak, amelyekben összesen több mint 1210 feladat található. A vizsgált heurisztikák által adott eredményeket a best fit és más heurisztikák eredményeivel hasonlították össze.

A [41]-es publikációban a szerzők egy dinamikus ládapakolási probléma egy konkrét változatát alkalmazták, amely egy ütemezési feladat részfeladataként fordul elő. A feladat megoldására a szerzők különböző előfeldolgozó technikát javasolnak a változók és a feltételek számának csökkentése érdekében. Az elvégzett számítások alapján az új megközelítés a korábbiaknál jobb teljesítményt eredményezett mind a megoldások, mind pedig a futási idő tekintetében.

1.3. Ládafedés

A 4. fejezetben egy viszonylag új területtel foglalkozom, amelynek a neve ládafedés szállítással (*Bin Covering with Delivery*, BCD). Ebben a problémában, hasonlóan a ládapakolási problémához, tárgyakat pakolunk ládákba, amelyeket, ha fedetté válnak, 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 [42]-ben lett bemutatva. A 4. fejezetben ennek a problémának a kiterjesztéséről és alapos vizsgálatáról lesz szó. A probléma offline változatával a [43] foglalkozik, továbbá néhány kapcsolódó probléma a [44]-ben kerül bemutatásra.

Emlékeztetünk arra, hogy a klasszikus ládapakolási probléma esetében a tár-

gyakat be kell pakolni valamelyik ládába úgy, hogy a tárgyak összmérete a láda kapacitását nem lépheti túl, és a cél a felhasznált ládák számának minimalizálása. A ládafedési feladat esetében viszont a lehető legtöbb ládát akarjuk lefedni. A ládát fedettnek tekintjük, ha a ládába bepakolt tárgyak összmérete legalább a láda kapacitásával egyenlő. Ismert, hogy mindkét probléma (ládapakolás és ládafedés) NP-nehéz [6]. Ez azt jelenti, hogy az optimális megoldás eléréséhez akár exponenciálisan sok lépés is szükséges lehet (rossz esetben). Azonban számos olyan eset van, amikor rövid időn belül sikerült optimális megoldást találni.

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 az [5], [45] és [46] áttekintő cikkek foglalkoznak.

A kutatásom ezen területén a ládafedési probléma online változatával foglalkoztam. Ahogy fentebb volt róla szó, a megoldás nem csak a pakolás minőségén (azaz, hogy az egyes tárgyakat milyen másik tárgyakkal együtt pakoljuk), hanem a felhasznált ládák számán is múlik. A feladat megoldása során *"gyors és jó"* pakolást szeretnék elérni az algoritmussal, amely esetében a célfüggvény bünteti azt, ha túl sok láda van nyitva egyszerre. Ez az ötlet természetesen adódik abból a megállapításból, hogy minél több láda van nyitva, annál nehezebb őket kezelni.

Megjegyezzük, hogy a *"Scheduling with delivery"* témakör (pl. [47]) hasonló probléma a mostanihoz. Vannak olyan munkák is ([48], [49]), amelyek a várakozási időt büntetik. Azaz azt az időt, amely a láda nyitása és elszállítása között telik el.

Az általam alkalmazott modell eltér az utóbbi két munkától, ugyanis ebben az esetben nem az eltelt időt bünteti a célfüggvény, hanem egyebek mellett a túl sok nyitott ládát.

A 4. fejezetben pontosan meghatározom a probléma definícióját és néhány, az offline modellre vonatkozó tulajdonságot is megadok. Bemutatom azokat a feladatosztályokat, amelyekre vonatkozó vizsgálatokat végeztem. Ezután ismertetek néhány természetesen adódó online algoritmust, majd ezek alapján bemutatok egy új, rugalmas algoritmus osztályt, amit MMask-nak neveztem el. Az új algoritmushoz kapcsolódóan bemutatok egy metaheurisztikus megoldást *Evolution of Algorithms (EoA)* néven, amelynek korábbi változata a [42] cikkben már megtalálható. A fejezet végén az új algoritmus hatékonyságát számítógépes futásokból származó eredményekkel demonstrálom, majd konklúziók levonása mellett összegzem az elvégzett munkát.

1.4. A dolgozat szerkezete

A dolgozat négy fő fejezetre tagolódik, amelyek közül az 2., 3. és 4. fejezetek a három kutatási területet foglalják magukba, a dolgozat elején pedig egy általános bevezetés olvasható.

A 2. fejezet a megerősítéses tanulás területéről ismert Q-tanulás algoritmusát

alkalmazza egy bizonyos, bonyolult ütemezési feladat megoldására. A megoldás alapötlete az, hogy az algoritmus a tevékenységeknek egy sorrendjét határozza meg, amely sorrendben az ütemezés végrehajtásra kerül. A sorrend kialakításában játszik nagy szerepet a Q-tanulás. Az algoritmusban a megerősítést az ütemezés végeredménye, a teljes átfutási idő jelenti.

A 3. fejezetben ládapakolási feladatoknak bizonyos mohó algoritmusait vizsgáltam. A mohó algoritmusok a feladatok különböző tulajdonságait használják ki, ezáltal egyszerű és könnyen implementálható eljárásokkal a megoldandó probléma bizonyos esetekben optimálisan megoldható.

A 4. fejezet a ládafedési feladat egy bizonyos általánosításával foglalkozik. A megoldás során korábbi, természetesen adódó algoritmusokat vizsgáltam, továbbá bevezettem és vizsgáltam egy új, flexibilis algoritmuscsaládot, amelynek optimalizáltam a paramétereit.

2. fejezet

Egy megerősítéses tanulás által motivált algoritmus alkalmazása bizonyos típusú ütemezési feladatra

2.1. Q-tanulás által motivált algoritmus (QLM)

A Q-tanulás egy olyan megerősítéses tanulási módszer, amely az élet számos területén felmerülő helyzet kezelésére alkalmas. Ebben a fejezetben a Q-tanulást bizonyos fajta ütemezési feladat megoldására fogjuk alkalmazni. A feladat jellemzésére lássunk néhány példát. Első példaként a hivatali ügyintézést említhetném (pl. adóhivatal, bevándorlási hivatal, önkormányzati hivatal stb.), ahol naponta rengeteg dokumentumot kell feldolgozni. A példa kedvéért a dokumentumokat különböztessük meg. Az első típusú dokumentumok feldolgozása egyszerű, azt bármilyen hivatali dolgozó képes elvégezni. A második típusú dokumentum feldolgozását már csak a megfelelő képzettséggel rendelkező dolgozók tudják elvégezni. Tehát, azoknál a munkáknál, amelyet mindenki el tud végezni, a p_j idők azonosak. Azon munkák esetében, amelyeket nem mindenki tud elvégezni, ott a dolgozók egy részénél azonosak csak a p_j idők. A feladatok pedig nem megszakíthatóak, ugyanis a dokumentumok feldolgozása gyorsan történik és feltételezzük, hogy egy dokumentum feldolgozását nem hagyják félbe. Továbbá itt nincs megelőzési reláció. A cél a dokumentumok feldolgozási idejének a minimalizálása.

Egy másik példa lehet az építkezési munkálatok. A munkálatok több alfeladatra oszthatóak fel, pl. alapozás, falak felhúzása, mérnöki feladatok stb. Természetesen ezen részfeladatok között megelőzési relációk vannak. Például a falak felhúzása előtt nyilvánvalóan az alapnak kell elkészülnie. Ebben a példában az erőforrások az építkezésen dolgozó emberek. A munkások különböző feladatokat tudnak elvégezni (pl. a segédmunkás keveri a betont, viszont villanyt nem szerelhet, de a villanyszerelő tudja felügyelni a beton keverését, de ő inkább a villanyszereléshez ért). Az épitkezési munkálatok elvégzését napokban mérjük. A munkások egyszerre csak egy helyen dolgoznak, és addig nem mennek máshova, amíg az adott helyen el nem készülnek. Ebből a szempontból a munka nem megszakítható. A cél pedig az, hogy a dolgozókat vagy azoknak a csoportjait úgy osszuk szét a feladatok között, hogy egyrészt azt az adott dolgozó vagy dolgozók képesek legyenek elvégezni, másrészt pedig a munka

ideje minimális legyen.

A harmadik példa egy családi ház felújítása. Itt is hasonló a megoldandó probléma az előző példához. Egy cég, amely a felújítást végzi, munkásokat küld a feladat elvégzésére. Azonban mind a cégnek, mind pedig a ház tulajdonosának érdeke, hogy olyan munkások végezzék el a munkát, akiknek megvan hozzá a megfelelő kvalitásuk, azaz precízen, szakszerűen el tudják végezni azt. Továbbá az egyes munkások elosztása a feladatok között úgy történjen, hogy a munkavégzés hatékony és minél gyorsabb legyen.

Természetesen, ha egy megelőzési reláció van, akkor ha az egyik munkás befejezi az adott részfeladatot, akkor a tervben következő feladat más munkás által is végezhető. Mint láttuk, bizonyos esetekben feltehető, hogy a munkák végzése nem megszakítható, ez a munkafolyamat szervezéséből, vagy a munkák jellegzetességeiből következik. (Pl. porszívózás az megszakítható, de piskóta sütése az nem megszakítható.) Megjegyezzük, hogy sok esetben a munkavégzés esetén kibocsátási és elvárt befejezési idők is vannak. Ez azt jelenti, hogy például valamely munkáknak egy bizonyos határidőre el kell készülnie. Kétfajta határidő van a szakirodalomban. Egyik esetben eddig az időpontig a munkának mindenképp el kell készülnie. Például, taxival megyünk a reptérre. Akkor a repülőt mindenképp el kell hogy érjük, különben nem fizetünk. Másik esetben "jó lenne" ha a munka egy adott időpontra befejeződne, de nem történik tragédia akkor sem, ha kicsit csúszik a munka, legföljebb valamennyi kötbért fizetünk majd. Kibocsátási idő pedig azt jelenti, hogy az adott munkát nem kezdhetjük el ennél az időpontnál korábban. Ebben a dolgozatban ilyen kibocsátási időkkel és elvárt befejezési időkkel nem foglalkoztam, egyrészt mert a vizsgálat enélkül is eléggé bonyolult, másrészt azért, mert a kapcsolódó cikkekben sem voltak sem kibocsátási, sem befejezési idők.

A fenti példák segítségével egy könnyen érthető áttekintést kívántam adni arról, hogy a felvetett problémának mi a lényege. A problémát olyan ütemezési problémaként modelleztem, ahol az egyes tevékenységek között megelőzési relációk vannak definiálva, valamint az egyes tevékenységek végrehajtásának ideje a hozzárendelt erőforrástól függ. Ezeket az erőforrásokat független erőforrásoknak nevezzük. Az első példában a második típusú dokumentum feldolgozása két tevékenységre bontható fel: feldolgozás és ellenőrzés. Látható, hogy a két tevékenység között egyértelmű megelőzési reláció van, ugyanis a dokumentum feldolgozását előbb kell elvégezni, mint az ellenőrzést.

Tehát a felvetett probléma az ütemezés területéhez tartozik. Az egyes tevékenységek végrehajtását az erőforrásokkal tudjuk elvégezni, amelyeket hozzá kell rendelni a tevékenységekhez úgy, hogy az előre meghatározott feltételek teljesüljenek. A munkámban a cél az, hogy a teljes átfutási időt minimalizáljam.

Az ütemezési feladatok általában számításigényes, nehéz feladatok. Ezeket a problémákat gyakran valamilyen heurisztikus módszerrel oldják meg. Az ebben a fejezetben bemutatott munkám egy, a megerősítéses tanulással kiegészített ütemező eljárás. A következő alfejezetekben részletesen bemutatom a felvetett problémát, a megerősítéses tanulást, és azon belül a Q-tanulást, a kidolgozott eljárást. Kitérek az alkalmazott példákra és azok eredményeinek kiértékelésére, végül összefoglalásként összegzem az elért eredményeket.

A problémával és annak megoldásával az első impakt faktoros cikkemben [50]

foglalkoztam.

2.1.1. Kapcsolódó munkák a megerősítéses tanulás ütemezésben való alkalmazására

Orhean és szerzőtársai [51] 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 [52] 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 alkalmazták. Stefán [53] a Q-tanulás algoritmusát alkalmazta egy permutációs flow shop problémára, ahol a cél a gépek üresjárati idejének a minimalizálása volt. Stefán a disszertációjában [54] bővebb leírást adott az algoritmusról, amely a flow shop típusú probléma megoldására készült. A cikk [53] és a disszertáció [54] az általam bemutatott probléma megerősítéses tanulás oldalról való megközelítésében segített. Gabel és Riedmiller [55] szintén a Q-tanulást alkalmazták, viszont ők egy job shop típusú problémára, amelynél a Q-függvényt neurális háló segítségével közelítették. Shahrabi és szerzőtársai [56] a megerősítéses tanulást alkalmazták egy job shop típusú problémára kifejlesztett eljárás továbbfejlesztéséhez. További példákat találunk a megerősítéses tanulás alkalmazására az ütemezés területén az [57–59] cikkekben. A [60]-ban például a Q-tanulásnak a neurális hálózatokkal összekapcsolt változatát alkalmazták, amelyet Deep Reinforcement Learning-nek hívnak.

2.2. Megerősítéses tanulás

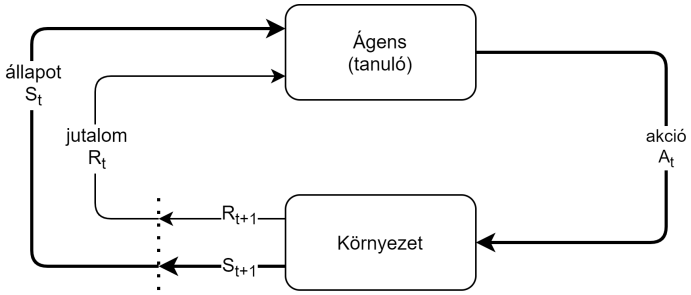
A megerősítéses tanulás [61] a gépi tanuláson belül a harmadik terület a felügyelt és a nem felügyelt tanulás mellett. A megerősítéses tanulás olyan technikák, algoritmusok gyűjteménye, amelyek segítségével a tanuló azt tanulja meg, hogy adott szituációban mit tegyen. A döntéseket stratégiák megtanulásával hozza meg. A stratégia az adott állapot leképezése egy akcióra, cselekvésre. A megerősítéses tanulással megoldható problémák jellemzője, hogy adott állapotban a választott akció pontos megfigyelésére nincs lehetőség, csak az ún. késleltetett jutalmakon keresztül, amely egy becslés. A cél az, hogy ezen jutalmakat a tanuló ágens maximalizálja.

Formálisan, egy megerősítéses tanulási probléma Markov döntési folyamattal írható le. Az érthetőség kedvéért a Markov folyamatok bemutatását a Markov tulajdonságtól kezdem, és megmutatom, hogy ezek a folyamatok hogyan épülnek fel attól függően, hogy milyen új komponenseket veszünk be a rendszerbe.

2.2.1. A megerősítéses tanulás általános modellje

A tanuló entitást vagy döntéshozót ágensnek nevezzük. Az ágens lehet bármi (ember, robot, jármű ...), ami érzékeli a környezetét, az érzékelt információk alapján döntést hoz és visszahat a környezetére. Környezet az, amelyben az ágens működni képes. Az ágens a meghozott döntések hatására a környezetével interakcióba

lép, azaz valamilyen cselekvést végrehajt. Ezekre az akciókra a környezet válaszol, amelynek hatására az ágens új állapotba kerül. Továbbá a környezet az ágens számára egy megerősítést, egy ún. jutalmat is biztosít. Ez a jutalom egy számérték, amelyet az ágens igyekszik maximalizálni. Természetesen a jutalom értéke lehet negatív, pozitív vagy éppen nulla is.



2.1. ábra. A megerősítéses tanulás általános modellje

Formálisan megfogalmazva a fentieket, az ágens minden diszkrét $t = 1, 2, 3, 4 \ldots$ időpillanatban kapcsolatba kerül a környezetével. Minden egyes t időpillanatban az ágens egy adott $S_t \in \mathcal{S}$ állapotban van, ahol $\mathcal{S}$ a lehetséges állapotok halmaza. Az aktuális állapotban választ egy $A_t \in \mathcal{A}(S_t)$ akciót, ahol $\mathcal{A}(S_t)$ az S_t állapotban elérhető akciók halmaza. Egy időpillanattal később a végrehajtott akció hatására a környezettől kap egy $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ jutalmat, és egy új S_{t+1} állapotba kerül. Az előbbi képletben $\mathcal{R}$ a lehetséges jutalmak halmaza.

Adott állapotban az akció kiválasztása valamilyen előre definiált mechanizmus szerint történik. Ezt a mechanizmust az ágens stratégiájának vagy politikájának nevezzük. A stratégia nem más, mint egy függvény, amely egy valószínűségi eloszlást ír le az akciók felett. A függvényt π-vel jelöljük. A $\pi(a|s)$ kifejezés azt adja meg, hogy az $A_t = a$ akció mekkora valószínűséggel kerül kiválasztásra az $S_t = s$ állapotban. Az előbbi kifejezésekben a szokásos, egyszerű jelöléseket használtuk.

2.2.2. Célok és jutalmak

Ahogy a korábbiakban volt róla szó, az ágens célja az, hogy a környezettől kapott jutalmat maximalizálja. Itt nem az azonnal megkapott jutalomra kell gondolni, hanem a hosszútávon szerzett jutalmakra. Ezt nevezzük kumulatív jutalomnak. A kumulatív jutalom formális felírásakor könnyebb dolgunk van abban az esetben, ha az ágens-környezet páros közötti interakciók száma véges, azaz az ágens futása véges lépésben befejeződik. Egy ilyen szekvenciát epizódnak nevezünk és az epizód akkor fejeződik be, amikor az ágens egy speciális állapotba, a végállapotba kerül. A végállapotokat is tartalmazó halmazt $\mathcal{S}^+$ szimbólummal jelöljük. Ebben az esetben a G_t kumulatív jutalom a t időpillanatban nem más, mint a t időpillanat után kapott jutalmak összege. Az ágens célja pedig a várható kumulatív jutalom maximalizálása. Mivel nem tudjuk előre, hogy mennyi lesz a tényleges jutalom a folyamat végén, a hangsúly a "várhatón" van.

1. Definíció. *Várható kumulatív jutalom*

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \tag{2.1}$$

ahol a $T < \infty$ az utolsó időpillanat.

Azonban vannak olyan helyzetek, amikor egy probléma nem bontható fel egyértelműen epizódokra, tehát a folyamat nem fejeződik be véges számú lépésben. Ezeket a problémákat folytonos vagy végtelen időhorizontú feladatoknak nevezzük, ahol a $T = \infty$. Ekkor könnyen látható, hogy a (2.1) összegzés végtelenné válhat, azaz egy olyan végtelen sort kapunk, ami divergens. Ennek a problémának a feloldására egy új módszert kell bevezetnünk, amit diszkontálásnak hívunk. Innentől kezdve az ágens célja a várható diszkontált kumulatív jutalom maximalizálása. A (2.1) kifejezés az alábbiak szerint módosul.

2. Definíció. *Várható diszkontált kumulatív jutalom*

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.2}$$

ahol a γ a diszkontálási paraméter és $\gamma \in [0, 1]$. A diszkontálásnak köszönhetően a végtelen sor konvergenssé válik. A diszkontálási paraméter másik alkalmazása a jövőben esedékes jutalom jelenértékének a kiszámítása, azaz, hogy mennyit ér most a k időpillanat múlva kapott jutalom. Pontosabban kifejezve a k időpillanat múlva esedékes jutalom most, a jelenben az eredeti érték γ^{k-1}-szeresét éri, ahhoz képest mintha azt most, azonnal megkapnánk. Ezzel az ágens viselkedését lehet befolyásolni. Ha a γ értéke nullához közelít, úgy az ágens egyre mohóbb, és egyre nagyobb mértékben csak az azonnali jutalmat veszi figyelembe. Ha a γ értéke egyhez közelít, úgy az ágens egyre nagyobb mértékben veszi figyelembe a jövőbeni jutalmakat is.

2.2.3. Markov tulajdonság

Egy probléma Markov tulajdonságú [62] akkor, ha a jövőbeni állapotok nem függenek a múltbéli állapotoktól, csak a jelentől.

A megerősítéses tanulás esetében azt mondjuk, hogy egy probléma (és a környezet is) Markov tulajdonságú, ha a rendszer minden jövőbeni állapotára igaz az, hogy csak a jelentől függ, a múltbéli állapotoktól nem.

3. Definíció. *Markov tulajdonság*

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2, \ldots, S_t] \tag{2.3}$$

Ha a (2.3) tulajdonság igaz, akkor az S_t állapotot Markov állapotnak nevezzük. A Markov tulajdonsággal rendelkező állapot minden hasznos információt tartalmaz a múltban megtörtént eseményekről, kompakt formában.

2.2.4. A rendszer dinamikája

A rendszer dinamikáját az állapotok közötti átmenetek valószínűségét leíró, ún. átmenet valószínűség mátrix adja meg. Bármely tetszőleges két szomszédos állapot közötti átmenet az alábbi valószínűséggel írható le.

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s] \tag{2.4}$$

A valószínűségeket a $\mathcal{P}$ állapotátmenet mátrix tartalmazza minden s állapotból minden s' állapotba. A mátrix egy sorában található valószínűségek összege pontosan 1. Ez azt jelenti, hogy valamilyen szomszédos s' állapotba biztosan átlép az ágens.

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix} \tag{2.5}$$

$$\sum_{j=1}^{n} \mathcal{P}_{ij} = 1, \;\; i = 1, \ldots, n \tag{2.6}$$

Ahhoz, hogy egy probléma értelmezhető és megoldható legyen, minden esetben előre definiálni kell az állapotátmenet mátrixot. Bonyolultabb feladatok esetében ennek a definiálása kényelmetlen és nehézkes lehet.

Markov döntési folyamat

Az olyan megerősítéses tanulási problémát, amely Markov tulajdonságú, Markov döntési folyamatnak nevezzük. Az alábbiakban áttekintést nyújtok az egyszerű Markov lánctól indulva az összetettebb Markov döntési folyamatig a megerősítéses tanulás szemszögéből.

Markov-lánc

A Markov-lánc egy memória nélküli, véletlenszerű állapotok sorozatából álló folyamat.

4. Definíció. *Markov folyamat: A Markov folyamat egy $\langle \mathcal{S}, \mathcal{P} \rangle$ rendezett kettes, ahol:*

- $\mathcal{S}$ *az állapotok véges halmaza*
- $\mathcal{P}$ *az állapotátmenet valószínűségi mátrix*
  $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$

Markov-folyamat jutalmazással

Annak a Markov-láncnak a neve, amelyben megjelenik a jutalomfüggvény, Markov-folyamat jutalmazással (*Markov Reward Process*), vagy röviden MRP. Ez nem más, mint a Markov-lánc kiterjesztése, ahol az egyes állapotváltásokra a környezet egy jutalomfüggvény segítségével visszajelez.

5. Definíció. *Markov Reward Process: A Markov Reward Process egy $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ rendezett négyes, ahol:*

- $\mathcal{S}$ *az állapotok véges halmaza*
- $\mathcal{P}$ *az állapotátmenet valószínűségi mátrix*
  $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$

- $\mathcal{R}$ *a jutalomfüggvény*
  $R_s = \mathbb{E}[R_{t+1}|S_t = s]$

- $\gamma \in [0, 1]$ *a diszkontálási faktor (leszámítolási tényező)*

Megjegyezzük, hogy itt $\mathcal{R}$ nem függ a választott lépéstől csak az adott állapottól.

A várható diszkontált jutalom a (2.2) képlet alapján számítható. A 2.2.2. fejezetben már volt róla szó, hogy miért szükséges a jutalom értékeit diszkontálni az idő függvényében. Egyrészt, a folytonos feladatok esetében ezzel biztosítható a végtelen sor konvergenciája, másrészt pedig a jövőbeni jutalmak fontossága is beállítható. Utóbbi esetben, ha a γ értéke nullához közelít, akkor az ágens rövidlátó, ha az egyhez közelít, akkor a jövőbeni jutalmak egyre jobban felértékelődnek.

6. Definíció. *Állapotértékelő függvény MRP esetén: Az MRP állapotértékelő függvénye, $v(s)$ egyenlő az elvárt diszkontált jutalommal az s állapotból indulva.*

$$v(s) = \mathbb{E}[G_t|S_t = s] \tag{2.7}$$

Az állapotok hasznosságát a belőle kiinduló állapotsorozatok várható hasznosságának összegével tudjuk leírni. Az állapotok hasznosságának meghatározására a Bellman-egyenletet [63] alkalmazzák, amely Richard Bellmantól származik. A Bellman-egyenlet szerint egy adott állapot hasznossága a benne tartózkodás értéke és a szomszédos állapotok várható hasznosságának az összege. A Bellman-egyenlet szerint az értékelő függvény két részre bontható fel:

- az azonnali jutalomra (R_{t+1})

- a szomszédos állapot diszkontált hasznosságára ($\gamma v(S_{t+1})$)

Az MRP esetében az értékelőfüggvény Bellman-egyenlete a következő.

7. Definíció. *Bellman-egyenlet*

$$\begin{aligned} v(s) &= \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots |S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots)|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \end{aligned} \tag{2.8}$$

azaz,

$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s'). \tag{2.9}$$

Az MRP esetében egy állapot hasznosságának kiszámítása a (2.9) kifejezés alapján történik. A kifejezésben az $\mathcal{R}_s$ az azonnali jutalom az s állapotban, a $\mathcal{P}_{ss'}$ annak a valószínűsége, hogy s állapotból az s' állapotba kerül az ágens, a $v(s')$ pedig a szomszédos állapot hasznossága. A szumma művelet pedig amiatt kell, mert egy adott állapotnak több szomszédja is lehet.

A Bellman-egyenlet lineáris, így közvetlenül is megoldható. Ehhez a (2.9) kifejezést mátrixok segítségével kell felírni. Így kapunk egy lineáris egyenletet, amelyet átrendezve megkapjuk a megoldást.

$$\begin{aligned} \mathsf{v} &= \mathcal{R} + \gamma\mathcal{P}\mathsf{v} \\ (\mathcal{I} - \gamma\mathcal{P})\mathsf{v} &= \mathcal{R} \\ \mathsf{v} &= (\mathcal{I} - \gamma\mathcal{P})^{-1}\mathcal{R} \end{aligned} \tag{2.10}$$

Az előbbi képletben feltételezzük, hogy a megfelelő mátrix invertálható. A direkt megoldás csak kis méretű MRP-k esetén lehetséges. Továbbá számos iteratív módszer is rendelkezésre áll:

- Dinamikus programozás
- Monte-Carlo kiértékelés
- TD (Temporal Difference - Időbeni különbség) tanulás

Markov döntési folyamat

A Markov döntési folyamat (*Markov Decision Process*, MDP) egy Markov Reward Process (MRP), ahol már megjelenik a döntés akciók formájában. Azaz a modell egy újabb taggal bővül.

8. Definíció. *Markov Decision Process: A Markov Decision Process egy* $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ *rendezett ötös, ahol:*

- $\mathcal{S}$ *az állapotok véges halmaza*
- $\mathcal{A}$ *az akciók véges halmaza*
- $\mathcal{P}$ *az állapotátmenet valószínűségi mátrix*
  $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$
- $\mathcal{R}$ *a jutalomfüggvény*
  $R_s = \mathbb{E}[R_{t+1} | S_t = s]$
- $\gamma \in [0, 1]$ *a diszkontálási faktor (leszámítolási tényező)*

Az akciók olyan elemei a rendszernek, amelyek az ágens cselekvéseit írják le. Minden állapotban az ágens valahány akció közül választhat, amelynek hatására átkerül egy új állapotba. Az akció kiválasztása is valószínűségek alapján történik. Az, hogy milyen a valószínűségek eloszlása az akciók felett, azt a stratégia vagy más néven a politika határozza meg. Egy stratégia teljes mértékben meghatározza az ágens viselkedését.

9. Definíció. *Stratégia: Egy* π *stratégia egy valószínűségi eloszlás az adott állapotban elérhető akciók felett.*

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s] \tag{2.11}$$

Tehát a fenti definíció szerint egy stratégia azt írja le, hogy az ágens egy adott s állapotban mekkora valószínűséggel választja az a akciót. Az MRP-hez hasonlóan az MDP esetében is az állapotok a hasznosságukkal jellemezhetőek. Azonban az MDP definíciója szerint itt már megjelennek az akciók is, mint döntés. Ezért itt nem csak állapotértékelő, hanem akcióértékelő függvényről is beszélhetünk. Az MDP esetében az értékelő függvények definíciója a következő.

10. Definíció. *Állapotértékelő függvény MDP esetén: Az MDP állapotértékelő függvénye, $v_\pi(s)$ egyenlő az elvárt diszkontált jutalommal az s állapotból indulva és a π stratégiát követve.*

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{2.12}$$

11. Definíció. *Akcióértékelő függvény MDP esetén: Az MDP akcióértékelő függvénye, $q_\pi(s, a)$ egyenlő az elvárt diszkontált jutalommal az s állapotból indulva, az a akciót végrehajtva és a π stratégiát követve.*

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{2.13}$$

Az MRP-hez hasonlóan az MDP értékelő függvényei is két részre bonthatóak. Viszont, itt az MRP-hez képest az ágens úgy próbálja meghatározni egy állapot vagy akció hasznosságát, hogy egy π stratégiát követ. Emiatt itt a Bellman-egyenletet Bellman várhatóérték-egyenletnek [64] nevezzük. A két függvény dekompozíciója a következőképpen írható fel.

12. Definíció. *Bellman várhatóérték-egyenlet*

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \tag{2.14}$$

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \tag{2.15}$$

Az előbbiek, vagyis az állapotértékelő és az akcióértékelő függvény között rekurzív összefüggés van. Később megmutatjuk, hogy kölcsönösen, egyikből a másik levezethető. Az s állapot értéke az elérhető akciók hasznosságának súlyozott összege.

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) \tag{2.16}$$

Egy akció értékének a súlya az adott akció kiválasztásának a valószínűsége. A választott a akció hasznossága függ azon s' utódállapotok hasznosságától, amelyekbe az ágens a környezet dinamikája alapján kerülhet. Továbbá függ az $\mathcal{R}_s^a$ azonnali jutalomtól is.

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \tag{2.17}$$

Az $\mathcal{R}_s^a$ az s állapotban végrehajtott a akció után kapott azonnali jutalom. A $\mathcal{P}_{ss'}^a$ annak a valószínűségét adja meg, hogy s állapotban az a akciót végrehajtva az s' állapotba kerül az ágens.
Ha a (2.16) kifejezésbe behelyettesítjük a (2.17) kifejezést, az állapotértékelő függvény alábbi alakját kapjuk.

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')) \tag{2.18}$$

Tehát az s állapot értéke (hasznossága) függ a választott akció értékétől (ennek kiválasztása az adott stratégia mentén történik) és azon utódállapotok értékétől, amelyekbe az adott akció végrehajtása után a rendszer az előre definiált dinamika alapján vihet. Továbbá az akció kiválasztásáért kapott azonnali jutalomtól.

Ha a (2.17) kifejezésbe behelyettesítjük a (2.16) kifejezést, az akcióértékelő függvény alábbi alakját kapjuk.

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a') \tag{2.19}$$

Az MDP-k esetében a Bellman-egyenlet lineáris, így a (2.10) alapján közvetlenül is megoldható.

Az értékelő függvény optimalitása MDP-ben

13. Definíció. *Állapotértékelő függvény optimalitása: Az optimális állapotértékelő függvény $v_*(s)$ a maximális állapotértékelő függvény minden π stratégia felett.*

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.20}$$

14. Definíció. *Akcióértékelő függvény optimalitása: Az optimális akcióértékelő függvény $q_*(s,a)$ a maximális akcióértékelő függvény minden π stratégia felett.*

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{2.21}$$

15. Definíció. *Stratégiák összehasonlítása: Két tetszőleges stratégia közül π legalább olyan jó, mint π' ($\pi \geq \pi'$) ha*

$$v_\pi(s) \geq v_{\pi'}(s),\ \forall s \tag{2.22}$$

Megjegyezzük, hogy az optimális stratégia nyilvánvalóan legalább olyan jó, mint bármelyik másik.

Az optimális értékelő függvény garantálja az ágens legjobb teljesítményét az MDP-ben, és egy MDP akkor megoldott, ha ismerjük ezt az optimális függvényt.

1. Tétel. *[61] Bármely Markov döntési folyamat esetében*

- *létezik legalább egy optimális π_* stratégia, ami legalább olyan jó, mint a többi, $\pi_* \geq \pi,\ \forall \pi$,*
- *minden optimális stratégia megadja az optimális állapot- és akcióértékelő függvényt, $v_{\pi_*}(s) = v_*(s)$ és $q_{\pi_*}(s,a) = q_*(s,a)$.*

Az optimális stratégia megtalálható, ha $q_*(s,a)$ felett maximalizálunk.

$$\pi_*(a|s) = \begin{cases} 1 & \text{ha } a = \arg\max_{a \in \mathcal{A}} q_*(s,a) \\ 0 & \text{egyébként} \end{cases} \tag{2.23}$$

Azaz adott s állapotban az a akció kiválasztásának valószínűsége pontosan 1, ha az az akció kerül kiválasztásra, amelyik a legnagyobb jutalmat eredményezi. Ha ismert $q_*(s,a)$, akkor ismert az optimális stratégia is. Az optimális értékelőfüggvények rekurzívan kapcsolódnak egymáshoz a Bellman optimalitási egyenlet [61] alapján.

$$v_*(s) = \max_a q_*(s,a) \tag{2.24}$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \tag{2.25}$$

ebből behelyettesítéssel,

16. Definíció. *Bellman optimalitási egyenlet*

$$v_*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right) \tag{2.26}$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a'). \tag{2.27}$$

A Bellman optimalitási egyenlet a *max* operátor miatt nem lineáris, így közvetlenül nem megoldható.

2.2.5. Q-tanulás

A Q-tanulás a megerősítéses tanuláson belül az ún. időbeli különbség tanulás, azon belül pedig az irányítási eljárások csoportjába tartozó, modell-független eljárás. Az algoritmus kidolgozása Christopher J.C.H. Watkins nevéhez fűződik [65].

Azokat az MDP-ket, amelyeknél ismert a rendszer dinamikája és a jutalomfüggvény is, jellemzően az értékiteráció vagy a stratégia iteráció eljárásokkal oldjuk meg, amelyek a dinamikus programozás témakörébe tartoznak és modell-függőek.

Az időbeli különbség tanulás módszer lényege, hogy egy adott s állapot esetében az értékelő függvények frissítése kizárólag csak a megfigyelt szomszédos s' állapotokat veszi figyelembe. Az időbeli különbség tanulása módszernél az ágensnek nincs szüksége sem a rendszer dinamikájának, sem pedig a jutalomfüggvénynek az ismeretére, ezért ezt a módszert modell-függetlennek nevezzük. Megjegyezzük, hogy semmilyen tréning adatbázis nem szükséges, mert a kapott jutalmak alapján tanul az ágens.

Eljárás

A Q-tanulás egy irányítási eljárás, amely azt jelenti, hogy egy meglévő stratégiát igyekszik javítani. Azaz olyan π stratégiát keres, amelyre igaz a 15. Definíció. Az eljárás egy Q akcióértékelő függvényt tanul meg, azaz minden állapot esetében az ott végrehajtható akciók hasznosságát. Továbbá a Q-tanulás az optimális q_* függvényt közelíti, függetlenül attól, hogy milyen viselkedési stratégiát követ. Legyen $(S_1, A_1, R_1)(S_2, A_2, R_2)\ldots$ a megfigyelt S_t állapotok, A_t akciók és R_t jutalmak sorozata és $t = 1, 2, 3, \ldots$ diszkrét időpillanatok. Az optimális akcióértékelő függvény közelítése az alábbi szabállyal történik.

$$Q_{t+1}(S_t, A_t) = (1 - \alpha_t)Q_t(S_t, A_t) + \alpha_t(R_t + \gamma_t \max_a Q_t(S_{t+1}, a)), \tag{2.28}$$

ahol

$$Q_{t+1}(s, a) = Q_t(s, a). \tag{2.29}$$

A (2.28) kifejezésben az $\alpha \in [0, 1]$ a tanulási paraméter (vagy bátorsági faktornak is hívják), amely azt határozza meg, hogy az utódállapot becsült maximális akcióértéke, amelyet a környezetből vett minta alapján kaptunk, és az azonnali jutalom összege mekkora mértékben lesz figyelembe véve. Amennyiben az $\alpha = 0$, úgy az ágens nem tanul semmit; ha $\alpha = 1$, akkor pedig az aktuális $Q_t(s, a)$ érték teljesen felülíródik. A Q-értékek egy kétdimenziós Q mátrixban kerülnek eltárolásra, ahol a sorok jelentik az állapotokat, az oszlopok pedig az akciókat. A Q-tanulás további fontos tulajdonsága az off-policy tanulás. Ez azt jelenti, hogy a (2.28) kifejezés két stratégia mentén működik. Ebből az egyik az ún. célstratégia (π), a másik pedig a viselkedési stratégia (μ). Ebben az esetben a π stratégia a (2.28) kifejezésben a *max* operátor, amely egy mohó stratégia. Azaz, a következő állapotban az akció választásának értékét úgy becsüli, hogy a választható akciók közül a legnagyobb értékűt választja. Ez a stratégia fix, a frissítő szabály része. A μ stratégia pedig az akció kiválasztását végző stratégia, amely az eljárást implementáló személy választása (pl. ϵ-mohó, softmax ...).

Watkins a publikációjában [65] megmutatta, hogy a Q_t függvény $p = 1$ valószínűséggel konvergál a q^* optimális akcióértékelő függvényhez adott feltételek mellett.

2. Tétel. *[61] Adott, véges időhorizontú MDP esetén a Q-tanulás algoritmusa az optimális q^* függvényhez konvergál, ha*

- $\sum_t \alpha_t = \infty$ *és*
- $\sum_t \alpha_t^2 < \infty$ *és*
- *minden (s, a) pár végtelen sokszor megjelenik az $(S_1, A_1)(S_2, A_2)\ldots$ sorozatban.*

Az első két feltétel azt írja le, hogy az α_t paraméterek négyzetösszege véges, de a paraméterek nullához tartása ne legyen túl gyors. Azaz az α_t paraméterek négyzetének összege már ne végtelenhez tartson. Az utolsó feltétel pedig azt követeli meg, hogy minden akciónak az adott π stratégia mellett bármely állapotban nem lehet 0 a kiválasztási valószínűsége. Az utolsó feltételt nem a Q-tanulás frissítő szabálya vezérli, hanem egy megfelelő π stratégia. Számos stratégia létezik, amelyek közül a legnépszerűbbek az ϵ-mohó és a Boltzmann felfedezési (softmax) stratégiák. Az ϵ-mohó stratégia működése nagyon egyszerű. A stratégia szerint az ágens minden t időpillanatban $1 - \epsilon_t$ valószínűséggel a legjobbnak gondolt a akciót választja, azaz ami maximalizálja a $Q_t(S_t, a)$-t, és ϵ_t valószínűséggel egy véletlenszerű akciót választ egyenletes eloszlás mellett. A munkámban a Boltzmann felfedezési stratégiát alkalmaztam, amelyet a következő részben nagy vonalakban tárgyalok.

Boltzmann felfedezési stratégia

Az akciók kiválasztása során alkalmazott akcióválasztó stratégia két fő, időben és viselkedésben eltérő szakaszra bontható. Az egyik a felfedezés, a másik pedig a kiaknázás szakasz. A felfedező szakaszban az ágens véletlenszerűen választ akciókat, ezzel megadva az esélyt, hogy addig még nem próbált cselekvéseket is végrehajt. Ennek előnye, hogy egyre több információja lesz a környezetéről, ami hosszútávon megtérül, rövidtávon azonban nyilván veszteséget is okozhat, ha éppen olyan cselekvést választott. A másik szakasz, a kiaknázás azt jelenti, hogy a már meglévő információkra támaszkodva mindig a legjobb (mohó) akciót választja az ágens.

A Boltzmann felfedezési stratégia pl. az ϵ-mohó stratégiával szemben az egyes akciók kiválasztását eltérő valószínűségi értékek alapján végzi.

$$\mathbb{P}(a|S_t) = \frac{e^{\frac{Q(S_t,a')}{\tau_t}}}{\sum_{a'} e^{\frac{Q(S_t,a')}{\tau_t}}} \tag{2.30}$$

Az egyes akciók valószínűségének kiszámítása függ az akció becsült értékétől és a $\tau_t \in \mathbb{R}^+$ paramétertől. A felfedezés és a hasznosítás közötti átmenet a τ_t paraméter függvénye. Ha a τ_t értéke magas, akkor az akciók kiválasztása közel azonos eloszlás mellett történik. Alacsonyabb τ_t esetén ezek a valószínűségi értékek a jobb (magasabb értékű) akciók esetében nagyobbak lesznek, míg a rosszabb (alacsonyabb értékű) akciók esetében alacsonyabbak. Minél jobban közelíti a nullát, az akciók kiválasztása egyre jobban mohó lesz. A (2.30) kifejezés az a akció kiválasztásának valószínűségét adja meg az S_t állapotban a Boltzmann eloszlásfüggvény segítségével. Másképpen fogalmazva, a függvény bemenete egy vektor, amely az adott S_t állapotban elérhető akciók értékeit tartalmazza, a kimenet pedig minden akció számára egy valószínűség.

Epizodikus Q-tanulás

Egy MDP-t epizodikusnak tekintünk, ha minden egyes

$$(S_1, A_1, R_1), (S_2, A_2, R_2), \ldots, (S_n, A_n, R_n), S_{n+1} \tag{2.31}$$

epizód után újraindul a megoldás a kiinduló állapotból. Az S_{n+1} az epizód végállapota. Egy epizód vége meghatározható egy maximális lépésszámban vagy valamilyen leállási feltételként is. Az epizodikus Q-tanulás epizodikusan alkalmazza a (2.28) frissítő szabályát, azaz csak az adott epizód befejezése után történik meg a frissítés. Az általam fejlesztett algoritmus is epizodikus Q-tanulást alkalmaz. Ez azt jelenti, hogy egy következő S_{t+1} állapot nem egy akció végrehajtásával áll elő, hanem akciósorozatok végrehajtásával, amely egy teljes epizódot ölel fel. Az epizód befejeződése az állapotsorozat végrehajtásának a befejeződése, amelynek eredménye egy új S_{t+1} következő állapot.

2.3. Problémafelvetés

Ebben a fejezetben részletesen bemutatom az általam felvetett ütemezési problémát. Az ütemezési feladat típusa független gépek ütemezése megelőzési relációk figyelembevételével (*unrelated machine scheduling with precedence constraints*), amely az alábbiak szerint írható le:

$$R_m|prec|C_{max} \tag{2.32}$$

ahol 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 C_j jelöli a j. munka befejezési idejét. A j. feladat végrehajtási idejét az M_i gépen p_{ij} jelöli, a cél pedig az átfutási idő minimalizálása (kibocsátási idők nincsenek a modellben). Lenstra és szerzőtársai [66] az $R_m||C_{max}$ problémára (vagyis az általunk vizsgált feladatnak arra a speciális esetére, amikor nincsenek megelőzési relációk) egy polinomiális idejű, 2-approximációs algoritmust adtak meg. Továbbá azt is megmutatták, hogy ezen probléma esetén nem létezik olyan polinomiális idejű algoritmus, amelynek az approximációs aránya $\frac{3}{2}$-nél kisebb, feltéve, hogy $P \neq NP$. Ez már mutatja, hogy a feladat (megelőzési relációk nélkül is) nagyon nehéz, hiszen a feladatot összehasonlítva a hasonló (*uniformly related*) gépek ütemezésével, erre viszont létezik polinom idejű approximációs séma. Lenstra és szerzőtársai [66] azt is megmutatták, hogy az $R_m||C_{max}$ problémának még az a speciális esete is NP-nehéz, amikor a végrehajtási idők kétfajták lehetnek csak, vagyis $p_{ij} \in \{p, q\}$ ahol $p < q, 2p \neq q$. Lenstra és szerzőtársai [66] eredményét Shchepin és Vakhania [67] kicsit tudta javítani úgy, hogy m gép esetében $2 - \frac{1}{m}$ approximációt értek el.

Az általam felvetett problémában azonban még a tevékenységek közötti megelőzési relációk is megjelennek. Ezek a megelőzési relációk egy irányított G gráffal adhatók meg. A gráf csomópontjai a tevékenységek, az irányított élek pedig a végrehajtási sorrendet definiálják. Azaz, a megelőző tevékenységnek minden esetben előbb be kell fejeződnie, mielőtt az utódtevékenység végrehajtása elkezdődhetne. Egy gép egyszerre csak egy tevékenységet hajthat végre és a végrehajtás nem megszakítható. Továbbá, a megelőzési relációban résztvevő munkák külön gépekre is ütemezhetőek. Nincs kibocsájtási és befejezési határidő. Csak olyan relációkat vettem figyelembe, ahol minden csomópont bemenő és kimenő éleinek száma legfeljebb

egy. Ez azt jelenti, hogy a gráf diszjunkt utak és izolált csomópontok uniója.

Herrmann és szerzőtársai [68] valamint Liu és Yang [69] publikációi azok, amelyek a független gépek ütemezésével foglalkoznak úgy, hogy a modellbe beépítik a megelőzési relációkat, amelyek az én munkámhoz hasonlóan diszjunkt utak és csomópontok uniója. A [68]-ban három alsó korlát (LB_1, LB_2 és LB_3) kerül definiálásra. Az esettanulmányban 33 input végeredményét hasonlítja a szerző a három alsó korlát közül a legnagyobbhoz. A [69]-ben Liu és Yang egy hatékonyabb heurisztikát közöl, amely általánosabb problémák esetén is alkalmazható. A szakirodalomban ehhez a problémakörhöz még számos más publikáció található, azonban a probléma specialitását tekintve az [68] az, amely az általam vizsgált problémához leginkább hasonlítható, ugyanis ez olyan munka, amely kizárólag diszjunkt utak formájában veszi figyelembe a megelőzési relációkat. A [69] esetében viszont a megelőzési relációk nem feltétlenül diszjunkt utak és csomópontok uniója, ugyanis a modelljük megengedi, hogy egy tevékenység több másik tevékenységgel álljon megelőzési relációban egyszerre. A modelljükben ezt tetszőleges megelőzési relációnak nevezik (arbitrary standard precedence constraints). Liu és Yang a bemutatott algoritmust a Herrmann és szerzőtársai [68] által kidolgozott heurisztikával hasonlítják össze. Az input azonos a Herrmann munkájában bemutatottal. Az eredmények azt mutatták, hogy sikerült egy hatékony algoritmust létrehozni.

Az elmúlt években megnőtt az érdeklődés olyan új modellek iránt, amelyek a független gépek ütemezésével valamilyen módon kapcsolatosak. Alább néhány releváns publikációt szeretnék felsorolni. [70]-ben a szerzők egy nehéz problémával foglalkoznak, amelyet Team Work Scheduling-nek hívnak. Ebben a feladatban vannak munkások, amelyek egy halmazt alkotnak. Ebből a halmazból munkacsapatokat hozhatunk létre és néhány feltétel mellett adott, hogy a csapat milyen hatékony bizonyos munkák elvégzésében. Egy munkás egyszerre csak egy csapatba tartozhat. De ha ez a csapat elvégzett egy adott munkát, a munkás másik csapathoz is csatlakozhat. A csapatok uniója bármely pillanatban részhalmaza az összes munkásból álló halmaznak. Minden egyes munkára választunk egy csapatot. Az, hogy melyik csapatot választjuk, befolyásolja a munkavégzés időtartamát, pl. egy nagyobb csapat ugyanazt a munkát gyorsabban tudja elvégezni. Minden csapat egy időben csak egy munkán dolgozhat, és minden egyes munkát valamelyik csapatnak el kell végeznie. A csapatokat úgy kell megválasztani, hogy a munkák minél hamarabb készen legyenek. Ez a modell a független gépek ütemezésének az általánosítása.

Egy másik, ide vonatkozó modell az ún. Multiprofessor Scheduling [71]. Ebben a modellben adottak a professzorok, a tanársegédek és néhány oktatói munka. A feladat az, hogy ezeket a munkákat a professzorokhoz és a tanársegédekhez rendeljük meghatározott feltételek betartása mellett. Ez a probléma egy újabb általánosítása a független gépek ütemezésének, ugyanis a professzoroknak eltérő a tudásuk az egyes szakterületeket tekintve. Például az egyiknek az algebra a szakterülete, a másiknak pedig a geometria. Továbbá, néhány oktatási tevékenységben a tanársegédeknek is részt kell vennie (pl. azért, hogy megtanulja, hogyan is kell oktatni a tárgyat). A publikáció approximációs algoritmusokat és komplexitással kapcsolatos eredményeket közöl.

A Multiprofessor Scheduling problémának egy speciális esetével foglalkozik a [72], ahol az erőforrásokra vonatkozó korlátozások is megjelennek. Ez a publikáció

is approximációs algoritmusokat és komplexitással kapcsolatos eredményeket közöl.

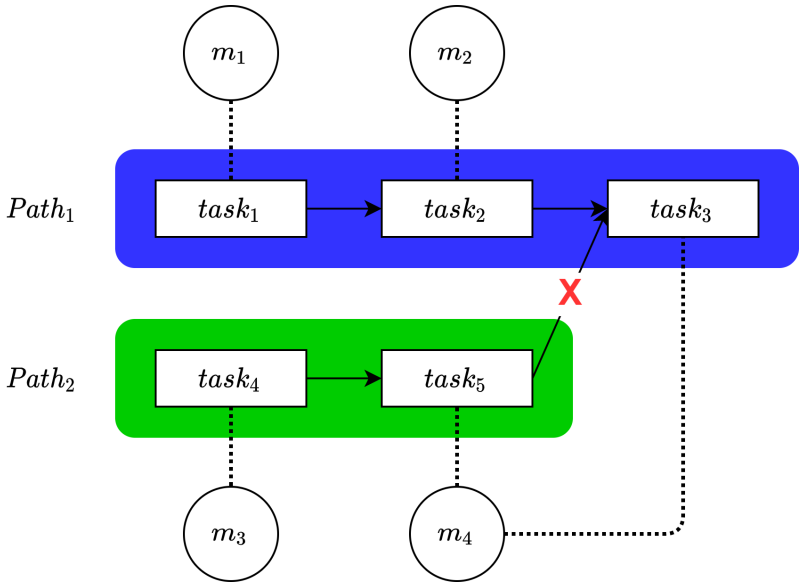
Az én esetemben a feladat modellje a

$$\langle \mathcal{T}, \mathcal{M}, G \rangle \tag{2.33}$$

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.

Továbbfejlesztési lehetőségként meg lehet vizsgálni, hogy némileg más struktúrák esetén pl. fa struktúra esetén hogyan működik a módszerünk. Természetesen ezekre is új tesztfeladatokat kellene generálnunk. Speciális esetek (például két gép vagy kevés számú lánc, vagy csak kétfajta feldolgozási idő) vizsgálata is érdekes lehet. Ezekkel a kérdésekkel a 2.3. fejezet, vagyis a jelen fejezet végén foglalkozom. Mindez további kutatásnak lehet a tárgya. Előfordulhat, hogy egy út csak egy tevékenységből áll. A csomópontok közötti irányított él pedig a megelőzési reláció. Az ütemezés célja a teljes átfutási idő minimalizálása.



2.2. ábra. Egy példa a modell alapján

A 2.2. ábrán egy egyszerű példa látható a modell alapján felépítve. Látható, hogy van egy erőforráshalmaz, azaz $\mathcal{M} = \{m_1, m_2, m_3, m_4\}$ és egy tevékenységhalmaz, azaz $\mathcal{T} = \{task_1, task_2, task_3, task_4, task_5\}$. Az erőforrások pontozott vonallal kapcsolódnak azokhoz a tevékenységekhez, amelyeket egy lehetséges ütemezés szerint végrehajtanak. Az egyes tevékenységek között megfigyelhetőek a megelőzési

relációk, pl. $task_1 \rightarrow task_2$. Ezek a relációk szigorúan meghatározzák a tevékenységek egymáshoz viszonyított végrehajtási sorrendjét, az ezektől való eltérés nem megengedett. A $task_5 \rightarrow task_3$ reláció nem megengedett, ugyanis ez megsérti a diszjunkt utak feltételét.

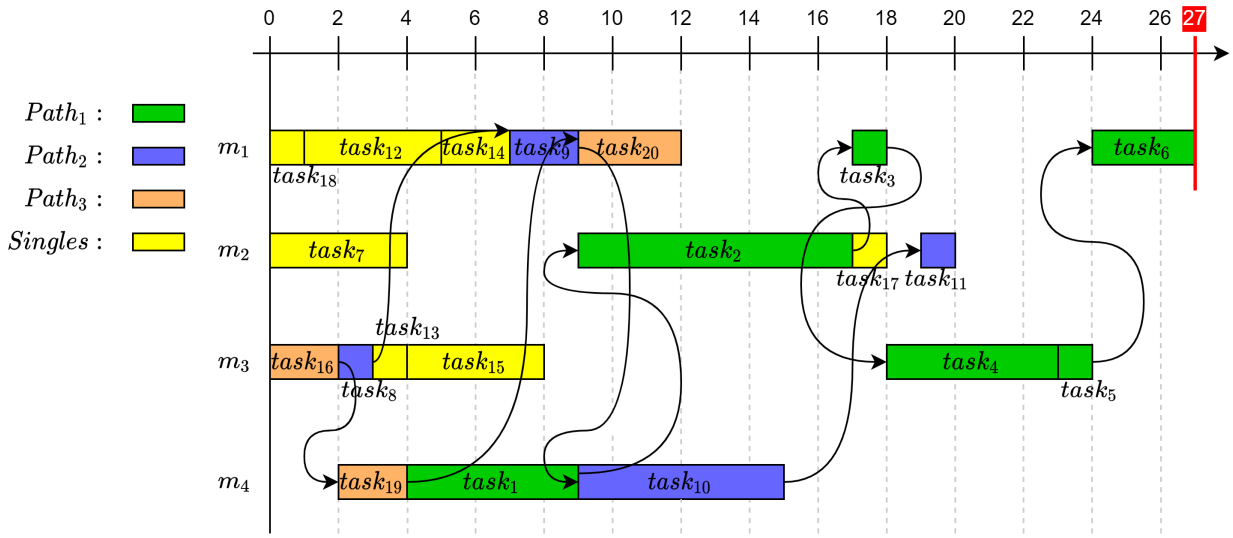
Ezen az ábrán (2.2. ábra) még nem tüntettük fel a különböző feldolgozási időket a gépeken. Egyelőre csak azt mutatja az ábra (esetleg meglehetősen ad-hoc módon), hogy már hozzá vannak rendelve a munkák a gépekhez.

A probléma illusztrálására ugyanazon példa két különböző ütemezési megoldása látható a 2.3. és a 2.4. ábrákon. A két megoldás között egy apró különbség volt, azonban a végeredményt nagyban befolyásolta. Az ábrákon a sárga kivételével az azonos színű tevékenységek között a nyilak jelzik a megelőzési relációkat. Értelemszerűen, ahonnan indul a nyíl, az a megelőző tevékenység, ahová tart, az pedig a későbbi. A feladat az alábbiak szerint írható fel.

- $\mathcal{T} = \{task_1, task_2, task_3, task_4, \ldots, task_{20}\}$
- $\mathcal{M} = \{m_1, m_2, m_3, m_4\}$
- G gráf (diszjunkt utak és izolált pontok)
  - diszjunkt utak:
    - $task_1 \rightarrow task_2 \rightarrow task_3 \rightarrow task_4 \rightarrow task_5 \rightarrow task_6$
    - $task_8 \rightarrow task_9 \rightarrow task_{10} \rightarrow task_{11}$
    - $task_{16} \rightarrow task_{19} \rightarrow task_{20}$
  - izolált pontok:
    - $task_7, task_{12}, task_{13}, task_{14}, task_{15}, task_{17}, task_{18}$

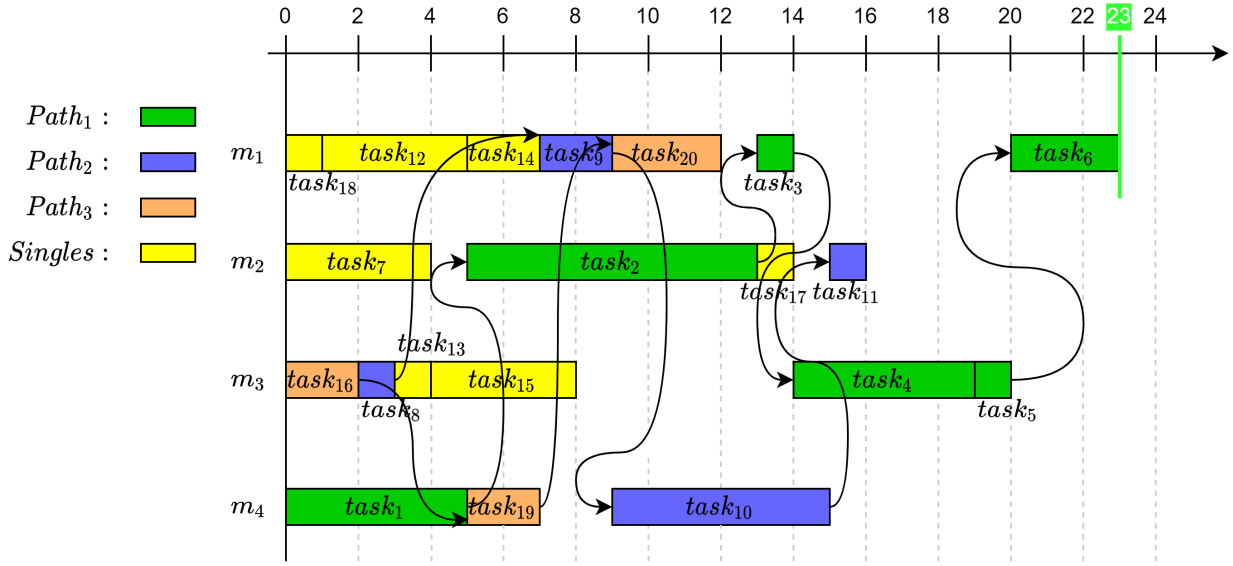
Összesen 20 tevékenység van, amelyek végrehajtására 4 gép áll rendelkezésre. A tevékenységekből felépülő utak a precedencia relációkkal vannak leírva. Az olyan utat, ami csak egy tevékenységből áll, az egyszerűség kedvéért egységesen jelöltem (sárga: $task_7, task_{12}, task_{13}, task_{14}, task_{15}, task_{17}, task_{18}$), ugyanis ezek esetében a precedencia reláció nem értelmezett, azaz nem függnek más tevékenységektől, így nem szükséges ezeket a tevékenységeket egymástól élesen megkülönböztetni. Továbbá még három lánc van (kék: $task_8 \rightarrow task_9 \rightarrow task_{10} \rightarrow task_{11}$, zöld: $task_1 \rightarrow task_2 \rightarrow task_3 \rightarrow task_4 \rightarrow task_5 \rightarrow task_6$, narancssárga: $task_{16} \rightarrow task_{19} \rightarrow task_{20}$), ahol a tevékenységek között megelőzési relációk találhatóak. A függőleges tengelyen a gépeket, a vízszintes tengelyen az időt ábrázoltam. Az egyes tevékenységek végrehajtási ideje a gépektől függ. A cél az, hogy a teljes feladat átfutási ideje minimális legyen. A tevékenységeknek a gépeken történő végrehajtási idejeit itt most nem adjuk meg, tegyük fel, hogy egy bizonyos ütemezés a következő (2.3 ábra).

A 2.3. ábrán a feladat egy megengedett megoldása látható. Ez a megoldás 27 időegységet jelentő átfutási időt mutat. Ahhoz, hogy egy jobb megoldást megkapjunk, egyszerűen fel kell cserélni a $task_{19}$ és a $task_1$ tevékenységek sorrendjét az m_4 gépen. Ennek hatására a végrehajtási idők nem változnak, azonban a precedenciák miatt mégis 4 időegységet sikerül nyerni. Ha megnézzük a zöld színnel jelölt út



2.3. ábra. Egy példafeladat megengedett ütemezése

tevékenységeit, azonnal látható, hogy ez a leghosszabb út, ami azt jelenti, hogy az átfutási időt ez határozza meg. Természetesen mindez csak az adott inputra vonatkozik, általában a helyzet sokkal bonyolultabb lehet. Az is látható, hogy a többi út (beleértve a sárga tevékenységeket is) mindegyik jócskán a zöld út előtt már befejeződik, így ezeknek nincs túl sok jelentőségük az átfutási idő tekintetében. Azonban (lásd: 2.4. ábra), ha a két említett tevékenységet felcseréljük az m_4-es gépen, úgy a teljes zöld út átfutási ideje 4 időegységet fog csökkenni az időtengelyen való balratolódás miatt. Megjegyezzük, hogy a feladatra vonatkozó alsó korlátok kérdésével pl. a 2.5. alfejezetben részletesebben foglalkozunk.



2.4. ábra. A példafeladat egy másik lehetséges ütemezése

A 2.4. ábrán már egy jobb megoldás látható, ami 23 időegység. Mivel a $task_1$ tevékenység az m_4-es gépen legelöl került a sorba, így a tőle közvetlenül és közvetet-

ten függő többi zöld tevékenység is eltolódott balra. Látható ebből a példából, hogy az ütemezés során a tevékenységek beütemezési sorrendje kritikus pont lehet. Egy egyszerű sorrendcsere két tevékenység között egy megengedett megoldásból rögtön jobb megoldást generált.

Az előző ábrákon (2.3., 2.4. ábrák) láthatjuk, hogy ha van egy hosszú utunk, amelyet mindenütt a leggyorsabb gépre ütemezünk, az egy alsó korlát a feladatra. És ha ráadásul az is teljesül, hogy amikorra a hosszú útban szereplő munkák befejeződnek, addig a többi munka is elkészül, akkor a makespan-t ez a hosszú út adja, tehát ez felső korlát is a feladatra. Amint látjuk, a többi munkát jó koránra ütemezve a végére már alig marad. Ez persze csak egy speciális input volt szemléltetés végett, az általános eset ennél sokkal bonyolultabb lehet.

A következő fejezetben egy olyan általam kidolgozott módszert mutatok be, amely a tevékenységeket mohó módon, az előre meghatározott sorrendjük szerint ütemezi. A tevékenységek sorrendjének kiszámítását pedig egy, a megerősítéses tanulás területéről ismert algoritmus által inspirált eljárás végzi.

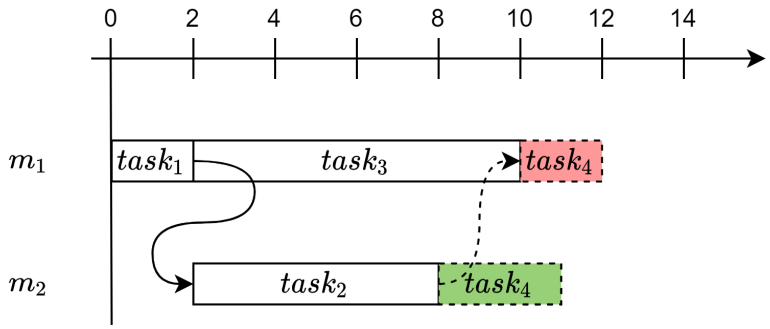
Megjegyzendő, hogy az algoritmus a fentebb megadott megelőzési reláció helyett más struktúrákra (pl. fa) is működőképes lenne. Annyi különbség van csak, hogy a fa (illetve erdő) típusú megelőzési relációk esetén a sorban következő munka esetén több olyan munka is lehet (nem csak legfeljebb egy), amelyek az ő közvetlen megelőzői, és csak ezek befejezése után szabad az aktuális feladatot elkezdeni. Ez a kutatási irány további vizsgálatokat igényel, emiatt későbbi kutatás tárgya.

2.4. A javasolt módszer

A 2.3. alfejezetben felvetett ütemezési probléma megoldására egy, a megerősítéses tanuláson belül ismert Q-tanulási eljárás által támogatott heurisztikus algoritmust fejlesztettem. Az eljárást Q-Learning Motivated Algorithm-nek (QLM) neveztem el. Ebben az alfejezetben az algoritmus működését mutatom be.

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 ennek a permutációnak 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 2.5. ábrán egy egyszerű szemléltető példa látható a mohó algoritmus működésére. Tegyük fel, hogy a $task_1, task_2$ és $task_3$ tevékenységek már ütemezésre kerültek. Most a $task_4$ következik, amely esetében figyelembe kell venni a $task_1 \rightarrow task_2 \rightarrow task_4$ relációt. A $task_4$ tevékenység nélkül az addig elért átfutási idő 10 egység. A megelőzési reláció figyelembevételével azt a gépet kell választani, ahol az eddig elért átfutási idő minimálisan növekszik. Az m_1 gépen a $task_4$ végrehajtási ideje 2 időegység. Az m_2 gépen 3 időegység. A jó választás az m_2 gép, annak ellenére, hogy ott a tevékenység végrehajtási ideje (3) nagyobb mint az m_1 gép esetén (2). Ugyanis ezzel a választás-

sal a teljes, addig elért átfutási idő csak 1 időegységgel növekszik. Az m_1 esetében viszont a kisebb végrehajtási idő ellenére is 2 időegységet növekedne. Azaz, a $task_4$ mindkét gépre tehető: az m_1 gépen így a teljes átfutási idő 12 egység lesz, az m_2 gépen pedig 11 egység.



2.5. ábra. A mohó algoritmus működése

Látható tehát, hogy a mohó algoritmus nem az alapján választ, hogy melyik gépen a legkisebb a végrehajtási ideje az adott tevékenységnek, hanem az alapján, hogy az adott gépre ütemezve a tevékenységet az addig elért átfutási idő mennyivel növekszik meg.

Algorithm 1: A mohó algoritmus működése

Result: Átfutási idő
Input: L_t permutáció
$t_{sum} \leftarrow 0$
$\mathcal{M}$ - erőforrások halmaza
L_t - tevékenységek egy permutációja
$min \leftarrow maxInt$;
$selectedRes$;
for $i \leftarrow 0$ **to** n **do**
  for $j \leftarrow 0$ **to** m **do**
    if $m_j(task_i) + t_{sum} < min$ **then**
      $min \leftarrow m_j(task_i) + t_{sum}$;
      $selectedRes \leftarrow j$;
    end
  end
  az i. tevékenység hozzárendelése az $selectedRes$ erőforráshoz
  $t_{sum} \leftarrow t_{sun} + m_{selectedRes}(task_i)$;
  $min \leftarrow maxInt$;
end
Átfutási idő kiszámítása az ütemezés alapján

Graham [2, 3] által tekintett modell a $P_m|prec|C_{max}$, ami különbözik az általam vizsgált feladat modelljétől.

A tevékenységek permutációjának előállítása az [53] publikációban közölt öt-

letek alapján történik. Az általam fejlesztett algoritmus implementációja az [53] cikkben bemutatott logikát követi (lásd a [73] cikket is). Az implementált algoritmus azonban Stefán munkájával (flow-shop scheduling) ellentétben egy teljesen más, speciális feladat (unrelated machine scheduling) megoldására készült, amely egyrészt tevékenységek végrehajtási idejével dolgozik, másrészt pedig kezeli a megelőzési relációk által leírt megszorító szabályokat. Stefán munkájában a flow shop ütemezéshez kapcsolódóan a tevékenységek egy olyan sorrendjének kiszámításával foglalkozott, amelynek eredményeképpen a végrehajtás közben keletkező üresjáratok ideje minimálisra csökkenthető. Ennek a sorrendnek a kiszámításához a megerősítéses tanulást, azon belül pedig a Q-tanulást alkalmazta. A megoldás lényege, hogy minden $task_i$ és $task_j$ párra az eljárás kiszámítja a $Q_t(i,j)$ értéket a (2.28) szabály segítségével, amely azt reprezentálja, hogy a kiszámítandó permutációban mennyire hasznos, ha $task_i$ tevékenységet a $task_j$ tevékenység követ. Másképpen fogalmazva, annak az akciónak a hasznosságát keresi az eljárás, amely szerint a $task_i$ után a sorrend következő tagjának a $task_j$ tevékenységet választjuk. A QLM algoritmusnál a megállási feltétel 2000 iteráció volt. A vizsgálatok alapján 2000 iteráció felett már nem javult az előállított megoldás.

Az algoritmus az alábbi öt lépésben összegezhető, amelyeket a következő alfejezetekben részletesen bemutatok.

Algorithm 2: QLM algoritmus

Result: Ütemezett tevékenységek
Input: $\mathcal{M}$, $\mathcal{T}$, *precConstraints*, *epoch* (iterációk száma)
1 **for** $i \leftarrow 0$ **to** *epoch* **do**
2 Permutáció meghatározása (Algorithm 3 és Algorithm 4)
3 Mohó algoritmussal az ütemezés megvalósítása és az átfutási idő kiszámítása (Algorithm 1)
4 A Q-mátrix értékeinek frissítése (Algorithm 5)
5 **end**

2.4.1. A tevékenységek egy permutációjának generálása

A tevékenységek permutációjának generálása a (2.30) Boltzman felfedezési stratégia alapján történik. Ahhoz, hogy a sorrendet az eljárás generálni tudja, már rendelkeznünk kell Q-értékekkel. Ez már az első lépésben biztosítva van, ugyanis a Q-értékek tárolására alkalmazott mátrix kezdeti értékei nullák, azaz $Q_t(i,j) = 0, \forall i,j$. Legyen $L_t = (i_1, \ldots, i_n)$ a tevékenységek permutációjának indexhalmaza a t. epizód után. Továbbá legyen $H = \{j_1, \ldots, j_l\}$ azon tevékenységek indexhalmaza, amelyek valamelyik megelőzési relációban részt vesznek és a megelőzési relációk függvényében a $task_j$ tevékenység kiválasztása csak akkor lehetséges, ha a $task_{i_1}, \ldots, task_{i_k}$ tevékenységek, mint megelőző tevékenységek már szerepelnek az L_t listában. Amennyiben olyan $task_i$ tevékenységről van szó, amely nem vesz részt megelőzési relációban, azaz $i \notin H$, az automatikusan választható.

Algorithm 3: A permutáció első elemének a kiválasztása

Result: L_t permutáció első eleme
Input: Q, B, D, τ
1 Q - Q mátrix
2 $B \leftarrow \emptyset$ - a választható tevékenységek indexhalmaza
3 $D \leftarrow \emptyset$ - a beválasztott tevékenységek indexhalmaza
4 τ - a hőmérséklet
5 **for** $i \leftarrow 0$ **to** $n-1$ **do**
6 **if** $i \notin H$ ***or*** *($i \in H$* ***and*** *$task_i$ az első elem a láncban)* **then**
7 $B \leftarrow B \cup \{i\}$;
8 **end**
9 **end**
10 **for** $i \leftarrow 0$ **to** $n-1$ **do**
11 **if** $i \in B$ **then**
12 $p_i \leftarrow \frac{e^{\frac{Q(i,i)}{\tau}}}{\sum_{i \in B} e^{\frac{Q(i,i)}{\tau}}}$
13 **end**
14 **end**
15 A permutáció első elemének kiválasztása a p_i értékek alapján;
16 i hozzáadása az L_t listához;
17 $selectedTaskIndex \leftarrow i$;
18 $B \leftarrow B \setminus \{i\}$;
19 $D \leftarrow D \cup \{i\}$;

Az 3. algoritmus feladata a tevékenységek permutációjának első elemét kiválasztani. Az algoritmus bemenetei (1-4 sorok) a kezdetben nullára inicializált Q mátrix, a választható tevékenységek B indexhalmaza, a beválasztott tevékenységek D indexhalmaza, amelyek kezdetben üresek és a τ hőmérséklet paraméter. Első lépésben (5-9 sorok) az eljárás az összes tevékenység közül kiválasztja azok indexeit, amelyek a permutációba beválaszthatóak. Ez azt jelenti, hogy az adott tevékenység nem vesz részt megelőzési relációban, vagy ha igen, akkor a lánc legelső eleme. Ezek a tevékenységek a B indexhalmazba kerülnek. A második lépésben (10-14 sorok) azon tevékenységekhez, amelyeknek az indexei a B halmazban vannak, a Boltzmann felfedezési stratégia szerint kiszámolja a választási valószínűségeket. Harmadik lépésben (15-19 sorok) a kiszámolt p_i valószínűségek mentén kiválaszt egy tevékenységet és annak indexét hozzáadja az L_t listához, mint első elem. A kiválasztott tevékenység indexét elmenti a $selectedTaskIndex$ változóba, törli az indexet a B halmazból, ugyanis ez a tevékenység már nem választható és hozzáadja a már beválasztott tevékenységek D indexhalmazához.

Algorithm 4: A permutáció maradék elemeinek a kiválasztása

Result: Az összes tevékenységek L_t permutációja
Input: Q, B, D, τ

```
1  Q - Q mátrix
2  B ≠ ∅ - a választható tevékenységek indexhalmaza
3  D ≠ ∅ - a beválasztott tevékenységek indexhalmaza
4  τ - a hőmérséklet
5  while length of L_t < n do
6      for j ← 0 to n − 1 do
7          if j ∉ D and (j ∉ H or (j ∈ H and task_j-t megelőző tevékenységek
             (ha vannak) már kiválasztásra kerültek)) then
8              B ← B ∪ {j};
9          end
10     end
11     for j ← 0 to n − 1 do
12         if j ∈ B then
13             p_j ← e^(Q(selectedTaskIndex,j)/τ) / Σ_{j∈B} e^(Q(selectedTaskIndex,j)/τ)
14         end
15     end
16     A permutáció következő elemének kiválasztása a p_j értékek alapján;
17     j hozzáadása az L_t listához;
18     selectedTaskIndex ← j;
19     B ← B \ {j};
20     D ← D ∪ {j};
21 end
22 Mohó ütemezés az L_t permutáció szerint;
```

A 4. algoritmus a fennmaradó tevékenységek kiválasztását végzi, logikailag a 3. algoritmus folytatása. A szétválasztás oka a jobb átláthatóság. Az algoritmus bemenete ugyanaz, mint az 3. algoritmus esetében, azonban látható, hogy mivel már a permutáció első eleme kiválasztásra került, így a B és D halmazok nem üresek. A B halmaz tartalmazza a választható tevékenységek indexeit, a D halmaz pedig a beválasztott első tevékenység indexét.

Mivel itt az összes fennmaradó tevékenységet beválasztja az eljárás, így a teljes művelet egy *while* ciklusban foglal helyet, amely addig fut, amíg az L_t lista számossága kisebb, mint a tevékenységek száma, azaz n. A ciklus első lépésében (6-10 sorok) az algoritmus frissíti a B halmazt, azaz a választható tevékenységek indexeit. Erre azért van szükség minden egyes iteráció elején, mert a tevékenységek beválasztása új választható tevékenységeket eredményez. Tehát, pl. ha egy lánc első eleme beválasztásra kerül, úgy a következő iterációban a második eleme választhatóvá válik és így tovább. Második lépésben (11-15 sorok) minden, az aktuális iterációban választható tevékenységhez az algoritmus hozzárendel egy valószínűségi értéket a Boltzmann felfedezési stratégia alapján. Harmadik lépésben (16-20 sorok) megtörténik a kiválasztás, j index hozzáadása az L_t listához és elmentése a

$selectedTaskIndex$ változóban. Majd az index törlődik a B halmazból, és hozzáadásra kerül a D halmazhoz.

Az eljárás kimenete az összes tevékenység $L_t = (i_1, \ldots, i_n)$ permutációja, azaz egy sorrendje, amelynek számítása a Q mátrix alapján történt. Az L_t permutáció kiszámítása után következik a mohó algoritmus alkalmazása, amely a tevékenységeket a permutációnak megfelelő sorrendben ütemezi be az Algorithm 1 mohó algoritmus szerint.

2.4.2. Q-értékek kiszámítása

A Q-tanulás és a bemutatott ütemezési probléma fúziójában egy állapotnak a tevékenységek egy teljes L_t permutációja felel meg, az akcióknak pedig egy tevékenység permutációba való beválasztása a megelőzési relációkat figyelembe véve.

Ezt a Q-tanulást epizodikusnak nevezzük, ugyanis egy következő S_{t+1} állapot kiszámításához az epizód végéig kell várakozni, azaz addig, amíg a permutáció minden eleme nem kerül meghatározásra, majd az új állapot alapján frissíteni kell a Q mátrix értékeit. A frissítéshez a (2.28) szabályt alkalmazza az eljárás. Az alábbiakban bemutatom az értékek frissítésének menetét és a jutalom értékének kiszámítását.

Tegyük fel, hogy az algoritmus permutációt generáló szakaszában a következő $L_t = (task_{t,2}, task_{t,4}, task_{t,1}, task_{t,3})$ permutáció adódott. A négy tevékenység között a $task_{t,2} \to task_{t,3}$ és a $task_{t,4} \to task_{t,1}$ megelőzési relációk vannak definiálva. A t epizódban a Q mátrix állapota legyen a következő.

Q	$task_1$	$task_2$	$task_3$	$task_4$
$task_1$	2.21	6.67	1.13	0
$task_2$	3.39	5.53	8.81	7.76
$task_3$	2.24	0	1.17	5.90
$task_4$	3.33	5.58	4.02	7.48

2.1. táblázat. Példa Q mátrix értékekkel

A 2.1. táblázatban a pirossal jelölt cellák értékei soha nem frissülnek a definiált megelőzési relációk miatt. Azaz, soha nem lehet olyan permutációt generálni, amelyben a megelőzési relációk sérülnek, azaz, hogy $task_{t,3}$ megelőzi $task_{t,2}$ tevékenységet, vagy $task_{t,1}$ megelőzi $task_{t,4}$ tevékenységet. Az R_t jutalom a t epizódban az alábbi függvénnyel van definiálva.

$$R_t = \begin{cases} -1 & \text{ha } z_t > Z \\ 0 & \text{ha } z_t = Z \\ 10 & \text{ha } z_t < Z \end{cases} \tag{2.34}$$

ahol Z az eddigi legjobb átfutási idő, z_t pedig a t epizódban az ütemezés által adott átfutási idő. Az $\alpha = 0.8$ és a $\gamma = 0.7$.

A jutalmazási stratégia paraméterei a vizsgálatok alapján csak enyhén befolyásolják a nehéz feladatok megoldását. A megválasztásuk egyrészt próbálgatással történt, másrészt pedig szimulált hűtéssel. Az utóbbi megoldás során a paramétereket véletlenszerűen változtattam 1 egységgel. A változtatás iránya is véletlenszerűen

történt, de úgy, hogy a büntetés értéke negatív maradjon, a jutalom pedig pozitív. A megválasztott paraméterekkel ezután az algoritmus előállított egy megoldást. Ha ez jobb volt, mint az addigi legjobb, akkor az új paraméter-beállítás rögzítésre került, ellenkező esetben egy adott valószínűséggel elfogadta a rosszabb megoldást is. Tapasztalatunk szerint a szimulált hűtés alkalmazása nem javított lényegesen a megoldáson, emiatt maradtam az egyszerű, előre beállított paramétereknél. A paraméterek optimalizálásával a [74]-ben foglalkoztam.

Algorithm 5: A Q mátrix értékeinek frissítése

Result: Frissített Q értékek az L_t permutáció alapján
Input: Q, L_t

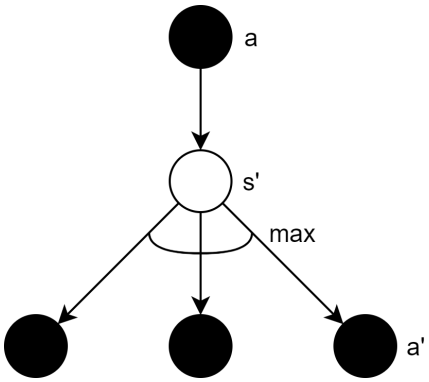
```
1  Q - Q mátrix
2  L_t - permutáció a t epizódban
3  for i ← 0 to n − 2 do
4      if i + 1 < n − 1 then
5          maxQ ← Q[L_t[i + 1]][L_t[i + 2]];
6      else if i + 1 == n − 1 then
7          maxQ ← 0;
8      for j ← i + 2 to n − 1 do
9          if Q[i + 1][j] > maxQ  and  prec. szabályok nem sérülnek then
10             maxQ ← Q[L_t[i + 1]][L_t[j]];
11     end
12     Q mátrix értékeinek frissítése a (2.28) szabállyal;
13 end
14 maxQ ← Q[L_t[0]][L_t[0]];
15 A permutáció első eleméhez tartozó Q érték frissítése a (2.28) szabállyal;
```

A 5. algoritmus feladata, hogy a Q mátrix értékeit frissítse a (2.28) szabály alapján. Az algoritmusnak két bemenete van; a frissítendő Q mátrix, és az előző lépésben kiszámított L_t permutáció. A kimenet pedig a frissített Q mátrix lesz. Az algoritmust a fentebb bevezetett példán keresztül mutatom be a könnyebb érthetőség miatt. Az algoritmus 4-11 soraiban történik a frissítő szabály max operátora által definiált művelet végrehajtása. A max operátor segítségével a Q-tanulás az aktuális állapotban kiválasztott akció hatására kapott következő állapotban megbecsüli a legjobb akciót, azaz a választható akciók közül azt választja, amelynek az értéke maximális.

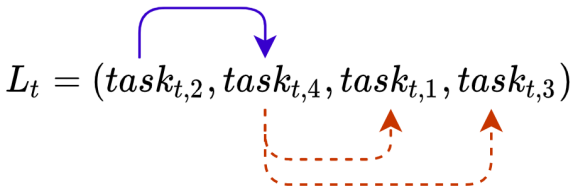
A 2.6. ábrán a max operátor vizualizációja látható. Egy adott állapotban az a akció kerül kiválasztásra, amely egy s' következő állapotba viszi az ágenst. A (2.28) szabályban található max operátor az s' utódállapotban visszaadja az ott választható akciók közül a legnagyobb értékűt. Ezzel a Q-tanulás mohó módszerrel igyekszik becsülni az aktuális s állapotban választott a akció jóságát.

A $task_{t,2}$ és $task_{t,4}$ egymásutániságának az értéke

Legyen az aktuális legjobb átfutási idő $Z = 15$, és legyen az aktuális L_t permutáció alapján kapott új ütemezés $z_t = 12$. Mivel $z_t < Z$, ezért $R_t = 10$. Az adott L_t



2.6. ábra. Q-tanulás *max* operátorának működése



2.7. ábra. A $task_{t,2}$ és $task_{t,4}$ sorrend Q értékének a számítása

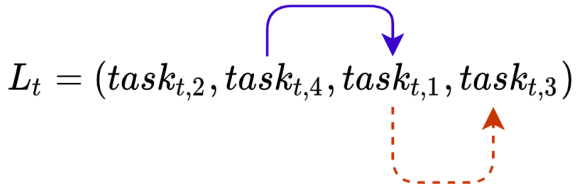
permutáció esetében az első lépés a $task_{t,2}$ és $task_{t,4}$ tevékenységek egymásutániságát leíró Q érték kiszámítása. A sorrendet a 2.7. ábrán a kék nyíl szimbolizálja. A két narancssárga nyíl pedig a *max* operátor vizualizációja. Azaz, ha a sorrendben a $task_{t,2}$ után a $task_{t,4}$ tevékenység következik, akkor a frissítő szabály értelmében ki kell számítani azt is, hogy a $task_{t,4}$ tevékenység után milyen tevékenységek következhetnek a sorrendben (természetesen a precedenciákat nem megsértve), és ezek közül melyik tevékenység választása (akció) adja a maximális értéket. A példában a $task_{t,4}$ tevékenység után következhet majd a $task_{t,1}$ és a $task_{t,3}$ is.

Az algoritmus elsőként a *max* operátor értékét számítja ki. Ehhez tekintsük a 2.1. táblázatban példaként megadott Q mátrixot. Ebben az esetben az algoritmusnak azt kell eldöntenie, hogy a $task_{t,4}$ után melyik tevékenység választása adja a nagyobb akcióértéket. Ehhez a Q mátrixban a ($task_{t,4}$, $task_{t,1}$) és a ($task_{t,4}$, $task_{t,3}$) cellákat kell kiolvasni, és a maximálisat kiválasztani. Azaz $max((task_{t,4}, task_{t,1}), (task_{t,4}, task_{t,3})) = max(3.33, 4.02) = 4.02$. Tehát a $maxQ = 4.02$. Ezután következik a frissítő szabály alkalmazása.

$$Q_{t+1}(task_{t,2}, task_{t,4}) = (1 - 0.8) * 7.76 + 0.8 * (10 + 0.7 * 4.02) = 11.8032$$

A $task_{t,4}$ és $task_{t,1}$ egymásutániságának az értéke

A második lépésben az algoritmus a permutáció következő, egymás utáni párját vizsgálja. Azaz annak az akciónak a hasznosságát, hogy $task_{t,4}$ után a $task_{t,1}$ következik. Ahogy az előző lépésben is, itt is a *max* operátor által meghatározott érték kinyerése az első. Ebben az esetben a $task_{t,1}$ után már csak egy tevékenység maradt,

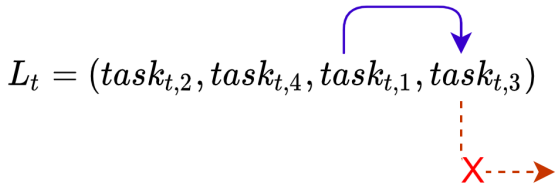


2.8. ábra. A $task_{t,4}$ és $task_{t,1}$ sorrend Q értékének a számítása

így a max operátor által visszaadott érték a $(task_{t,1}, task_{t,3}) = 1.13$ lesz. Tehát a $maxQ = 1.13$. Ezután a $Q_{t+1}(task_{t,4}, task_{t,1})$ értékének számítása következik.

$$Q_{t+1}(task_{t,4}, task_{t,1}) = (1 - 0.8) * 3.33 + 0.8 * (10 + 0.7 * 1.13) = 8.6328$$

A $task_{t,1}$ és $task_{t,3}$ egymásutániságának az értéke



2.9. ábra. A $task_{t,1}$ és $task_{t,3}$ sorrend Q értékének a számítása

Harmadik lépésben a permutáció utolsó egymás utáni párját vizsgálja az algoritmus, azaz annak az akciónak a hasznosságát, hogy $task_{t,1}$ után a $task_{t,3}$ kerül kiválasztásra. Mivel a $task_{t,3}$ tevékenység az utolsó a permutációban, így nincsenek következő kiválasztható tevékenységek. Emiatt a max operátor nem használható már, tehát a $maxQ = 0$. Ezután a $Q_{t+1}(task_{t,1}, task_{t,3})$ értékének számítása következik.

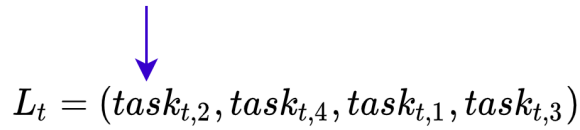
$$Q_{t+1}(task_{t,1}, task_{t,3}) = (1 - 0.8) * 1.13 + 0.8 * (10 + 0.7 * 0) = 8.2260$$

A $task_{t,2}$ tevékenység, mint első elem Q értékének kiszámítása

A Q értékek frissítésének befejező lépése az, hogy a permutáció első elemére az algoritmus kiszámolja annak Q értékét. Azaz azt az értéket, amely leírja, hogy az adott tevékenységet tekintve (a példa esetében a $task_{t,2}$) mennyire hasznos, hogy a sorrend első tagja.

A $maxQ$ értéke ekkor a $(task_{t,2}, task_{t,2})$ értékkel lesz egyenlő, amely nem más, mint a $task_{t,2}$ tevékenység önmagával vett Q értéke. Azaz $maxQ = 5.53$. Ezután a $Q_{t+1}(task_{t,2}, task_{t,2})$ értékének számítása következik.

$$Q_{t+1}(task_{t,2}, task_{t,2}) = (1 - 0.8) * 5.53 + 0.8 * (10 + 0.7 * 5.53) = 12.2028$$



2.10. ábra. A $task_{t,2}$ (mint a permutáció első eleme) Q értékének a számítása

Tehát az adott L_t permutáció alapján az algoritmus frissítette a Q értékeket a mátrixban, amely ezek után az alábbiak szerint néz ki.

Q	$task_1$	$task_2$	$task_3$	$task_4$
$task_1$	2.21	6.67	8.2260	0
$task_2$	3.39	12.2028	8.81	11.8032
$task_3$	2.24	0	1.17	5.90
$task_4$	8.6328	5.58	4.02	7.48

2.2. táblázat. Példa frissített Q mátrix értékekkel

2.5. Eredmények

Emlékeztetek arra, hogy az általam kifejlesztett algoritmus két komponensből áll. A Q-tanulás alapú keret megállapítja a munkáknak egy sorrendjét és utána a munkákat ebben a sorrendben ütemezzük a klasszikus LS algoritmussal természetesen a megelőzési relációkat is figyelembe véve. Megjegyzem, hogy sok más lehetőség is lenne a tárgyalt megoldáson kívül.

Elsőként a [68]-ban és a [69]-ben közölt kis méretű feladatot oldottam meg, amely összesen hét tevékenységből ($task_1$–$task_7$), két gépből (m_1, m_2) és három megelőzési relációból áll, továbbá a gépi idők is ismertek. A megelőzési relációkat megadó láncok a következők:

- $task_1 \rightarrow task_3 \rightarrow task_7$
- $task_2 \rightarrow task_6$

A kis méretű példában használt gépi idők a [68] és [69] cikkekből származnak.

A fenti példát optimálisan oldották meg a [69]-ben. Az optimális megoldás 13. Azonban a [68]-ban alkalmazott heurisztikus eljárás megoldása csak 15 lett. A QLM algoritmusom szintén megtalálta az optimális megoldást.

Megjegyezzük, hogy ez a feladat akár "kézzel" megoldható, mert viszonylag kevés kombináció jöhet szóba. Ha mindegyik munkát arra a gépre tesszük, ahol rövidebb idő alatt végrehajtható, először az m_1 gépnek 17 lesz az átfutási ideje, az m_2 gépnek kevesebb. Ha tudjuk, hogy 13 a feladat optimum értéke, akkor elég a $task_2$ munkát az m_1 gépről áttenni az m_2 gépre. Ha a megelőzési relációkat nem vesszük figyelembe, máris optimális ütemezést kaptunk. A gépek átfutási ideje 13 illetve 11. Ha azt

	m_1	m_2
$task_1$	3	9
$task_2$	4	5
$task_3$	8	2
$task_4$	2	6
$task_5$	5	10
$task_6$	9	4
$task_7$	3	8

2.3. táblázat. A kis méretű példában használt gépi idők

akarjuk, hogy a megelőzési relációk is teljesüljenek, ehhez elég a gépeken a munkákat megfelelő sorrendbe tenni.

Általában viszont könnyen lehetséges az, hogy az az ütemezés, amelyik a megelőzési relációkat nem veszi figyelembe nem javítható ki olyanná, amelyik esetén a megelőzési relációk is teljesülnek, pusztán azáltal, hogy az egyes gépeken a munkák sorrendjét megcseréljük. Tehát ebből a szempontból ez a kisméretű feladat nem jellemző, de mindenképp fontosnak tartottam ennek a feladatnak a megoldását is, mert az említett cikkek is megoldják ezt a feladatot. Érdekes, hogy a [68] cikkbeli algoritmus erre a könnyű feladatra se találta meg az optimális megoldást.

Az általam fejlesztett algoritmus hatékonyságának vizsgálatára további feladatok megoldására került sor. Ehhez a [68]-ban és a [69]-ben megoldott további feladatokat vettem alapul. A [68]-ban 33 ütemezési problémát oldottak meg a szerzők, azonban a feladatok részletei, mint a gépi idők és a megelőzési relációk, egyáltalán nem kerültek közlésre, csak a gépek, tevékenységek és a megelőzési relációk száma. A [69]-ben a problémákat osztályokba sorolták a gépek száma, a tevékenységek száma és a megelőzési relációk száma alapján. Azonban a gépi idők és a megelőzési relációk ebben az esetben sem ismertek. Emiatt a gépi időket és a megelőzési relációkat én hoztam létre. A gépi idők generálása véletlenszerűen történt diszkrét, egyenletes eloszlás szerint. A megelőzési relációk megadása is véletlenszerű volt. Mivel a cikkekben megoldott feladatok részletei nem kerültek közlésre, így nem tudtam pontosan ugyanazokat a gépi időket és megelőzési relációkat alkalmazni. Próbáltam felvenni a kapcsolatot a szerzőkkel mindkét cikk esetében a feladatok konkrét adatainak megismerése céljából, de sajnos egyik esetben sem kaptam választ a megkeresésre.

A módszer teszteléséhez a [68]-ban található 33 problémából hármat választottam úgy, hogy ezek a rendelkezésre álló adatok (gépek száma, tevékenységek száma és a megelőzési relációk száma) alapján különbözzenek egymástól. A feladatok [68]-ban használt számozását megtartva a #1, #2 és #5 került kiválasztásra. Ezek kis méretű feladatok. A [69]-ből pedig egy nagy méretű feladatot választottam, az eredeti számozás szerint a #28-at. Az alábbi táblázatban megadom a feladatokhoz tartozó, a fenti cikkekből származó adatokat.
A 2.4. táblázatban n a tevékenységek, m a gépek és NC (number of chains) a megelőzési relációk száma. Ezen adatok alapján négy osztályt hoztam létre.

- *Class* #1 $\rightarrow n = 14$, $m = 8$ és $NC = 5$
- *Class* #2 $\rightarrow n = 28$, $m = 7$ és $NC = 8$

	#1	#2	#5	#28
n	14	28	27	74
m	8	7	4	19
NC	5	8	1	10

2.4. táblázat. A kiválasztott feladatok ismert adatai

- *Class* #3 $\rightarrow n = 27$, $m = 4$ és $NC = 1$
- *Class* #4 $\rightarrow n = 74$, $m = 19$ és $NC = 10$

Minden feladatosztályhoz generáltam 10 db feladatot a megfelelő gép, tevékenység és megelőzési reláció számmal. A gépi idők generálása egyenletes eloszlás mellett történt az $\{1, 2, \ldots, 10\}$ halmazból. Továbbá az eredmények egyszerűbb elemzéséhez két alsó korlátot számoltam, amelyek számítási módszerét a [68]-ból vettem át és LB_1-gyel és LB_2-vel jelölöm a cikk alapján. Az LB_1 alsó korlát segítségével a problémában a megelőzési relációk által leírt láncokat vizsgálom. Ez az alsó korlát azt adja meg, hogy ha az egyes láncokban szereplő tevékenységekhez a végrehajtási idő szerint leggyorsabb gépet rendeljük, mekkora a végrehajtásban legtovább tartó lánc átfutási ideje. Az LB_2 esetében pedig minden tevékenységhez azt a gépet rendeljük, amely a tevékenységet a leggyorsabban hajtja végre, ezeknek a minimumoknak vesszük az összegét és elosztjuk a gépek számával. A két alsó korlát informatívnak tekinthető a következők miatt. Egyrészt, ezek egyszerűen kiszámíthatók, emiatt könnyen alkalmazhatóak. Ezeknél jobb alsó korlátot nem találtam az irodalomban. Magunk is igyekeztünk ezeknél erősebb alsó korlátot előállítani, azonban nem sikerült. Ennek az az oka, hogy a munkák $p(i, j)$ végrehajtási idejei változatosak, nem találtam semmilyen egyszerű szabályszerűséget köztük, nem is lehet ilyet találni, mert egyenletes eloszlás szerint, véletlenszerűen vannak a végrehajtási idők megválasztva. Továbbá, eléggé erős alsó korlátokról van szó, amit az is bizonyít, hogy sok esetben az algoritmus által talált megoldás értéke megegyezik, vagy majdnem megegyezik az LB_1 és LB_2 alsó korlátok maximumával, és sok esetben ez a maximum erősebb alsó korlátot biztosít, mint a CPLEX megoldó által szolgáltatott alsó korlát. Ezen okok miatt, az LB_1 és LB_2 alsó korlátok használatát alkalmaztam. Megjegyezhető, hogy bizonyos feladatosztály esetén az LB_1, mások esetén az LB_2 szolgáltatja az erősebb alsó becslést, de van olyan feladatosztály is, amikor néha az egyik, néha a másik adja az erősebb becslést. Továbbá a probléma speciális eseteire semmilyen felső korlát nem ismert a legjobb tudásom szerint.

Minden feladat esetén a QLM algoritmus tíz, egymástól független futást hajtott végre, így minden egyes problémára tíz eredmény született. Az algoritmuson belül egy futás esetén a Q-tanuló fázis iterációszáma (azaz az epizódok, vagy más néven az epoch száma) 2000 volt. Az optimális megoldások meghatározására a [69]-ben alkalmazott kevert egész értékű modellt használtam, a megoldásokat pedig a CPLEX megoldóval állítottam elő. A feladat modelljét a teljesség kedvéért megadjuk a C függelékben. A [68]-ban és a [69]-ben közölt algoritmusokkal, HH és SS, a QLM összehasonlítása nem lett volna korrekt úgy, hogy nem ismerem azon feladatok részleteit, amelyeket a szerzők a saját algoritmusaikkal megoldottak a cikkben. (A szerzők sajnos nem publikálták az általuk megoldott feladatok részletes adatait és

ez irányú megkeresésemre sem reagáltak. Az én általam generált új feladatok adatai megtalálhatók itt: [75].) A QLM algoritmus eredményeit a CPLEX által adott eredményekkel hasonlítottam össze. A CPLEX egy széles körben ismert és alkalmazott kereskedelmi szoftver, amely (többek között) kevert egész értékű lineáris modellek megoldására alkalmazható.

A négy feladatosztályon belül összesen 44 feladatot hoztam létre és oldottam meg a QLM algoritmussal, majd a CPLEX megoldóval. A következő alfejezetekben az eredményeket és azok értékelését mutatom be.

2.5.1. Eredmények kiértékelése

Első alkalommal mind a négy feladatosztályhoz egy-egy feladatot generáltam, majd ezek QLM-mel és CPLEX-szel való megoldásával teszteltem az algoritmusom. Megjegyzem, hogy a CPLEX úgynevezett "fekete dobozként" funkcionál, mivel kereskedelmi szoftverről van szó, nem tudhatjuk pontosan milyen algoritmusok vannak beépítve. Az biztos, hogy ezen belül van lineáris programozási programcsomag relaxáció kezelésére, B&B illetve Branch and Price típusú algoritmusok, valamint heurisztikus megoldók is. Ezeket a feladatokat tekintsük az alap feladatoknak. Az ezekhez a feladatokhoz tartozó gépi idők és megelőzési relációk az A.1. függelékben találhatók. A megoldásokat tartalmazó táblázatban a következő jelöléseket alkalmazom:

- n - a tevékenységek száma,
- m - a gépek száma,
- NC - a megelőzési relációk száma,
- LB_1, LB_2 - a két alsó korlát értéke,
- CPLEX LB, CPLEX UB - a CPLEX által kiszámított alsó és felső korlát,
- QLM - a QLM algoritmus által kiszámított megoldás,
- QLM-freq - a tíz futásból hányszor találta meg a QLM az optimumot.

Megjegyzem, hogy csak akkor tudható pontosan hogy mekkora az optimum értéke, ha az alsó korlátok maximuma megegyezik a felső korlátok minimumával. Tehát az LB_1, LB_2 illetve a CPLEX által szolgáltatott alsó korlát maximuma megegyezik a CPLEX által talált megoldás, illetve a QLM 10 futásából származó megoldások értékeinek minimumával. Mint látható az #1 feladat esetén például LB_1 = CPLEX LB = 10, valamint CPLEX UB = QLM = 10. Itt a QLM mindegyik futása 10-es értéket adott. Vagy például a #28 feladat esetén LB_2 = CPLEX LB = 5, a CPLEX UB értéke ennél nagyobb (6), de a QLM 10 futásból 9-szer szintén 5-ös értéket adott. Most lássuk a feladatok megoldásának részletes kiértékelését.

Az első feladat (#1) könnyűnek bizonyult mind a QLM, mind pedig a CPLEX számára. Mindkét esetben sikerült megtalálni az optimális megoldását, továbbá látható, hogy a QLM a 10 futásból tízszer találta meg az optimumot. A feladat

Feladatok:	#1	#2	#5	#28
n	14	28	27	74
m	8	7	4	19
NC	5	8	1	10
LB_1	10	11	7	4
LB_2	4	9	18	5
CPLEX LB	10	11	8	5
CPLEX UB	10	11	18	6
QLM	10	11	18	5
QLM-freq	10/10	3/10	1/10	9/10
CPLEX idő (s)	0,188	28,8	28 526	28 247

2.5. táblázat. Az elsőként generált négy feladat megoldásának eredménye

optimális megoldása 10 volt. A CPLEX alsó és felső korlátjai és az LB_1 is ezt mutatja. A CPLEX nagyon gyorsan megtalálja a megoldást, 0,188 másodperc alatt. A CPLEX eredményeire vonatkozó statisztikai adatok a C függelékben találhatóak. A feladat modelljében összesen 1625 feltétel, 239 változó található, utóbbiból 224 a bináris változók száma.

A második feladat (#2) nehezebb volt, látható, hogy a tíz futásból csak három alkalommal találta meg az optimumot a QLM. Az optimális megoldás 11 volt. Ez a feladat a CPLEX-nek is nehéz volt, 28 másodperc alatt oldotta meg. A CPLEX eredményeire vonatkozó statisztikai adatok a C függelékben találhatóak. A feladat modelljében összesen 21491 feltétel, 1009 változó található, utóbbiból 980 a bináris változók száma.

Ezekkel kapcsolatban megjegyzem a következőket: vannak olyan tanuló algoritmusok, amelyeket többször futtatva az algoritmus fölhasználja a korábbi futások eredményeit. Az én általam fejlesztett QLM algoritmus nem ilyen, minden egyes új futás esetén a Q értékeket újraszámolja. Ki lehetne próbálni azt a változatot, amelyik a korábbi futás által kapott legjobb célfüggvényértéket úgy használja fel egy későbbi futásnál, hogy ha az aktuális célfüggvényérték ennél nagyobb, akkor ott megnöveli a büntetést, ha pedig kisebb, akkor megnövelt jutalmat ad. Ez későbbi kutatás tárgya lehet.

A harmadik feladat (#5) már jelentősen nehezebb az előző kettőnél. A QLM algoritmus csak egy alkalommal találta meg az optimális megoldást a tíz futásból. Látható, hogy a CPLEX-nek is nehéz volt a feladat megoldása, ugyanis bár hamar megtalálja az optimális megoldást, de mivel a NEOS szerver úgy van beállítva, hogy nagyjából 28 000 másodperc (ami nagyjából 8 óra) után leáll, ennyi idő nem volt elég az optimum verifikálásához. A CPLEX által talált 18-as célfüggvény érték valójában optimális, de ezt a CPLEX nem tudta verifikálni (mert 8 óra futás után CPLEX UB nem egyenlő CPLEX LB-vel). Csak onnét tudható, hogy a 18-as célfüggvényérték az optimum, hogy megegyezik LB_2-vel, de a CPLEX erre "nem jött rá", ennél sokkal gyengébb alsó korlátot (8) talált csak. A CPLEX eredményeire vonatkozó statisztikai adatok a C függelékben találhatóak. A feladat modelljében összesen 17060 feltétel, 784 változó található, utóbbiból 756 a bináris változók száma.

A negyedik feladat (#28) az előző háromtól eltérően jóval nagyobb probléma. Ez

a gépek számában, a tevékenységek számában és a megelőzési relációk számában is látható. Azonban a feladat mérete ellenére nem volt nehéz a QLM számára, ugyanis a tíz futásból kilencszer megtalálta az optimális megoldást. A CPLEX számára azonban nehéz volt a feladat, ez látható a CPLEX UB és CPLEX LB értékekből, a kettő között nagy az eltérés. A CPLEX eredményeire vonatkozó statisztikai adatok a C függelékben találhatóak. A feladat modelljében összesen 412282 feltétel, 7105 változó található, utóbbiból 7030 a bináris változók száma.

Mind a 4 alapfeladatra lefuttattam 10-szer a QLM algoritmust. A futási idők 0,1 másodperc és 1 másodperc között mozognak, egy feladaton belül is változatosak, de mindig 1 másodperc alatt maradtak. A futásidők átlaga feladatonként 0,5, 0,64, 0,53 és 0,59.

A feladatokhoz tartozó részletes gépi idő táblázatok és megelőzési relációk az A.1 függelékben találhatók. A futási idők részletes adatai a C függelékben vannak megadva.

Mivel a QLM futásideje 1 másodperc alatt marad mind a négy alapfeladat esetén, a 2.5. táblázatban csak a CPLEX futásidejét adtam meg, mert az viszont erősen függ a feladattól.

2.6. Részletesebb vizsgálatok

A 2.5.1. fejezetben bemutatott eredményeken túl mind a négy osztály esetében további feladatokat generáltam, természetesen figyelembe véve az adott osztály paramétereit. Minden egyes osztály további tíz darab feladattal bővült. Ebben a fejezetben az ezekre a feladatokra vonatkozó eredményeket mutatom be. Az ezekhez a feladatokhoz tartozó, a gépi időkre és a megelőzési relációkra vonatkozó információk az A.2 számú függelékben találhatók. Az alábbi táblázatokban az egyes osztályokba tartozó feladatpéldányokat 1-10 számokkal jelöltem. Tehát minden inputhoz tartozik egy oszlop. Négy feladatosztály esetén ez összesen 40 input. Megjegyzem, hogy minden inputra a QLM 10-szer van futtatva. Ez összesen 400 futás. Minden oszlop alján a QLM* szám azt jelenti, hogy a 10 futásból az algoritmus hányszor találta meg az általa talált legjobb értéket. Ez bizonyos esetekben egyenlő az optimummal (ha a legjobb LB egyenlő a legjobb UB-vel) más esetekben csak annyit tudunk, hogy a legjobb felső korlát. Mivel a QLM futásideje mindig 1 másodperc alatt maradt, ezért azt a táblázatokban nem adtam meg. A CPLEX futásidejét a folyószövegben ismertetem.

A *Class* #1 osztályhoz generált 10 feladat esetében a QLM algoritmusa a 10 futásból 10 alkalommal találta meg az optimális megoldást. Mind a CPLEX, mind a QLM algoritmus esetén a futás gyors, kevesebb, mint 1 másodperc alatt ment végbe.
A 2.6. táblázatban látható, hogy minden feladat esetében sikerült megtalálni az optimális megoldást, amelyet a CPLEX is megerősített. A CPLEX által kiszámított alsó és felső korlátok minden feladatnál megegyeznek. A QLM esetében számított korlátok közül az LB_1 korlát értékei minden oszlopban megegyeznek a CPLEX korlátaival. Mind a QLM, mind a CPLEX átlagosan 1 másodpercen belüli futási idővel dolgozott.

$n = 14$, $m = 8$, $NC = 5$										
Feladatok:	1	2	3	4	5	6	7	8	9	10
LB_1	9	5	9	8	8	4	9	17	7	10
LB_2	3	3	3	3	3	2	3	5	3	3
CPX LB	9	5	9	8	8	4	9	17	7	10
CPX UB	9	5	9	8	8	4	9	17	7	10
QLM	9	5	9	8	8	4	9	17	7	10
QLM* (/10)	10	10	10	10	10	10	10	10	10	10

2.6. táblázat. A bővített feladatok *Class* #1 osztályának eredményei

A *Class* #2 feladatai esetében már változóak az eredmények. A CPLEX mindössze egy esetben tudta megerősíteni az optimális megoldást, méghozzá a 6-os számú esetében. Ha a feladatok közül például az 1-es számút tekintjük, akkor látható, hogy a 10 futásból a QLM 6 alkalommal számolt 9-es értékű átfutási időt. Ugyanezen feladat esetében az $LB_1 = 8$ és $LB_2 = 7$. Ebből látható, hogy az optimális megoldás legfeljebb 9 és legalább 8.

$n = 28$, $m = 7$, $NC = 8$										
Feladatok:	1	2	3	4	5	6	7	8	9	10
LB_1	8	6	5	8	7	7	6	6	5	9
LB_2	7	7	7	8	9	6	7	6	7	9
CPLEX LB	8	6	5	8	7	7	6	6	5	9
CPLEX UB	9	8	8	9	12	7	7	8	7	11
QLM	9	8	8	9	12	7	7	8	7	11
QLM* (/10)	6	10	10	3	6	2	6	7	9	6

2.7. táblázat. A bővített feladatok *Class* #2 osztályának eredményei

A CPLEX is ugyanezeket az értékeket számolta, azaz az alsó korlát 8, a felső korlát pedig 9 volt. A CPLEX által számított alsó korlátok a 2-es, 3-as, 5-ös, 7-es és 9-es feladatok esetében rosszabbak, mint a $max(LB_1, LB_2)$. A 6-os számú feladat esetében a CPLEX alsó korlátja és felső korlátja megegyeznek, így itt sikerült megerősíteni az optimális megoldást, amelyet a QLM is megtalált. A 7-es és 9-es feladatoknál látható, hogy az LB_1 és LB_2 értékei megegyeznek a CPLEX alsó és felső korlátjával, továbbá a QLM által számított eredmények mindkét esetben egyenlők az LB_2 értékével, így a megoldás optimális, azonban ezt a CPLEX nem erősítette meg. A CPLEX átlagosan 12 másodperc alatt megtalálja az általa legjobb megoldást, de ezen utána nem tud már javítani. A 8 órás időkeretben nem képes a megoldás optimalitását verifikálni (kivéve az említett 6-odik input esetén.)

A *Class* #3 feladataihoz tartozó eredmények a 2.8. táblázatban láthatók. Megállapíthatóak a következők:

- a CPLEX alsó korlátja mindig az LB_1-el egyezik meg, de ennél lényegesen jobb az LB_2 érték, érdekes, hogy ezt a CPLEX nem találta meg,
- a CPLEX soha nem ad jobb megoldást, mint a QLM, azonban egy feladat esetén a QLM jobb megoldást ad (3. feladat).

$n = 27$, $m = 4$, $NC = 1$										
Feladatok:	1	2	3	4	5	6	7	8	9	10
LB_1	5	8	5	7	6	8	5	8	6	6
LB_2	16	18	15	18	16	19	17	18	19	16
CPLEX LB	5	8	5	7	6	8	5	8	6	6
CPLEX UB	18	20	18	19	16	20	17	19	21	17
QLM	18	20	17	19	16	20	17	19	21	17
QLM* (/10)	10	9	7	8	2	10	1	8	9	10

2.8. táblázat. A bővített feladatok *Class* #3 osztályának eredményei

A CPLEX itt sem képes garantáltan optimális megoldást találni 8 órás futásidő alatt.

$n = 74$, $m = 19$, $NC = 10$										
Feladatok:	1	2	3	4	5	6	7	8	9	10
LB_1	6	4	4	5	5	4	4	6	5	4
LB_2	5	5	5	5	5	5	5	5	5	5
CPLEX LB	6	4	4	5	5	4	3	5	5	4
CPLEX UB	13	9	11	9	9	15	9	9	8	10
QLM	6	5	6	6	6	5	6	6	5	5
QLM*	6	1	7	10	10	1	10	8	1	3

2.9. táblázat. A bővített feladatok *Class* #4 osztályának eredményei

A *Class* #4 feladataihoz tartozó eredmények a 2.9. táblázatban láthatók. Az eredményekből látható, hogy a QLM algoritmus az esetek több, mint a felében (1-es, 2-es, 6-os, 8-as, 9-es, 10-es feladatok) ugyanazt az eredményt adta, mint a $max(LB_1, LB_2)$. A 3-as, 4-es, 5-ös és 7-es feladatok esetében a QLM eredménye csak eggyel nagyobb, mint $max(LB_1, LB_2)$. Az 1-es, 2-es, 6-os, 8-as, 9-es és 10-es feladatok esetben a QLM eredményei egyben az optimális megoldások is. Az is látható, hogy a CPLEX számára nehezek voltak ezek a feladatok, mert nagyon magas felső korlátokat számolt ki, egyúttal egyik esetben sem tudta megerősíteni az optimális megoldást. A CPLEX átlagosan 9 másodperc alatt találja meg az általa talált legjobb megoldást, és a 8 órás időkeret nem elég annak eldöntésére, hogy ez optimális-e.

A vizsgálatok alapján elmondható, hogy a megoldások minősége nem a QLM paraméterein múlik. Ezek a feladatok felépítésükből eredően nehezek. Nem arról van szó, hogy a megoldások nem sikeresek, hanem a feladatok nehézsége folytán a CPLEX nem tudta igazolni, hogy az optimális megoldást kaptuk-e meg vagy sem.

A feladatokhoz tartozó összefoglaló gépi idő táblázatok és megelőzési relációk az A.2 függelékben találhatók.

2.7. Összefoglalás

Ebben a témakörben egy olyan, speciális ütemezési probléma megoldásával foglalkoztam, amelyben a tevékenységek között előre meghatározott megelőzési relációk

szerepelnek, és az egyes tevékenységek végrehajtási ideje a hozzájuk rendelt gépi erőforrástól függ. A feladat specialitását a megelőzési relációk adják, amelyek rövid, diszjunkt utakat definiálnak. A szakirodalomban csak kevés számú olyan algoritmus van, amely az általam bemutatott modellek esetében egyáltalán releváns lenne. Az általam létrehozott algoritmus azon kevés megoldások közé tartozik, amely ezen speciális probléma megoldására lett kidolgozva.

Megmutattam, hogy a QLM eljárás alkalmas az ebben a fejezetben bemutatott ütemezési probléma megoldására.

Az állapottér egyszerűsítésével 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 [68]-ban és a [69]-ben 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 csak sejtjük, hogy a QLM optimális megoldást talált, de 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 összhangban vannak a CPLEX megoldója által kiszámított eredményekkel.

Emellett érdekes és továbbra is nyitott kérdés az, hogy a Q-tanulás vagy más megerősítéses tanulási módszer hogyan és milyen hatékonysággal alkalmazható egyéb ütemezési problémák megoldására.

Jelenleg nincsen semmilyen eredmény a legrosszabb esetre vonatkozó közelítési arányról a bemutatott ütemezési problémával kapcsolatban. Ez is egy érdekes és nyitott terület, amely a jövőbeni kutatások részét képezheti.

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.

3. fejezet

Mohó algoritmusok ládapakolási benchmark feladatokhoz

3.1. Bevezetés

A ládapakolás egy klasszikus területe a kombinatorikus optimalizálásnak, amelynek számos felhasználási területe van a mindennapok során. A ládapakolási feladatok megoldásakor a rendelkezésre álló kapacitás optimális felhasználására törekszünk, legyen szó tárolásról vagy éppen szállításról. A minél hatékonyabb kapacitáskihasználásnak az eredménye a kevesebb számú tároló alkalmazása, ezáltal például a szállítás is kevesebb járművel oldható meg, ez pedig a környezet terhelésének csökkenéséhez vezet. Továbbá sokkal olcsóbbá tehető ezáltal a tárolás vagy a szállítás, hiszen kevesebb erőforrást kell felhasználni.

3.1.1. Az új megközelítés

A Bevezetésben (1. fejezet) nagyon röviden áttekintettem, hogy a ládapakolási feladatnak milyen főbb változatai vannak és ezekre milyen megoldásokat javasoltak.

Ebben az alfejezetben bemutatom az általam bevezetett új megközelítést. Ennek lényege, hogy a ládapakolási problémák megoldása előtt egy ún. előfeldolgozást végzek el, amellyel igyekszem egyszerűsíteni a megoldandó feladatot. Ha van egy ládapakolási feladat, amelyet szeretnénk optimálisan megoldani, akkor az nehéz lehet, ugyanis a ládapakolási probléma NP-nehéz. De ez nem jelenti azt, hogy az optimális megoldás megtalálása minden feladat esetében nehéz. Például, ha minden tárgy $w > 0$ mérete egyforma, akkor a feladat megoldása triviális. Ebben az esetben minden ládába pontosan $\lfloor \frac{1}{w} \rfloor$ tárgy pakolható és a feladatot megoldottuk optimálisan. Természetesen a valóságban a ládapakolási feladatok nehezebbek vagy sokkal nehezebbek. A későbbiekben azonban látni fogjuk, hogy bizonyos benchmark feladatok esetén tudunk olyan "trükkös" algoritmusokat javasolni, amelyek az esetek java részében mégis képesek megtalálni az optimális megoldást.

Az új módszer a következőképpen fogalmazható meg. Megpróbáljuk meghatározni az adott feladat optimális megoldását egy mohó algoritmussal; ha ez sikerült, a feladat megoldásával készen vagyunk. Ha viszont nem sikerült, akkor más, szofisztikáltabb megoldó algoritmus szükséges (amellyel itt most nem foglalkozom). 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 mohó algoritmusok ilyesfajta alkalmazása a mindennapi életben is jelen van, például rövidebb útvonalak megtalálása, két város között olyan útirány választása, ahol a forgalom kisebb. Az operációkutatás sok területén szintén alkalmaznak mohó módszereket segédalgoritmusként.

A következő alfejezetben részletes áttekintést nyújtok a vizsgált benchmark feladatokról.

3.2. A vizsgált benchmarkok

Ebben a fejezetben bemutatom azokat a benchmark feladatokat, amelyeket használtam az algoritmusok fejlesztése és tesztelése során. A feladatosztályok például a Bologna Egyetem Operációkutatás Csoportjának a weboldalán [76] elérhetők.

A feladatok a következő formában vannak megadva. A tárgyak számát n, a láda kapacitását pedig C jelöli. A tárgyak mérete w_i és a feladatok különböző osztályokba vannak sorolva. A weboldalon összesen 6195 db feladat érhető el és mindegyik feladat esetében a tárgyak méretük szerint csökkenő sorrendbe vannak rendezve. A benchmarkok döntő többségénél ismertek az optimális megoldások, azonban vannak olyan osztályok, ahol még vannak a feladatok között olyanok, ahol nem ismerjük az optimális megoldást. A Schwerin és a Falkenauer osztályok esetében minden feladathoz ismertek az optimális megoldások.

3.2.1. Schwerin benchmark

A Schwerin [77] benchmark két halmazra oszlik; Schwerin1 és Schwerin2. Mindkét halmazban 100 feladat található. A Schwerin1 esetében a tárgyak száma $n = 100$, a Schwerin2 esetében pedig $n = 120$. A ládák kapacitása egységesen $C = 1000$ és a tárgyak méretei a $[150, 200]$ intervallumból kerülnek ki egyenletes eloszlással. (Tehát minden ládába vagy 5 vagy 6 darab tárgy fog kerülni.) A Schwerin 1 esetében valamennyi feladat optimális megoldása 18, a Schwerin 2 esetében pedig 21 vagy 22.

Összes feladat: 200 db.

3.2.2. Falkenauer benchmark

A Falkenauer [78] benchmark két osztályra oszlik, amelyek mindegyikében 80 feladat található.

Az első osztály, a Falkenauer_U további négy alosztályra bomlik, mindegyik alosztályban 20 feladat van. Az alosztályokban a tárgyak száma $n = 120$, $n = 250$, $n = 500$ és $n = 1000$. A tárgyak mérete a $[20, 100]$ intervallumból kerül ki egyenletes eloszlással, a ládák kapacitása $C = 150$.

A második osztály, a Falkenauer_T ugyancsak négy alosztályra oszlik és mindegyik alosztályban 20 feladat van. A tárgyak száma $n = 60$, $n = 120$, $n = 249$ és $n = 501$. A tárgyak mérete a $[250, 500]$ intervallumból kerül ki egyenletes eloszlással,

a ládák kapacitása $C = 1000$. A Falkenauer_U esetében az optimális megoldások értékei feladatonként változó.

Összes feladat: 160 db.

3.2.3. További benchmarkok

Scholl

A Scholl benchmark [79] esetében a feladatok három részre oszlanak: DataSet1, DataSet2 és DataSet3. Ezekben a halmazokban 720, 480 és 10 feladat található. A tárgyak száma (n) az $[50, 500]$ intervallumból kerül ki. A ládák kapacitása (C) DataSet1 esetén az $[100, 150]$ intervallumból kerül ki, DataSet2 esetén $C = 1000$ és a DataSet3 esetén $C = 100000$. A DataSet1 esetében a tárgyak mérete az $[1, 100]$, a DataSet2 esetében az $[1, 500]$ és a DataSet3 esetében a $[20000, 35000]$ intervallumból kerül ki.

Összes feladat: 1210 db.

Wäscher

A Wäscher benchmark [80] feladatai az egyik legnehezebbek az összes benchmark közül. A tárgyak száma (n) a $[57, 239]$ intervallumból kerülnek ki, a ládák kapacitása $C = 10000$. A legnagyobb tárgyméret kicsivel 5000 alatti és egészen kicsi tárgyméretek is vannak (40). Mivel a későbbiekben csak a Schwerin és Falkenauer_U osztály inputjaival fogunk részletesen foglalkozni, ezért a többi benchmark osztályok tárgyméreteit a hely kímélése végett nem adjuk meg.

Összes feladat: 17 db.

Schoenfield Hard28

A Hard28 benchmark [81] 28 nehéz feladatot tartalmaz, ahol a tárgyak száma (n) 160 és 200 között alakul, a ládák kapacitása $C = 1000$. A tárgyak mérete változatos.

Összes feladat: 28 db.

RGI, Augmented Non-IRUP, Augmented IRUP, GI

Az RGI benchmarkban [18] a tárgyak száma $n \in [50, 1000]$, a ládák kapacitása (C) pedig 50 és 1000 között. A legkisebb tárgy mérete a $[0.1C, 0.2C]$ intervallumból, a legnagyobb tárgy mérete pedig a $[0.7C, 0.8C]$ intervallumból származik. Összesen 3840 feladat található ebben a csomagban.

Ugyanez a publikáció [18] mutatja be az Augmented Non-IRUP (ANI) és az Augmented IRUP (AI) benchmarkokat. Mindkét benchmark 250 feladatot tartalmaz, így összesen 500 feladatról van szó. Az ANI esetében a feladatok öt halmazra vannak bontva és a tárgyak száma az öt alosztályban $n \in \{201, 402, 600, 801, 1002\}$, továbbá a tárgyak méretei az $[1, 2500]$, $[1, 10000]$, $[1, 20000]$, $[1, 40000]$ és az $[1, 80000]$ intervallumokból kerülnek ki. A ládák kapacitása az öt halmaz esetén sorra 2500, 10000, 20000, 40000, és 80000. Az AI osztály az ANI osztályból lett generálva annyi módosítással, hogy minden alosztály esetében a tárgyak száma $n + 1$.

A harmadik benchmark a GI [82], amely nagyon változatos feladatokat tartalmaz, összesen 240 darabot.

3.3. Algoritmusok

Ebben a fejezetben korábban már létező, illetve az általam kifejlesztett algoritmusokat mutatom be. Az FFD algoritmust arra használtam, hogy a lehető legtöbb feladatot megoldja az osztályból. Ez azért fontos, mert ha egy olyan egyszerű algoritmus, mint az FFD a feladatok egy jelentős részét optimálisan képes megoldani, akkor a teljes problémakör gyorsan egyszerűsödik, ugyanis az FFD-vel megoldott feladatokkal már nem kell foglalkozni. Természetesen ez az előbbi megjegyzés csak akkor igaz, ha a tekintett feladatra létezik olyan alsó becslés, amelynek az értéke egyenlő az FFD eredményével. Viszont az említett benchmarkok jelentős részében ez igaz. A legegyszerűbb alsó korlát a következő: a tárgyak összméretét elosztjuk a ládamérettel és a kapott számot fölfelé kerekítjük. Ez a triviális alsó korlát is elég erős volt ahhoz a benchmarkok jelentős részében, hogy az FFD által kapott megoldás optimalitását bizonyítsa. Tehát egy természetesen adódó lehetőség, hogy egy adott benchmark osztályon belül az inputokra először lefuttatjuk az FFD algoritmust, ha ez optimális megoldást ad, akkor kész is vagyunk. Mivel ez nem minden esetben történik meg, szükségünk van más, bonyolultabb algoritmusokra is.

Következő lépésként a ládapakolási feladatot egyszerűsítjük az alábbi módszerrel. Ahelyett, hogy arra figyelnénk, hogy sok ládát hogyan lehet egyszerre jól megtölteni, egyesével fogunk ládákat megtölteni. Vagyis egymás után fogunk hátizsák feladatokat megoldani. Egy hátizsák megfelel egy ládának. Azt figyeltük meg, hogy (legalábbis a Schwerin osztályon belül) ha "jól" telepakolunk egyesével ládákat, akkor így globálisan is jó megoldást kapunk. Vagyis más szóval mohó döntések sorozatát lehet alkalmazni. A hátizsák "jó" megtöltésére pedig egy egyszerű útkereső segédalgoritmust alkalmaztam.

3.3.1. First Fit Decreasing (FFD)

Az FFD algoritmus az FF algoritmus módosított változata oly módon, hogy az első lépés egy előfeldolgozás. Ahogy az algoritmus nevében is benne van, a bemeneti elemeket méret szerinti csökkenő sorrendbe rendezzük, majd ezen a sorrenden alkalmazzuk a First Fit algoritmust. Tulajdonképpen az ismertetett benchmarkok esetében erre már nincs szükség, ugyanis alapértelmezetten csökkenő sorrendben szerepelnek a tárgyak a méretük alapján minden feladat esetében. A továbbiakban az FFD teljesítményét mutatom be a fenti benchmarkok esetén.

Az FFD algoritmussal kapcsolatos első tapasztalat az volt, hogy a fentebb említett benchmarkok közül néhányra elég jó eredményeket adott. Elsőként az AI és az ANI benchmarkok FFD-vel történő megoldásait mutatom be. Ezek olyan feladatosztályok, amelyek esetében a bennük található feladatok között vannak még meg nem oldottak, azaz az optimális megoldásuk nem ismert, csak a megoldás alsó (LB) és felső (UB) korlátja. Néhány esetben igaz, hogy $LB = UB$.

A 3.1. táblázatban az FFD algoritmus teljesítménye látható az AI és az ANI osztályokon. Látható, hogy az FFD algoritmus a felső korláttal megegyező ered-

	AI		ANI	
	#	%	#	%
FFD = UB	131	52,4	250	100
FFD >UB	119	47,6	0	0
Összes	250	100	250	100

3.1. táblázat. Az FFD teljesítménye az AI és az ANI feladatosztályokon

ményt hozott az AI osztályban található feladatok 52,4%-ra és az ANI osztályban található feladatok 100%-ra. A felső korlát az ismert legjobb megoldást jelenti. Tehát ezzel az egyszerű algoritmussal az AI osztály feladatainak több, mint a felét, az ANI osztály feladatai közül pedig mindet sikerült megoldani. Azaz a megoldandó AI osztálybeli feladatok aránya 47,6%-ra csökkent, az ANI feladatoké pedig 0%-ra. Így drasztikusan csökkentettük a megoldatlan feladatok számát mindenféle bonyolult algoritmus alkalmazása nélkül. Természetesen továbbfejlesztési lehetőségként felmerül, hogy jobb alsó korlátot is lehetne alkalmazni, azonban dolgozatom e részében csak azt szeretném illusztrálni, hogy nagyon sok benchmark feladatra már a klasszikus és általánosan ismert FFD algoritmus is hatékony. Általánosságban a különféle benchmarkok vizsgálatát viszont itt nem részletezem, tehát az FFD-vel kapcsolatos vizsgálatok itt csak illusztráció céljából vannak. Két benchmark osztályt (Schwerin és Falkenauer_U) viszont részletesen megvizsgálok a következőkben. Ezekben az FFD nem elég hatékony, emiatt újonnan kifejlesztett algoritmusokat mutatok be.

A fejezet további részében a maradék benchmarkok eredményeit tekintem át. Ezek az osztályok a megoldás szempontjából abban különböznek az AI és ANI osztálytól, hogy itt minden egyes feladatnak ismert az optimális megoldása.

	DataSet1		DataSet2		DataSet3		Falkenauer_U		Falkenauer T	
	#	%	#	%	#	%	#	%	#	%
FFD = UB	546	75,8%	236	49,2%	0	0%	6	7,5%	0	0%
FFD >UB	174	24,2%	244	50,8%	10	100%	74	92,5%	80	100%
Összes	720	100%	480	100%	10	100%	80	100%	80	100%

3.2. táblázat. Az FFD teljesítménye a többi feladatosztályon (I)

	GI Instances		RGI Instances		Schwerin		Wäscher		Hard28	
	#	%	#	%	#	%	#	%	#	%
FFD = UB	1	0,4%	1601	41,7%	0	0%	2	11,8%	5	17,9%
FFD >UB	239	99,6%	2239	58,3%	200	100%	15	88,2%	23	82,1%
Összes	240	100%	3840	100%	200	100%	17	100%	28	100%

3.3. táblázat. Az FFD teljesítménye a többi feladatosztályon (II)

A 3.2. és a 3.3. táblázatokban láthatók a többi feladatosztály esetén kapott eredmények. Ezek közül a DataSet1 esetében volt a leghatékonyabb az FFD. Itt a feladatok több mint 75%-át sikerült optimálisan megoldani. A DataSet2 esetében közel a feladatok felében optimális megoldás született. A harmadik legjobb eredmény az RGI feladatoknál látható, itt a feladatok 41,69%-át sikerült optimálisan

megoldani. Sajnos a többi feladatosztály jócskán elmarad ezektől az eredményektől. A Falkenauer T és a Schwerin esetében 0 feladat esetében találta meg az FFD az optimális megoldást. A GI esetében $\frac{1}{240}$, a Wäscher-nél $\frac{2}{17}$, a Falkenauer_U-nál $\frac{6}{80}$, a Hard28-nál pedig $\frac{5}{28}$ ez az arány.

Amennyiben az FFD nem talált optimális megoldást, úgy jellemzően egy ládával többet töltött meg a kelleténél. Összesítve a 6195 feladatból az FFD-nek sikerült 2778 darabot optimálisan megoldani, ami 44,84%-os teljesítmény. Mint említettem, az FFD-vel kapcsolatos vizsgálataim itt csak illusztrációként szerepelnek, azt mutatják be, hogy sok esetben nem szükséges bonyolult algoritmus alkalmazása. Megjegyzem, hogy amikor az FFD a "szükségesnél" egy ládával többet használ, várhatóan, lokális cserék alkalmazásával sok esetben el lehet jutni az optimális megoldásig. Ez azonban nem képezte vizsgálatok tárgyát.

A [83]-as publikációban egy új, viszonylag bonyolult algoritmust mutatnak be, amely a Scholl feladatosztályból 120 feladatot optimálisan megold az összes 1210 feladatból. Ugyanezt a 120 feladatot az FFD is megoldja optimálisan. Adódik a kérdés, hogy akkor miért használjunk bonyolult eljárásokat, ha sokkal egyszerűbb módszerekkel is megoldhatjuk ugyanazt? Feltehető, hogy a cikk szerzői az 1210 feladatból olyan 120-at választottak (három alosztály a sok alosztály közül), amelyekre az FFD hatékony, mert nem túl nehezek a feladatok. Az alábbi konklúziókat vonhatjuk le.

- Az FFD algoritmus nem minden esetben hatékony, de jó ötletnek tűnik először az FFD-t futtatni. Látható, hogy az összes feladat közel felét megoldotta optimálisan.

- Ezek után két feladatosztályt választottam: Schwerin és Falkenauer_U. Ezekre az FFD nem hatékony. Viszont a későbbiekben megmutatom, hogy összetettebb, de még mindig mohó algoritmusok képesek lesznek megoldani vagy az összes feladatot vagy a feladatok jelentős részét ezekből az osztályokból. Mint később részletesen bemutatom, ennek az az oka, hogy ezekben az osztályokban található feladatok tárgyméretei véletlenszerűen és egyenletesen vannak kiválasztva valamely intervallumból.

3.3.2. Előfeldolgozó eljárások az irodalomban

Az előfeldolgozás egy jól ismert és gyakran alkalmazott technika az optimalizálásban. Ez lényegében azt jelenti, hogy az adott feladat megoldása előtt megpróbáljuk egyszerűsíteni a problémát, amennyiben lehetséges. Például, ha egy lineáris optimalizálási problémáról van szó, akkor első lépésben eltávolítják a redundáns korlátozásokat, azaz egyszerűsítik a modellt. Vélhetően a legelső, a lineáris programozásban alkalmazott előfeldolgozó módszerrel foglalkozó publikáció [84], amely 1975-ben jelent meg. Majd ezt több hasonló publikáció is követte; 1983-ban Tomlin és Welch [85, 86], 1995-ben Andersen és Andersen [87] vagy 1997-ben Gondzio [88] munkája.

Savelsbergh munkájában [89] javasolja az előfeldolgozást a kevert értékű programozási problémákra. A témában néhány további publikáció [90–92].

Mészáros és Suhl [93] a lineáris és kvadratikus programozás kapcsán foglalkozott az előfeldolgozással. [94] pedig egy olyan publikáció, amely különböző előfeldolgozási

technikákkal foglalkozik az egész értékű programozás kapcsán.

Az egyes tárgyak méret szerinti rendezése is előfeldolgozásnak tekinthető, ami láthatóan nagyon hatékony tud lenni. Ha a First Fit algoritmust futtatjuk egy adott L tárgylistával, akkor a legrosszabb esetben az $FF(L) = 1,7 \cdot OPT(L)$, függetlenül attól, hogy milyen nagy $OPT(L)$ [12]. Viszont, ha az elemek nem növekvő sorrendbe vannak rendezve, és a rendezés után alkalmazzuk az First Fit algoritmust (ami innentől kezdve tulajdonképpen First Fit Decreasing), akkor a legrosszabb esetben $FFD(L) \approx \frac{11}{9} \cdot OPT(L)$ nagy $OPT(L)$ értékek esetén [7, 15].

3.3.3. Segédalgoritmusok

Ebben a fejezetben néhány kiegészítő algoritmust mutatok be. Ezek nem újak, gyakran alkalmazott algoritmusokról van szó. Az itt bemutatott algoritmusokat a különböző feladatosztályok (Schwerin és Falkenauer) megoldása során alkalmaztam.

A hátizsák feladat és kapcsolata a ládapakolással

A hátizsák feladat a következő: adott n tárgy, minden i-re az i. tárgynak van egy w_i súlya és egy g_i haszon értéke. Továbbá a hátizsák rendelkezik egy C kapacitással is. A cél az, hogy a tárgyak egy részhalmazát úgy pakoljuk a hátizsákba, hogy a tárgyak méreteinek összege legfeljebb C legyen, a nyereség pedig a lehető legnagyobb. Köztudott [95], hogy a hátizsák probléma NP-nehéz, a ládapakolási probléma pedig erősen NP-nehéz. Természetesen adódik a következő ötlet. Tekintsünk egy ládát egy hátizsáknak, azt pakoljuk "jó alaposan" tele, zárjuk le ezt a ládát, aztán pakoljunk meg hasonlóképpen egy újabb ládát és így tovább. Az algoritmust az egyszerűség kedvéért Hátizsáknak nevezzük és pontos leírása a következő:

Algorithm 6: Hátizsák

Input: a ládapakolási feladat elemei
Output: az elemek pakolása C méretű ládákba

1 Amíg valamely megállási feltétel nem teljesül **do**
2 Néhány elemet kiválasztunk a még nem pakoltak közül, valamely később meghatározandó elv alapján. Ezeket bepakoljuk a hátizsákba (ládába).
3 A hátizsákba (ládába) pakolt tárgyak megjelölése "már pakolt" tárgyként, majd folytatás az 1. lépéssel.

Fontos leszögezni, hogy nincs garancia arra, hogy a fenti algoritmus optimális megoldást ad egy adott ládapakolási feladatra. Azonban, később látható lesz, hogy egy ilyen egyszerű mohó algoritmus mégis képes sok esetben optimális megoldást előállítani. Az algoritmus mohó, ugyanis egyszerre egy ládával dolgozik, és arra törekszik, hogy a még nem pakolt tárgyakkal a lehető legjobb pakolást érje el.

Ami a tárgyak hasznosságát illeti a hátizsák problémában, számos lehetőség van ennek az értéknek a megállapítására. Erre az egyik legegyszerűbb, ha a tárgyak méretével azonos haszonértékeket választunk, azaz minden i. tárgy esetében $g_i = w_i$. A haszonfüggvény vagy más néven profitfüggvény ekkor arányos költség (*proportional cost*).

1. Megfigyelés. *Tegyük fel, hogy a láda kapacitása $C > 0$ (egész érték) és minden tárgy mérete egy pozitív egész szám az $[1, C]$ intervallumból, a haszonfüggvény pedig $g_i = w_i$. Ez esetben a Hátizsák feladat egy speciális esetét kapjuk, amelynek neve Subset Sum. Figyeljük meg, hogy ilyen módon csak egy ládát figyelünk egyszerre.*

Ha úgy akarjuk pakolni a tárgyakat a ládákba, hogy azok minél jobban fel legyenek töltve, akkor a fenti haszonfüggvény ($g_i = w_i$) is egy jó választás lehet. Azonban annyi hátránya van, hogy nem tesz különbséget két pakolás között, ha például mindkettő teljesen megtölti a ládát, de az egyik kevesebb, a másik több tárgyat használ ehhez. Nyilvánvaló, hogy két ilyen pakolás közül jobb egy olyat választanunk, ahol kevés nagy tárgyat pakolunk a ládába, mint ha sok kicsit. Azért van így, mert akkor a sok kicsi tárgy megmarad a későbbi ládák pakolásához és így nagyobb mozgásterünk marad a későbbi ládák ügyes pakolására.

Emiatt az előzőnél jobb választás lehet a $g_i = w_i - 1$. Ezzel a haszonfüggvénnyel az algoritmus inkább a nagyobb tárgyakat fogja választani a kisebbek helyett, ha lehetséges. Például legyen a ládaméret 10 és legyen hét tárgy, amelyek méretei 6, 4 és öt darab 2 méretű. Többféleképpen is fel lehet tölteni a hátizsákot, de csak egy esetben lesz a haszon maximális. Mégpedig akkor, ha a lehető legkevesebb tárggyal töltjük fel. Azaz, a $g_i = w_i$ haszonfüggvény minden teljes feltöltést azonosan jónak venne, azonban a $g_i = w_i - 1$ haszonfüggvény esetében az algoritmus a {6,4} választást fogja preferálni ahelyett, hogy pl. öt darab 2 méretű tárggyal töltse fel a hátizsákot.

Útkereső algoritmus alkalmazása

A következő algoritmus egy jól ismert útkereső algoritmus. Az algoritmus olyan tárgyhalmazokat keres, amelyek együttesen beleférnek egy ládába és az összméretük pontosan K, ahol $K \leq C$. Az algoritmus ennek megfelelően keres irányított utakat egy gráfban. Van egy kezdőcsúcs: k_0, amely az üres ládának felel meg. A k_0-ból egy másik k_j csúcsba vezető irányított útnak pedig megfelel egy olyan pakolás a ládába, ahol a pakolt tárgyak összmérete éppen j. Az algoritmus ügyes szervezésével (visszafelé keresés) biztosítható, hogy minden tárgy csak egyszer van figyelembe véve. Megjegyzem, hogy C, vagyis a láda mérete egész szám, és minden tárgyméret is egész.

Kezdetben adottak az $i = 1, 2, \ldots, n$ tárgyak. Az algoritmus egy irányított gráfot épít fel, ahol minden csúcs rendelkezik egy címkével. Induláskor a gráf $C + 1$ csúccsal rendelkezik, élek nélkül. A gráf j-edik csúcsa a számegyenesen a j egész számnál van. A leginkább balra levő csúcs a 0 helyen, a leginkább jobbra levő a C helyen található. Kezdetben minden k_j csomópont l_j címkéje -1, a k_0 csomóponté pedig 0. Azaz $l_j = -1 \; \forall j > 0$ esetén és $l_0 = 0$. Ha egy csomópont címkéje -1, az azt jelenti, hogy a csomópontot még nem értük el. Ha a címke értéke nem -1, akkor a csomópontot már elértük. Ha egy k_j csomópontot már elértünk, akkor az l_j címke értéke a k_0-ból induló és a k_j csomópontban végződő irányított út utolsó élszakaszának a hossza, ami nem más, mint az utolsóként pakolt tárgy mérete. Az algoritmus egymás után tekinti a tárgyakat az adott sorrendben. Az aktuális, i. tárgy esetén megvizsgálja, hogy milyen összméret pakolása lehetséges, ahol az első i darab tárgyat vesszük csak figyelembe. Ezt úgy teszi, hogy az i. tárgynak megfeleltet

egy olyan élt, amelynek a hossza ugyanakkora mint az i. tárgy mérete, és ezt az élt minden már elért csúcshoz hozzáilleszti, és így új csúcsokat érhetünk el.

Algorithm 7: Útkereső algoritmus

Input: a ládapakolási feladat elemei

Output: az elemek pakolása C méretű ládákba

1 Az i. tárgynak megfeleltetünk egy előre mutató ($\rightarrow$) irányított élt, amelynek a hossza ugyanakkora, mint a tárgy mérete.

2 Tekintsük a már elért k_j csúcsokat jobbról balra haladó sorrendben. Adjunk a gráfhoz egy irányított élt a k_j-ből a k_t csúcsba ($k_j \rightarrow k_t$), ahol $t = j + w_i$ (ha $t \leq C$ és w_i az i. tárgy mérete). Ha a k_t csúcsot még korábban nem értük el, akkor legyen k_t címkéje $l_t = i$. Ha a k_t csomópontot korábban már elértük, akkor vagy változatlanul hagyjuk az l_t korábbi címkét, vagy felülírjuk.

3 Ha van még tárgy, akkor menjünk az 1. lépésre, ha nincs, akkor megáll az algoritmus.

Az útkereső algoritmusnak többféle implementációja is létezik. Mi egy egyszerű változatot alkalmaztunk. Megjegyzem, hogy az előbb ismertetett útkereső algoritmus a későbbi 8. algoritmusnak (REM SW algoritmus) egy segédalgoritmusa, amelynek segítségével pakoljuk egyesével a ládákat.

Alsó korlátok

A ládapakolási feladatok megoldása során az ún. alsó korlátoknak fontos szerepe van. Például, ha ismerjük a feladatnak egy megengedett megoldását és a megoldás értékével megegyezik valamely alsó korlát értéke, akkor ebből arra tudunk következtetni, hogy a megengedett megoldásunk optimális megoldás is egyben. A ládapakolási feladat alsó korlátainak bőséges irodalma van, két releváns publikáció a következő: [96, 97].

Nézzünk meg két, jól ismert alsó korlátot, amelyeket az egyszerűség kedvéért LB_1-gyel és LB_2-vel jelölünk. Az alábbi LB_1 úgy adódik, hogy vesszük a tárgyak összméretét, elosztjuk a ládamérettel és ezt felfelé kerekítjük. Ez egy természetesen adódó alsó korlát.

$$LB_1 = \left\lceil \frac{\sum_{i=1}^{n} w_i}{C} \right\rceil \tag{3.1}$$

Az LB_2 is egy triviális alsó korlát, ami nem más, mint azoknak a tárgyaknak a száma, amelyek a ládakapacitás felénél nagyobbak. Nyilvánvalóan ezek mind külön ládába kerülnek.

$$LB_2 = \sum_{i=1}^{n} 1 \text{ ha } w_i > \frac{C}{2} \tag{3.2}$$

LB_2 általánosításaként kapjuk a következő alsó korlátot: vesszük azon tárgyak számát, amelyek mérete nagyobb, mint $\frac{C}{k}$ valamilyen $k > 1$ esetén, és az előbbi számot

elosztjuk $(k-1)$-gyel, és ezt a hányadost felfelé kerekítjük. Ugyanis az ilyen tárgyakból (amelyek mérete nagyobb, mint $\frac{C}{k}$) legfeljebb $k-1$ darab férhet egy ládába. Így kapjuk tehát az alábbi alsó korlátot:

$$LB_3^k = \left\lceil \frac{\sum_{i=1}^{n} 1 \text{ ha } w_i > \frac{C}{k}}{(k-1)} \right\rceil \tag{3.3}$$

3.4. Egyes benchmark feladatok megoldása

3.4.1. Schwerin osztály

A Schwerin feladatosztály tulajdonságai a 3.2. alfejezetben kerültek ismertetésre.

Schwerin 1: Az első 100 feladat

A Schwerin 1 csoportba tartozó feladatok esetében a láda kapacitása $C = 1000$ és a tárgyak száma $n = 100$. A tárgyak méretei a $[150, 200]$ intervallumból kerülnek ki véletlenszerűen, egyenletes eloszlással. Az ebbe a csoportba tartozó feladatok mindegyikénél az optimális megoldás 18 darab felhasznált láda. Emiatt, ha közelebbről megvizsgáljuk a feladatokat, akkor látható, hogy a 100 tárgyból 40 darabot automatikusan, "gondolkodás nélkül" tudunk pakolni, és csak 60 azon tárgyak száma, ahol már figyelni kell a pakolásokra. Ennek az oka a láda kapacitása, valamint a tárgyméretekhez tartozó alsó és felső korlát. Ugyanis a következő két tulajdonságot tudjuk észrevenni:

- semelyik hét tárgyat nem tudjuk egy ládába pakolni, ugyanis ha a legkisebb tárgyméretből (150) hetet veszünk, akkor $7 \times 150 = 1050 > 1000$, és
- bármely öt tárgyat tudjuk egy ládába pakolni, ugyanis ha a legnagyobb tárgyméretet vesszük (200), akkor $5 \times 200 = 1000$.

Ezért minden láda 5 vagy 6 darab tárgyat fog tartalmazni. Következésképpen 8 láda (a 18-ból) 5 tárgyat tartalmaz. Így a 40 legnagyobb tárgy elpakolható 8 ládába úgy, hogy az optimalitás nem sérül. A más feladatosztályokra való általánosítási lehetőségekkel a néhány oldal múlva következő *Skálázhatóság és komplexitás* részben foglalkozom. Ezután már csak a maradék 60 tárgyat kell elpakolni, itt viszont már jól meg kell gondolni az eljárást. Fontos kiemelni, hogy már ezen a ponton egyszerűsödött a feladat, hiszen a tárgyak 40%-át máris sikerült a ládákba bepakolni kevés idő befektetésével, hiszen csak két egyszerű tulajdonságot kellett felismerni a láda kapacitása és a tárgyak mérete alapján, valamint az optimális megoldás ismeretében. Mint nemsokára látni fogjuk ezek az inputok olyanok, hogy az alsó korlát megegyezik az optimum értékével. Emiatt valójában nincs szükség arra, hogy előre ismerjük az optimumot: megpróbáljuk annyi ládába pakolni a tárgyakat, amennyi az alsó korlát. Mivel azt tapasztaljuk, hogy ez sikerül, tudjuk, hogy optimális megoldást kaptunk. Hangsúlyozom, hogy ez nem minden feladatosztályra igaz, de mint látni fogjuk a Schwerin feladatosztály esetén teljesül ez a szerencsés tulajdonság.

Mivel ismerjük a feladatok optimális megoldását, ami 18, így tudjuk, hogy a maradék 60 tárgyat 10 darab ládába kell elosztani, hiszen így jutunk el az optimális megoldáshoz. Az ezt a célt megvalósító algoritmust Algorithm Rem SW-nek neveztem el. A "Rem" jelző az angol "remaining" szó rövidítése, amely ebben az esetben a maradék elpakolandó tárgyakat jelenti, az "SW" pedig a Schwerin típusra utal.

Algorithm 8: REM SW algoritmus

Input: a ládapakolási feladat elemei
Output: az elemek pakolása C méretű ládákba

1 Legyen k a legnagyobb olyan szám, amelyre $0 \leq k \leq 6$, továbbá teljesül az, hogy a k legnagyobb és $6-k$ legkisebb tárgy belefér egy ládába. (Ha nincs ilyen k, az algoritmus megáll.)
2 Pakoljuk a k darab legnagyobb tárgyat egy ládába.
3 Alkalmazzuk az útkereső algoritmust a ládában fennmaradó hely lehető legjobb betöltésére.
4 A pakolt tárgyak eltávolítása a rendszerből.
5 Ugorjunk az 1. lépésre, ha van még tárgy, egyébként az algoritmus leáll.

Megjegyezzük, hogy az algoritmus általában nem a k legnagyobb és $6-k$ legkisebb tárgyat pakolja, hanem a k legnagyobb tárgy pakolása után $6-k$ olyan további tárgyat pakol, amelyek a fennmaradó helyet a lehető legjobban megtöltik a ládában. A Schwerin 1 osztállyal történő futtatásból származó tapasztalatok azt mutatták, hogy a k értéke kezdetben 3, majd innen növekszik egészen 6-ig. Ritkán, de előfordult, hogy a k értéke kezdetben 2 volt és innen növekedett 6-ig. Soha nem fordult elő olyan eset, hogy a 6 legkisebb tárgy ne fért volna bele egy ládába. Emlékeztetek arra, hogy a Schwerin 1 halmaz esetén minden input 100 darab tárgyat tartalmaz és minden inputra az alsó korlát 18. Emiatt megpróbáljuk a tárgyakat 18 ládába pakolni. Először a 40 legnagyobb tárgyat ötösével 8 ládába pakoljuk. A megmaradt 60 tárgyat pedig *megpróbáljuk* hatosával 10 ládába pakolni. Arra azonban nincs semmi garancia, hogy ez sikerülni fog! Vagyis a k szám definiálásakor "optimista" módon járunk el, feltételezzük, hogy van ilyen k szám. Valójában az történik, hogy a 100 input esetén minden egyes alkalommal sikerült a maradék 60 tárgyat 6 ládába pakolni, ilyen értelemben a k szám jól definiált volt, tehát az algoritmus soha nem állt meg amiatt, hogy nem talált volna megfelelő k számot. Az algoritmus beágyazottan tartalmazza az útkereső algoritmust. Az Algorithm Rem SW eljárás futási ideje polinomiális n tárgy esetében, mert k értéke legföljebb 6 és az útkereső algoritmus lépésszáma C függvényében lineáris. 60 tárgyra vetítve a futási idő nagyon alacsony, milliszekundumokban mérhető.

A futási eredmények alapján a fenti egyszerű algoritmus segítségével mind a 100 feladatra sikerült megkapni az optimális megoldást. Az általánosítási lehetőségeket a kicsit később következő *Skálázhatóság és komplexitás* alfejezetben tárgyalom.

Schwerin 2: A második 100 feladat

A Schwerin 2 esetében a láda kapacitása a Schwerin 1-hez hasonlóan $C = 1000$, viszont a tárgyak száma $n = 120$. A tárgyak méretei a $[150, 200]$ intervallumból

kerülnek ki. A feladatok esetében itt is ismertek az optimális megoldások, amely a feladattól függően 21 vagy 22 felhasznált láda. Az LB_1 értéke minden feladat esetében megegyezik az optimális megoldással. Ha a Schwerin 1 esetében bemutatott algoritmust alkalmaztam a Schwerin 2 feladatainak megoldására, akkor a 100 feladatból 99-et tudott megoldani optimálisan. Egy apró módosítással azonban elérhető volt, hogy minden feladat esetén megtalálja az optimális megoldást. A k értékét kellett korlátozni 2 és 3 közé, azaz $2 \leq k \leq 3$ (ahelyett, hogy $0 \leq k \leq 6$). Továbbá, ha az algoritmust ezzel a megszorítással a Schwerin 1 feladatokra alkalmaztam, ott is mind a 100 esetben sikerült optimális megoldást találni. Így ezzel a beállítással sikerült mind a 200 feladatot optimálisan megoldani. A feladatok lépésről lépésre való megoldásának menete megtekinthető a következő honlapon [75].

A következő alfejezetben foglalkozunk kicsit részletesebben általánosítási lehetőségekkel. Már itt megjegyzem a következőt: a tárgyméretek intervalluma erősen meghatározza, hogy alkalmazható-e valamilyen fajta mohó algoritmus a tárgyak ügyes pakolására. Mint láttuk, ha a tárgyméretek a $[0,15; 0,2]$ intervallumból kerülnek ki (a C ládamérethez viszonyítva), az előbb ismertetett algoritmus jól működik. Ha ez az intervallum például $(\frac{1}{3}; 1]$, akkor ismert, hogy az FFD algoritmus optimális pakolást határoz meg. Még egy példát véve, ha az intervallum $(\frac{1}{4}; \frac{1}{2}]$, akkor pedig a jól ismert 3-partíciós feladatot kapjuk (további feltételek mellett), ami erősen NP-nehéz.

Skálázhatóság és komplexitás

Felvetődik a kérdés, hogy a kidolgozott algoritmus csak erre a benchmark osztályra alkalmazható, vagy esetleg másokra is. A kérdésre a válasz első közelítésben az, hogy a kidolgozott mohó módszer sajnos nagyon specifikus, tehát erősen kihasználja a Schwerin osztály jellemzőit. Melyek ezek a jellemzők? A tárgyak mérete egyenletes eloszlás szerint van választva egy viszonylag szűk intervallumból, és a tárgyak száma nem túl kevés. Ezen feltételek mellett az algoritmus bővebb osztályon is alkalmazható az alább ismertetett módon.

Tehát nézzük, mi történik akkor, ha a fent bemutatott algoritmus bemenete nem a Schwerin osztályba tartozó feladat? Például, ha a láda kapacitása nem 1000, vagy a tárgyak mérete nem a [150, 200] intervallumból kerülnek ki. Másképpen, a kérdés az, hogy az algoritmus miképpen skálázható, hogy eltérő bemenetekre is megfelelően működjön? Ha a tárgyak mérete egyenletes eloszlás mentén véletlenszerűen kerül kiválasztásra egy "szűk" intervallumból (ilyen volt például a [150, 200] intervallum is), akkor az optimális megoldás megtalálása rendszerint egyszerű. A következő lépésekben adjuk meg az algoritmus általánosabb osztályon való működését:

1. Az LB_1 alsó korlát kiszámítása.

2. Az egy ládába pakolható tárgyak minimális és maximális számának meghatározása. Feltételezzük, hogy ez a szám csak $K-1$ vagy K lehet. (A Schwerin osztály esetében $K = 6$).

3. Néhány láda megtöltése a legnagyobb méretű tárgyakkal, $K-1$ tárgy pakolása minden ládába. Ezután már csak a még nem pakolt tárgyakkal kell foglalkozni.

4. Az Algorithm Rem SW alkalmazása úgy, hogy k értéke egy korlátozott intervallumból kerüljön ki a $0 \leq k \leq K$ helyett. Például, mint a Schwerin 2 esetében, ahol $2 \leq k \leq 3$.

A 2. lépésben feltételeztük, hogy az egy ládába pakolható tárgyak száma vagy K vagy $K-1$. Ha több eset lehetséges, a pakolás sokkal bonyolultabb lehet, ilyen esettel nem foglalkoztam. Ez további kutatás tárgya lehet.

Visszatérve a $K = 6$ esetre, nézzük meg az Algorithm Rem SW eljárás becsült lépéseinek a számát, ahol a tárgyak száma $n = 100$ Schwerin 1 esetében és $n = 120$ Schwerin 2 esetében. Függetlenül attól, hogy mekkora n értéke, az 1. lépésben az algoritmus konstans számítást végez, mivel maximum K darab tárgy méretét összegzi. A 2. és 4. lépés esetében is konstans számításról van szó, hiszen csak K darab tárgyat kell kezelni. A 3. lépés viszont már több időt vesz igénybe, amely függ n értékétől. Ennek a lépésnek a futási ideje $\mathcal{O}(C \cdot n)$, ugyanis legfeljebb C darab csomópont tartozik minden i elemhez. Mivel ez a lépés körülbelül $\frac{n}{6}$-szor kerül futtatásra (figyelmen kívül hagyva azt a tényt, hogy az algoritmus által kezelt tárgyak száma a 2. lépésben redukálásra kerül), így a futási idő nem lehet több, mint $\mathcal{O}(\frac{C}{6} \cdot n^2)$. Ha K értéke szabadon változhat, akkor is hasonló számítási idő figyelhető meg, ezzel garantálva az algoritmus futási idejének felső korlátját. A feladatok megoldása egy Intel Core i5-4300M processzorral és 8 GB RAM-mal szerelt számítógépen történt.

A következőkben összehasonlítom algoritmusom eredményeit egy másik, hatékony (state-of-the-art) algoritmuséval.

Osztály	Összes (s)	Átlag (s)	HEA átlag (s)
Schwerin 1	0,3817811	0,003817811	0,34
Schwerin 2	0,408763	0,00408763	0,47

3.4. táblázat. Schwerin futási idők összehasonlítása a HEA átlaggal

A HEA algoritmus Borgulya [24] munkájában került bemutatásra (és implementálásra). Ez az algoritmus az egyik leghatékonyabb (és az egyik legújabb) ezen benchmark feladatok megoldására. A HEA algoritmus egy Intel Core i5 processzorral és 16 GB RAM-al szerelt iMAC számítógépen került futtatásra. A 3.4. táblázat alapján az alábbi következtetések vonhatók le. A teljes futási idő az én algoritmusom esetében a Schwerin 1 osztályra nézve 0,3817811 másodperc, a Schwerin 2 esetében pedig 0,408763 másodperc. Az átlagidőt tekintve a saját algoritmusom futási ideje 0,003817811 másodperc volt a Schwerin 1, és 0,00408763 másodperc volt a Schwerin 2 esetében. A HEA algoritmus átlagos futási ideje a Schwerin 1 esetében 0,34 másodperc, a Schwerin 2 esetében pedig 0,47 másodperc. Az átlagos futási időket összehasonlítva látható, hogy az algoritmusom futási ideje a Schwerin 1 esetében 90×, a Schwerin 2 esetében pedig 115× gyorsabb. Viszont fontos megjegyezni, hogy a HEA egy sokkal általánosabb eljárás, amely az összes benchmark feladatra alkalmazható, míg az Algorithm Rem SW specializált eljárás, amely kifejezetten a Schwerin osztályra lett kifejlesztve, és ezen az osztályon működik nagyon hatékonyan. Továbbá az általam kidolgozott algoritmusok nem ugyanabban a programozási környezetben készültek, mint a HEA, emiatt a futásidők tekintetében csak hozzávetőleges összehasonlítás volt készíthető.

3.4.2. Falkenauer_U osztály

A Falkenauer_U feladatosztály tulajdonságai a 3.2. alfejezetben kerültek ismertetésre. A Falkenauer_U feladatosztály és a Schwerin feladatosztály között a két nagy különbség az intervallum, ahonnan a tárgyak méretei származnak és a láda kapacitása. Pontosabban, hogy a tárgyak intervalluma hogyan aránylik a láda méretéhez.

Az arány a legnagyobb és a legkisebb méretek között a Schwerin esetében $\frac{200}{150} = \frac{4}{3}$, a Falkenauer_U esetében pedig $\frac{100}{20} = 5$. Ez azt jelenti, hogy a tárgyak mérete az utóbbi esetben sokkal változatosabb. A Falkenauer_U esetében tehát, a változatosság miatt, nem jelenthető ki az, hogy a megoldásban minden láda nagyjából azonos számú tárgyat fog tartalmazni. Ennek ellenére ennél a feladatosztálynál is megadható egy nagyon egyszerű, de fontos megfigyelés.

2. Megfigyelés. *Tételezzük fel, hogy van két tárgy, $i.$ és $j.$ úgy, hogy $w_i + w_j = C$, azaz a két tárgy méretének összege pontosan a láda kapacitásával egyenlő. Feltételezhető, hogy az optimális megoldásban ez a két tárgy egy ládába fog kerülni.*

A fenti megfigyelésben szereplő két tárgyat nevezzük el "jó pár"-nak. A megoldás során, ha egy "jó pár" egy ládába kerül, akkor az nem rontja el a feladat megoldásának minőségét, azaz megmarad az optimális megoldás megtalálásának lehetősége.

A részletesebb vizsgálathoz tekintsük a Falkenauer_u120_00 feladatot a Falkenauer_U120 alosztályból. Ebben a feladatban 15 darab "jó pár" található. Ez azt jelenti, hogy a 120 darab tárgyból már csak 90 tárggyal kell foglalkozni. Ez egy jelentős egyszerűsítése a feladatnak. A többi Falkenauer_U120 feladat is nagyon hasonló ehhez, azok is hordozzák ezt a tulajdonságot.

1. Megjegyzés. *Megjegyzem, hogy a "jó pár" fogalma egyszerűen általánosítható. Tegyük fel tehát, hogy a ládaméret $c = 1000$ és van például egy 600-as méretű tárgyunk. Ha találunk ehhez egy "jó párt", amelynek a mérete 400, azt láttuk, hogy jobban járunk, hogy ha ezt a 400-as tárgyat tesszük ebbe a ládába a 600-as tárgy mellé, minthogy ha több kicsi tárgyat tennénk ide, amelyeknek az összmérete 400. Ugyanakkor, az is igaz (elemi úton belátható), hogy ha például egy 390 méretű tárgyat teszünk a 600-as méretű tárgy ládájába, az is optimális választás akkor, ha nincsenek olyan kicsi tárgyak, amelyeknek az összmérete 391 és 400 között van. A következőkben bevezetjük a tartalék fogalmát, ahol mint látni fogjuk akkor is megengedünk párokat (vagy hármasokat, négyeseket) pakolni, ha nem teljesen töltik meg a ládát, de a tartalék elég nagy; implicit módon pontosan ezt az általánosítást fogom alkalmazni.*

A feladat megoldásához vezessük be a *kezdeti tartalék* fogalmát. Ez a mérőszám megmutatja, hogy mennyi hely marad kihasználatlanul az optimális megoldásban, ha az $OPT = LB_1$ egyenlőség fennáll, azaz az optimális megoldás egyben az alsó korlát is (szerencsére a Falkenauer_U osztályban szereplő legtöbb feladatra ez igaz). Jelölje a kezdeti tartalékot res_0.

17. Definíció. *A kezdeti tartalék kiszámítása*

$$res_0 = LB_1 \cdot C - \sum_{i=1}^{n} w_i \tag{3.4}$$

Ahol w_i a tárgy mérete, C pedig a láda kapacitása. Emlékeztetek arra, hogy ugyan tudjuk, hogy az optimum érték megegyezik az alsó korláttal, de ezt az algoritmus futása során nem használjuk. Emiatt van az előző képletben LB_1, nem pedig OPT. A feladatok vizsgálata során kiderült, hogy, mivel a tárgyak mérete egyenletes eloszlás mellett, véletlenszerűen lett generálva, így jellemző, hogy jelentős mennyiségű tartalék keletkezik. Ez azt jelenti, hogy nem szükséges minden esetben telepakolni a ládákat. Azaz, ha van egy elempár, amik nem teljesen, de "majdnem teljesen" megtöltik a ládát, akkor ez a két elem is pakolható a "jó pár" helyett, ami ugyancsak jó választás lehet. Így, néhány esetben elegendő, ha a ládák csak egy megadott L_b szintig vannak megtöltve, ahol L_b értéke közel van a láda C kapacitásához. A tartalék értéke a feladat megoldása közben dinamikusan generált érték, amely kezdetben a res_0 a (3.4) alapján, mint kezdeti tartalék, majd a későbbi lépésekben már res-ként hivatkozok rá.

Értelemszerűen, ha egy láda legfeljebb L_b szintig feltöltésre kerül, akkor a tartalék értéke $C - L_b$ mennyiséggel lesz csökkentve. Például, ha a kezdeti tartalék $res_0 = 50$, és egy láda $L_b = 146$ szintig feltöltésre kerül, akkor, mivel $C = 150$ a fennmaradó tartalék $res = 46$ lesz. Ez azért van, mert a fennmaradó 4 szabad egység már semmivel sem tölthető ki, nincs ilyen kis méretű tárgy a feladatban, így ez a hely elveszik.

18. Definíció. *Az i. tárgyat nagynak tekintjük, ha*

$$w_i > \frac{C}{2} \tag{3.5}$$

A (3.5) feltétel alapján egy tárgyat akkor tekintünk nagynak, ha annak a tárgynak a mérete nagyobb, mint a láda kapacitásának a fele. A Falkenauer_U osztály esetében egy tárgy akkor nagy, ha a mérete nagyobb, mint 75. Ebből könnyen látható, hogy két nagynak minősített tárgy már nem fér bele egy ládába. Ugyanis, a legkisebb méretű, de már nagynak tekintett tárgy mérete 76, amiből ha kettő van, az már 152, ez pedig a $C = 150$ kapacitást túllépi. Emiatt nem célszerű a pakolás végére túl sok nagy tárgyat hagyni, jobb tőlük már a pakolás elején megszabadulni, amikor még a kis méretű tárgyakkal összepakolhatók.

A Falkenauer_U osztályhoz készült algoritmus tulajdonképpen jól elhatárolható eljárásokból épül fel. Ezek azonban szekvenciális sorrendben vannak, emiatt az algoritmus nem párhuzamosítható. Az algoritmus bemenete a feladathoz tartozó összes tárgy. A már pakolt tárgyakat az eljárás eltávolítja a listából. Amint egy láda pakolt lesz, a res értéke, azaz a tartalék a fentebb bemutatott módon csökken. Ahogy korábban már volt róla szó, a tárgyak minden esetben nem növekvő sorrendben vannak és az algoritmus is így dolgozza fel őket.

Az első eljárás a $Pair(L_b, r)$. Az eljárás olyan (i, j) elempárokat keres, amelyek méretének összege pontosan L_b, továbbá, ha a tárgyat pakolnánk, akkor az azután megmaradó tartalék értékének legalább akkorának kell még lennie, mint az r korlát. Az r a tartalékra vonatkozó alsó korlát. Az algoritmus minden esetben olyan párt keres, amelynek a nagyobbik tagja a lehető legnagyobb még nem pakolt elem.

Algorithm 9: Pair(L_b, r)

Input: L_b, r
Output: a megtalált elempárok pakolása

1 Legyen az i. a legnagyobb még nem pakolt tárgy.
2 Ha $w_i < \frac{L_b}{2}$ vagy a pakolás után a tartalék $res < r$ lenne, akkor az algoritmus leáll.
3 Ha létezik olyan j. tárgy, ahol $w_i + w_j = L_b$, akkor pakoljuk ezt a két tárgyat egy új ládába. Töröljük a tárgyakat a még nem pakolt tárgyak halmazából és csökkentjük res értékét $C - L_b$ mértékével. Majd lépjünk az 1. lépésre.
4 Legyen $i = i + 1$ a következő nem pakolt tárgy indexe és lépjünk a 2. lépésre.

A következő segédalgoritmus a $Triplet(L_b, r)$. Az algoritmus a futása során elemhármasokat keres és pakol új ládákba. Ha egy megfelelő hármas megvan, akkor ezeket egy új ládába pakoljuk, töröljük őket a még nem pakolt tárgyak közül és a tartalék értékét csökkentsük. A megfelelő hármas kiválasztásánál a három tárgy együttes méretét és a tartalék pakolás utáni értékét veszem figyelembe.

Algorithm 10: Triplet(L_b, r)

Input: L_b, r
Output: a megtalált elemhármasok pakolása

1 Legyen az i. a legnagyobb még nem pakolt tárgy.
2 Ha $w_i < \frac{L_b}{3}$ vagy a pakolás után a tartalék $res < r$, akkor az algoritmus leáll.
3 Ha léteznek olyan j. és k. különböző tárgyak, ahol $w_i + w_j + w_k = L_b$, akkor pakoljuk ezt a három tárgyat egy új ládába. Töröljük a tárgyakat a még nem pakolt tárgyak halmazából és csökkentjük res értékét $C - L_b$ mértékével. Majd lépjünk az 1. lépésre.
4 Legyen $i = i + 1$ a következő nem pakolt tárgy indexe és lépjünk a 2. lépésre.

Az utolsó segédalgoritmus a $Quadret(L_b, r)$. Ez a segédalgoritmus már négyesével pakolja az elemeket egy ládába, az előzőekben látott feltételek mentén. Ha van olyan négy tárgy, amely megfelel a feltételeknek, akkor ezeket a tárgyakat egy ládába pakoljuk, töröljük a még nem pakolt tárgyak közül mind a négyet és csökkentjük a tartalék res értékét. Az algoritmus kiválasztja a két legnagyobb, még nem pakolt tárgyat. Ha a két tárgy méretének összege $(w_i + w_j)$ és a tartalék (res) értéke megfelel a feltételeknek, akkor ehhez a két legnagyobb elemhez keres egy párt úgy, hogy a négy elem összege pontosan L_b legyen. (Mivel az algoritmusok futása során bizonyos tárgyakat kiválasztunk és pakolunk, mások még pakolásra várnak, ezért a pakolatlan tárgyak sorrendjében "lyukak" keletkeznek.) Emiatt kell az előbbi képletben $(w_i + w_j)$ különböző indexeket alkalmazni, vagyis nem biztos, hogy $j = i + 1$.

Algorithm 11: Quadret(L_b, r)

Input: L_b, r

Output: a megtalált elemnégyesek pakolása

1 Legyen az i. a legnagyobb még nem pakolt tárgy.

2 Ha $w_i + w_j < \frac{L}{2}$ vagy a pakolás után a tartalék $res < r$, akkor az algoritmus leáll.

3 Ha létezik olyan k. és l. különböző tárgy, ahol $w_i + w_j + w_k + w_l = L$, akkor pakoljuk ezt a négy tárgyat egy új ládába. Töröljük a tárgyakat a még nem pakolt tárgyak halmazából és csökkentjük res értékét $C - L$ mértékével. Majd lépjünk az 1. lépésre.

4 Legyen $i = j$ és j a következő nem pakolt tárgy indexe, majd lépjünk a 2. lépésre.

Még egy definícióra van szükségünk. Megkülönböztetünk "kicsi" és "nagy" tárgyakat. Egy tárgy akkor nagy, ha a mérete nagyobb mint a ládaméret fele, egyébként kicsi. Ezt a megkülönböztetést azért tesszük, mert nyilvánvaló módon minden nagy tárgyat külön ládába kell tenni. Az előbb felsorolt segédalgoritmusokat fogom alkalmazni egy *Master* algoritmusban. Ennek során, ha van még hátra pakolatlan nagy tárgy, akkor óvatosabbak kell, hogy legyünk, ha már nincs hátra pakolatlan nagy tárgy, akkor "bátrabban" alkalmazhatjuk a segédalgoritmusokat.

Most, hogy a segédalgoritmusok ismertek, összeállítható a fő (*master*) algoritmus, ami az *FU* nevet kapta. Az algoritmus elsődlegesen a $Falkenauer\_U\_120$ alosztályhoz készült, azonban a paraméterek módosításával a további Falkenauer alosztályokra is alkalmazható. Az alábbiakban a segédalgoritmusokból felépítve, röviden bemutatom a teljes algoritmus működését. Elsőként, az algoritmus megpróbálja "jól" megtölteni a ládákat a lehető legnagyobb elempárokkal. Itt igyekszik az algoritmus megszabadulni a nagy méretű tárgyaktól, ugyanis ha ezek a tárgyak a legvégén kerülnek pakolásra, akkor nem biztos, hogy már nyitott ládába még belefférnek. Ha a tartalék értéke "nagy", az algoritmus jobban engedi az olyan ládákat használni, amelyekben a töltöttség nem éri el a kapacitást, azaz a láda nincs tele. Ahogy a tartalék értéke csökken, az algoritmus egyre szigorúbb a ládák töltöttségével kapcsolatban, azaz egyre kevésbé engedi meg a nem telepakolt ládákat. Amennyiben az elempárokkal sikerült a lehető legtöbb ládát "jól" megtölteni, utána következnek az elemhármasok, majd az elemnégyesek. Ezután a maradék (rendszerint csak néhány) tárgyat az algoritmus az FFD-vel pakolja. Az algoritmus pontos leírását az alábbi pszeudokód (Algorithm 12) szemlélteti. A második lépésben, $r_{b2(j+1)}$ a tartalék éppen aktuális értékét jelöli úgy, hogy a nem pakolt tárgyak között még van nagy méretű. (Ezt jelenti a b index, mint *big*.) Hasonlóan az r_{n2i} is a tartalék aktuális értékét jelöli úgy, hogy már nincs a még nem pakolt tárgyak között nagy méretű. Erre utal az n index, mint *no*. A $2(j+1)$ pedig azt jelenti, hogy a második lépés tartaléka (2-es index) és a tartalékokat tartalmazó vektor hányadik eleméről van szó $(j+1)$.

Az FU algoritmus hatékonysága a következő alfejezetben kerül tárgyalásra. Itt, most csak annyit érdemes megjegyezni, hogy az algoritmus a második lépésben (7-22. sorok) sokkal szigorúbb, ha még vannak nem pakolt nagy tárgyak. A harmadik

lépésben (23-30. sorok) az algoritmus egyáltalán nem szigorú a tartalék értékét illetően. Ez azt jelenti, hogy bármi is a tartalék értéke, a ládák pakolva lesznek, ha a töltöttség szintje minimum 148.

Az FU algoritmus kiértékelése

A 3.5 és 3.6. táblázatokban a Falkenauer_U osztályhoz készült algoritmusban alkalmazott paramétereket tartalmazzák alosztályok szerint. Az algoritmus leírásában az U120-as csoport paraméterei szerepelnek, de természetesen, ahogy a táblázatból is látható, minden alosztályra eltérő beállításokat alkalmaztunk. Az algoritmus működése minden beállítás esetében természetesen ugyanaz.

Algorithm 12: FU algoritmus

```
1  for i = 0, ..., 5 do
2  |  Pair(150 - i, r_1i)
3  end
4  while még van nem pakolt "nagy" tárgy do
5  |  for i = 0, ..., 5 do
6  |  |  Triplet(150 - i, r_b2i)
7  |  end
8  end
9  if nincs már nem pakolt "nagy" tárgy then
10 |  for i = 0, ..., 5 do
11 |  |  Triplet(150 - i, r_n2i)
12 |  end
13 end
14 if nincs már nem pakolt "nagy" tárgy then
15 |  for i = 0, ..., 2 do
16 |  |  Quadret(150 - i, r_3i)
17 |  end
18 end
19 A maradék tárgyak pakolása FFD-vel történik
```

	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{b20}	r_{b21}	r_{b22}	r_{b23}	r_{b24}	r_{b25}
U120	0	1	2	3	4	5	0	5	10	15	20	26
U250	0	1	2	30	40	50	0	5	10	30	30	30
U500	0	10	15	30	30	30	0	5	10	15	20	25
U1000	0	30	45	60	90	100	0	10	10	10	10	10

3.5. táblázat. A Falkenauer_U osztály paraméter-beállítása (v3)

	r_{n20}	r_{n21}	r_{n22}	r_{n23}	r_{n24}	r_{n25}	r_{30}	r_{31}	r_{32}
U120	0	5	10	15	30	30	0	0	0
U250	0	5	10	30	30	30	0	0	0
U500	0	5	15	30	30	30	0	0	0
U1000	0	30	40	50	60	70	0	0	0

3.6. táblázat. A Falkenauer_U osztály paraméter-beállítása (v3)

Vegyük észre, hogy az $r_{11}, r_{12}, \ldots, r_{32}$ értékek optimalizálhatók. Az értékek automatizált optimalizálása nem történt meg, csupán manuális beállítással kerültek kipróbálásra az értékek, amelynek a célja az volt, hogy "elég jó" eredményt érjünk el. Lehetséges, hogy egy optimalizáló eljárással még jobb eredményeket kaphatunk. További, a paraméterek beállításával, és általánosítási lehetőségekkel kapcsolatos észrevételek a 3.4.2.1. alfejezetben szerepelnek.

A 3.7. táblában az FU algoritmus eredményei láthatók a Falkenauer_U120 osztályra vonatkozóan, különböző beállítások mellett. A zöld színnel jelölt értékek azt jelentik, hogy ebben az esetben az algoritmus megtalálta az optimális megoldást. A piros jelölés esetében pedig nem találta meg. Minden sorra igaz, hogy $OPT = LB_1$.

	v1						**v2**	**v3**		
	FFD			**Útkereső**						
	150	**149**	**148**	**150**	**149**	**148**	**145**	**145**	**LB**	**Kezdeti tartalék**
u120_00	49	48	48	50	48	49	48	48	48	122
u120_01	49	49	49	49	49	49	49	49	49	145
u120_02	46	46	46	46	46	46	46	46	46	106
u120_03	50	49	49	50	49	49	49	49	49	65
u120_04	50	50	50	50	50	50	50	50	50	146
u120_05	48	48	48	48	48	48	48	48	48	78
u120_06	48	48	48	48	48	48	48	48	48	63
u120_07	50	50	49	50	49	49	49	49	49	55
u120_08	51	50	50	51	50	50	51	50	50	22
u120_09	47	46	46	47	46	46	47	46	46	30
u120_10	52	52	52	52	52	52	52	52	52	120
u120_11	50	49	50	51	50	50	49	49	49	103
u120_12	48	48	48	48	48	48	48	48	48	20
u120_13	49	49	49	49	49	49	49	49	49	148
u120_14	50	50	50	50	50	50	50	50	50	127
u120_15	48	48	48	48	48	48	48	48	48	98
u120_16	52	52	52	52	52	52	52	52	52	112
u120_17	53	52	52	54	52	52	52	52	52	97
u120_18	49	49	49	49	49	49	49	49	49	95
u120_19	50	50	50	50	50	50	50	49	49	28

3.7. táblázat. A Falkenauer_U120 osztály feladatainak megoldásai

Az eredmények alapján az alábbi konklúziókat és megfontolásokat teszem.

3.4.2.1. Észrevételek

1. Az első tapasztalat a ládák töltöttségére vonatkozik. A 2-4. oszlopban (vagyis az FFD-re vonatkozó eredmények oszlopaiban) látható, hogy minél szigorúbb, azaz nagyobb a töltöttségre vonatkozó L_b értéke, az FU algoritmus annál rosszabb eredményeket produkál. Így annak tiltása, hogy 149-nél alacsonyabb töltöttséget is elfogadok, nem hatékony. Emiatt a töltöttségi szinteket egészen 145-ös értékig engedtem akkor, ha a tartalék mértéke "nem túl kicsi".

2. Az FU algoritmust úgy módosítottam, hogy az utolsó lépésben (31. sor) nem az FFD pakolta a maradék tárgyakat, hanem a korábban bemutatott útkereső algoritmus. Ennek az eredményei a 5-7. oszlopokban láthatók. Látható, hogy nincs túl sok különbség az FFD-hez képest, azaz az algoritmus nem igazán érzékeny arra, hogy az utolsó elemeket milyen módon pakoljuk. Azaz, az utolsó tárgyak pakolása nem igazán befolyásolja az eredményt, ha a tárgyak többsége a korábbi lépésekben már "jól lettek pakolva". Emiatt később a sokkal egyszerűbb FFD került vissza az utolsó lépésbe a bonyolultabb útkereső algoritmus helyett.

3. A v2 és v3 algoritmusverziók esetében az L_b értékét tovább csökkentettem, egészen 145-ig. (Már ez a csökkentett érték szerepel az FU algoritmus leírásakor.) Látható, hogy a v3 verziójú algoritmus (9. oszlop) sikeresen megoldotta az összes feladatot. A v2 és a v3 közötti különbséget az eltérő paraméterbeállítás jelenti.

4. Fontos megjegyezni, hogy a Fakenauer U osztály majdnem minden feladata esetében az ismert optimális megoldás egyenlő az LB_1 értékével. Ez alól csak a Falkenauer_U250-es osztály 13-as számú feladata a kivétel, ahol $LB_1 = 102$ és $OPT = 103$.

5. A táblázatból látható, hogy számos esetben az algoritmus összes verziója megtalálta az optimális megoldást. Például ilyen a Falkenauer_u120_01. Ellenpéldaként a Falkenauer_u120_19 említhető, ahol az optimális megoldást ($OPT = 49$) csak a v3 verziójú algoritmus találta meg. Kiemelendő, hogy a Falkenauer_u120_12 esetében meglehetősen kicsi a tartalék mértéke, mindössze 20. Ennek ellenére az összes algoritmus megtalálta az optimális megoldást. Ezzel szemben a Falkenauer_u120_00 és a Falkenauer_u120_11 esetében sokkal több a tartalék, de mégsem talált optimális megoldást az algoritmus minden esetben.

6. Az itt bemutatott algoritmus a 80 feladatból 73 feladatot tudott optimálisan megoldani. Az u120-as csoportból mindet. A maradék 60 feladat esetében hasonló a helyzet, de néhány esetben az algoritmus egyik változata sem képes megtalálni az optimális megoldást. Ezeket nem részleteztük úgy, mint az U120-at a 3.7. táblázatban.

7. A futási idő meglehetősen kicsi. A 80 feladat megoldásához 31,8 másodpercre volt szükség. Az algoritmus komplexitása $O(n^3)$, a rejtett együttható a $O(.)$ szimbólumban körülbelül 20.

8. Látható, hogy a v3 változat az első alosztály (u120) esetében mindenhol optimális megoldást talált. A vizsgálatok alapján ez nem járt a futási idő megnövekedésével: a különböző változatok futásideje lényegében megegyezik. Azonban a v3-as változat csak az első alosztály esetében optimális. Sajnos nincs olyan univerzális beállítás, amely mindegyik alosztályra optimális lenne. Bizonyos beállítás az egyik alosztály esetén jobb, másik beállítás pedig másik alosztály esetén. Mivel a futási idő kicsi, ezért több változatot is kipróbálhatunk, amelyek közül a legjobbat választjuk.

9. Az FU algoritmus hatékonyságát a különböző paraméter beállítások kipróbálásával és mindig a legjobb beállítás rögzítésével lehetne növelni. Az is egy lehetőség, hogy a különböző alosztályok esetén más és más beállítást alkalmazunk.

A 3.8. táblázatban a Falkenauer_U osztály futtatásának futási idejei láthatók összehasonlítva a HEA algoritmussal. Az FU algoritmus átlagos futási ideje az összes feladatosztályra nézve közel 0,4 másodperc, a HEA algoritmusé viszont 1,48 másodperc. Az FU algoritmus teljes futási ideje mindössze 31,8683809 másodperc, míg a HEA teljes futási ideje 118,4 másodperc. Az új algoritmus a jelenlegi implementációban 3,7-szer gyorsabb mint a HEA, és ez továbbfejlesztéssel még várhatóan növelhető lesz a jövőben. Fontos tudni, hogy a jelenlegi implementáció az első verzió, így bőven van továbbfejlesztési lehetőség. Azt is jegyezzük meg, hogy a Falkenauer_U osztály esetén ugyan a HEA algoritmus lassabb, de minden esetben megtalálja az optimumot, míg az FU algoritmus az esetek 91%-ban. Korábban megjegyeztem (az előbbi észrevételek 8. pontjában), hogy mivel egyik paraméterbeállítás sem ad minden esetben optimális megoldást, emiatt több (például 4 féle) paraméterbeállítással is próbálkozhatunk. Ez azonban nem jelent négyszeres futási időt! Ugyanis, mint láttuk a v3 beállítással a 80 feladatból 73-at sikerül optimálisan megoldani. Akkor más paraméterbeállítással csak a maradék hét feladat esetén kell próbálkoznunk. Így összességében csak némileg lesz nagyobb futásidő, mintha csak egy paraméterbeállítással futtatjuk az algoritmust.

Osztály	FU Összes (s)	FU Átlag (s)	HEA átlag (s)
Falkenauer_U120	0,241859	0,01209295	1,48
Falkenauer_U250	1,1633459	0,058167295	
Falkenauer_U500	4,1469928	0,20734964	
Falkenauer_U1000	26,3161832	1,31580916	
Falkenauer_U	31,8683809	0.3983548	1,48

3.8. táblázat. Az FU algoritmus és a HEA futási ideje

Az FU algoritmus v1 és v2 verzióinak részletes paraméter-beállításai a D függelékben olvashatók.

3.5. Összefoglalás

A fő konklúzió az, hogy az előfeldolgozás egy nagyon hasznos eljárás, amelyet demonstráltam is ebben a fejezetben, felhasználva 2 ládapakolási benchmark feladat-

osztályt. Jól ismert, hogy a ládapakolási feladat NP-nehéz, így ebből az következhetne, hogy egy ládapakolási feladat megoldása szükségszerűen nehéz. De ez nem így van. Néhány konkrét feladat vagy akár egy teljes feladatosztály első ránézésre nehezen megoldható, azonban kiderül, hogy mégis viszonylag egyszerűen megoldható.

Másképpen megfogalmazva, a ládapakolási feladatok egy részét egyszerű trükkökkel könnyen megoldhatjuk. Nagyon érdekes probléma, hogy milyen módszerekkel lehet hatékonyan kiszűrni azt, hogy melyik feladat a nehéz. Jelenleg nem tudok ilyen általános módszert, amely hatékonyan eldönti a feladatokról, hogy azok nehezek vagy sem. A nehéz feladatok kiszűrésével a [80] foglalkozik. Ezzel ellentétben munkámban a könnyű feladatok kiszűrésével foglalkoztam.

A bemutatott mohó eljárások gyorsak, egyszerűek és a legtöbb esetben olyannyira segítenek a feladat egyszerűsítésében, hogy az algoritmus végül megtalálja az optimális megoldást. Összetettebb algoritmus alkalmazása csak abban az esetben indokolt, ha az egyszerűbb módszerek nem adnak optimális vagy elfogadhatóan jó megoldást.

Továbbá, 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. Ennek részletezése szerepel a 3.5.2. és 3.5.3. fejezetekben.

Az optimális megoldás ismeretére vonatkozóan kijelenthető, hogy nem kell ismerni az optimális megoldást. A Falkenauer osztály esetében 80 esetből 1 esetben az OPT nem egyenlő az LB-vel. A Schwerin 1 esetében véletlenszerűen vannak választva a tárgyak méretei 150 és 200 között. Emiatt átlagosan 175 a méretük. A 100 tárgy mérete összegének várható értéke 17 500. Nagyon kicsi a valószínűsége (elemi úton kiszámolható, hogy ez a valószínűség kisebb, mint 0,0003), hogy az összeg legfeljebb 17 000 vagy több mint 18 000. Emiatt mindig 18 lesz az alsó korlát. Ez igaz az osztály többi feladatára is.

3.5.1. A fő konklúzió részletesen bemutatva

Ahogy a fejezet korábbi szakaszában bemutattam, a Schwerin és Falkenauer_U osztályok esetében számos olyan tárgy volt a konkrét feladatokban, amelyek könnyen pakolhatók voltak úgy, hogy ezáltal nem sértettük az optimalitást, azaz a pakolásukkal továbbra is elérhető maradt az optimális megoldás lehetősége.

A Schwerin osztály esetében ez úgy nézett ki, hogy mindig az öt legnagyobb tárgyat pakolta az algoritmus egy ládába. Ezzel a módszerrel a tárgyak körülbelül 40%-a nagyon gyorsan pakolható volt az optimalitás megsértése nélkül.

A Falkenauer_U osztály esetében "jó párokat" keresett. Ezek olyan párok vol-

tak, amelyek mérete pontosan a láda kapacitásával volt egyenlő. Ezzel a módszerrel a tárgyak körülbelül 30%-a volt pakolható, és azok a ládák, amikbe pakolásra kerültek, pont tele lettek. A 3.4.2. fejezetbeli 1. Megjegyzés szerint a "jó párok" általánosabban is definiálhatók, és ezáltal esetleg további tárgyak pakolása is lehetséges.

Az egyszerűség kedvéért vegyük ketté a tárgyak csoportját *könnyű* és *nehéz* tárgyakra. A könnyű tárgyak azok, amelyeket előfeldolgozással könnyen pakolhatunk. A maradék tárgyak pedig a nehéz tárgyak.

1. Megállapítás. *Mindkét benchmark osztály esetében a nehéz tárgyak száma számottevően kisebb, mint a teljes tárgyhalmaz mérete.*

Ez azt jelenti, hogy a pakolandó tárgyak száma nagy mértékben csökkenthető, így a komplexebb eljárásokat csak a nehéz tárgyak halmazán kell alkalmazni. Természetesen minél kisebb a maradék tárgyak száma, annál könnyebb az optimális megoldás elérése.

Emlékeztetünk arra, hogy a Schwerin osztályban az optimális megoldás megtalálásának aránya 100%, a Falkenauer_U osztályban pedig 91% volt. Algoritmusaink kifejezetten erre a két osztályra lettek kifejlesztve. A *Skálázhatóság és komplexitás* fejezetben foglalkoztunk azzal, hogy a Schwerin algoritmusát hogyan lehet más feladatosztályra alkalmazni és a 3.5. fejezet elején foglalkozunk azzal, hogy a Falkenauer osztályra kidolgozott algoritmusokat hogyan lehet más osztályra alkalmazni.

2. Megállapítás. *Létezik olyan mohó algoritmus, amely a feladatosztályban szereplő feladatok többségében optimális pakolást végez.*

3.5.2. További kutatási lehetőségek

Ebben az alfejezetben 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. A 3.5. és a 3.6. táblázatban bemutatott paraméterek helyett más paraméterek kiválasztása. A paraméterek 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 [98] a feladat polinomiális időben megoldható, ahol n a tárgyak

száma és n tetszőlegesen nagy lehet. A polinomiális időben való megoldhatóság elenére 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. Ezekkel a kérdésekkel dolgozatomban nem foglalkoztam.

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

1. Sejtés. *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, ez azonban túlmutat dolgozatom keretein.

3.5.3. Mohó algoritmusok alkalmazásának korlátai

Ebben az alfejezetben azzal a kérdéssel foglalkozom, hogy melyek az algoritmusok alkalmazásának korlátai és, hogy az optimum értékét nem kell előre ismerni.

Természetesen a mohó algoritmusoknak vannak korlátaik, mivel a ládapakolási feladat NP-nehéz. Mik is ezek a korlátok pontosan?

Vegyük észre, hogy mindkét vizsgált feladatosztály (Schwerin és Falkenauer_U) esetén a tárgyak méretei egy megadott intervallumból kerültek ki, ráadásul egyenletes eloszlással. A Schwerin feladatok esetében a láda kapacitása 1000, a tárgyak mérete pedig 150 és 200 között változik. Ez egy meglehetősen szűk intervallum. A Falkenauer_U esetében már nem ilyen szűk. A láda mérete 150, a tárgyak méretei pedig 20 és 100 között változik. Ezek alapján a következő konklúzió vonható le.

3. Megállapítás. *Tegyük fel, hogy az inputban minden tárgy valamely, viszonylag kicsi d számnál kisebb méretű (például $d = 0,2$, mint a Schwerin osztály esetén). Ekkor, minél szűkebb az az intervallum, ahonnan a tárgyak méretei kerülnek kiválasztásra, annál nagyobb az esélye annak, hogy hatékony mohó algoritmus adható a feladatra.*

Vegyük észre, hogy mindkét feladatosztály esetében a tárgyak méretei véletlenszerűen kerülnek kiválasztásra egyenletes eloszlás mellett. Ez azt jelenti, hogy a jövőbeni kutatásokban érdemes megvizsgálni olyan példákat, ahol a tárgyak méretei nem egyenletes eloszlás szerint vannak generálva.

A bemutatott eredmények alapján látható, hogy a Falkenauer_U osztály esetében a bemutatott mohó algoritmus jól működik akkor, ha a tartalék (azaz a még

fel nem használt hely) mérete megfelelően nagy. Ha a kezdeti tartalék kevés, akkor annak az esélye, hogy a mohó algoritmus optimális megoldást talál, nagyon kicsi.

4. fejezet

Egy új feladat: ládafedés szállítással

Ebben a fejezetben, amint a Bevezetésben már említettük, egy új feladattal foglalkozom, amelynek neve ládafedés szállítással, röviden BCD. Ebben a problémában, hasonlóan a ládapakolási problémához, tárgyakat pakolunk ládákba, amelyeket, ha fedetté válnak, 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 [42]-ben szerepel. Most ebben a fejezetben a problémának a kiterjesztésével és alapos vizsgálatával foglalkozunk. A probléma offline változatával a [43] foglalkozik, továbbá néhány kapcsolódó probléma a [44]-ben kerül bemutatásra. A fejezetben szereplő eredmények a [99] cikkben lettek bemutatva.

4.1. Problémafelvetés és néhány tulajdonság az offline és online modellek esetében

A probléma a következőképpen fogalmazható meg. A tárgyak egyenként érkeznek az L lista alapján. 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

$$G : \{1, \ldots, K\} \to R^+. \tag{4.1}$$

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.

Vegyünk egy egyszerű példát. Egy kiskereskedésben többféle gyümölcsöt pakolnak kis dobozokba. Minden dobozt egy minimum súlyig meg kell tölteni, de a

szükséges mennyiségnél *lehetőleg* ne legyen túl sokkal több a dobozban (mert akkor kevesebb dobozt fogunk tudni megtölteni). Ebben az esetben a kevesebb nyitott doboz előny, mert így könnyebb a dobozokat kezelni. A minimális súly (amennyinek mindenképp meg kell lenni) ebben az esetben a láda kapacitását jelenti, és szeretnénk, hogy ennél ne legyen sokkal több a ládában (mert az nekünk veszteséget jelent), tehát a láda ne legyen nagyon túltöltve. Azaz, megengedjük, hogy a kapacitás fölé menjen a töltöttség, de ne túlságosan.

Az offline és az online algoritmusok hatékonyságát rendszerint versenyképességi analízissel vizsgáljuk. Ez azt jelenti, hogy a megoldás $C_A(I)$ értéke (ami egy A offline vagy online algoritmus által lett meghatározva az I bemenetre) van összehasonlítva (elosztva) az offline $C^*(I)$ optimummal. A maximalizálás esetén az összes I bemenetre vett $\frac{C_A(I)}{C^*(I)}$ arány alsó határát (infimumát) az A algoritmus approximációs arányának nevezzük offline esetben.

A probléma offline változata esetén a tárgyak sorrendje és mérete előre ismert, továbbá ismerjük a G profit függvényt is. Továbbá a tárgyak pakolása az L lista szerinti sorrendben történik. Az offline algoritmust optimálisnak nevezzük, ha az algoritmus megoldásához tartozó összes haszon, amit a ládákért kaptunk az L lista esetén, a lehető legnagyobb.

Az online változat esetén viszont a bemenetről nem tudunk előre semmit. Nem ismerjük a tárgyak sorrendjét és a méreteiket sem. Azonban a G függvény itt is ismert, ugyanis ez nem része a bemenetnek. Online esetben minden döntést a következő tárgy érkezése előtt kell meghozni. A döntés azt jelenti, hogy az aktuális tárgyat melyik ládába pakolja az algoritmus.

Minden véges L listára (ami a pakolandó tárgyak listája) és minden G profit függvényre legyen $C_A(L,G)$ a megoldás értéke, amit egy A offline vagy online algoritmus ért el a pakolás során. Ezt az értéket összehasonlítjuk az $C^*(L,G)$ offline optimummal. Egy A online algoritmus ρ-kompetitív ($0 \leq \rho \leq 1$), ha a

$$\frac{C_A(L,G)}{C^*(L,G)} \geq \rho \tag{4.2}$$

feltétel teljesül minden L listára és G profit függvényre. Online esetben a legnagyobb olyan ρ-t, amelyre A algoritmus kompetitív, versenyképességi aránynak nevezzük. Másrészről, ha létezik olyan L valamilyen G esetén, amelyre teljesül a

$$\mu \geq \frac{C_A(L,G)}{C^*(L,G)} \tag{4.3}$$

feltétel minden A online algoritmusra, akkor μ a probléma felső korlátja. Egy online algoritmus akkor optimális, ha ρ értéke egyenlő a μ infimumával.

4.1.1. Az offline modell tulajdonságai

Mind a ládapakolási, mind a ládafedési feladat erősen NP-nehéz, mert 3-partíciós feladatra visszavezethető. (A 3-partíciós probléma esetén adott $3n$ tárgyunk (számunk), amelyek összege $n \times B$, ahol $B \in \mathbb{R}$. A tárgyak méretei az $(\frac{1}{4}B, \frac{1}{2}B)$ intervallumból kerülnek ki. A kérdés az, hogy lehet-e úgy n darab hármas csoportot

létrehozni, hogy minden csoportban a három darab tárgy méreteinek összege pontosan B.) Tehát, ha nincs semmilyen speciális kikötés az L listára vagy a G függvényre, akkor a feladat már erősen NP-nehéz. Ha a G függvény konstans, akkor a klasszikus ládafedési problémát kapjuk. Ha G nem konstans, akkor a feladat megoldása még nehezebb lehet.

A következőkben két tételt adok meg, amelyek bizonyítását a [43] tartalmazza. Az első tétel szerint, az offline optimum hatékonyan megtalálható, ha a tárgyak méretei egy pozitív alsó korláttal rendelkeznek (a felső korlátot a láda kapacitása jelenti) tetszőlegesen megválasztott G függvény mellett.

3. Tétel. *[43] Legyen K és b rögzített egészek, $G : \{1, \ldots, K\} \to R^+$ tetszőleges profit függvény, és $c > 0$ egy rögzített valós szám. Ekkor*

(i) ha az összes tárgy méretének legalább c-nek kell lennie, akkor bármilyen L lista esetében az offline optimum polinomiális időben kiszámítható. Nevezetesen, a szükséges lépések száma legfeljebb $O(n^{1+\frac{K}{c}})$,

(ii) ha a bemenetben szereplő tárgyak méretei legfeljebb b különböző érték közül kerülnek ki és egyik sem kisebb, mint c, akkor bármilyen L esetében az offline optimum lineáris időben kiszámítható. A futási idő nagyságrendje $nb^{O(\frac{K}{c})}$.

Továbbá az is megmutatható, hogy offline problémára nem létezik APTAS (asymptotic polynomial time approximation scheme), azaz aszimptotikus polinomiális idejű approximációs séma. Pontosabban, a következő tétel igaz.

4. Tétel. *[43] Van olyan választása a K értéknek és a G függvénynek, amelyek esetén létezik olyan L listákat tartalmazó osztály, amelyre nem létezik olyan polinomiális futási idejű algoritmus, amelynek az aszimptotikus approximációs aránya $\frac{6}{7}$-nél jobb lenne, ha $P \neq NP$.*

Fontos megjegyezni, hogy a 3. tétel szerint, a 4. tétel esetében említett L listának tetszőlegesen kicsi méretű tárgyakat kell tartalmaznia.

4.1.2. Benchmark osztályok

Az optimalizálási problémák jelentős részére jellemző, hogy valamilyen benchmark feladatot vesznek alapul. A ládapakolási probléma esetében a bolognai egyetem operációkutatási csoportja rendelkezik egy bőséges feladatgyűjteménnyel [76]. Az ebben a fejezetben vizsgált probléma nem tisztán ládapakolási vagy ládafedési probléma, ugyanis egy profitfüggvény is a modell része.

Egy nem régi publikációnkban [100] és a 3. fejezetben két típust választottam ki a benchmark feladatok közül: Schwerin és a Falkenauer_U. Most is ezekkel dolgoztam (de ezeket ki kellett egészíteni megfelelő G haszonfüggvénnyel). A Schwerin osztály a [77]-ben került definiálásra. A másik feladatosztály a Falkenauer_U, amely a [78]-ban definiált. A Schwerin és a Falkenauer_U osztályok tulajdonságait a 3.2. alfejezet részletezi.

4.1.3. A feladatok előkészítése

Ebben az alfejezetben bemutatom, hogy a meglévő feladatosztályok esetében milyen módosításokat hajtottam végre, definiálom az általam létrehozott új feladatosztályt, valamint bevezetem a nyereségfüggvényeket.

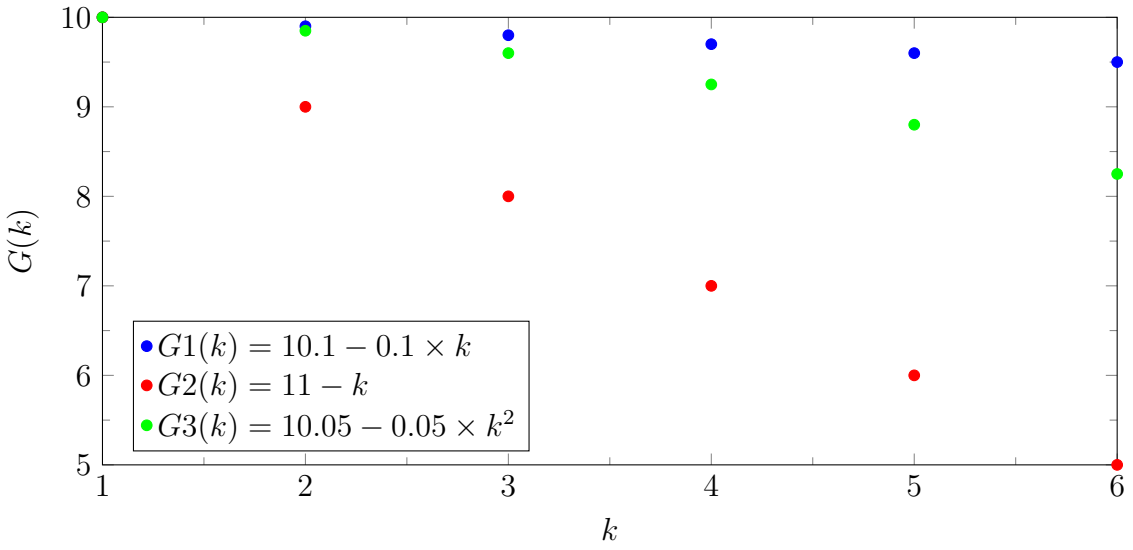
(a) **Rendezettség megbontása.** A feladatok esetében a tárgyak méret szerint csökkenő sorrendbe vannak rendezve. Ezért az első módosítás az alkalmazott feladatosztályokon az volt, hogy a tárgyakat méretük alapján véletlenszerűen összekevertem.

(b) **Falkenauer osztály normalizálása.** A Falkenauer osztály normalizálásra került úgy, hogy a láda kapacitását 1000-nek vettem. Emiatt a tárgyak méreteit is újra kellett számolni. Minden tárgy méretét $\frac{1000}{150}$ értékkel kellett megszorozni. Az (a) és a (b) lépések után a bemenettípusokat S1, S2, F1, F2, F3 és F4 jelöli (ahol az S betű a Schwerin típusra, az F betű pedig a Falkenauer típusra utal).

(c) **Új feladatosztály.** Ahogy korábban volt róla szó, a Schwerin osztály esetében a tárgyak méretei egy szűk intervallumból kerülnek ki, a láda mérete 1000. A Falkenauer osztály esetében a normalizálást követően a láda kapacitása 1000 és a tárgyak száma immáron a $[133, 666]$ intervallumból kerül ki. Az ehhez a témához kapcsolódó, korábbi publikációban [42] három osztály szerepelt, ahol a tárgyak méretei egy szélesebb intervallumból kerültek ki. Emiatt létrehoztam egy új osztályt, ahol a tárgyak méretei az $[1, 1000]$ intervallumból valók. Látható, hogy az intervallum maximuma megegyezik a láda kapacitásával. Az osztályt LR-nek neveztem el, amely a Large Range rövidítése. Ebben az osztályban összesen 400 darab feladat található, mindegyik esetében a tárgyak száma legfeljebb 1000. Az egyik alosztálynak a neve LR4, ahol 100 darab tárgy van. Ezután még három osztály lett kialakítva úgy, hogy csak az első 120, 250 vagy 500 tárgyat tartottam meg. Ezek az alosztályok az LR1, LR2 és LR3 neveket kapták. Az alábbiakban egy összefoglaló táblázat (4.1) látható az alkalmazott osztályokról.

Osztály	Feladatok száma	Ládaméret	Intervallum	Tárgyak száma
S1	100	1000	[150;200]	100
S2	100	1000	[150;200]	120
F1	20	1000	[133;666]	120
F2	20	1000	[133;666]	250
F3	20	1000	[133;666]	500
F4	20	1000	[133;666]	1000
LR1	100	1000	[1;1000]	120
LR2	100	1000	[1;1000]	250
LR3	100	1000	[1;1000]	500
LR4	100	1000	[1;1000]	1000

4.1. táblázat. A feladatosztályok összefoglaló táblázata



4.1. ábra. Nyereségfüggvények

(d) **Nyereségfüggvények.** Az eredeti benchmarkok esetében nincs megadva semmilyen nyereségfüggvény. Emiatt három függvényt hoztam létre, amelyeket az alábbiakban szeretnék bemutatni. $k = 1, 2, \ldots$ esetén az alábbi függvényeket alkalmaztam. A függvények grafikus ábrázolása a 4.1 ábrán látható.

- $G1(k) = 10,1 - 0,1 \times k$, azaz $G1(1) = 10$, $G1(2) = 9,9$, $G1(3) = 9,8$ és így tovább. Látható, hogy egy nagyon lassan csökkenő függvényről van szó.
- $G2(k) = 11 - k$, azaz $G2(1) = 10$, $G2(2) = 9$, $G2(3) = 8$ és így tovább. Látható, hogy ez a függvény k egységnyi növekedése esetén meredeken csökken.
- $G3(k) = 10,05 - 0,05 \times k^2$, azaz $G3(1) = 10$, $G3(2) = 9,85$, $G3(3) = 9,6$ és így tovább. A függvény kezdetben lassan csökken, majd egyre meredekebben. Ahogy a 4.1. ábrán is látható, a $G1$ és a $G2$ között helyezkedik el a csökkenési ütemet vizsgálva.

Egy egyszerű gyakorlati példát az előbb megadott különböző nyereségfüggvényekre a következőképpen tudunk megadni: Tekintsünk megint egy kisüzemet, ahol valamilyen áru csomagolása folyik:

- G1 - egy precíz és gyors robot pakol
- G2 - egy ember pakol
- G3 - több ember pakol

Vagyis: Ha egy robot pakolja az árut, és feltételezzük, hogy maga a robot (a mozgása) gyors, valamint gyors a reagálása is, tehát az információt gyorsan tudja feldolgozni, akkor a robot számára nagyjából mindegy, hogy hány láda

van kinyitva, több láda esetén csak kicsit romlik a robot teljesítménye. Ezt fejezi ki a G1 nyereségfüggvény. Ellenkező esetben egy ember (egy alkalmazott van csak az adott helyen, aki a munkát végzi) pakol, neki (a 8 órás műszak alatt) lényeges hogy mennyit kell járkálnia a megnyitott ládák között, és nem is könnyen látja át, hogy mit hova tegyen, mert ezeket fejben kell kiszámolnia. Az ő esetében tehát egy gyorsan csökkenő haszonfüggvény alkalmazható, ezt fejezi ki a G2 függvény. A G3 eset pedig a kettő között van, itt több ember végzi a rakodást, kevés nyitott ládával még jól elboldogulnak, de egy bizonyos ládaszámon túl már hasonlóan problémás lesz az eset az ő számunkra is mint egy ember esetén. Ez tehát a G3 függvény esete.

(e) **Tárgytípusok és a 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 $u = 1, 2, 3$.

A következő fejezetekben bemutatok néhány algoritmust, amelyeket a kutatásom során vizsgáltam. Elsőként az algoritmusok teljesítményét osztályonként csak egy feladatpéldányon szemléltetem, de a 4.5. alfejezetben részletes vizsgálatok mentén mutatom be az algoritmusok hatékonyságát. Ez összesen 680 feladatot jelent és figyelembe véve a három nyereségfüggvényt is, így összesen $3 \times 680 = 2040$ esetre történnek a vizsgálatok.

Vegyük figyelembe, hogy az ehhez a területhez tartozó előzetes publikációban [42] csak 6 darab alosztályt vettek figyelembe, amelyek mindegyike 10 darab feladatot tartalmazott.

4.2. Természetesen adódó online algoritmusok

Ebben a fejezetben néhány, természetesen adódó algoritmust mutatok be. Az algoritmusok működése egyszerű, alkalmazásuk kényelmes és könnyen implementálhatóak.

4.2.1. Dual Next Fit algoritmus

A Dual Next Fit, vagy röviden DNF algoritmus működése rendkívül egyszerű. Egyszerre csak egy ládát tart nyitva és a következőképpen működik:

- 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.

A DNF algoritmus esetében mindig csak egyféleképpen lehet pakolni a következő tárgyat, hiszen egyszerre csak egy láda lehet nyitva. Az algoritmus az egyszerűsége ellenére is optimális, abban az esetben ha az elszállított ládákért kapott profit 0, ha egynél több láda van nyitva.

1. Lemma. *Tegyük fel, hogy $G(k) = 0$ minden $2 \leq k \leq K$ esetén. Ekkor a Dual Next Fit algoritmus optimális.*

Bizonyítás. Nyilvánvalóan nem érdemes kettő vagy több ládát nyitni. ■

Akkor is optimális az algoritmus, ha a tárgyak méretei megadott feltételeket teljesítenek a következő lemma alapján.

A következő lemmák bizonyítása egyszerűbb úgy, ha feltételezzük, hogy a ládaméretek és a tárgyméretek normalizálva vannak úgy, hogy a ládamérettel elosztjuk a tárgyméreteket és a ládaméretét $C = 1$-nek vesszük.

2. Lemma. *Tegyük fel, hogy $\frac{1}{k} \leq w_i < \frac{1}{(k-1)}$ feltétel minden w_i tárgyméretre igaz, ahol $k \geq 2$ és a ládaméret normalizálva van, azaz $C = 1$. Ekkor a Dual Next Fit algoritmus optimális.*

Bizonyítás. Minden fedett láda pontosan k tárgyat tartalmaz. Ezért a legjobb megoldás, ha egyszerre csak egy láda van nyitva. ■

Természetesen egyáltalán nem biztos, hogy a tárgyak méretei megfelelnek a 2. lemmában megfogalmazott feltételnek. Ezért általában néhány láda túl lesz pakolva, azaz néhány fedett láda töltöttsége jóval meghaladhatja majd a láda kapacitását.

3. Lemma. *A Dual Next Fit versenyképességi aránya legalább $\frac{1}{2}$ tetszőleges G nyereségfüggvény mellett.*

Bizonyítás. Az állítás alapja az a tény, hogy $G(k)$ egy nem növekvő függvény k szerint. Továbbá, az egy ládába pakolt tárgyak összmérete nem lehet több, mint 2, mivel minden tárgy mérete legfeljebb 1. ■

Vegyük figyelembe, hogy az S1 és S2 típusú inputok nagyon hasonlóak ahhoz az esethez, mint amikor a tárgyak méreteire igaz, hogy $\frac{1}{k} \leq w_i < \frac{1}{(k-1)}$ valamely k esetében. A normalizálás miatt a Schwerin osztály tárgyméretei az $\left(\frac{1}{7}; \frac{1}{5}\right]$ intervallumba esnek. Ez azt jelenti, hogy a DNF némileg túlpakolhatja a ládát: minden fedett láda töltöttsége legfeljebb $\frac{6}{5}$. Legyen I egy feladatpéldány az S1 vagy S2 osztályból. Tételezzük fel, hogy optimális megoldás esetén a fedett ládák száma C^*. Ekkor az optimum értéke legfeljebb $C^* \times G(1)$. Továbbá könnyű belátni, hogy a DNF legalább $\lfloor \frac{5}{6} \times C^* \rfloor$ ládát nyit, továbbá $G(1)$ profitot kap minden fedett láda után, emiatt a következő állításokat tehetjük:

1. Állítás. *A Schwerin osztálynál DNF esetén a célfüggvény értéke legalább $\lfloor \frac{5}{6} \times C^*(I) \rfloor$, ahol $C^*(I)$ az optimum értéke az I input esetén.*

A fenti állítást általános formában is megfogalmazhatjuk az alábbiak szerint:

2. Állítás. *Tegyük fel, hogy az összes tárgy mérete az $\left(\frac{1}{q}; \frac{1}{t}\right]$ intervallumból kerül ki, valamilyen q és t egészekre. Ekkor a DNF célfüggvény értéke legalább $\lfloor \frac{t}{(t+1)} \times C^*(I) \rfloor$.*

A DNF algoritmus ellenőrzése céljából számos teszt futtatása történt, viszont mivel minden esetben egymáshoz nagyon hasonló eredményeket kaptunk osztályon belül, így csak az $S1Gu$, $F1Gu$ és $LR1Gu$ ($u = 1, 2, 3$) osztályokat vesszük górcső alá, és ezekből is csak a legelső feladatot.

A tesztek során keletkező megoldások alapján kiszámítottam a ládák átlagos töltöttségét (lefelé kerekítve), az elszállított (azaz fedett) ládák számát és a nyereségfüggvény értékét. Mivel a DNF egyszerre csak egy ládát tart nyitva és mindhárom nyereségfüggvény értéke $k = 1$ esetén ugyanaz, ezért csak a $G1$ függvény értékét jelenítettem meg ($G2$ és $G3$ értéke ugyanaz, mint a $G1$ értéke).

Osztály	Ládaméret	Ládák átlagos töltöttsége	Ládák száma	Összes haszon (G1)
S1G1	1000	1072	16	160
F1G1	1000	1228	38	380
F4G1	1000	1236	322	3220
LR1G1	1000	1470	43	430
LR2G1	1000	1424	98	980
LR3G1	1000	1377	192	1920
LR4G1	1000	1359	365	3650

4.2. táblázat. A DNF algoritmus eredményei (1 feladat/osztály)

Emlékeztetek arra, hogy a Schwerin osztály esetében, a Schwerin_1 alosztály feladatainál tudjuk, hogy a tárgyak összmérete több, mint a ládaméret 17-szerese, de kevesebb, mint 18-szorosa. Emiatt nyilvánvaló, hogy 17-nél több ládát biztosan nem lehet lefedni a tárgyakkal. Ezt már a korábbi publikációmban [100] tisztáztam (hogy a tárgyakkal legfeljebb 17 láda fedhető). Tehát ehhez hasonlítva az $S1G1$ osztály esetében a DNF 16 ládát fedett a tárgyakkal úgy, hogy egy nagyon egyszerű algoritmusról van szó. Ez egy nagyon jó eredmény. Az alosztály sok inputja esetén az optimum érték valójában 17, mert amikor pakoljuk a tárgyakat (tehát nem a fedési, hanem a pakolási feladat esetében) a [100] publikáció vizsgálatai során azt találtuk, hogy a megfelelő pakolási algoritmusom majdnem 17 ládát pontosan megtölt és a többi láda is majdnem tele van töltve a 17. ládáig és még vannak kicsi tárgyak a 18. ládában. Emiatt a 18. láda kicsi tárgyaival le tudjuk fedni azt a néhány ládát, amelyik nem lett teljesen telepakolva.

Visszatérve a DNF algoritmushoz, ezen a ponton a 4.2. táblázat alapján még nem eldönthető, hogy a DNF a többi osztály esetében is ilyen hatékony-e.

Megfogalmazható egy további állítás az optimum és a DNF értékének összehasonlításával. Legyen I egy feladatpéldány. A fedett ládák átlagos töltöttsége legyen a, a ládák száma pedig n. Ekkor a tárgyak összmérete $s = a \times n$ plusz azon tárgyak összmérete, amelyek esetleg egy nem fedett ládában vannak. Figyelembe véve ezeket a tárgyakat is $s \leq a \times n + C$ adódik, és emiatt az optimális megoldás értéke legfeljebb $\left(\frac{a \times n}{C} + 1\right) \times G(1)$. A tárgyak összmérete minden fedett ládában legalább C, a nyereségfüggvény pedig nem növekvő. Továbbá, a DNF által adott megoldás értéke $n \times G(1)$ mert a G függvény nem növekvő, ezért az alábbi állítás tehető:

3. Állítás. *DNF esetén a célfüggvény értéke nem lehet kisebb, mint* $\frac{1}{\frac{a}{C} + \frac{1}{n}} \times C^*(I)$.

4.2.2. Dual Harmonic(K) algoritmus

Amennyiben a ládafedés során okosabban szeretnénk kihasználni a ládák kapacitását, úgy érdemes egyszerre több ládát is nyitva tartani. Tekintsük a következő klasszikus algoritmust, amely a Harmonic(K) algoritmus, vagy röviden H(K), ahol $K \geq 1$ egy egész szám. Az algoritmus alapötlete, hogy egy ládába méretükben hasonló tárgyak kerüljenek. Egyidejűleg maximum K darab láda lehet nyitva. A tárgyakat az algoritmus osztályozza a méretük alapján, és minden láda egy osztályt reprezentál. 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. Az algoritmus az alábbiak szerint működik:

- Az algoritmus a következő tárgyat a k. típusú ládába pakolja ($k = 1, \ldots, K$), ha létezik ilyen nyitott láda és a tárgy mérete az I_k intervallumba esik. Ha a láda fedetté válik, az algoritmus lezárja. Ha nincs ilyen láda nyitva, akkor nyit egy ilyen típusút. Ha nincs több elem, akkor az algoritmus megáll.

A H(K) algoritmus esetében is, hasonlóan a DNF-hez, megtörténtek a tesztfuttatások, $K = 2, 3, 4, 5$ esetekre. Az alkalmazott osztályok majdnem teljesen ugyanazok voltak mint a DNF esetében. Viszont, mivel a H(K) esetében az algoritmus több ládát is tarthat nyitva a futás során, így mindhárom nyereségfüggvény be lett vonva a futtatásokba. Minden osztály esetében mindhárom nyereségfüggvény kiértékelésre került. Az eredményeket tartalmazó táblázatokban csak a nyereségfüggvények értékei szerepelnek, továbbá a korábban a DNF esetében kiszámított nyereségfüggvény értékei is helyet kaptak az összehasonlíthatóság végett. Minden érték lefelé kerekített. Minden sorban a legjobb értékeket színes háttérrel jelöltem.

Osztály	DNF	H(2)	H(3)	H(4)	H(5)
S1G1	160	160	160	160	160
S1G2	160	160	160	160	160
S1G3	160	160	160	160	160
F1G1	380	397	385	382	368
F1G2	380	373	340	311	265
F1G3	380	395	381	373	349
F4G1	3220	3369	3314	3258	3214
F4G2	3220	3183	2905	2613	2358
F4G3	3220	3358	3276	3171	3064

4.3. táblázat. A H(K) algoritmus által szerzett profit az S és F osztályok esetében

2. Megjegyzés. *A 4.3. táblázatban az eredmények közül az S1 osztálynál minden esetben a profit értéke 160. Ez meglepő lehet. Viszont vegyük figyelembe, hogy itt a Schwerin_1 osztályról van szó, ahol a tárgyak mérete 150 és 200 között változik, a ládaméret pedig 1000. Azaz minden tárgy K típusú ládába kerül, mivel csak egy láda van használatban egyszerre. Ez akkor változhatna, ha $K \geq 6$ lenne, azonban ekkor*

a nyereségfüggvény nagyon lecsökkenne és sok nyitott láda használata nem lenne előnyös.

Ennél érdekesebbek az F1 osztály eredményei mindhárom nyereségfüggvényt tekintve. Látható, hogy H(2) jobb eredményt ért el, mint a DNF a G1 és G3 esetében. Az is látható, ahogy K értéke növekszik, úgy mindhárom nyereségfüggvény értéke csökken. Azt mondhatjuk, ha a nyereségfüggvény lassan csökken (G1 csökken a leglassabban, utána a G3), akkor előnyös több ládát nyitva tartani, ugyanis így az algoritmus nagyobb eséllyel tud "jó" pakolást csinálni.

A helyzet hasonló az F4 osztályok esetében is. Viszont itt sokkal több tárgy van feladatonként, mint az előző osztályoknál. Emiatt van nagyobb különbség az értékekben a DNF-hez hasonlítva.

Összefoglalva elmondható, hogy a H(K) algoritmus jobban vagy sokkal jobban képes teljesíteni, mint a DNF abban az esetben ha K "nem túl nagy" és az elszállított (lezárt) ládák függvényében számított nyereségfüggvény nem csökken "túl gyorsan".

Osztály	DNF	H(2)	H(3)	H(4)	H(5)
LR1G1	430	437	423	420	407
LR1G2	430	415	362	332	298
LR1G3	430	436	417	407	387
LR2G1	980	953	935	918	911
LR2G2	980	900	802	724	653
LR2G3	980	951	922	890	862
LR3G1	1920	1877	1841	1806	1792
LR3G2	1920	1763	1582	1418	1273
LR3G3	1920	1870	1815	1751	1694
LR4G1	3650	3546	3507	3459	3428
LR4G2	3650	3333	3036	2738	2427
LR4G3	3650	3534	3461	3358	3250

4.4. táblázat. A H(K) algoritmus által szerzett profit az LR osztály esetében

Az LR1 alosztály esetében az eredmények hasonlóak a 4.3. táblázatban közöltekhez. Az LR2, LR3 és LR4 esetében azonban más a helyzet. Továbbra is igaz az a megállapítás, hogy ha a K értéke növekszik, akkor a H(K) eredményei romlanak. Látható az is, hogy a H(K) az LR osztály esetében nem tudja megverni a DNF algoritmust. Még H(2) is rosszabbul teljesít, akkor is, ha G1 értékét vesszük, amely a leglassabban csökkenő függvény a három közül. Az ok az lehet, hogy az LR osztályban a tárgyak méret szerinti diverzitása nagy, mivel a $[1, 1000]$ intervallumból kerülnek ki, a ládaméret pedig 1000. Szóval, például a H(2) esetében feltételezhetően az történik, hogy "mindig" két ládát tart nyitva az algoritmus. Ezért G1(1) értékét nem éri el, csak G1(2)-t, de az kevesebb, mint G1(1).

4.2.3. Smart Dual Harmonic(K) algoritmus

Ebben az alfejezetben a H(K) algoritmus egy továbbfejlesztését mutatom be, amivel az eredeti eljárás "okosabbá" tehető. Ezt az új algoritmust Smart Harmonic(K)-

nak, vagy röviden SH(K)-nak nevezzük. Az algoritmus az alábbi egyszerű szabályok mentén működik.

- 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.

Ahogy látható, az SH(K) algoritmus majdnem ugyanúgy működik, mint a H(K). Az egyetlen különbség akkor van, ha az algoritmus a következő tárgynak talál fedhető ládát. Ebből több is lehet, így mindig a legkisebb töltöttségűbe fogja pakolni. Ezzel a lépéssel a nyitott ládák számát szeretnénk leszorítani, hogy a profit értéke minél nagyobb legyen.

Osztály	**DNF**	**SH(2)**	**SH(3)**	**SH(4)**	**SH(5)**
S1G1	160	160	160	160	160
S1G2	160	160	160	160	160
S1G3	160	160	160	160	160
F1G1	380	408	413	414	412
F1G2	380	394	393	360	341
F1G3	380	407	415	408	402
F4G1	3220	3359	3423	3431	3459
F4G2	3220	3264	3187	2997	2822
F4G3	3220	3354	3406	3386	3369

4.5. táblázat. A SH(K) algoritmus által szerzett profit az S és F osztályok esetében

A 4.5. táblázatban látható futási eredmények ebben az esetben is a nyereségfüggvény értékeit mutatják. Látható, hogy az S1 osztály esetében ugyanazt az eredményt kaptuk, mint a H(K) algoritmus esetében. Itt az SH(K) algoritmus új lépése nem hozott jobb eredményt. Ellenben az F1 osztály esetében az SH(K) sokkal jobb eredményt ért el, mint a H(K). Az F1G1 esetében $K = 4$ értékig a nyereségfüggvény értéke növekszik. A helyzet hasonló az F1G3 esetében, itt a maximumot a $K = 3$ érték esetében éri el a függvény. Az F4 osztály esetében is látható, hogy az SH(K) algoritmus szignifikánsan jobb, mint a DNF és szintén jobb, mint a H(K) algoritmus. A legnagyobb profitot az F4 osztályon belül az F4G1 esetében tudta elérni az algoritmus $K = 5$ esetén.

A H(K) algoritmus esetében azt mutatták az eredmények, hogy a H(K) algoritmus nem tudott jobb teljesítményt nyújtani az LR2, LR3 és LR4 esetében, mint a DNF algoritmus. Azonban az SH(K) algoritmus jobb eredményeket hozott még a gyorsan csökkenő G2 nyereségfüggvény esetében is. Az LR1 alosztálynál a legnagyobb profitot a G1 függvény produkálta $K = 4$ esetén. Az LR2, LR3 és LR4 esetében hasonló a helyzet.

Összességében tehát elmondható, hogy az összes feladatot tekintve az SH(K) algoritmus többségében jobb eredményeket ért el, mint a H(K) algoritmus.

Osztály	DNF	SH(2)	SH(3)	SH(4)	SH(5)
LR1G1	430	478	487	516	515
LR1G2	430	465	460	481	477
LR1G3	430	477	485	513	512
LR2G1	980	1047	1074	1132	1131
LR2G2	980	1021	1023	1060	1054
LR2G3	980	1045	1071	1126	1125
LR3G1	1920	2023	2088	2165	2160
LR3G2	1920	1963	1987	2035	1988
LR3G3	1920	2019	2082	2155	2145
LR4G1	3650	3798	3928	4029	4086
LR4G2	3650	3694	3732	3754	3693
LR4G3	3650	3792	3915	4007	4046

4.6. táblázat. Az SH(K) algoritmus által szerzett profit az LR osztály esetében

4.2.4. Az eredmények összefoglalása

Osztály	DNF	H(2)	H(3)	H(4)	H(5)	SH(2)	SH(3)	SH(4)	SH(5)	MAX
S1G1	160	160	160	160	160	160	160	160	160	160
S1G2	160	160	160	160	160	160	160	160	160	160
S1G3	160	160	160	160	160	160	160	160	160	160
F1G1	380	397	385	382	368	408	413	414	412	414
F1G2	380	373	340	311	265	394	393	360	341	394
F1G3	380	395	381	373	349	407	415	408	402	415
F4G1	3220	3369	3314	3258	3214	3359	3423	3431	3459	3459
F4G2	3220	3183	2905	2613	2358	3264	3187	2997	2822	3264
F4G3	3220	3358	3276	3171	3064	3354	3406	3386	3369	3406

4.7. táblázat. Összesített eredmények az S és F osztályok esetében

Ebben az alfejezetben összegyűjtöttem az előzőleg bemutatott algoritmusok eredményeit, így könnyebben összehasonlíthatók. (Annak érdekében, hogy a táblázatok beférjenek, a H(2) és H(3) oszlopokat elhagytam, a maximálisak soha nem ezekből az oszlopokból kerültek ki, emiatt a maximum értékét ezeknek az oszlopoknak az elhagyása nem érinti.) Minden sorban az adott alosztályokra vonatkozó, lefelé kerekített eredmények vannak megadva algoritmusonként. Az utolsó oszlopban az adott sorban található maximum érték van megadva, azaz a legjobb eredmény az adott alosztály esetében.

4.3. Egy új, rugalmas, paraméteres algoritmus: MMask

4.3.1. Az algoritmus bemutatása

Ebben az alfejezetben egy új algoritmuscsalád kerül definiálásra, amely megfelelően flexibilis ahhoz, hogy versenyképes legyen a korábban bemutatott algoritmusokkal.

Osztály	DNF	H(2)	H(3)	H(4)	H(5)	SH(2)	SH(3)	SH(4)	SH(5)	MAX
LR1G1	430	437	423	420	407	478	487	516	515	516
LR1G2	430	415	362	332	298	465	460	481	477	481
LR1G3	430	436	417	407	387	477	485.2	513	512	513
LR2G1	980	953	935	918	911	1047	1074	1132	1131	1132
LR2G2	980	900	802	724	653	1021	1023	1060	1054	1060
LR2G3	980	951	922	890	862	1045	1071	1126	1125	1126
LR3G1	1920	1877	1841	1806	1792	2023	2088	2165	2160	2165
LR3G2	1920	1763	1582	1418	1273	1963	1987	2035	1988	2035
LR3G3	1920	1870	1815	1751	1694	2019	2082	2155	2145	2155
LR4G1	3650	3546	3507	3459	3428	3798	3928	4029	4086	4086
LR4G2	3650	3333	3036	2738	2427	3694	3732	3754	3693	3754
LR4G3	3650	3534	3461	3358	3240	3792	3915	4007	4046	4046

4.8. táblázat. Összesített eredmények az LR osztály esetében

Az algoritmus működése különböző paraméterek beállításától függ. Az algoritmust Modified Mask-nak, vagy röviden MMask algoritmusnak neveztem el. Első verziója egy korábbi cikkben [42] már megjelent. A *modified* kifejezés az algoritmus továbbfejlesztése miatt eszközölt módosítások tényét jelöli.

Az MMask algoritmusnak három paramétere van, amelyek az alábbiak:

- K - az egy időben nyitott ládák maximális számát meghatározó pozitív egész szám ($K > 0$),
- α - K-dimenziós nemnegatív vektor ($0 \leq \alpha_k \leq C$ $\forall k$-ra),
- β - egy pozitív egész szám ($\beta > 0$).

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. Az elfogadó és elutasító intervallumok az alábbiak szerint kerültek meghatározásra:

- 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 ($1 \leq k \leq K$): $(C - \alpha_k; C) \cup (C + \beta; \infty)$

Mint látható, nem csak az a célunk, hogy egy láda fedett legyen, hanem azt szeretnénk, hogy a tárgyak összmérete csak "kicsivel" legyen több, mint a láda mérete.

A tartományok megadásánál az α és β paraméterek a láda kapacitásának (C) értékét módosítva állítják be az intervallumokat. A szakirodalomban a formális definíciók jellemzően $C = 1$ értéket használnak. Az általam vizsgált feladatok esetében azonban a ládák kapacitása 1000 és a tárgyak méretei is arányosak ezzel a kapacitással. A feladatok megoldása során az eredeti értékeket hagytam meg, felesleges lett volna a láda kapacitását és ezzel együtt a tárgyak méretét a $C = 1$ értékhez normálni.

Az elfogadó és elutasító politikával a cél az, hogy megfelelő α_k és β értékek megválasztásával egyrészt elkerüljük azt a helyzetet, hogy egy éppen pakolt tárggyal a láda nem lesz fedett, csak "majdnem". Ekkor a láda töltöttsége az $(1-\alpha_k; 1)$ intervallumba fog esni, tehát ez a pakolás elutasításra kerül. A "majdnem" fedett ládák magukban hordozzák azt a veszélyt, hogy később fedetté válnak, viszont egy nagy méretű tárggyal túlpakolt lesz a láda. Másrészt, azt is szeretnénk elkerülni, hogy a láda túlpakolt legyen, ezért az algoritmus nem engedi a pakolás mértékét $1+\beta$ érték fölé nőni. Ezen stratégiák alkalmazásával azt reméljük, hogy a ládák töltöttsége kiegyensúlyozott lesz, ezáltal pedig a profit is javulhat.

Ha több olyan láda van, amelyik az aktuális tárgy pakolásával elfogadottá válik, hogy ezek közül hogyan válasszunk, ezzel a kérdéssel majd később részletesebben foglalkozom (4.3.3. alfejezet).

Rendszerint az α vektor elemeit különbözőnek választjuk meg, amivel néhány láda töltöttségét alacsony (ha C alatt van) szinten tudjuk tartani, másokét pedig magasan. Ennek azért van jelentősége, hogy a nagyobb méretű tárgyak az alacsony töltöttségű ládákat, míg a kisebb tárgyak a jobban töltött ládákat töltsék meg. Ez egyfajta egyensúlyt biztosít a tárgyak elhelyezésében. Az algoritmus a következők szerint működik.

Algorithm 13: MMask(K, α, β) algoritmus

Input: L lista

1. 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 és az 5. lépésre ugrik.
2. 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 az 5. lépésre ugrik.
3. Ha $k < K$ (ahol k az éppen nyitott ládák száma), 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 az 5. lépésre ugrik az algoritmus.
4. Ha $k = K$ teljesül, akkor az aktuális tárgy a legkisebb töltöttségű ládába kerül. Ha a láda fedett lesz, az algoritmus lezárja.
5. Ha nincs több tárgy, az algoritmus leáll. Egyébként az 1. lépésre ugrik.

Megjegyzem, hogy az 1. lépés esetén a tárgy elfogadó tartományba kerül és a láda fedetté válik, a 2. lépés esetén elfogadó tartományba kerül, de nem válik fedetté. A következő lépések esetén tehát a tárgy biztos, hogy nem elfogadó tartományba kerül. Ha $k < K$, akkor a tárgy új ládába kerül (3. lépés). Ellenkező esetben a 4. lépés során már nem tudjuk növelni a ládák számát, és a tárgyat nem tudjuk elfogadó tartományba pakolni. Mivel jobbat nem tehetünk (a tárgyat muszáj pakolni), a legkisebb töltöttségű ládába tesszük. Lehet, hogy lefedi, lehet, hogy nem.

Az algoritmus megértéséhez tekintsünk egy egyszerű példát, amelynek a beállí-

tásai a következők:

- $K = 4$
- $C = 100$
- $\alpha = [10, 20, 30, 40]$
- $\beta = 30$

A tárgyak méretei a $[10; 40]$ intervallumból kerülnek ki és legyenek a következők: 24, 35, 18, 22, 16, 29, 20, 17, 38, 14, 31, 28, 32. A tárgyak a felsorolás sorrendjében érkeznek.

Az algoritmus indulásakor csak egy láda van nyitva, az alkalmazott paraméterek pedig $\alpha_1 = 10$ és $\beta = 30$. Ezek alapján az elfogadó tartomány $[0; 90] \cup [100; 130]$, az elutasító tartomány pedig $(90; 100) \cup (130; \infty)$ az aktuálisan nyitott, egyetlen ládára nézve. Az MMask az első tárgyat (24) bepakolja a nyitott (és egyetlen) ládába. A következő tárgyat (35) szintén ide pakolja, így a láda töltöttsége $24 + 35 = 59$ lesz. A harmadik tárgy (18) még mindig pakolható ebbe a ládába, hiszen így a töltöttség $59 + 18 = 77$ lesz. Viszont a negyedik tárgy (22) már nem pakolható ide, ugyanis a pakolás után a láda töltöttsége $77 + 22 = 99$ lenne, ami az elutasító tartományba esik.

Emiatt a negyedik tárgynak (22) egy új ládát nyit az algoritmus. Az új láda esetében $\alpha_2 = 20$ és $\beta = 30$, így az elfogadó tartomány $[0; 80] \cup [100; 130]$, az elutasító tartomány pedig $(80; 100) \cup (130; \infty)$. Immáron két nyitott láda van, az egyiknek 77, a másiknak 22 a töltöttsége. Az ötödik tárgy (16) nem pakolható az első ládába, mert $77 + 16 = 93$, ami az első láda elutasító tartományába esik. A második ládába viszont pakolható, mivel $22 + 16 = 38$ a második láda elfogadó tartományába esik. Ebben a pillanatban az első láda töltöttsége 77, a másodiké pedig 38.

A hatodik tárgy (29) elég nagy, így az első ládába pakolható, aminek a töltöttsége így $77 + 29 = 106$ lesz és elszállítható. A láda zárása előtt a nyereségfüggvényt kiszámítja az algoritmus. Mivel két nyitott láda van, így $k = 2$, azaz $G(2)$ profitot kapunk a láda elszállítása után. A megmaradt láda töltöttsége 38 továbbra is.

A következő három tárgy (20, 17, 38) pakolható a megmaradt nyitott ládába, ugyanis $38 + 20 = 58$, $58 + 17 = 75$ és $75 + 38 = 113$. Mind a három tárgy pakolása után a láda töltöttsége az elfogadó tartományban volt. A láda elszállítása előtt az algoritmus kiszámítja a $G(1)$ profitot, majd elszállítja a ládát. Most nincs egyetlen nyitott láda sem.

Még hátra van négy tárgy (14, 31, 28, 32), amelyhez az algoritmus nyit egy ládát. Mind a négy tárgy ebbe a ládába kerül, ugyanis $0 + 14 = 14$, $14 + 31 = 45$, $45 + 28 = 73$ és $73 + 32 = 105$. Azaz minden egyes tárgy pakolása után a töltöttségi szintek elfogadó tartományban voltak. A láda fedetté vált, amiért $G(1)$ profit jár. Így összesen a szerzett profit $2G(1) + G(2)$.

Abban az esetben, ha ugyanerre a példára az MMask helyett a DNF algoritmust alkalmaztam volna, akkor a megszerzett profit csak $2 \times G(1)$ lenne, valamint maradt volna egy nyitott (azaz nem fedett) láda is, aminek a töltöttsége $14 + 31 + 28 = 73$ lenne. Tehát ebben a helyzetben az MMask algoritmus sokkal jobb teljesítményt nyújtott, mint a DNF nyújtott volna. Természetesen az algoritmusok teljesítménye

erősen függ a konkrét feladattól, az α_k $(k = 1, \ldots, K)$ és a β paraméterek megválasztásától.

A vizsgálatok során az derült ki, hogy a K, α_k és β paraméterek megfelelő megválasztása esetén az MMask versenyképes a korábban bemutatott algoritmusokkal.

4.3.2. Az eredmények vizsgálata

Ebben az alfejezetben bemutatom, hogy a stratégiai paraméterek megfelelő beállítása mellett milyen teljesítményre képes az algoritmus. Látható lesz, hogy a megfelelő értékek megválasztásával az MMask majdnem minden feladatosztály esetén képes túlteljesíteni a korábbi algoritmusokat. (Ebben az alfejezetben ezeket az eseteket jelöltük zöld háttérrel.) Az M0-M8 paraméterbeállítások adatait nemsokára megadom. Az algoritmus ugyanazokra a feladatosztályokra lett alkalmazva, mint amelyekkel az előző algoritmusokat vizsgáltam. Ennek köszönhetően az MMask és a korábbi algoritmusok közül a legjobb teljesítményt nyújtó összehasonlíthatóvá vált minden bemenet esetében.

Megjegyzem, hogy a paraméterek hangolása során érdemes lehet figyelembe venni, hogy a tárgyak mérete milyen intervallumban van. Jelenleg ezzel a kérdéssel nem foglalkoztam, későbbi kutatás tárgya lehet. Azt azonban megjegyzem, hogy a 4.4. fejezetben foglalkozom a paraméterek optimalizálásával, és kiderül, hogy az optimalizáló algoritmus valójában implicit módon, "automatikusan" figyelembe veszi a tárgyméreteket az α és β értékek beállításánál.

Osztály	Max	M0	M1	M2	M3	M4	M5	M6	M7	M8
S1G1	160	160	136	134	158	158	158	136	136	136
S1G2	160	160	102	88	147	142	142	100	100	100
S1G3	160	160	130	127	158	157	157	130	130	130
F1G1	414	380	424	424	417	417	409	407	408	415
F1G2	394	380	374	373	395	395	401	389	391	379
F1G3	415	380	419	418	416	416	408	406	407	412
F4G1	3459	3220	3585	3603	3517	3569	3399	3503	3487	3496
F4G2	3264	3220	3182	3090	3319	3297	3307	3268	3280	3287
F4G3	3406	3220	3539	3531	3506	3547	3394	3482	3471	3480

4.9. táblázat. Összesített eredmények az S és F osztályok esetében

A korábbi legjobb eredményt a Max oszlop jelöli feladatosztályonként. A paraméterek beállítása itt még kézzel történt, próbálkozás útján. A 4.9. táblázat az S és F osztályokon végzett futtatások eredményét szemlélteti. A zölddel jelölt cellák esetében az MMask felülmúlta a korábbi legjobb eredményt, a kék cellák esetében pedig a korábbi legjobbal egyenlő az MMask eredménye. Az S1 osztály esetében a korábbi legjobb eredményt a DNF állította elő, és minden korábbi algoritmus is ugyanazt az eredményt adta, mint a DNF. Ez amiatt volt, mert a tárgyak méretei egy nagyon szűk intervallumból kerültek ki. Ennek következménye, hogy jobb eredményt itt nem lehetett elérni, ha az MMask legalább két ládát nyitott. Viszont, ha $K = 1$ volt a beállítás (M0 paraméter beállítás), akkor ugyanazt az eredményt érte el, mint a korábbi algoritmusok.

Látható, hogy a paraméter-beállítások (M0-M8) nagyon változatosan képesek befolyásolni az MMask teljesítményét. Van ahol az adott beállítás a vizsgált feladatosztályok több mint a felére jobb eredményt ért el, de volt olyan, ahol kevesebb, mint a felénél ért el jobb eredményt. Ez teljesen természetes, hiszen a feladatosztályok különböznek egymástól. Pontosan emiatt nem cél egy "univerzális" beállítás megtalálása, amely minden feladatosztály esetében jobb eredményeket ad. A cél az volt, hogy minden feladatosztályra találjak egy olyan beállítást, amely jobb eredményt ad, mint a korábbi algoritmusok. A táblázatban foglalt osztályokra ez sikerült is.

Az alkalmazott beállítások (M0-M8) az alábbiak voltak.

Beállítás	**K**	$\alpha_k, k = 1, \ldots, K$	β
M0	1	[200]	200
M1	4	[100; 200; 300; 400]	200
M2	5	[100; 200; 300; 400; 500]	200
M3	2	[100; 100]	200
M4	3	[100; 100; 100]	200
M5	4	[100; 100; 100; 100]	500
M6	4	[225; 225; 230; 230]	560
M7	4	[200; 200; 200; 200]	600
M8	4	[200; 200; 300; 400]	350

4.10. táblázat. MMask kézi paraméter-beállításai

Az LR osztály esetében is elvégeztem a futtatásokat, ugyanazokat a paraméter-beállításokat alkalmazva. Mivel az M0 beállítás a Schwerin osztályra lett kitalálva, ami azt jelenti, hogy csak egy láda lehet nyitva egyszerre, ezért nem alkalmazható a többi osztályra. Így nem került be az LR osztályon végzett futások eredményeit tartalmazó táblázatba (4.11). Továbbá a futások alapján az M3 és M4 beállítások sem voltak versenyképesek egyik LR alosztály esetében sem, így azok is kimaradtak az eredmények közül.

Az eredményeket tartalmazó táblázatban látható, hogy az LR2G2, LR2G3 és LR3G2 esetében nem volt jobb eredmény a korábbi legjobbnál. Vegyük figyelembe, hogy a témához kapcsolódó korábbi publikációban [42] az SH algoritmus teljesítménye nem volt vizsgálva a $K = 4$ és $K = 5$ esetekre, most viszont ez megtörtént. Kiderült, hogy az SH(4) algoritmus teljesítménye néhány esetben sokkal jobb, mint akkor, ha $K \neq 4$ (4.5. és 4.6. táblázatok). Mivel most SH(4) teljesítménye is ismert, így az MMask-nak egy sokkal jobb eredményt kell felülmúlnia. Habár, a 4.11. táblázatban a másik 9 feladat esetében sikerült jobb eredményeket elérni, ami az alosztályok 75%-át jelenti.

4.3.3. Továbbfejlesztési lehetőségek

Néhány, az MMask algoritmus továbbfejleszthetőségére vonatkozó lehetőséget szeretnék itt megadni.

- A fent bemutatott paraméter-beállítások manuálisan kerültek meghatározásra. Előfordulhat, hogy más beállításokkal sokkal jobb eredmények is elérhetők.

Osztály	Max	M1	M2	M5	M6	M7	M8
LR1G1	516	518	538	507	517	508	524
LR1G2	481	414	431	485	498	491	470
LR1G3	513	503	521	505	516	506	518
LR2G1	1132	1122	1146	1119	1122	1103	1131
LR2G2	1060	876	846	1022	1059	1046	962
LR2G3	1126	1086	1091	1108	1116	1098	1109
LR3G1	2165	2198	2232	2124	2127	2089	2189
LR3G2	2035	1730	1625	1980	2015	1991	1911
LR3G3	2155	2130	2119	2109	2117	2080	2156
LR4G1	4086	4194	4242	4032	4014	3968	4139
LR4G2	3754	3338	3186	3784	3786	3777	3681
LR4G3	4046	4073	4057	4008	3994	3953	4086

4.11. táblázat. Összesített eredmények az LR osztály esetében

- Kiderült, hogy az LR osztály esetében az SH(4) algoritmus több esetben is nagyon hatékony a 4.6. táblázat alapján. 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.

4.3.4. Összefoglalás

Az eredmények alapján az látható, hogy megfelelő paraméterek megválasztása esetén az MMask igen hatékonyan működik, és a legtöbb esetben jobb eredményt ad, mint a korábban vizsgált algoritmusok. Lehetséges, hogy azoknál az alosztályoknál, ahol nem sikerült javítani a korábbi eredményeken, ott is van egy olyan paraméterbeállítás, amivel sikerülhet jobb eredményt produkálni. A kérdés csupán az, hogy milyen stratégia mentén kell a paramétereket megválasztani, hogy jobb eredményeket érjen el az algoritmus a korábbiaknál. A következő alfejezetben ezzel a kérdéssel foglalkozom.

4.4. Paraméter optimalizálás

A vizsgálatok során kiderült, hogy amennyiben az MMask algoritmus paraméterei megfelelően vannak megválasztva, akkor az MMask igen jó eredményeket tud elérni. Ezért, a legtöbb problémaosztály választásakor (amelynek ismertek a paraméterei: ládaméret, a tárgyak méretének eloszlása, a tárgyak száma és a G nyereségfüggvény), az MMask képes messze túlteljesíteni a korábban tárgyalt algoritmusokat (DNF, H(K) és SH(K)). A nehézség viszont pont az MMask paramétereinek megválasztása. Az egyik megoldás a manuális választás, amikor egyszerűen próbálgatjuk a paramétereket és megvizsgáljuk, hogy az egyes beállításokkal milyen eredményt kaptunk. Ez nyilvánvalóan nehézkes, lassú és nem biztos, hogy megtaláljuk azt a paraméter-beállítást az adott feladatosztályhoz, ami legalább olyan eredményt garantál, mint a korábbi algoritmusok eredményei.

Ebben a fejezetben egy természetesen adódó eljárást mutatok be, amelynek feladata a paraméterek automatizált keresése. Az ilyen optimalizálást hiperparaméter optimalizálásnak nevezik. Az implementált eljárást EoA-nak neveztem el (ugyanilyen néven szerepel a már korábban, a témához kapcsolódóan megjelent publikációban [42] is).

Ezzel a módszerrel szeretném az MMask algoritmus paramétereit optimalizálni. Ez azt jelenti a gyakorlatban, hogy az optimalizáló eljárás a paraméter-beállításokat vizsgálja, megváltoztatja és a kapott futási eredmények alapján képes különbséget tenni az egyes beállítások jósága között. A hiperparaméterek optimalizálása legtöbbször offline probléma, hiszen az optimalizálás előre megadott (azaz ismert) adathalmazokat alkalmazva történik.

A megoldás során szomszédsági struktúrát definiálunk a paraméter-beállítások között. Létezik hasonló algoritmus az irodalomban, például [101], amely a legjobb paraméter-beállítást igyekszik megtalálni. A megoldásom kulcsa az, hogy létrehozok egy flexibilis algoritmuscsaládot, ahol az algoritmusok között definiálásra kerül egy lehetséges szomszédsági struktúra. A megoldások előállítására a lokális keresést alkalmazom, ahol a meta-algoritmus néhány szabály betartása mellett lépked az egyik algoritmusról a másikra azért, hogy a lehető legjobb algoritmust megtalálja. A lokális keresés megállási feltétele jelenleg 1000 iteráció. Vizsgáltam magasabb iterációszámra is, azonban a tapasztalatok alapján 1000 iteráció felett már nem javult a megoldás.

A szomszédsági struktúra a különböző MMask algoritmusok között természetes módon definiálható. Egy konkrét $MMask(K, \alpha, \beta)$ algoritmus 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. (Egy másik lehetőség lehetne, hogy egyszerre több paramétert változtatunk. Ennek várhatóan az lenne a hatása, hogy a keresés felgyorsul, viszont nagyobb eséllyel maradnának felderítetlenül a keresési tér egyes részei.)

Az MMask algoritmusra a lokális keresést alkalmaztam a korábban bemutatott feladatosztályokat felhasználva. A 4.12. táblázatban az F osztályra vonatkozóan látható a korábbi algoritmusok (DNF, H(K), SH(K)) által elért eredmények közül a

legjobb (MAX), az MMask által kiszámított legjobb eredmény (BMM, mint Best of MMask rövidítése). A fennmaradó oszlopokban (O1-O5) pedig az MMask új verziói által elért legjobb eredmények láthatók a paraméterek optimalizálása után, amely lokális kereséssel valósult meg.

A 4.12. táblázatból láthatóan kimaradt az S osztály. Ennek oka, hogy korábban a DNF már megtalálta az optimális megoldást, amit nyilvánvalóan nem lehet már javítani, így felesleges volna erre is futtatni az MMask algoritmust, kiegészítve a paraméter optimalizálással. Azonban az F osztályra a fentebbi állítás már nem igaz. Így elsőként erre a feladatosztályra próbáltam először manuálisan megfelelő paraméter-beállítást találni. Látható, hogy az MMask alapbeállításával (BMM oszlop) már sikerült felülmúlni a korábbi algoritmusok legjobb teljesítményét (MAX oszlop) minden esetben. Az MMask kezdeti alapbeállítása a 4.10. táblázatban olvasható.

Osztály	**MAX**	**BMM**	**O1**	**O2**	**O3**	**O4**	**O5**
F1G1	414	424	436				
F1G2	394	401					
F1G3	415	419		427			
F4G1	3459	3603			3625		
F4G2	3287	3319				3331	
F4G3	3406	3547					3553

4.12. táblázat. Az MMask optimalizált paramétereinek eredménye az F osztály esetében

Az O1-O5 paraméter-beállítások az egyes problémákra külön-külön lettek meghatározva. Például az O1 az F1G1-re, az O2 az F1G3-ra és így tovább. Látható, hogy az F1G2 probléma kivételével mindegyikre sikerült paraméter optimalizálással olyan beállításokat találni, amely a korábbi legjobb MMask eredményt (BMM oszlop) is javították. A 4.13. táblázatban megadom az O1-O5 beállítások lokális keresés által kapott pontos értékeit.

Beállítás	**K**	α_k	β
O1	4	[113,222,286,388]	175
O2	4	[94,221,316,410]	267
O3	5	[64,197,286,395,495]	199
O4	2	[100,101]	198
O5	3	[89,103,104]	218

4.13. táblázat. Az MMask paramétereinek lokális kereséssel való beállítása (O1-O5)

A következőkben az LR osztályra szeretném bemutatni a hasonlóan elvégzett vizsgálatot. Itt a paraméter-beállításokat O6-O14 jelöléssel láttam el, az eredmények a 4.14. táblázatban láthatók. Az egyéb oszlopok jelölése ugyanaz, mint a 4.12. tábla esetében. Látható, hogy a BMM oszlopban olvasható érték a MAX oszlop értékéhez viszonyítva az LR2G3 és LR3G2 esetében nem javult (ezt *-gal jelöltem). Továbbá a BMM oszlop eredményét nem sikerült javítani semmilyen, a lokális kereséssel optimalizált paraméter-beállítással sem az LR1G2 esetében (ezt **

karakterekkel jelöltem). Minden más problémánál a BMM érték javult az optimalizált paraméterválasztással. A javulás 9 sort érintett a 12-ből, azaz 75%-ban sikerült javítani az eredményeken.

Osztály	MAX	BMM	O6	O7	O8	O9	O10	O11	O12	O13	O14
LR1G1	516	538	539								
LR1G2	481	498**									
LR1G3	513	521		527							
LR2G1	1132	1146			1158						
LR2G2	1060	1059				1060					
LR2G3	1126	1116*									
LR3G1	2165	2232					2245				
LR3G2	2035	2015*									
LR3G3	2155	2156						2166			
LR4G1	4086	4242							4257		
LR4G2	3754	3786								3788	
LR4G3	4046	4086									4087

4.14. táblázat. Az MMask optimalizált paramétereinek eredménye az LR osztály esetében

A 4.15. táblázatban megadom az O6-O14 beállítások lokális keresés által kapott paraméterek pontos értékeit.

Beállítás	K	α_k	β
O6	5	[106, 205, 292, 413, 490]	238
O7	5	[130, 204, 304, 393, 473]	238
O8	5	[128, 194, 308, 414, 481]	241
O9	4	[218, 197, 211, 185]	558
O10	5	[100, 205, 299, 402, 498]	214
O11	4	[220, 224, 320, 409]	373
O12	5	[102, 199, 302, 399, 498]	214
O13	4	[225, 220, 206, 214]	563
O14	4	[201, 199, 301, 400]	353

4.15. táblázat. Az MMask paramétereinek lokális kereséssel való beállítása (O6-O14)

3. Megjegyzés. *A fent bemutatott eredmények esetében az O1-O14 beállítások a lokális kereséssel lettek előállítva úgy, hogy a kiindulási beállítás a manuálisan megtalált legjobb volt minden esetben. Továbbá két alapbeállítást is alkalmaztam, amelyeket BASIC_1 és BASIC_2-nek neveztem el, amelyek a következők voltak:*

- ***BASIC_1:*** *K ne legyen se túl kicsi, se túl nagy, azaz pl.: $K = 4$, $\alpha_i = 200$ $\forall i$-re, ahol $1 \leq i \leq K$, $\beta = 200$*
- ***BASIC_2:*** *$K = 4$, $\alpha = [100, 200, 300, 400]$, $\beta = 200$*

Az ok, amiért két kitüntetett alapbeállítást is alkalmaztam az, hogy a korábbi manuális beállítások keresése során ez a két beállítás jó eredményeket hozott. Ellenben az kiderült, hogy ha a lokális keresést egy korábbi manuális beállításból indítom vagy az egyik alapbeállításból, az szignifikánsan nem befolyásolja az optimalizáció végeredményét.

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 témához kapcsolódó korábbi publikációban [42] a paraméterek beállítására a szimulált hűtést alkalmazták a szerzők, ami kicsit jobb eredményeket mutatott, mint a lokális keresés.

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 paraméteroptimalizáló fúziója hatékonynak bizonyult. A paraméter beállításokra vonatkozó további lehetőségekkel a 4.6. alfejezetben foglalkozom.

4.5. Részletes vizsgálat

Ebben az alfejezetben részletesebb vizsgálatokat szeretnék bemutatni az SH(K) és MMask algoritmusok összehasonlításában. Itt az MMask algoritmusnak már a lokális kereséssel való kiegészített változatát tekintjük. A vizsgálatok során a korábban már bemutatott feladatosztályokat alkalmaztam. Tekintsük az F1G1 osztályt. Válasszuk ki azt a korábbi algoritmust (DNF, H(K) és SH(K) közül), amely a legjobb eredményt adta ezen az osztályon. A 4.7. táblázat alapján ez az SH(K) algoritmus volt, $K = 4$ beállítással, azaz az SH(4). Az SH(4) eredménye 414 volt, a többi algoritmusé rosszabb. Ezért a 4.16. táblázatban az SH(4) algoritmus eredményeit hasonlítom össze az MMask eredményeivel az F1 osztályban található összes feladatra nézve a G1 nyereségfüggvényt kiértékelve. Az MMask esetében az F1G1 osztályra vonatkozóan a lokális kereséssel megtalált legjobb beállítás az O1 volt a 4.12. táblázat alapján, így ezt a beállítást alkalmaztam itt is. Az O1 beállítással futtatott MMask eredménye 436 volt.

Fontos hangsúlyozni, hogy ez az eredmény az F1 osztály legelső feladatára volt csak érvényes, így az O1 beállítás csak erre a feladatra lett kiszámítva. A vizsgálat tárgya az, hogy az F1G1 osztályban az első feladatra optimalizált paraméter-beállítás a többi feladat esetében milyen eredményt hoz az SH(4) algoritmussal összehasonlítva. Az eredmények itt is lefelé vannak kerekítve.

A 4.16. táblázatban foglalt eredmények ígéretesek, ugyanis azt mutatják, hogy egy feladat kivételével (Falkenauer_u120_19_G1) az MMask az optimalizált paraméter-beállításokkal a 20 feladatból 19 feladat esetén jobb eredményeket ért el, mint az SH(4) úgy, hogy az O1 paraméter-beállítások csak a legelső feladatra (Falkenauer_u120_00_G1) lettek optimalizálva. A Falkenauer_u120_19_G1 feladat esetében az SH(4) kerekítés nélküli, eredeti értéke 423,1 volt, míg az MMask értéke

Feladat példány	SH(4)	O1
Falkenauer_u120_00_G1	414	436
Falkenauer_u120_01_G1	394	434
Falkenauer_u120_02_G1	385	403
Falkenauer_u120_03_G1	404	432
Falkenauer_u120_04_G1	414	444
Falkenauer_u120_05_G1	403	425
Falkenauer_u120_06_G1	413	432
Falkenauer_u120_07_G1	405	435
Falkenauer_u120_08_G1	435	453
Falkenauer_u120_09_G1	384	406
Falkenauer_u120_10_G1	444	474
Falkenauer_u120_11_G1	402	442
Falkenauer_u120_12_G1	413	434
Falkenauer_u120_13_G1	413	423
Falkenauer_u120_14_G1	423	445
Falkenauer_u120_15_G1	404	431
Falkenauer_u120_16_G1	443	465
Falkenauer_u120_17_G1	444	466
Falkenauer_u120_18_G1	405	435
Falkenauer_u120_19_G1	423	422

4.16. táblázat. SH(4) és az MMask összehasonlítása O1 beállítás mellett az F1G1 osztályon

422,9. Látható, hogy a különbség csupán 0,2. Az eredmények összességében azt mutatják, hogy a lokális kereséssel valóban tudunk olyan beállítást találni, ami jobb eredményeket hoz, mint a korábbi algoritmusok. Megjegyzendő, hogy a feladatosztályban szereplő feladatok esetén ismeretes, hogy a tárgyak milyen intervallumból kerülnek ki és az, hogy a generálásuk egyenletes eloszlás mellett történt. Mivel minden feladatban szereplő tárgyméretek azonos módszerrel kerültek generálásra, így a feladatok hasonlóságot mutatnak egymással.

Hasonló vizsgálatot végeztem az F1G2 és F1G3 osztályok esetében is. Itt a feladatosztály ugyanaz, mint az előző esetben, viszont a nyereségfüggvények nem, azaz a profit kiszámítására a G2 és G3 nyereségfüggvényeket alkalmaztam. Az eredmény hasonló volt, azaz az MMask minden esetben jobb eredményt ért el, mint az SH(4). Az alábbiakban egy összefoglaló táblázatot (4.17) ismertetek, amelyben az F1 és F4 osztályok eredményét mutatom be mindhárom nyereségfüggvény esetében. A táblázatban a feladatosztályok esetében a legjobb, legrosszabb és az átlagos eredmények kerültek megadásra. Minden feladatosztály esetében az MMask alkalmazása előtti algoritmusok közül (DNF, H(K), SH(K)) az SH algoritmus volt a legjobb, valamilyen K beállítása mellett. Az MMask esetében pedig a legjobbnak bizonyult paraméter-beállítást adtam meg.

4. Megállapítás. *Az MMask minden esetben jobb eredményeket ért el, mint az SH algoritmus. Ez alapján kijelenthető, hogy a lokális keresés alapú optimalizáló eljárás hatékony.*

Osztály		SH				MMask		
	K	legrosszabb	átlag	legjobb	beállítás	legrosszabb	átlag	legjobb
F1G1	4	384	413	444	O1	403	437	474
F1G2	2	367	394	431	BASIC_1	386	406	440
F1G3	3	383	411	444	BASIC_1	404	433	464
F4G1	5	3372	3460	3555	O3	3552	3623	3715
F4G2	2	3215	3301	3403	O4	3272	3334	3479
F4G3	3	3381	3443	3528	BASIC_1	3528	3587	3678

4.17. táblázat. Az SH és az MMask összehasonlítása az F1 és F4 osztályokon

5. Megállapítás. *Az MMask még jobb eredményeket érne el, ha az adott feladatosztály esetében nem csak az első feladatra, hanem mindegyikre, külön-külön elvégeznénk a paraméterek beállítását. Viszont ebben az esetben az algoritmust nem hívhatnánk online algoritmusnak. Emiatt az aranyközépút az lehet, hogy egy feladatosztály esetében nem egy feladatra és nem is az összesre, hanem csak néhányra alkalmazzuk a paraméterek optimalizálását. Például, ha van 20 feladat egy osztályban, akkor 5 feladaton elvégezzük a paraméter-beállítást, majd a maradék 15 feladaton alkalmazzuk.*

Az F osztály vizsgálata után következzen az LR osztály vizsgálata is, amelynek eredményei a 4.18. táblázatban láthatók. Zölddel van jelölve az a néhány eset, ahol MMask nem javított.

Osztály		SH				MMask		
	K	legrosszabb	átlag	legjobb	beállítás	legrosszabb	átlag	legjobb
LR1G1	4	465	510	548	O6	460	530	565
LR1G2	4	428	481	530	M6	430	480	525
LR1G3	4	462	507	546	O7	447	508	545
LR2G1	4	993	1061	1132	O8	1021	1110	1158
LR2G2	4	930	999	1061	O9	945	999	1061
LR2G3	4	988	1056	1126	M6	1009	1065	1120
LR3G1	4	1997	2099	2186	O10	2106	2202	2295
LR3G2	4	1868	1964	2061	M6	1884	1977	2095
LR3G3	4	1988	2088	2176	O11	2050	2142	2232
LR4G1	5	3934	4069	4196	O12	4095	4259	4386
LR4G2	4	3569	3743	3872	O13	3684	3795	3912
LR4G3	5	3525	3703	3850	O14	3989	4120	4262

4.18. táblázat. Az SH és az MMask összehasonlítása az LR osztályon

Ebben az esetben az MMask már nem volt jobb minden esetben. A táblázatban zölddel jelöltem azokat az eredményeket, ahol az SH algoritmus jobban teljesített. A 4.17. és a 4.18. táblázatokban foglalt eredményekhez két fontos konklúzió tartozik.

6. Megállapítás. *Csak néhány esetben volt jobb az SH, mint az MMask. Ezekben az esetekben ráadásul az MMask nem sokkal maradt el az SH eredményeitől. Ennek oka lehet a nem megfelelő paraméter-beállítás, vagy az SH algoritmus azon tulajdonsága, hogy az azonos méretű tárgyakat igyekszik egy ládába pakolni. Ráadásul az LR*

osztályban a tárgyak méreteit tekintve nagy a diverzitás. Valószínűleg a paraméterek további javításával az SH algoritmus ezen előnye eltűnne.

7. Megállapítás. *Az LR1 osztály esetében a 9 értékből 5 érték esetében teljesített rosszabbul az MMask, mint az SH. Az LR2 esetében csak egy ilyen érték látható, az LR3 és LR4 esetében pedig az MMask volt a jobb minden tekintetben. Ez azt jelenti, hogy ha az adott feladatnál a tárgyak mérete minél nagyobb, az annál jobb az MMask algoritmus teljesítményére nézve.*

4.6. További lehetőségek

Ebben az alfejezetben néhány további lehetőséget említek meg, amely a téma jövőbeni folytatása is lehet. Elsőként a MMask algoritmusra vonatkozó továbbfejlesztési lehetőségeket adom meg:

- Az alkalmazott paraméter optimalizálási eljárás, a lokális keresés egy könnyen implementálható megoldás. Helyette alkalmazható lenne más eljárás is, például a szimulált hűtés vagy tabukeresés. Az általam alkalmazott lokális keresés megállási feltétele jelenleg 1000 iteráció, mert azt tapasztaltam, hogy ennyi elegendő a lokális maximum eléréséhez.

- A szomszédságon alapuló keresés kiváltására más optimalizáló algoritmus tesztelése is érdekes lehet, például genetikus algoritmus alkalmazása. Genetikus Algoritmus (GA) esetén adott egy populáció, ebben a feladat megengedett megoldásai az egyedek. Valamilyen szelekciós szabályt alkalmazunk bizonyos egyedek kiválasztására, és a kiválasztott egyedeken valamilyen műveleteket (operációkat) alkalmazunk, ezek lényegében a mutáció (*mutation*) és a keresztezés (*crossover*). Ezek után feltöltünk az új egyedekkel egy új populációt, és valamely populáció után megállunk. A vizsgált feladatunk esetén a mutáció operátor minden gond nélkül alkalmazható. Ez ugyanis a kiválasztott megengedett megoldás (egyed) kis mértékű megváltoztatását jelenti, épp úgy, mint az általunk alkalmazott lokális keresés esetén. Fejlett genetikus algoritmusoknál egyébként gyakran előfordul, hogy mutáció helyett lokális keresést alkalmaznak. A keresztezés műveletét vizsgáljuk meg részletesebben, hogy a feladatunkra (az MMask algoritmus paraméterei optimalizálására) hogyan lehet alkalmazni. Csak akkor alkalmazunk keresztezést, ha a kiválasztott mindkét egyed esetén a megengedett ládák számát meghatározó K paraméter ugyanaz (legyen például K=4). Legyenek tehát a kiválasztott egyedek a következők:

  - $(K = 4, \alpha_{11}, \alpha_{12}, \alpha_{13}, \alpha_{14}, \beta_{11})$
  - $(K = 4, \alpha_{21}, \alpha_{22}, \alpha_{23}, \alpha_{24}, \beta_{21})$

  A gyerek egyed (offspring) esetén is természetesen $K = 4$ lesz. Az alfa és béta értékek között pedig alkalmazhatjuk azt a szabályt, hogy egyesével választunk az első illetve a második egyedből, a szokásos módon: Egy véletlen számot generálunk 0 és 1 között. Ha ez 0.5-nél kisebb, akkor az első vektorból

választjuk a megfelelő értéket, egyébként pedig a másodikból. Az alfa értékek választása történhet másképpen is, választunk egy véletlen számot 1 és 4 között (és kerekítjük). Legyen például 2. Akkor az első kettő alfát az első vektorból választjuk, a maradék kettőt pedig a másik vektorból. Összefoglalva, úgy gondolom hogy a paraméter optimalizálás lehetséges GA alkalmazásával is. Továbbá a lokális kereső helyettesíthető lenne tabukereséssel vagy szimulált hűtéssel. Mindezekre később, további kutatás során kerülhet sor.

- A vizsgálatok során kiderült, hogy az SH(K) algoritmus $K = 4$ esetében igen hatékony volt számos feladatra az LR osztályból (4.6. táblázat). Ez alapján érdekes lehet az SH(4) és az MMask algoritmusok valamilyen fúziója. Egy egyszerű lehetőség lenne az, hogy mindkét algoritmust futtatjuk és a jobb eredményt adót választjuk.

- Az MMask algoritmus a 2. lépésben az aktuális tárgyat véletlenszerűen pakolja valamelyik megfelelő ládába. Lehetséges, hogy ha ebben a lépésben a ládát nem véletlenszerűen választja az algoritmus, hanem valamilyen szabályrendszer alapján, akkor azzal növelhető lesz az MMask teljesítménye. Két módosítást próbáltam ki ezzel kapcsolatban:

  1. Az aktuális tárgyat a megfelelő ládák közül a legnagyobb töltöttségűbe pakolta az algoritmus. Ennek a logikája az lenne, hogy ezáltal a megengedett ládatartományon belül változatosabb lesz a ládák szintje (növekszik a maximális és minimális ládaszint közötti különbség). Ettől a módosítástól azt vártam, hogy az algoritmus rugalmasabb lesz, jobban tudja kezelni a következő tárgy pakolását: ha kicsi a mérete, olyan ládába tesszük, amelyik eléggé meg van már töltve, ha pedig nagy a mérete, olyan ládába tesszük, amelynek kisebb a szintje. Tehát, ha változatosabb a ládák szintje, attól azt reméltem, hogy az algoritmus hatékonysága javul. Viszont a tapasztalat azt mutatta, hogy ez a módosítás semmilyen szignifikáns javulást nem mutatott. Néhány bemenetre kicsit jobb eredményt ért el, más bemenetek esetében viszont rontott.
  2. Legyenek $B_1, B_2, \ldots, B_t$ azok a ládák, amelyekbe pakolható tárgy a 2. lépésben. Ha $t = 1$, akkor pontosan egy ilyen megfelelő láda van és ebbe kerül az aktuális tárgy. Ha $t > 1$, akkor az algoritmus kiszámolja a $B_1, B_2, \ldots, B_t$ ládák mindegyikére a bennük lévő tárgyak átlagméretét, amelyek rendre legyenek $x_1, x_2, \ldots, x_t$. Az aktuális tárgyat abba a ládába pakolja az algoritmus, amely ládának az átlagtöltöttsége a "legközelebb van" a tárgy x méretéhez, azaz amely láda esetében az $(x-x_i)^2$ a legkisebb $1 \leq i \leq t$ esetén. Sajnos ez a módosítás sem javított szignifikánsan az eredményeken. Néhány feladat esetében kicsit jobb, mások esetében kicsit rosszabb eredményt számolt.

A bemutatott algoritmusok és eredmények a vizsgálat első szakaszát jelentették. Természetesen lehetnek további opciók is a továbbfejlesztésre és a további vizsgálatokra vonatkozóan. Ezenkívül számos kérdés felmerül még, amely ugyancsak a vizsgálatok kiterjeszthetőségére vonatkozik.

- Az eredeti feladatosztályokban szereplő problémák tárgysorrendjét megváltoztattam, ami azt jelenti, hogy az eredeti csökkenő sorrendet véletlenszerű sorrendre alakítottam át. Mi történik, ha más sorrendet választok? A rendezetlen sorrend maradna, viszont nem véletlenszerűen, hanem más szabály szerint alakítanám ki azt. Vajon ugyanazok az eredmények születnének, mint most?
- Mi történne, ha az eredeti csökkenő sorrendet növekvő sorrendé alakítanám?

A korábban bemutatott algoritmusokon kívül egy ötödik verzió is implementálásra került, amelyet DN-nel jelöltem, ugyanis ez az algoritmus a DNF algoritmus párjának is tekinthető. Megadott K érték esetén az algoritmus minden esetben k ládát tart nyitva, és $k = K$. Az eredeti DNF algoritmus is így működött, viszont ott a K értéke minden esetben 1 volt, azaz egy láda lehetett csak nyitva egyszerre. A DN algoritmus esetében a cél pedig az, hogy $K > 1$ esetekre vizsgáljuk meg az eredményeket. Megjegyzem, hogy mivel mindig több láda van nyitva, ez az algoritmus változat biztosan nem lehet hatékony abban az esetben, ha a haszonfüggvény gyorsan csökkenő, de azt nem lehetett tudni előre, hogy talán más esetben hatékony lehet-e. Az algoritmus minden tárgyat a nyitott ládák közül abba pakol, amelynek a legkisebb a töltöttsége.

Az algoritmus tesztelése megtörtént $K = 2, 3, 4, 5$ értékekre. Az eredményeket összehasonlítottam a korábbi algoritmusok (DNF, H(K), SH(K)) eredményei közül a legjobbal. Az eredmények a 4.19. és a 4.20. táblázatokban láthatók. A K értékek alatt a DN algoritmus futási eredményei láthatók, a MAX oszlopban pedig az előző algoritmusok eredményei közül a legjobb, mellette pedig a DN algoritmus legjobb eredménye.

Osztály	**K**				**MAX**	
	2	**3**	**4**	**5**	**előző**	**DN**
S1G1	148	137	135	124	160	148
S1G2	135	112	98	78	160	135
S1G3	147	134	129	114	160	147
F1G1	366	352	349	336	414	366
F1G2	333	288	252	210	394	333
F1G3	364	345	333	308	415	364
F4G1	3187	3145	3103	3071	3459	3187
F4G2	2898	2568	2240	1920	3264	2898
F4G3	3171	3081	2960	2816	3406	3171

4.19. táblázat. A DN algoritmus eredményei az S és F osztályokra

A táblázatokban foglalt eredmények alapján látható, hogy a DN algoritmus a DNF, H(K) és SH(K) legjobb eredményeit sem tudta javítani, ebből pedig az következik, hogy az MMask eredményeitől is messze elmarad. A gyenge teljesítmény oka az lehet, hogy az algoritmus feleslegesen tart nyitva sok ládát (pl. $K = 4$ esetén mindig négyet) és emiatt a nyereségfüggvény értéke minden fedett láda után ugyanaz. Azaz, ha pl. $K = 4$, akkor az algoritmus mindig négy ládát tart nyitva és egy fedett ládáért mindig $G(4)$ profitot kap. Ebből könnyen belátható, hogy a profit értéke konstanssá válik, azaz nem tud tovább növekedni (bár csökkenni sem).

Osztály	K				MAX	
	2	**3**	**4**	**5**	**előző**	**DN**
LR1G1	425	411	397	384	516	425
LR1G2	387	336	287	240	481	387
LR1G3	423	403	379	352	513	423
LR2G1	970	950	931	912	1132	970
LR2G2	882	776	672	570	1060	882
LR2G3	965	931	888	835	1126	965
LR3G1	1900	1871	1843	1814	2165	1900
LR3G2	1728	1528	1330	1134	2035	1728
LR3G3	1891	1833	1757	1663	2155	1891
LR4G1	3613	3567	3521	3475	4086	3613
LR4G2	3285	2912	2541	2172	3754	3285
LR4G3	3595	3494	3357	3185	4046	3595

4.20. táblázat. A DN algoritmus eredményei az LR osztályra

5. fejezet

Összefoglalás

Az alábbiakban egy összefoglalást kívánok adni az elvégzett munkáról és az új tudományos hozzájárulásokról. A dolgozatban alapvetően három területtel foglalkoztam, amelyek az ütemezés, a ládapakolás és a ládafedés szállítással voltak. Mindhárom terület NP-nehéz.

Az ütemezés területén a független gépek ütemezése megelőzési relációkkal típusú feladatokat igyekeztem megoldani megerősítéses tanulás támogatásával. A megoldás célja az erőforrások munkákhoz való rendelése úgy, hogy figyelembe vesszük a megelőzési relációkat és igyekszünk a teljes átfutási időt (makespan) minimalizálni. A megelőzési relációk leírása irányított és egyszerű gráf segítségével történt, ahol a gráf diszjunkt utak és izolált pontok uniója. A megelőzési reláció két tevékenység között azt adja meg, hogy a tevékenységek végrehajtása milyen sorrendben történhet meg. A megelőzési relációk miatt a tevékenységek sorrendje fontos, befolyásolja a teljes átfutási időt. Az általam kifejlesztett megoldás két fő komponensre bontható fel: a tevékenységek sorrendjét meghatározó eljárás és a mohó módon ütemező algoritmus. A sorrend meghatározását a megerősítéses tanulás területén ismert Q-tanulással valósítottam meg. A kidolgozott algoritmus neve **Q-Learning Motivated Algorithm (QLM)**. A kidolgozott algoritmus tesztelésére négy feladatosztályt hoztam létre (*Class* #1, *Class* #2, *Class* #3 és *Class* #4), amelyek közül a *Class* #1 és *Class* #2 könnyűnek tekinthetők, a *Class* #3 közepesen nehéz, a *Class* #4 pedig extrém nehéz. Az algoritmus teljesítményét az $LB1$, $LB2$ alsó korlátokkal mértem, továbbá megadtam ezek maximumát az $LB = \max\{LB1, LB2\}$ segítségével. A QLM minden esetben megtalálta az optimális megoldást, amikor a CPLEX is. Bizonyos esetekben a CPLEX-nél jobb vagy lényegesen jobb megoldást adott a QLM, a többi esetben pedig az optimálishoz közeli megoldás született, azaz közel volt az LB értékéhez. Ezen a területen az alábbi tudományos hozzájárulások valósultak meg:

1. Az $R_m|prec|C_{max}$ feladat megoldására létrehoztam egy megerősítéses tanulás által támogatott algoritmust (QLM).

2. Az algoritmus vizsgálatára létrehoztam új feladatosztályokat.

3. A QLM algoritmus futását összehasonlítottam az irodalomban levő, valamint a CPLEX megoldó által szolgáltatott eredményekkel. Az eredmények alapján a QLM algoritmus a vizsgált feladatosztályokon belül hatékonyan működik.

A ládapakolási feladat esetében tárgyakat pakolunk ládákba úgy, hogy az egy ládába pakolt tárgyak összmérete ne lépje túl a láda kapacitását és a felhasznált ládák száma minimális legyen. A problémát új megközelítésben vizsgáltam, amely szerint egy adott feladatosztályba tartozó inputok közül a lehető legtöbbet igyekszünk megoldani optimálisan mohó algoritmusokkal. A cél a lehető legtöbb input optimális megoldása. Mivel a ládapakolás NP-nehéz, ezért nem elvárható, hogy az összes feladat esetén megkapjuk az optimális megoldást. Feladat-osztályoknak a Schwerin és a Falkenauer_U osztályokat választottam. Mindkét feladatosztály esetében az optimális megoldások az irodalomból már ismertek, azonban az algoritmusaim futásához nem szükséges az optimumértékek előzetes ismerete. A Schwerin és a Falkenauer_U feladat-osztályokban található inputok megoldására két algoritmust készítettem: a Schwerin osztályhoz a **REM SW**, a Falkenauer_U osztályhoz az **FU algoritmust**. Mindkét algoritmus a feladatosztályok különböző tulajdonságait kihasználva oldja meg az ott található inputokat. A REM SW algoritmus a Schwerin feladatosztály mind a 200 darab feladatát, míg az FU algoritmus a Falkenauer_U osztály 80 feladatából 73-at oldott meg optimálisan. Ezen a területen az alábbi tudományos hozzájárulások valósultak meg:

1. Megvizsgáltam, hogy a Schwerin és a Falkenauer_U osztályok milyen, a megoldás szempontjából kihasználható tulajdonságokkal rendelkeznek.

2. Mohó algoritmusokat hoztam létre a Schwerin és a Falkenauer_U osztályok számára.

3. A REM SW algoritmus a Schwerin osztály mind a 200 feladatát optimálisan megoldotta.

4. Az FU algoritmus a Falkenauer_U feladatosztály 80 inputjából 73 darabot (91%) optimálisan megoldott.

A ládafedés szállítással egy viszonylag új terület. Mint a ládapakolás esetében, itt is tárgyakat pakolunk ládákba. Azonban itt ahelyett, hogy az egy ládába pakolt tárgyak összmérete ne lépje túl a láda kapacitását, az a feltétel van, hogy a tárgyak összmérete legalább akkora kell hogy legyen, mint a láda kapacitása. Egy ládát fedettnek tekintünk, ha a benne levő tárgyak összmérete legalább a láda kapacitásával egyenlő. Minden elszállított (azaz fedetté vált) láda után profitot realizálunk előre meghatározott célfüggvény alapján. A cél a profit maximalizálása. A megoldás során számon tartunk egy $K > 0$ pozitív egész számot, amely megadja, hogy hány láda lehet nyitva egyszerre. A kutatás során már az irodalomból ismert természetes algoritmusokat implementáltam (**DNF**, **H(K)** és **SH(K)**), valamint kidolgoztam egy új, rugalmas, paraméteres algoritmust is, 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 először lassan, majd négyzetesen csökkenő és egy meredeken csökkenő változatot. A feladat-osztályokon módosításokat hajtottam végre: megbontottam a rendezettséget, normalizálást hajtottam végre és összekapcsoltam a

feladat-osztályokat a nyereség-függvényekkel. Az MMask algoritmus három paraméterrel dolgozik: K - az egyszerre nyitva tartható ládák maximális száma, α - K-dimenziós vektor, β - pozitív egész szám. Az α és β paraméterek az algoritmus elfogadó-elutasító politikájában kerültek felhasználásra. Az MMask algoritmus paramétereinek beállítása kezdetben kézzel, próbálgatással történt. Majd később ez automatizálásra került a lokális kereséssel. Az eredmények alapján az látható, hogy már a manuális paraméter-beállítás mellett is az MMask az esetek döntő többségében javított a természetesen adódó algoritmusok által elért addigi legjobb eredményeken, a lokális kereséssel optimalizált paraméter-beállítás pedig ezen az esetek többségében még tovább javított. Ezen a területen az alábbi tudományos hozzájárulások valósultak meg:

1. Részletes vizsgálatokat folytattam a területen, és ennek érdekében létrehoztam egy új benchmark feladatosztályt Large Range (LR) néven.

2. Az irodalomban már létező algoritmusokat implementáltam a feladatra (DNF, H(K), SH(K)). Az implementált algoritmusokra részletes vizsgálatokat végeztem, és megállapítottam, hogy hatékonyan oldják meg a vizsgált feladatosztály elemeit.

3. Egy új MMask-nak nevezett algoritmust dolgoztam ki (amely a korábbi Mask javított változata). Megállapítottam, hogy a paraméterek megfelelő beállításával az MMask algoritmus a korábbiaknál is jobb eredményeket szolgáltat.

4. A paraméterek optimalizálására létrehoztam egy lokális keresés alapú metaheurisztikát, amely a legjobb eredményeket elérő beállításokat hozta létre. Részletesen megvizsgáltam a további javítási lehetőségeket.

Függelékek

A. függelék

Gépi idők a QLM algoritmus feladataihoz

A.1. Az elsőként generált, alap feladatokhoz tartozó gépi idők és megelőzési relációk

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8
$task_1$	2	10	4	9	1	4	1	10
$task_2$	2	6	9	2	6	7	3	5
$task_3$	7	9	6	2	8	1	5	4
$task_4$	6	10	8	3	8	5	4	3
$task_5$	6	8	7	7	10	5	10	6
$task_6$	3	5	2	1	10	9	1	3
$task_7$	2	7	1	5	10	9	2	9
$task_8$	2	8	6	10	8	6	2	7
$task_9$	9	4	1	3	8	6	5	9
$task_{10}$	8	1	8	7	9	5	5	6
$task_{11}$	1	1	4	4	8	2	5	8
$task_{12}$	5	8	3	3	5	7	10	10
$task_{13}$	3	3	2	3	10	10	3	2
$task_{14}$	2	7	3	3	2	7	5	6

A.1. táblázat. A #1 számú feladat gépi idő táblázata

$$
\begin{array}{ccccc}
task_4 & \rightarrow & task_{11} & \rightarrow & task_{12} \\
task_1 & \rightarrow & task_3 & \rightarrow & task_7 \\
task_0 & \rightarrow & task_5 & &
\end{array}
$$

A.2. táblázat. A #1 számú feladat megelőzési relációi

	m_1	m_2	m_3	m_4	m_5	m_6	m_7
$task1$	6	3	4	9	2	8	10
$task2$	9	6	7	4	10	8	3
$task3$	9	10	1	1	3	6	5
$task4$	1	1	4	1	6	4	3
$task5$	10	7	5	3	2	8	6
$task6$	7	4	10	4	10	8	7
$task7$	4	7	10	3	10	7	3
$task8$	8	7	9	2	2	5	5
$task9$	5	6	9	2	6	6	4
$task10$	6	1	4	7	8	8	6
$task11$	2	3	3	7	10	6	2
$task12$	2	9	8	4	4	5	8
$task13$	6	4	2	6	9	6	2
$task14$	10	10	6	3	10	4	7
$task15$	1	1	9	2	4	6	7
$task16$	9	7	8	9	6	3	9
$task17$	9	6	4	6	6	2	8
$task18$	5	10	5	8	4	5	2
$task19$	10	9	8	2	2	5	5
$task20$	1	10	9	6	10	5	4
$task21$	8	10	4	8	3	4	3
$task22$	5	1	8	5	9	7	6
$task23$	6	3	3	8	6	7	3
$task24$	6	2	1	7	3	5	10
$task25$	7	10	6	2	5	4	2
$task26$	5	7	6	6	10	5	10
$task27$	8	9	9	2	2	6	2
$task28$	4	10	7	4	9	5	6

A.3. táblázat. A #2 számú feladat gépi idő táblázata

$task_1$	$\rightarrow$	$task_5$	$\rightarrow$	$task_{27}$			
$task_2$	$\rightarrow$	$task_8$	$\rightarrow$	$task_{13}$	$\rightarrow$	$task_{20}$	
$task_4$	$\rightarrow$	$task_{10}$					
$task_{12}$	$\rightarrow$	$task_{18}$	$\rightarrow$	$task_{19}$			

A.4. táblázat. A #2 számú feladat megelőzési relációi

	m_1	m_2	m_3	m_4
$task1$	7	2	2	8
$task2$	7	5	2	6
$task3$	10	1	4	5
$task4$	4	1	8	7
$task5$	1	5	7	9
$task6$	8	3	7	2
$task7$	5	6	3	2
$task8$	3	4	7	1
$task9$	3	8	10	6
$task10$	3	10	8	7
$task11$	9	1	8	1
$task12$	2	7	3	3
$task13$	6	4	8	6
$task14$	9	6	3	1
$task15$	8	10	7	9
$task16$	7	7	5	7
$task17$	10	6	7	1
$task18$	4	2	6	8
$task19$	5	5	5	5
$task20$	1	5	4	8
$task21$	9	6	5	5
$task22$	6	3	5	6
$task23$	3	6	5	10
$task24$	6	6	9	9
$task25$	10	3	1	2
$task26$	7	10	10	6
$task27$	1	10	1	1

A.5. táblázat. A #5 számú feladat gépi idő táblázata

$$task_{10} \quad \rightarrow \quad task_{26}$$

A.6. táblázat. A #5 számú feladat megelőzési relációja

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}	m_{16}	m_{17}	m_{18}	m_{19}
$task_1$	5	10	2	4	6	10	3	7	3	4	3	2	2	2	7	5	1	10	10
$task_2$	3	9	6	9	2	8	8	7	8	3	9	4	7	6	8	3	1	1	1
$task_3$	10	10	6	5	3	4	10	5	3	10	3	5	2	4	7	3	7	1	3
$task_4$	1	1	3	9	1	6	7	1	3	1	1	6	6	9	2	2	10	5	4
$task_5$	4	5	2	10	4	4	6	9	10	9	7	7	5	2	2	8	8	9	9
$task_6$	10	8	8	9	2	4	5	2	7	6	6	5	4	6	2	2	8	1	3
$task_7$	9	8	7	2	9	1	2	6	10	4	4	5	5	9	1	7	8	1	3
$task_8$	4	3	7	1	3	6	7	2	9	8	5	2	5	4	4	8	2	3	7
$task_9$	3	2	1	9	10	8	4	6	6	8	7	2	1	6	8	7	3	3	2
$task_{10}$	3	4	5	6	10	7	10	8	3	3	6	5	9	7	2	4	1	3	3
$task_{11}$	2	10	8	4	1	9	9	8	1	3	3	7	2	6	3	3	8	2	6
$task_{12}$	10	9	7	7	2	10	10	2	10	6	8	9	4	3	3	10	7	7	2
$task_{13}$	1	7	4	10	9	3	8	2	1	4	8	7	3	3	8	5	7	3	9
$task_{14}$	10	1	4	5	6	8	7	3	4	4	3	4	6	1	9	8	3	2	9
$task_{15}$	3	5	10	5	4	6	6	7	10	3	9	5	6	2	2	5	1	9	10
$task_{16}$	6	2	2	2	10	5	9	8	5	6	8	1	3	7	5	4	7	2	2
$task_{17}$	8	8	8	6	3	8	8	4	10	9	7	1	6	1	8	3	8	10	9
$task_{18}$	6	5	6	2	10	6	9	10	3	4	2	1	6	5	8	6	10	7	1
$task_{19}$	7	1	1	2	6	1	5	7	3	10	2	5	8	2	10	8	9	3	4
$task_{20}$	5	3	7	5	8	3	7	8	7	10	9	4	4	4	6	1	9	5	7
$task_{21}$	5	6	7	10	2	8	5	4	1	5	3	10	1	7	7	3	5	7	1
$task_{22}$	4	10	1	9	9	3	7	1	5	2	5	4	3	6	1	1	2	3	10
$task_{23}$	10	7	4	1	8	5	3	5	5	3	3	1	5	10	9	4	4	9	8
$task_{24}$	10	5	7	3	3	8	3	3	4	5	7	9	5	2	7	10	4	10	2

A.7. táblázat. A #28 számú feladat gépi idő táblázata (első részlet)

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}	m_{16}	m_{17}	m_{18}	m_{19}
$task_{25}$	8	3	2	9	1	3	3	9	4	8	4	7	3	2	8	9	8	8	7
$task_{26}$	5	6	3	4	10	1	6	9	3	2	9	6	8	10	1	9	10	5	2
$task_{27}$	6	6	2	10	8	8	5	5	6	8	1	4	6	8	7	7	10	3	10
$task_{28}$	9	2	3	10	5	8	7	9	5	9	10	10	4	8	8	5	10	6	6
$task_{29}$	4	7	7	3	8	7	9	6	3	5	8	3	4	1	3	1	6	2	3
$task_{30}$	5	6	1	8	5	6	7	4	3	9	6	4	5	2	1	5	8	8	10
$task_{31}$	9	9	3	2	10	9	6	5	4	9	9	9	1	5	6	1	10	7	4
$task_{32}$	5	7	10	5	1	1	6	8	1	8	8	6	10	7	9	1	5	10	6
$task_{33}$	2	1	4	2	4	3	10	6	4	5	4	4	3	2	1	4	8	8	2
$task_{34}$	9	5	2	8	5	9	6	8	6	5	9	10	1	1	4	2	8	9	9
$task_{35}$	10	4	4	8	3	10	3	7	10	2	6	7	2	2	3	4	3	7	5
$task_{36}$	1	1	1	8	7	8	2	10	3	9	1	6	5	10	9	8	6	2	8
$task_{37}$	8	4	2	9	3	3	3	7	8	7	5	6	8	3	7	6	4	8	8
$task_{38}$	6	4	3	2	5	9	3	6	9	10	5	7	6	2	5	3	10	10	7
$task_{39}$	2	6	9	1	2	2	3	5	7	1	10	3	6	2	4	5	10	6	9
$task_{40}$	3	7	5	1	10	9	6	8	3	5	4	6	8	7	1	9	8	10	10
$task_{41}$	3	10	10	6	9	8	2	1	2	5	6	3	3	4	1	10	2	4	7
$task_{42}$	10	6	8	8	3	9	7	2	4	8	5	3	8	7	8	9	3	9	5
$task_{43}$	6	2	8	3	8	7	9	10	2	10	6	3	8	10	6	10	6	10	6
$task_{44}$	2	4	2	8	10	10	9	3	8	5	1	4	8	10	6	2	10	4	4
$task_{45}$	5	7	4	1	2	2	2	5	2	8	3	4	4	9	1	9	8	4	7
$task_{46}$	10	1	3	1	6	6	8	4	9	1	1	1	3	4	2	4	2	5	10
$task_{47}$	1	1	6	2	6	8	7	7	1	1	2	3	1	6	10	9	7	4	2
$task_{48}$	2	2	10	1	9	10	1	1	7	2	9	4	4	8	4	2	8	2	5
$task_{49}$	6	3	1	6	4	5	5	10	4	9	8	3	4	5	8	6	5	5	4

A.8. táblázat. A #28 számú feladat gépi idő táblázata (második részlet)

	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}	m_{16}	m_{17}	m_{18}	m_{19}
$task_{50}$	6	8	2	1	8	4	9	8	4	9	7	7	5	5	8	8	2	1	3
$task_{51}$	7	4	8	8	4	7	8	3	1	2	6	3	5	9	9	4	4	7	1
$task_{52}$	9	2	9	5	7	10	7	8	1	7	8	1	3	8	3	5	2	4	10
$task_{53}$	10	8	9	1	10	10	2	9	6	1	3	5	8	8	5	8	2	2	3
$task_{54}$	10	3	6	9	4	4	3	5	10	5	2	2	5	4	6	9	7	1	6
$task_{55}$	8	10	6	1	4	6	6	1	9	3	3	1	3	5	4	2	8	6	10
$task_{56}$	1	10	3	1	7	3	2	6	2	8	7	2	6	6	10	4	5	1	7
$task_{57}$	7	10	8	4	8	6	2	7	9	10	6	10	3	9	5	7	9	1	8
$task_{58}$	7	7	2	6	8	4	5	2	1	2	5	6	7	1	5	1	9	3	7
$task_{59}$	7	2	9	6	10	3	9	2	6	6	8	9	8	2	7	4	10	7	5
$task_{60}$	3	9	9	4	1	3	7	3	10	7	8	4	1	6	9	1	7	2	9
$task_{61}$	7	3	7	7	7	2	6	9	8	7	7	5	6	1	9	7	10	3	10
$task_{62}$	3	4	2	9	10	7	10	5	1	7	10	6	9	1	6	2	4	8	10
$task_{63}$	9	3	1	7	9	10	8	6	9	7	9	10	5	10	9	4	5	7	10
$task_{64}$	8	10	1	7	9	5	4	7	5	9	5	1	8	8	2	8	4	8	1
$task_{65}$	5	5	4	1	8	7	6	2	2	5	8	9	2	7	9	4	7	4	1
$task_{66}$	3	10	10	5	9	7	5	2	6	7	7	10	7	9	2	6	3	9	9
$task_{67}$	2	3	3	3	4	4	5	8	6	10	3	8	1	1	6	3	5	2	1
$task_{68}$	8	6	7	5	2	3	3	1	10	3	10	1	10	2	1	1	5	6	2
$task_{69}$	5	9	3	8	9	5	3	6	1	9	5	1	3	2	9	1	7	1	1
$task_{70}$	6	2	8	8	10	2	7	6	6	7	10	8	7	7	8	4	5	6	5
$task_{71}$	3	9	5	5	6	6	10	9	4	1	1	5	6	6	2	4	3	4	6
$task_{72}$	1	5	7	1	2	1	3	7	6	7	10	10	4	4	6	10	10	1	10
$task_{73}$	5	5	10	7	5	2	3	8	8	1	2	1	10	10	4	2	9	6	2
$task_{74}$	3	1	5	8	6	6	6	2	3	6	10	8	10	4	3	8	4	6	2

A.9. táblázat. A #28 számú feladat gépi idő táblázata (harmadik részlet)

$task_2$	$\rightarrow$	$task_{30}$	$\rightarrow$	$task_{33}$	$\rightarrow$	$task_{34}$
$task_{10}$	$\rightarrow$	$task_{12}$	$\rightarrow$	$task_{17}$		
$task_{60}$	$\rightarrow$	$task_{65}$	$\rightarrow$	$task_{66}$		
$task_{67}$	$\rightarrow$	$task_{68}$	$\rightarrow$	$task_{69}$		
$task_7$	$\rightarrow$	$task_9$				

A.10. táblázat. A #28 számú feladat megelőzési relációi

A.2. A bővített feladatokhoz tartozó gépi idők és megelőzési relációk összefoglaló táblázatai

A bővített feladatok esetében osztályonként tíz feladatról van szó, azaz összesen negyven darabról. Nyilván mindegyikhez tartozik egy gépi idő táblázat és megelőzési relációk. Ezen táblázatok teljes terjedelmű közlése nem célszerű helyhiány miatt.

A feladatosztályokra vonatkozó összefoglaló táblázatban megadom a gépi idők generálásának módját, az alkalmazott diszkrét intervallumot és a megelőzési relációkat.

Class #1				
Feladatok:	1	2	3	4
Intervallum	$[1, 10]$, diszkrét			
Eloszlás	egyenletes			
	Megelőzési relációk			
	$task_5 \rightarrow task_{12}$	$task_4 \rightarrow task_{14}$	$task_{11} \rightarrow task_{13}$	$task_2 \rightarrow task_{10}$
	$task_{12} \rightarrow task_{13}$	$task_2 \rightarrow task_3$	$task_7 \rightarrow task_9$	$task_{10} \rightarrow task_{11}$
	$task_2 \rightarrow task_4$	$task_3 \rightarrow task_8$	$task_6 \rightarrow task_{10}$	$task_{11} \rightarrow task_{12}$
	$task_4 \rightarrow task_8$	$task_{10} \rightarrow task_{11}$	$task_1 \rightarrow task_4$	$task_{12} \rightarrow task_7$
	$task_1 \rightarrow task_6$	$task_1 \rightarrow task_9$	$task_4 \rightarrow task_5$	$task_4 \rightarrow task_3$

A.11. táblázat. A *Class* #1 osztály feladatainak gépi idő adatai és megelőzési relációi (első részlet)

Class #1							
Feladatok:	5	6	7	8	9	10	
Intervallum	$[1, 10]$, diszkrét						
Eloszlás	egyenletes						
	Megelőzési relációk						
	$task_5 \rightarrow task_6$	$task_3 \rightarrow task_5$	$task_1 \rightarrow task_3$	$task_2 \rightarrow task_4$	$task_4 \rightarrow task_3$	$task_1 \rightarrow task_4$	
	$task_6 \rightarrow task_7$	$task_9 \rightarrow task_7$	$task_3 \rightarrow task_5$	$task_4 \rightarrow task_6$	$task_8 \rightarrow task_{12}$	$task_4 \rightarrow task_7$	
	$task_7 \rightarrow task_8$	$task_4 \rightarrow task_6$	$task_5 \rightarrow task_7$	$task_6 \rightarrow task_8$	$task_5 \rightarrow task_6$	$task_7 \rightarrow task_{10}$	
	$task_8 \rightarrow task_9$	$task_6 \rightarrow task_8$	$task_7 \rightarrow task_9$	$task_8 \rightarrow task_{10}$	$task_7 \rightarrow task_{10}$	$task_{10} \rightarrow task_{13}$	
	$task_9 \rightarrow task_{10}$	$task_{13} \rightarrow task_{14}$	$task_9 \rightarrow task_{11}$	$task_{10} \rightarrow task_{12}$	$task_1 \rightarrow task_5$	$task_{13} \rightarrow task_{14}$	

A.12. táblázat. A *Class* #1 osztály feladatainak gépi idő adatai és megelőzési relációi (második részlet)

Class #1 osztály megelőzési relációi (összevont diszjunkt utak formájában)

1. feladat:

$$\begin{array}{lllll} task_5 & \rightarrow & task_{12} & \rightarrow & task_{13} \\ task_2 & \rightarrow & task_4 & \rightarrow & task_8 \\ task_1 & \rightarrow & task_6 & & \end{array}$$

2. feladat:

$$\begin{array}{lllll} task_4 & \rightarrow & task_{14} & & \\ task_2 & \rightarrow & task_3 & \rightarrow & task_8 \\ task_{10} & \rightarrow & task_{11} & & \\ task_1 & \rightarrow & task_9 & & \end{array}$$

3. feladat:

$$\begin{array}{lllll} task_{11} & \rightarrow & task_{13} & & \\ task_7 & \rightarrow & task_9 & & \\ task_1 & \rightarrow & task_4 & \rightarrow & task_5 \end{array}$$

4. feladat:

$$\begin{array}{lllllllll} task_2 & \rightarrow & task_{10} & \rightarrow & task_{11} & \rightarrow & task_{12} & \rightarrow & task_7 \\ task_4 & \rightarrow & task_3 & & & & & & \end{array}$$

5. feladat:

$$task_5 \rightarrow task_6 \rightarrow task_7 \rightarrow task_8 \rightarrow task_9 \rightarrow task_{10}$$

6. feladat:

$$\begin{array}{lllll} task_3 & \rightarrow & task_5 & & \\ task_9 & \rightarrow & task_7 & & \\ task_4 & \rightarrow & task_6 & \rightarrow & task_8 \\ task_{13} & \rightarrow & task_{14} & & \end{array}$$

7. feladat:

$$task_1 \rightarrow task_3 \rightarrow task_5 \rightarrow task_7 \rightarrow task_9 \rightarrow task_{11}$$

8. feladat:

$$task_2 \rightarrow task_4 \rightarrow task_6 \rightarrow task_8 \rightarrow task_{10} \rightarrow task_{12}$$

9. feladat:

$$\begin{array}{lllll} task_4 & \rightarrow & task_3 & & \\ task_8 & \rightarrow & task_{12} & & \\ task_1 & \rightarrow & task_5 & \rightarrow & task_6 \\ task_7 & \rightarrow & task_{10} & & \end{array}$$

10. feladat:

$$task_1 \rightarrow task_4 \rightarrow task_7 \rightarrow task_{10} \rightarrow task_{13} \rightarrow task_{14}$$

Class #2				
Feladatok:	1	2	3	4
Intervallum	[1, 10], diszkrét			
Eloszlás	egyenletes			
	Megelőzési relációk			
	$task_2 \rightarrow task_6$	$task_1 \rightarrow task_4$	$task_2 \rightarrow task_5$	$task_4 \rightarrow task_{10}$
	$task_6 \rightarrow task_{28}$	$task_8 \rightarrow task_{23}$	$task_9 \rightarrow task_{24}$	$task_3 \rightarrow task_{11}$
	$task_3 \rightarrow task_9$	$task_{11} \rightarrow task_{14}$	$task_{12} \rightarrow task_{15}$	$task_{24} \rightarrow task_{26}$
	$task_9 \rightarrow task_{14}$	$task_{10} \rightarrow task_{11}$	$task_{11} \rightarrow task_{12}$	$task_8 \rightarrow task_{12}$
	$task_{14} \rightarrow task_{20}$	$task_{19} \rightarrow task_{27}$	$task_{20} \rightarrow task_1$	$task_{22} \rightarrow task_{21}$
	$task_5 \rightarrow task_{11}$	$task_{24} \rightarrow task_{26}$	$task_{25} \rightarrow task_{27}$	$task_{27} \rightarrow task_5$
	$task_{13} \rightarrow task_{19}$	$task_{15} \rightarrow task_{17}$	$task_{16} \rightarrow task_{18}$	$task_{18} \rightarrow task_{19}$
	$task_{19} \rightarrow task_{20}$	$task_2 \rightarrow task_3$	$task_3 \rightarrow task_4$	$task_5 \rightarrow task_{15}$

A.13. táblázat. A *Class #2* osztály feladatainak gépi idő adatai és megelőzési relációi (első részlet)

Class #2						
Feladatok:	5	6	7	8	9	10
Intervallum	$[1, 10]$, diszkrét					
Eloszlás	egyenletes					
	Megelőzési relációk					
	$task_{12} \rightarrow task_{14}$	$task_2 \rightarrow task_1$	$task_{13} \rightarrow task_3$	$task_{11} \rightarrow task_8$	$task_7 \rightarrow task_3$	$task_{10} \rightarrow task_4$
	$task_{14} \rightarrow task_{15}$	$task_3 \rightarrow task_{21}$	$task_4 \rightarrow task_{10}$	$task_{24} \rightarrow task_7$	$task_{22} \rightarrow task_{26}$	$task_{11} \rightarrow task_{13}$
	$task_{22} \rightarrow task_{24}$	$task_{14} \rightarrow task_{24}$	$task_{14} \rightarrow task_{24}$	$task_5 \rightarrow task_6$	$task_4 \rightarrow task_1$	$task_6 \rightarrow task_7$
	$task_{23} \rightarrow task_{25}$	$task_5 \rightarrow task_8$	$task_7 \rightarrow task_9$	$task_7 \rightarrow task_9$	$task_1 \rightarrow task_2$	$task_7 \rightarrow task_8$
	$task_8 \rightarrow task_{10}$	$task_8 \rightarrow task_{10}$	$task_5 \rightarrow task_{11}$	$task_{10} \rightarrow task_{22}$	$task_2 \rightarrow task_{10}$	$task_8 \rightarrow task_{27}$
	$task_{10} \rightarrow task_{11}$	$task_{10} \rightarrow task_{11}$	$task_{18} \rightarrow task_8$	$task_{18} \rightarrow task_{16}$	$task_{15} \rightarrow task_{24}$	$task_{14} \rightarrow task_{19}$
	$task_{18} \rightarrow task_{19}$	$task_{23} \rightarrow task_{27}$	$task_{20} \rightarrow task_{21}$	$task_{20} \rightarrow task_{21}$	$task_{17} \rightarrow task_{19}$	$task_{19} \rightarrow task_{21}$
	$task_{13} \rightarrow task_{17}$	$task_{12} \rightarrow task_{16}$	$task_{22} \rightarrow task_{16}$	$task_9 \rightarrow task_{19}$	$task_{18} \rightarrow task_{25}$	$task_1 \rightarrow task_{23}$

A.14. táblázat. A *Class #2* osztály feladatainak gépi idő adatai és megelőzési relációi (második részlet)

Class #2 osztály megelőzési relációi (összevont diszjunkt utak formájában)

1. feladat:

$$\begin{array}{llllllll} task_2 & \rightarrow & task_6 & \rightarrow & task_{28} & & & \\ task_3 & \rightarrow & task_9 & \rightarrow & task_{14} & \rightarrow & task_{20} & \\ task_5 & \rightarrow & task_{11} & & & & & \\ task_{13} & \rightarrow & task_{19} & \rightarrow & task_{20} & & & \end{array}$$

2. feladat:

$$\begin{array}{lllll} task_1 & \rightarrow & task_4 & & \\ task_8 & \rightarrow & task_{23} & & \\ task_{10} & \rightarrow & task_{11} & \rightarrow & task_{14} \\ task_{19} & \rightarrow & task_{27} & & \\ task_{24} & \rightarrow & task_{26} & & \\ task_{15} & \rightarrow & task_{17} & & \\ task_2 & \rightarrow & task_3 & & \end{array}$$

3. feladat:

$$\begin{array}{lllll} task_2 & \rightarrow & task_5 & & \\ task_9 & \rightarrow & task_{24} & & \\ task_{11} & \rightarrow & task_{12} & \rightarrow & task_{15} \\ task_{20} & \rightarrow & task_1 & & \\ task_{25} & \rightarrow & task_{27} & & \\ task_{16} & \rightarrow & task_{18} & & \\ task_3 & \rightarrow & task_4 & & \end{array}$$

4. feladat:

$$\begin{array}{lllll} task_4 & \rightarrow & task_{10} & & \\ task_3 & \rightarrow & task_{11} & & \\ task_{27} & \rightarrow & task_5 & \rightarrow & task_{15} \\ task_8 & \rightarrow & task_{12} & & \\ task_{24} & \rightarrow & task_{26} & & \\ task_{22} & \rightarrow & task_{21} & & \\ task_{18} & \rightarrow & task_{19} & & \end{array}$$

5. feladat:

$$
\begin{array}{lllll}
task_{12} & \rightarrow & task_{14} & \rightarrow & task_{15} \\
task_{22} & \rightarrow & task_{24} & & \\
task_{23} & \rightarrow & task_{25} & & \\
task_{8} & \rightarrow & task_{10} & \rightarrow & task_{11} \\
task_{18} & \rightarrow & task_{19} & & \\
task_{13} & \rightarrow & task_{17} & &
\end{array}
$$

6. feladat:

$$
\begin{array}{lllllll}
task_{2} & \rightarrow & task_{1} & & & & \\
task_{3} & \rightarrow & task_{21} & & & & \\
task_{14} & \rightarrow & task_{24} & & & & \\
task_{5} & \rightarrow & task_{8} & \rightarrow & task_{10} & \rightarrow & task_{11} \\
task_{23} & \rightarrow & task_{27} & & & & \\
task_{12} & \rightarrow & task_{16} & & & &
\end{array}
$$

7. feladat:

$$
\begin{array}{lll}
task_{13} & \rightarrow & task_{3} \\
task_{4} & \rightarrow & task_{10} \\
task_{14} & \rightarrow & task_{24} \\
task_{7} & \rightarrow & task_{9} \\
task_{5} & \rightarrow & task_{11} \\
task_{18} & \rightarrow & task_{8} \\
task_{20} & \rightarrow & task_{21} \\
task_{22} & \rightarrow & task_{16}
\end{array}
$$

8. feladat:

$$
\begin{array}{lllll}
task_{11} & \rightarrow & task_{8} & & \\
task_{24} & \rightarrow & task_{7} & & \\
task_{5} & \rightarrow & task_{6} & & \\
task_{7} & \rightarrow & task_{9} & \rightarrow & task_{19} \\
task_{10} & \rightarrow & task_{22} & & \\
task_{18} & \rightarrow & task_{16} & & \\
task_{20} & \rightarrow & task_{21} & &
\end{array}
$$

9. feladat:

$$
\begin{array}{lllllll}
task_7 & \rightarrow & task_3 & & & & \\
task_{22} & \rightarrow & task_{26} & & & & \\
task_4 & \rightarrow & task_1 & \rightarrow & task_2 & \rightarrow & task_{10} \\
task_{15} & \rightarrow & task_{24} & & & & \\
task_{17} & \rightarrow & task_{19} & & & & \\
task_{18} & \rightarrow & task_{25} & & & &
\end{array}
$$

10. feladat:

$$
\begin{array}{lllllll}
task_{10} & \rightarrow & task_4 & & & & \\
task_{11} & \rightarrow & task_{13} & & & & \\
task_6 & \rightarrow & task_7 & \rightarrow & task_8 & \rightarrow & task_{27} \\
task_{14} & \rightarrow & task_{19} & \rightarrow & task_{21} & & \\
task_1 & \rightarrow & task_{23} & & & &
\end{array}
$$

Class #3				
Feladatok:	1	2	3	4
Intervallum	$[1, 10]$, diszkrét			
Eloszlás	egyenletes			
	Megelőzési relációk			
	$task_{10} \rightarrow task_4$	$task_{11} \rightarrow task_8$	$task_{24} \rightarrow task_{15}$	$task_{27} \rightarrow task_{12}$

A.15. táblázat. A *Class #3* osztály feladatainak gépi idő adatai és megelőzési relációi (első részlet)

Class #3							
Feladatok:	5	6	7	8	9		10
Intervallum	[1, 10], diszkrét						
Eloszlás	egyenletes						
	Megelőzési relációk						
	$task_8 \rightarrow task_{24}$	$task_{15} \rightarrow task_{26}$	$task_1 \rightarrow task_{27}$	$task_8 \rightarrow task_{18}$	$task_{14} \rightarrow task_{19}$		$task_{21} \rightarrow task_2$

A.16. táblázat. A *Class #3* osztály feladatainak gépi idő adatai és megelőzési relációi (második részlet)

Class #3 osztály megelőzési relációi (összevont diszjunkt utak formájában)

1. feladat:

$$task_{10} \rightarrow task_4$$

2. feladat:

$$task_{11} \rightarrow task_8$$

3. feladat:

$$task_{24} \rightarrow task_{15}$$

4. feladat:

$$task_{27} \rightarrow task_{12}$$

5. feladat:

$$task_8 \rightarrow task_{24}$$

6. feladat:

$$task_{15} \rightarrow task_{26}$$

7. feladat:

$$task_1 \rightarrow task_{27}$$

8. feladat:

$$task_8 \rightarrow task_{18}$$

9. feladat:

$$task_{14} \rightarrow task_{19}$$

10. feladat:

$$task_{21} \rightarrow task_2$$

Class #4				
Feladatok:	1	2	3	4
Intervallum	[1, 10], diszkrét			
Eloszlás	egyenletes			
	Megelőzési relációk			
	$task_3 \rightarrow task_{31}$	$task_1 \rightarrow task_{13}$	$task_4 \rightarrow task_5$	$task_{13} \rightarrow task_{15}$
	$task_{31} \rightarrow task_{34}$	$task_5 \rightarrow task_9$	$task_5 \rightarrow task_9$	$task_{17} \rightarrow task_{19}$
	$task_{34} \rightarrow task_{35}$	$task_9 \rightarrow task_{10}$	$task_9 \rightarrow task_{23}$	$task_{22} \rightarrow task_{23}$
	$task_{11} \rightarrow task_{13}$	$task_{14} \rightarrow task_{27}$	$task_{15} \rightarrow task_8$	$task_{21} \rightarrow task_{34}$
	$task_{13} \rightarrow task_{18}$	$task_{71} \rightarrow task_{34}$	$task_{72} \rightarrow task_{70}$	$task_{34} \rightarrow task_{35}$
	$task_{61} \rightarrow task_{66}$	$task_{33} \rightarrow task_{37}$	$task_{37} \rightarrow task_{44}$	$task_{35} \rightarrow task_{36}$
	$task_{66} \rightarrow task_{67}$	$task_{57} \rightarrow task_{48}$	$task_{12} \rightarrow task_{41}$	$task_{67} \rightarrow task_{64}$
	$task_{68} \rightarrow task_{69}$	$task_3 \rightarrow task_{35}$	$task_{58} \rightarrow task_{26}$	$task_{62} \rightarrow task_{26}$
	$task_{69} \rightarrow task_{70}$	$task_{45} \rightarrow task_{60}$	$task_{11} \rightarrow task_{12}$	$task_{18} \rightarrow task_{12}$
	$task_8 \rightarrow task_{10}$	$task_8 \rightarrow task_{32}$	$task_{12} \rightarrow task_{20}$	$task_{12} \rightarrow task_{56}$

A.17. táblázat. A *Class #4* osztály feladatainak gépi idő adatai és megelőzési relációi (első részlet)

Class #4							
Feladatok:	5	6	7	8	9	10	
Intervallum	[1, 10], diszkrét						
Eloszlás	egyenletes						
	Megelőzési relációk						
	$task_2 \rightarrow task_3$	$task_{34} \rightarrow task_{35}$	$task_{24} \rightarrow task_{25}$	$task_1 \rightarrow task_2$	$task_{15} \rightarrow task_{24}$	$task_{11} \rightarrow task_{12}$	
	$task_3 \rightarrow task_4$	$task_{35} \rightarrow task_{24}$	$task_{25} \rightarrow task_{14}$	$task_2 \rightarrow task_3$	$task_{31} \rightarrow task_6$	$task_{12} \rightarrow task_{23}$	
	$task_4 \rightarrow task_5$	$task_{58} \rightarrow task_{59}$	$task_{48} \rightarrow task_{49}$	$task_3 \rightarrow task_{49}$	$task_6 \rightarrow task_{12}$	$task_{23} \rightarrow task_{24}$	
	$task_6 \rightarrow task_7$	$task_{68} \rightarrow task_{70}$	$task_{58} \rightarrow task_{59}$	$task_4 \rightarrow task_{59}$	$task_{73} \rightarrow task_{74}$	$task_{71} \rightarrow task_{73}$	
	$task_{56} \rightarrow task_{57}$	$task_{69} \rightarrow task_4$	$task_{59} \rightarrow task_{60}$	$task_{59} \rightarrow task_{68}$	$task_{71} \rightarrow task_{72}$	$task_{47} \rightarrow task_{51}$	
	$task_{57} \rightarrow task_{58}$	$task_{71} \rightarrow task_{72}$	$task_{61} \rightarrow task_{62}$	$task_{68} \rightarrow task_{71}$	$task_{72} \rightarrow task_{73}$	$task_{52} \rightarrow task_{68}$	
	$task_{59} \rightarrow task_{60}$	$task_3 \rightarrow task_6$	$task_2 \rightarrow task_7$	$task_{12} \rightarrow task_{66}$	$task_{45} \rightarrow task_{46}$	$task_4 \rightarrow task_5$	
	$task_{60} \rightarrow task_{61}$	$task_5 \rightarrow task_{10}$	$task_4 \rightarrow task_{11}$	$task_{66} \rightarrow task_{70}$	$task_{46} \rightarrow task_{47}$	$task_5 \rightarrow task_6$	
	$task_{18} \rightarrow task_{12}$	$task_4 \rightarrow task_5$	$task_5 \rightarrow task_6$	$task_5 \rightarrow task_6$	$task_{60} \rightarrow task_{61}$	$task_{10} \rightarrow task_{21}$	
	$task_{12} \rightarrow task_{56}$	$task_{18} \rightarrow task_{62}$	$task_{19} \rightarrow task_{63}$	$task_{19} \rightarrow task_{63}$	$task_{19} \rightarrow task_{63}$	$task_{42} \rightarrow task_{14}$	

A.18. táblázat. A *Class #4* osztály feladatainak gépi idő adatai és megelőzési relációi (második részlet)

Class #4 osztály megelőzési relációi (összevont diszjunkt utak formájában)

1. feladat:

$$
\begin{array}{lllllll}
task_3 & \rightarrow & task_{31} & \rightarrow & task_{34} & \rightarrow & task_{35} \\
task_{11} & \rightarrow & task_{13} & \rightarrow & task_{18} & & \\
task_{61} & \rightarrow & task_{66} & \rightarrow & task_{67} & & \\
task_{68} & \rightarrow & task_{69} & \rightarrow & task_{70} & & \\
task_8 & \rightarrow & task_{10} & & & &
\end{array}
$$

2. feladat:

$$
\begin{array}{lllll}
task_1 & \rightarrow & task_{13} & & \\
task_5 & \rightarrow & task_9 & \rightarrow & task_{10} \\
task_{14} & \rightarrow & task_{27} & & \\
task_{71} & \rightarrow & task_{34} & & \\
task_{33} & \rightarrow & task_{37} & & \\
task_{57} & \rightarrow & task_{48} & & \\
task_3 & \rightarrow & task_{35} & & \\
task_{45} & \rightarrow & task_{60} & & \\
task_8 & \rightarrow & task_{32} & &
\end{array}
$$

3. feladat:

$$
\begin{array}{lllllll}
task_4 & \rightarrow & task_5 & \rightarrow & task_9 & \rightarrow & task_{23} \\
task_{15} & \rightarrow & task_8 & & & & \\
task_{72} & \rightarrow & task_{70} & & & & \\
task_{37} & \rightarrow & task_{44} & & & & \\
task_{12} & \rightarrow & task_{41} & & & & \\
task_{58} & \rightarrow & task_{26} & & & & \\
task_{11} & \rightarrow & task_{12} & \rightarrow & task_{20} & &
\end{array}
$$

4. feladat:

$$
\begin{array}{lllllll}
task_{21} & \rightarrow & task_{34} & \rightarrow & task_{35} & \rightarrow & task_{36} \\
task_{13} & \rightarrow & task_{15} & & & & \\
task_{17} & \rightarrow & task_{19} & & & & \\
task_{22} & \rightarrow & task_{23} & & & & \\
task_{67} & \rightarrow & task_{64} & & & & \\
task_{62} & \rightarrow & task_{26} & & & & \\
task_{18} & \rightarrow & task_{12} & \rightarrow & task_{56} & &
\end{array}
$$

5. feladat:

$$
\begin{array}{lllllllll}
task_2 & \rightarrow & task_3 & \rightarrow & task_4 & \rightarrow & task_5 & & \\
task_6 & \rightarrow & task_7 & & & & & & \\
task_{59} & \rightarrow & task_{60} & \rightarrow & task_{61} & & & & \\
task_{18} & \rightarrow & task_{12} & \rightarrow & task_{56} & \rightarrow & task_{57} & \rightarrow & task_{58}
\end{array}
$$

6. feladat:

$$
\begin{array}{lllllll}
task_{34} & \rightarrow & task_{35} & \rightarrow & task_{24} & & \\
task_{58} & \rightarrow & task_{59} & & & & \\
task_{68} & \rightarrow & task_{70} & & & & \\
task_{69} & \rightarrow & task_4 & \rightarrow & task_5 & \rightarrow & task_{10} \\
task_{71} & \rightarrow & task_{72} & & & & \\
task_3 & \rightarrow & task_6 & & & & \\
task_{18} & \rightarrow & task_{62} & & & &
\end{array}
$$

7. feladat:

$$
\begin{array}{lllll}
task_{24} & \rightarrow & task_{25} & \rightarrow & task_{14} \\
task_{48} & \rightarrow & task_{49} & & \\
task_{58} & \rightarrow & task_{59} & \rightarrow & task_{60} \\
task_{61} & \rightarrow & task_{62} & & \\
task_2 & \rightarrow & task_7 & & \\
task_4 & \rightarrow & task_{11} & & \\
task_5 & \rightarrow & task_6 & & \\
task_{19} & \rightarrow & task_{63} & &
\end{array}
$$

8. feladat:

$$
\begin{array}{lllllll}
task_1 & \rightarrow & task_2 & \rightarrow & task_3 & \rightarrow & task_{49} \\
task_4 & \rightarrow & task_{59} & \rightarrow & task_{68} & \rightarrow & task_{71} \\
task_{12} & \rightarrow & task_{66} & \rightarrow & task_{70} & & \\
task_5 & \rightarrow & task_6 & & & & \\
task_{19} & \rightarrow & task_{63} & & & &
\end{array}
$$

9. feladat:

$$
\begin{array}{lllllll}
task_{71} & \rightarrow & task_{72} & \rightarrow & task_{73} & \rightarrow & task_{74} \\
task_{31} & \rightarrow & task_6 & \rightarrow & task_{12} & & \\
task_{45} & \rightarrow & task_{46} & \rightarrow & task_{47} & & \\
task_{15} & \rightarrow & task_{24} & & & & \\
task_{60} & \rightarrow & task_{61} & & & & \\
task_{19} & \rightarrow & task_{63} & & & &
\end{array}
$$

10. feladat:

$$
\begin{array}{ccccccc}
task_{11} & \rightarrow & task_{12} & \rightarrow & task_{23} & \rightarrow & task_{24} \\
task_{4} & \rightarrow & task_{5} & \rightarrow & task_{6} & & \\
task_{71} & \rightarrow & task_{73} & & & & \\
task_{47} & \rightarrow & task_{51} & & & & \\
task_{52} & \rightarrow & task_{68} & & & & \\
task_{10} & \rightarrow & task_{21} & & & & \\
task_{42} & \rightarrow & task_{14} & & & &
\end{array}
$$

B. függelék

Futási idők a QLM algoritmus feladataihoz

B.1. Az elsőként generált, alap feladatokhoz tartozó futási idők

Feladat	Futási idők futásonként (mp)										Átlag (mp)
	1	2	3	4	5	6	7	8	9	10	
#1	0,15	0,29	0,43	0,58	0,73	0,90	0,71	0,28	0,44	0,59	0,51
#2	0,91	0,85	0,66	0,46	0,32	0,20	0,62	0,97	0,81	0,59	0,64
#5	0,47	0,85	0,11	0,38	0,70	0,92	0,55	0,21	0,96	0,13	0,53
#28	0,84	0,87	0,55	0,67	0,92	0,14	0,26	0,45	0,89	0,28	0,59

B.1. táblázat. Az alap feladatokhoz tartozó futási idők

B.2. A Class #1, Class #2, Class #3 és Class #4 feladatosztályokhoz tartozó futási idők

Feladat-osztály	Futási idők futásonként (mp)										Átlag (mp)
	1	2	3	4	5	6	7	8	9	10	
Class #1	0,16	0,29	0,44	0,59	0,71	0,83	0,95	0,84	0,21	0,33	0,54
	0,13	0,25	0,37	0,49	0,61	0,73	0,85	0,97	0,94	0,21	0,55
	0,13	0,25	0,38	0,50	0,63	0,75	0,88	0,12	0,14	0,26	0,40
	0,14	0,27	0,39	0,52	0,65	0,78	0,91	0,39	0,17	0,27	0,45

B.2. táblázat. A Class #1 feladatokhoz tartozó futási idők

Feladat-osztály	Futási idők futásonként (mp)										Átlag (mp)
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Class #2	0,71	0,41	0,10	0,81	0,50	0,22	0,94	0,65	0,34	0,42	0,51
	0,73	0,48	0,20	0,93	0,65	0,04	0,12	0,83	0,53	0,24	0,47
	0,73	0,47	0,21	0,55	0,77	0,48	0,21	0,94	0,64	0,37	0,54
	0,74	0,47	0,20	0,97	0,71	0,45	0,22	0,92	0,63	0,33	0,56

B.3. táblázat. A Class #2 feladatokhoz tartozó futási idők

Feladat-osztály	Futási idők futásonként (mp)										Átlag (mp)
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Class #3	0,73	0,45	0,18	1,00	0,72	0,45	0,22	0,96	0,67	0,38	0,58
	0,76	0,53	0,26	0,98	0,70	0,41	0,15	0,99	0,79	0,51	0,61
	0,86	0,64	0,37	0,11	0,84	0,64	0,37	0,89	0,80	0,54	0,61
	0,73	0,48	0,23	0,98	0,74	0,48	0,29	0,83	0,82	0,57	0,62

B.4. táblázat. A Class #3 feladatokhoz tartozó futási idők

Feladat-osztály	Futási idők futásonként (mp)										Átlag (mp)
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Class #4	0,74	0,25	0,49	0,94	0,67	0,32	0,81	0,99	0,76	0,39	0,64
	0,49	0,79	0,15	0,25	0,40	0,65	0,17	0,56	0,35	0,35	0,42
	0,44	0,17	0,40	0,59	0,89	0,53	0,25	0,67	0,17	0,66	0,48
	0,67	0,48	0,52	0,98	0,53	0,90	0,46	0,90	0,31	0,41	0,62

B.5. táblázat. A Class #4 feladatokhoz tartozó futási idők

C. függelék

A CPLEXsz-el megoldott eredeti optimalizálási feladat modellje és a GAMS kód

C.1. Az eredeti modell

$$Min\ C_{max} = \max_{j \in J} FT_j \tag{C.1}$$

s. t.

$$\sum_{v=1}^{m} \sum_{r=1}^{UB} x_{jvr} = 1, \forall j \in J \tag{C.2}$$

$$\sum_{j=1}^{n} x_{jvr} \leq 1, \forall r \in R, \forall v \in M \tag{C.3}$$

$$\begin{gathered} \sum_{i=1}^{n} x_{ivr} - \sum_{j=1}^{n} x_{j,v,r-1} \leq 0, \\ \forall v \in M, \forall r \in \{2, \ldots, UB\} \end{gathered} \tag{C.4}$$

$$\begin{gathered} FT_j - FT_i + L(2 - x_{jvr} - x_{i,v,r-1}) \geq p_{jv}, \\ \forall i, j \in J, i \neq j, \forall v \in M, \forall r \in \{2, \ldots, UB\} \end{gathered} \tag{C.5}$$

$$FT_j \geq \sum_{r=1}^{UB} p_{jv} x_{jvr}, \forall j \in J, \forall v \in M \tag{C.6}$$

$$\begin{gathered} FT_j - FT_i \geq \sum_{v=1}^{m} \sum_{r=1}^{UB} p_{jv} x_{jvr}, \forall i \in P_j \\ x_{jvr} \in \{0, 1\}, FT_j \geq 0, \forall j \in J, \forall v \in M, \forall r \in R \end{gathered} \tag{C.7}$$

J	munkák halmaza, $J = \{1, \ldots, n\}$
M	gépek halmaza, $M = \{1, \ldots, m\}$
UB	azon pozíciók maximális száma minden gépen, ahová a feladatok kerülnek, $UB = n - m + 1$
R	pozíciók halmaza, $R = \{1, \ldots, UB\}$
p_{jv}	a j. munka végrehajtási ideje a v gépen
P_j	a j. munkát közvetlenül megelőző munkák halmaza
L	egy nagy pozitív szám
x_{jvr}	1 az értéke, ha a j. munka az r. pozícióban került feldolgozásra a v. gépen, különben 0
FT_j	a j. munka befejezési ideje

C.1. táblázat. A modell paraméterei

(C.1) A célfüggvény, a teljes átfutási időt minimalizáljuk, amely nem lehet kisebb mint bármely munka befejezésének ideje.

(C.2) Minden munka esetén, a munkát pontosan egy gép fogja végrehajtani, és ezen a gépen a munka az r-edik pozícióba fog kerülni, valamely r-re.

(C.3) Minden v munka és r pozíció esetén, ide legfeljebb egy munka ütemezhető.

(C.4) Az r-edik pozícióba csak akkor teszünk munkát, ha az $r-1$-edik pozícióban is van munka, egyébként nem (emiatt a munkák folyamatosan lesznek ütemezve a gépeken, tehát pl. olyan nem fordulhat elő, hogy van munka az 1. pozícióban, aztán a 2-ban nincs, de utána a 3-ban meg megint van).

(C.5) Ha valamely v gép esetén valamely j munka közvetlenül az i munka utáni pozícióban van, akkor a j-edik munka befejezési ideje legalább akkora mint az előtte levő (tehát az i munka) befejezési ideje, plusz a j-edik munka megmunkálási ideje azon a gépen.

(C.6) A j-edik munka befejezési ideje legalább akkora mint a v-edik gépen a j-edik munka végrehajtási ideje, ha a munka erre a gépre van ütemezve.

(C.7) A j-edik munka befejezési ideje legalább akkora mint bármely azt közvetlenül megelőző munka befejezési ideje, plusz a j-edik munka megmunkálási ideje.

C.2. GAMS kód

```
sets v   gep indexe   / v0 * v7 /
     j   munka (task) / j0 * j13 /
     r   pozicio (hanyadik a sorrendben a munka valamely gepen) / r1 * r4 /;

Alias (i,j);
Alias (r,q);
Scalar L nagy szam / 100 /;

Table     p(v,j) processing time v-edik gepen a j-edik munka

     j0   j1   j2   j3   j4   j5   j6   j7   j8   j9  j10  j11  j12  j13
v0    2    2    7    6    6    3    2    2    9    8    1    5    3    2
v1   10    6    9   10    8    5    7    8    4    1    1    8    3    7
v2    4    9    6    8    7    2    1    6    1    8    4    3    2    3
v3    9    2    2    3    7    1    5   10    3    7    4    3    3    3
v4    1    6    8    8   10   10   10    8    8    9    8    5   10    2
v5    4    7    1    5    5    9    9    6    6    5    2    7   10    7
v6    1    3    5    4   10    1    2    2    5    5    5   10    3    5
v7   10    5    4    3    6    3    9    7    9    6    8   10    2    6
;

Parameter prec(i,j) ez akkor 1 ha i kozvetlenul megelozi j-t;
;
prec("j4","j11")=1;
prec("j11","j12")=1;
prec("j1","j3")=1;
prec("j3","j7")=1;
prec("j0","j5")=1;


variables
   x(j,v,r) akkor 1 ha a j munka a v-edik gepen az r-edik
   c(j)     a j-edik munka befejezesi ideje
   makespan      a teljes atfutasi ido

positive variable c;
binary variable x;
```

C.1. ábra. Az alapfeladatok közül az #1 feladathoz használt GAMS kód (első részlet)

```
equations
   obj(j)               makespan minimization
   valahol(j)           minden munka valahol van
   csakegy(v,r)         minden helyen legf egy munka van
   monoton(v,r,q)       nincs lyuk az x vektorban
   legalabb(i,j,v,r,q)  nem kezdodhet korabban mint ahogy az elotte levo befejezodik
   elso(j,v)          elso munka vege
   utanna(i,j)        csak a kozvetlenul megelozo utan
;

obj(j)..          makespan  =g= c(j) ;
valahol(j)..      sum(  (v,r) ,  x(j,v,r)   )     =e= 1;
csakegy(v,r)..    sum(  j  ,  x(j,v,r)   )     =l= 1;
monoton(v,r,q)$( ord(r) gt 1 and ord(r)=ord(q)+1 )..
                  sum( j ,  x(j,v,r) )  =l=  sum( j , x(j,v,q) )  ;

legalabb(i,j,v,r,q)$( ord(i)<>ord(j) and ord(r) gt 1 and ord(r) = ord(q)+1 )..
                 c(j)-c(i) + L * ( 2-x(j,v,r)-x(i,v,q) ) =g= p(v,j);

elso(j,v)..        c(j) =g= sum( r , p(v,j) *  x(j,v,r)   )  ;
utanna(i,j)$( prec(i,j)=1)..
                  c(j)-c(i) =g= sum( (v,r) , p(v,j) *  x(j,v,r)   )  ;

model Unrelated  / all / ;
option optcr=0.001;
option optca=0.1;
solve Unrelated using MIP minimizing makespan;
display x.l, c.l, makespan.l;
```

C.2. ábra. Az alapfeladatok közül az #1 feladathoz használt GAMS kód (második részlet)

C.3. Statisztika

Az #1 alapfeladat statisztikai adatai:

- Feltételek száma: 1625
- Változók száma: 239
- Bináris változók száma: 224

```
************************
MODEL STATISTICS

BLOCKS OF EQUATIONS           7     SINGLE EQUATIONS        1,625
BLOCKS OF VARIABLES           3     SINGLE VARIABLES          239
NON ZERO ELEMENTS         6,950     DISCRETE VARIABLES        224
S O L V E      S U M M A R Y

     MODEL   Unrelated           OBJECTIVE  makespan
     TYPE    MIP                 DIRECTION  MINIMIZE
     SOLVER  CPLEX               FROM LINE  74

**** SOLVER STATUS     1 Normal Completion
**** MODEL STATUS      1 Optimal
**** OBJECTIVE VALUE               10.0000

 RESOURCE USAGE, LIMIT          0.145 10000000000.000
 ITERATION COUNT, LIMIT        99    2147483647
--- GAMS/Cplex Link licensed for continuous and discrete problems.
--- GMO setup time: 0.00s
--- Space for names approximately 0.05 Mb
--- Use option 'names no' to turn use of names off
--- GMO memory 0.71 Mb (peak 0.71 Mb)
--- Dictionary memory 0.00 Mb
--- Cplex 22.1.0.0 link memory 0.02 Mb (peak 0.16 Mb)
--- Starting Cplex


--- MIP status (101): integer optimal solution.
--- Cplex Time: 0.13sec (det. 96.36 ticks)

--- Fixing integer variables and solving final LP...


--- Fixed MIP status (1): optimal.
--- Cplex Time: 0.00sec (det. 1.73 ticks)


Proven optimal solution
MIP Solution:           10.000000    (99 iterations, 0 nodes)
Final Solve:            10.000000    (11 iterations)

Best possible:          10.000000
Absolute gap:            0.000000
Relative gap:            0.000000
EXECUTION TIME       =        0.171 SECONDS      5 MB  39.1.0 5f04cd76 LEX-LEG
************************
```

C.3. ábra. Modellstatisztika

A #2 alapfeladat statisztikai adatai:

- Feltételek száma: 21491
- Változók száma: 1009
- Bináris változók száma: 980

```
***********************
MODEL STATISTICS

BLOCKS OF EQUATIONS           7     SINGLE EQUATIONS       21,491
BLOCKS OF VARIABLES           3     SINGLE VARIABLES        1,009
NON ZERO ELEMENTS        89,728     DISCRETE VARIABLES        980


GENERATION TIME      =        0.058 SECONDS     16 MB  39.1.0 5f04cd76 LEX-LEG
GAMS 39.1.0  5f04cd76 May 3, 2022           LEX-LEG x86 64bit/Linux - 07/18/22 07:42:01 Page 4
Unrelated gepek, chain tipusu megelozesi feltetelekkel
Solution Report     SOLVE Unrelated Using MIP From line 76


               S O L V E      S U M M A R Y

     MODEL   Unrelated           OBJECTIVE  makespan
     TYPE    MIP                 DIRECTION  MINIMIZE
     SOLVER  CPLEX               FROM LINE  76

**** SOLVER STATUS     1 Normal Completion
**** MODEL STATUS      1 Optimal
**** OBJECTIVE VALUE                11.0000

 RESOURCE USAGE, LIMIT         28.651 10000000000.000
 ITERATION COUNT, LIMIT    194202    2147483647
--- GAMS/Cplex Link licensed for continuous and discrete problems.
--- GMO setup time: 0.00s
--- Space for names approximately 0.72 Mb
--- Use option 'names no' to turn use of names off
--- GMO memory 7.00 Mb (peak 7.00 Mb)
--- Dictionary memory 0.00 Mb
--- Cplex 22.1.0.0 link memory 0.27 Mb (peak 2.05 Mb)
--- Starting Cplex


--- MIP status (101): integer optimal solution.
--- Cplex Time: 28.60sec (det. 42975.28 ticks)

--- Fixing integer variables and solving final LP...


--- Fixed MIP status (1): optimal.
--- Cplex Time: 0.03sec (det. 22.16 ticks)


Proven optimal solution
MIP Solution:           11.000000    (194202 iterations, 12477 nodes)
Final Solve:            11.000000    (26 iterations)

Best possible:          11.000000
Absolute gap:            0.000000
Relative gap:            0.000000
***********************
```

C.4. ábra. Modellstatisztika

A #5 alapfeladat statisztikai adatai:

- Feltételek száma: 17060
- Változók száma: 784
- Bináris változók száma: 756

```
MIP Presolve eliminated 3 rows and 1 columns.
MIP Presolve modified 29 coefficients.
Reduced MIP has 17060 rows, 784 columns, and 71103 nonzeros.
Reduced MIP has 756 binaries, 0 generals, 0 SOSs, and 0 indicators.

Elapsed time = 28285.39 sec. (40184658.11 ticks, tree = 27376.25 MB) Nodefile size = 25325.67 MB (3760.83 MB after compression)
  15856327 10826479       16.0000    33       18.0000        8.0000 3.12e+08   55.56%
  15861203 10830446        8.0000    34       18.0000        8.0000 3.12e+08   55.56%
  15866386 10833208       16.0000    35       18.0000        8.0000 3.12e+08   55.56%
  15871557 10837503        8.0000    38       18.0000        8.0000 3.12e+08   55.56%
  15876900 10842164       16.0000    43       18.0000        8.0000 3.13e+08   55.56%
  15881760 10843727       16.0000    22       18.0000        8.0000 3.13e+08   55.56%
  15887486 10849539       16.0000    31       18.0000        8.0000 3.13e+08   55.56%
  15892949 10854920       13.0000    37       18.0000        8.0000 3.13e+08   55.56%
  15898162 10856795       16.0000    23       18.0000        8.0000 3.13e+08   55.56%
  15903756 10859865        8.0000    44       18.0000        8.0000 3.13e+08   55.56%
Elapsed time = 28403.70 sec. (40337266.80 ticks, tree = 27464.38 MB) Nodefile size = 25412.79 MB (3772.37 MB after compression)
  15909089 10867410       12.0000    32       18.0000        8.0000 3.13e+08   55.56%
  15914333 10869771        cutoff              18.0000        8.0000 3.13e+08   55.56%
  15920140 10873097        cutoff              18.0000        8.0000 3.14e+08   55.56%
  15925481 10879571       17.0000    15       18.0000        8.0000 3.14e+08   55.56%
  15930738 10883578       17.0000    31       18.0000        8.0000 3.14e+08   55.56%
  15936078 10887447       16.0000    23       18.0000        8.0000 3.14e+08   55.56%
  15940690 10888415       17.0000    26       18.0000        8.0000 3.14e+08   55.56%
  15945668 10893245       17.0000    40       18.0000        8.0000 3.14e+08   55.56%
  15950973 10897715       16.0000    31       18.0000        8.0000 3.14e+08   55.56%
  15956319 10900420       15.0000    28       18.0000        8.0000 3.14e+08   55.56%
Elapsed time = 28526.78 sec. (40489876.48 ticks, tree = 27564.14 MB) Nodefile size = 25513.26 MB (3785.63 MB after compression)
  15962249 10905415       17.0000    21       18.0000        8.0000 3.15e+08   55.56%
  15967608 10912276        9.0000    44       18.0000        8.0000 3.15e+08   55.56%
  15972718 10914269       15.0000    21       18.0000        8.0000 3.15e+08   55.56%
  15978313 10917753       14.0000    25       18.0000        8.0000 3.15e+08   55.56%
  15983151 10923447       17.0000    31       18.0000        8.0000 3.15e+08   55.56%
  15988107 10926011       10.0000    34       18.0000        8.0000 3.15e+08   55.56%
  15993356 10929062       17.0000    30       18.0000        8.0000 3.15e+08   55.56%
  15998437 10933758       14.0000    32       18.0000        8.0000 3.15e+08   55.56%
  16003070 10937938        cutoff              18.0000        8.0000 3.16e+08   55.56%
  16008186 1094086
```

C.5. ábra. Modellstatisztika

A #28 alapfeladat statisztikai adatai:

- Feltételek száma: 412282
- Változók száma: 7105
- Bináris változók száma: 7030

```
IBM ILOG CPLEX   39.1.0 5f04cd76 May 3, 2022           LEG x86 64bit/Linux

--- GAMS/Cplex Link licensed for continuous and discrete problems.
--- GMO setup time: 0.04s
--- Space for names approximately 13.96 Mb
--- Use option 'names no' to turn use of names off
--- GMO memory 123.45 Mb (peak 123.50 Mb)
--- Dictionary memory 0.00 Mb
--- Cplex 22.1.0.0 link memory 4.85 Mb (peak 38.54 Mb)
--- Starting Cplex

MIP Presolve eliminated 5 rows and 1 columns.
MIP Presolve modified 964 coefficients.
Reduced MIP has 412282 rows, 7105 columns, and 1676805 nonzeros.

Elapsed time = 27755.68 sec. (21967597.07 ticks, tree = 3829.80 MB)
Nodefile size = 1768.54 MB (267.17 MB after compression)
 304931 73945        5.0000    54        6.0000        5.0000  7529278   16.67%
 305209 73831        5.0000    58        6.0000        5.0000  7510164   16.67%
 305480 73830        cutoff              6.0000        5.0000  7520743   16.67%
 305777 74057        cutoff              6.0000        5.0000  7545501   16.67%
 306042 73997        cutoff              6.0000        5.0000  7540093   16.67%
 306300 74125        5.0000    46        6.0000        5.0000  7565236   16.67%
 306599 74146        5.0000    47        6.0000        5.0000  7566734   16.67%
 306854 74161        5.0000    44        6.0000        5.0000  7568149   16.67%
 307024 74147        cutoff              6.0000        5.0000  7572724   16.67%
 307165 74137        cutoff              6.0000        5.0000  7574247   16.67%
Elapsed time = 28020.32 sec. (22120840.78 ticks, tree = 3852.49 MB)
Nodefile size = 1745.52 MB (263.79 MB after compression)
 307365 74205        cutoff              6.0000        5.0000  7580852   16.67%
 307613 74361        5.0000    66        6.0000        5.0000  7590516   16.67%
 307807 74461        5.0000    75        6.0000        5.0000  7592071   16.67%
 308001 74626        5.0000    51        6.0000        5.0000  7610370   16.67%
 308263 74682        cutoff              6.0000        5.0000  7612305   16.67%
 308518 74717        5.0000    56        6.0000        5.0000  7614678   16.67%
 308765 74635        5.0000    59        6.0000        5.0000  7623425   16.67%
 309001 74638        cutoff              6.0000        5.0000  7625436   16.67%
 309207 74639        5.0000    55        6.0000        5.0000  7627207   16.67%
 309442 74657        5.0000    55        6.0000        5.0000  7628210   16.67%
Elapsed time = 28247.17 sec. (22273699.88 ticks, tree = 3878.98 MB)
Nodefile size = 1757.91 MB (265.66 MB after compression)
 309693 74840        5.0000    43        6.0000        5.0000  7640942   16.67%
 309906 74978        5.0000    61        6.0000        5.0000  7655617   16.67%
 310106 75026        5.0000    54        6.0000        5.0000  7670229   16.67%
 310351 75085        5.0000    51        6.0000        5.0000  7671620   16.67%
 310596 75004        5.0000    53        6.0000
```

C.6. ábra. Modellstatisztika

D. függelék

Az FU algoritmus paraméter-beállításai az algoritmus különböző verzióiban

D.1. Az FU algoritmus paraméter-beállításai a v1 változat esetén

	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{b20}	r_{b21}	r_{b22}	r_{b23}	r_{b24}	r_{b25}
U120	0	0	0	0	0	0	0	70	70	70	70	70
U250	0	0	0	0	0	0	0	70	70	70	70	70
U500	0	0	0	0	0	0	0	70	70	70	70	70
U1000	0	0	0	0	0	0	0	70	70	70	70	70

D.1. táblázat. A Falkenauer_U osztály paraméter-beállítása

	r_{n20}	r_{n21}	r_{n22}	r_{n23}	r_{n24}	r_{n25}	r_{30}	r_{31}	r_{32}
U120	0	0	0	0	0	0	0	0	0
U250	0	0	0	0	0	0	0	0	0
U500	0	0	0	0	0	0	0	0	0
U1000	0	0	0	0	0	0	0	0	0

D.2. táblázat. A Falkenauer_U osztály paraméter-beállítása

D.2. Az FU algoritmus paraméter-beállításai a v2 változat esetén

	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{b20}	r_{b21}	r_{b22}	r_{b23}	r_{b24}	r_{b25}
U120	0	5	10	15	20	25	0	5	10	15	20	25
U250	0	5	10	15	20	25	0	5	10	15	20	25
U500	0	10	20	30	40	50	0	15	20	30	35	40
U1000	0	5	15	25	35	0	0	15	20	25	30	35

D.3. táblázat. A Falkenauer_U osztály paraméter-beállítása

	r_{n20}	r_{n21}	r_{n22}	r_{n23}	r_{n24}	r_{n25}	r_{30}	r_{31}	r_{32}
U120	0	5	15	30	30	30	0	0	0
U250	0	10	20	30	40	50	0	0	0
U500	0	5	10	25	35	45	0	0	0
U1000	0	20	30	40	50	60	0	0	0

D.4. táblázat. A Falkenauer_U osztály paraméter-beállítása

Irodalomjegyzék

[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. M. Turing. Computing machinery and intelligence. In *Parsing the turing test*, pages 23–65. Springer, 2009.

[5] 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.

[6] 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.

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

[8] J. D. Ullman. The performance of a memory allocation algorithm. *Computer Science Laboratory*, 47, 1971.

[9] M. R. Garey, R. L. Graham, and J. D. Ullman. Worst-case analysis of memory allocation algorithms. In *Proceedings of the fourth annual ACM symposium on Theory of computing*, pages 143–150, 1972.

[10] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on computing*, 3(4):299–325, 1974.

[11] D. Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Research Logistics (NRL)*, 41(4):579–585, 1994.

[12] Gy. Dósa and J. Sgall. First fit bin packing: A tight analysis. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[13] Gy. Dósa and J. Sgall. Optimal analysis of Best Fit bin packing. In *International Colloquium on Automata, Languages, and Programming*, pages 429–441. Springer, 2014.

[14] Gy. Dósa. The tight bound of first fit decreasing bin-packing algorithm is $FFD(L) \leq 11/9 \cdot OPT(L) + 6/9$. In *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11. Springer, 2007.

[15] Gy. Dósa, R. Li, X. Han, and Zs. Tuza. Tight absolute bound for first fit decreasing bin-packing: $FFD(L) \leq 11/9 \cdot OPT(L) + 6/9$. *Theoretical Computer Science*, 510:13–61, 2013.

[16] D. L. Vega, W. Fernandez, and G. S. Lueker. Bin packing can be solved within $1+ \varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[17] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 312–320. IEEE, 1982.

[18] M. Delorme, M. Iori, and S. Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20, 2016.

[19] J. M. De Carvalho. Lp models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273, 2002.

[20] L. Wei, Z. Luo, R. Baldacci, and A. Lim. A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*, 32(2):428–443, 2020.

[21] T. Dokeroglu and A. Cosar. Optimization of one-dimensional bin packing problem with island parallel grouping genetic algorithms. *Computers & Industrial Engineering*, 75:176–186, 2014.

[22] K. Loh, B. Golden, and E. Wasil. Solving the one-dimensional bin packing problem with a weight annealing heuristic. *Computers & Operations Research*, 35(7):2283–2291, 2008.

[23] T. Kucukyilmaz and H. E. Kiziloz. Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem. *Computers & Industrial Engineering*, 125:157–170, 2018.

[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. Brugger, K. F. Doerner, R. F. Hartl, and M. Reimann. Antpacking–an ant colony optimization approach for the one-dimensional bin packing problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 41–50. Springer, 2004.

[26] A. Laurent and N. Klement. Bin packing problem with priorities and incompatibilities using pso: application in a health care community. *IFAC-PapersOnLine*, 52(13):2596–2601, 2019.

[27] M. Abdel-Basset, G. Manogaran, L. Abdel-Fatah, and S. Mirjalili. An improved nature inspired meta-heuristic algorithm for 1-d bin packing problems. *Personal and Ubiquitous Computing*, 22(5):1117–1132, 2018.

[28] Z. Zendaoui and A. Layeb. Adaptive cuckoo search algorithm for the bin packing problem. In *Modelling and implementation of complex systems*, pages 107–120. Springer, 2016.

[29] M. Jain, V. Singh, and A. Rani. A novel nature-inspired algorithm for optimization: Squirrel search algorithm. *Swarm and evolutionary computation*, 44:148–175, 2019.

[30] C. Munien, S. Mahabeer, E. Dzitiro, S. Singh, S. Zungu, and A. E. Ezugwu. Metaheuristic approaches for one-dimensional bin packing problem: A comparative performance study. *IEEE Access*, 8:227438–227465, 2020.

[31] A. Babalik. Implementation of bat algorithm on 2d strip packing problem. In *Intelligent and Evolutionary Systems*, pages 209–218. Springer, 2016.

[32] A. Gherboudj. African buffalo optimization for one dimensional bin packing problem. *International Journal of Swarm Intelligence Research (IJSIR)*, 10(4):38–52, 2019.

[33] J. B. Odili, M. N. M. Kahar, and S. Anwar. African buffalo optimization: a swarm-intelligence technique. *Procedia Computer Science*, 76:443–448, 2015.

[34] M. Gourgand, N. Grangeon, and N. Klement. An analogy between bin packing problem and permutation problem: A new encoding scheme. In *IFIP International Conference on Advances in Production Management Systems*, pages 572–579. Springer, 2014.

[35] H. Feng, H. Ni, R. Zhao, and X. Zhu. An enhanced grasshopper optimization algorithm to the bin packing problem. *Journal of Control Science and Engineering*, 2020, 2020.

[36] E. G. Coffman, J. Csirik, G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: survey and classification. In *Handbook of combinatorial optimization*, pages 455–531. 2013.

[37] T. Yang, F. Luo, J. Fuentes, W. Ding, and C. Gu. A flexible reinforced bin packing framework with automatic slack selection. *Mathematical Problems in Engineering*, 2021, 2021.

[38] T. G. Crainic, F. D. Fomeni, and W. Rei. Multi-period bin packing model and effective constructive heuristics for corridor-based logistics capacity planning. *Computers & Operations Research*, 132:105–308, 2021.

[39] M. Witteman, Q. Deng, and B. F. Santos. A bin packing approach to solve the aircraft maintenance task allocation problem. *European Journal of Operational Research*, 294(1):365–376, 2021.

[40] C. Munien and A. E. Ezugwu. Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications. *Journal of Intelligent Systems*, 30(1):636–663, 2021.

[41] J. Martinovic, N. Strasdat, and M. Selch. Compact integer linear programming formulations for the temporal bin packing problem with fire-ups. *Computers & Operations Research*, 132:105–288, 2021.

[42] 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.

[43] 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.

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

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

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

[47] W. Zhong, Gy. Dósa, and Z. Tan. On the machine scheduling problem with job delivery coordination. *European Journal of Operational Research*, 182(3):1057–1072, 2007.

[48] L. Epstein. On bin packing with clustering and bin packing with delays. *Discrete Optimization*, 41:100–647, 2021.

[49] L. Ahlroth, A. Schumacher, and P. Orponen. Online bin packing with delay and holding costs. *Operations Research Letters*, 41(1):1–6, 2013.

[50] **Gy. Ábrahám and P. Auer and Gy. Dósa and 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**).

[51] 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.

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

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

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

[55] 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.

[56] 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.

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

[58] 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.

[59] 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.

[60] 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.

[61] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[62] A. A. Markov. *The theory of algorithms*, volume 42. Acad. Sci. USSR, Moscow–Leningrad, 1954.

[63] R. Bellman. *Dynamic Programming*. Dover Publications, 1957.

[64] D. Silver. Lectures on reinforcement learning. `https://www.davidsilver.uk/teaching/`, 2015. Hozzáférés: 2021.06.15.

[65] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1989.

[66] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 217–224, 1987.

[67] E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2):127–133, 2005.

[68] 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.

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

[70] Gy. Dósa, H. Kellerer, and Zs. Tuza. Team work scheduling. *MATCOS-16, Middle-European Conference on Applied Theoretical Computer Science*, pages 80–82, 2016.

[71] Gy. Dósa and Zs. Tuza. Multiprofessor scheduling. *Discrete Applied Mathematics*, 234:195–209, 2018.

[72] Gy. Dósa, H. Kellerer, and Zs. Tuza. Restricted assignment scheduling with resource constraints. *Theoretical Computer Science*, 760:72–87, 2019.

[73] P. Stefán and L. Monostori. On the relationship between learning capability and the boltzmann-formula. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 227–236. Springer, 2001.

[74] **Gy. Ábrahám and T. Dulai and 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.

[75] Ábrahám Gyula oktatói weboldala. `https://virt.uni-pannon.hu/index.php/hu/a-tanszekrol/oktatoi-oldalak/184-abraham-gyula`. Hozzáférés: 2021.10.18.

[76] Bin packing benchmarks of homepage unibo. `http://or.dei.unibo.it/library/bpplib`. Hozzáférés: 2021.10.12.

[77] P. Schwerin and G. Wäscher. The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP. *International Transactions in Operational Research*, 4(5-6):377–389, 1997.

[78] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2(1):5–30, 1996.

[79] A. Scholl, R. Klein, and C. Jürgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627–645, 1997.

[80] G. Wäscher and T. Gau. Heuristics for the integer one-dimensional cutting stock problem: A computational study. *Operations-Research-Spektrum*, 18(3):131–144, 1996.

[81] J. E. Schoenfield. Fast, exact solution of open bin packing problems without linear programming. *Draft, US Army Space and Missile Defense Command, Huntsville, Alabama, USA*, 2002.

[82] T. Gschwind and S. Irnich. Dual inequalities for stabilized column generation revisited. *INFORMS Journal on Computing*, 28(1):175–194, 2016.

[83] A. Adamuthe and T. Nitave. Harmony search algorithm with adaptive parameter setting for solving large bin packing problems. *Decision Science Letters*, 9(4):581–594, 2020.

[84] A. L. Brearley, G. Mitra, and H. P. Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical programming*, 8(1):54–83, 1975.

[85] J. A. Tomlin and J. S. Welch. A pathological case in the reduction of linear programs. *Operations Research Letters*, 2(2):53–57, 1983.

[86] J. A. Tomlin and J. S. Welch. Formal optimization of some reduced linear programming problems. *Mathematical programming*, 27(2):232–240, 1983.

[87] E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Mathematical Programming*, 71(2):221–245, 1995.

[88] J. Gondzio. Presolve analysis of linear programs prior to applying an interior point method. *INFORMS Journal on Computing*, 9(1):73–91, 1997.

[89] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[90] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger. Multi-row presolve reductions in mixed integer programming. In *Proceedings of the Twenty-Sixth RAMP Symposium Hosei University, Tokyo, October 16-17*, pages 181–196, 2014.

[91] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020.

[92] P. Gemander, W. Chen, D. Weninger, L. Gottwald, A. Gleixner, and A. Martin. Two-row and two-column mixed-integer presolve using hashing-based pairing methods. *EURO Journal on Computational Optimization*, 8(3):205–240, 2020.

[93] Cs. Mészáros and U. H. Suhl. Advanced preprocessing techniques for linear and quadratic programming. *OR Spectrum*, 25(4):575–595, 2003.

[94] E. L. Johnson, G. L. Nemhauser, and N. W. P. Savelsbergh. Progress in linear programming-based algorithms for integer programming: An exposition. *Informs journal on computing*, 12(1):2–23, 2000.

[95] D. Pisinger and P. Toth. Knapsack problems. In *Handbook of combinatorial optimization*, pages 299–428. Springer, 1998.

[96] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics*, 28(1):59–70, 1990.

[97] S. P. Fekete and J. Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical programming*, 91(1):11–31, 2001.

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

[99] **Gy. Ábrahám and P. Auer and Gy. Dósa and T. Dulai and Zs. Tuza and Ágnes 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**).

[100] **Gy. Ábrahám and Gy. Dósa and T. Dulai and Zs. Tuza and Á. Werner-Stark**. Efficient Pre-Solve Algorithms for the Schwerin and Falkenauer_U Bin Packing Benchmark Problems for Getting Optimal Solutions with High Probability. *Mathematics*, 9(13):1540, 2021 (**IF: 2.542**).

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

[102] **Á. Werner-Stark and 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.

[103] **T. Dulai and 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.

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

[105] **Gy. Ábrahám and T. Dulai and 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.

[106] **Gy. Ábrahám and Á. Werner-Stark**. Emberi tényezők vizsgálata speciális hibafa elemzési módszerrel az üzleti és gyártási folyamatok végrehajtásának hatékonyabbá tétele érdekében. In *Pannon Tudományos Nap*, 2018.

[107] **Gy. Ábrahám and T. Dulai and 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.

[108] **Gy. Ábrahám and T. Dulai and Á. Werner-Stark and Gy. Dósa**. Decision supporting tool for scheduling of production processes considering human factors. In *Proceedings of the Pannonian Conference on Advances in Information Technology*, pages 133–138, 2019.

[109] **Gy. Ábrahám and T. Dulai and Á. Werner-Stark and Gy. Dósa**. A hybridization technique for improving a scheduling heuristic in mass production. In *3rd International Conference on Communication and Computational Technologies*, pages 45–45, 2021.