

Ősz Olivér, mérnök informatikus MSc I. évfolyam
Kovács Balázs, mérnök informatikus MSc, I. évfolyam

AUTOMATIZÁLT NEDVES MARÁSI RENDSZEREK ÜTEMEZÉSE

Témavezető: Hegyháti Máté, tudományos munkatárs

Pannon Egyetem,
Műszaki Informatikai Kar,
Rendszer- és Számítástudományi Tanszék



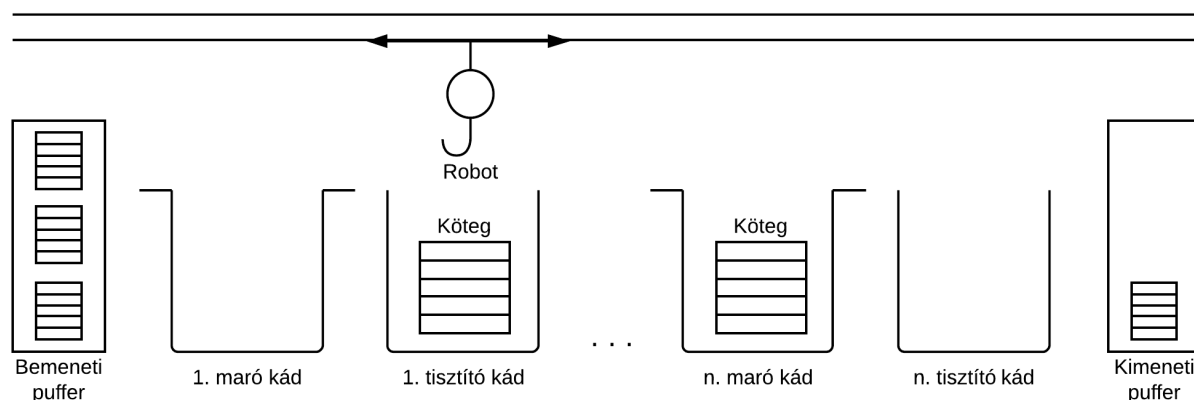
Tartalomjegyzék

1. Bevezetés	2
2. Szakirodalmi áttekintés	3
2.1. Feladatleírás	3
2.2. Szakirodalmi módszerek	5
2.3. Az S-gráf módszertan	6
2.3.1. Ütemezések modellezése S-gráfokkal	6
2.3.2. Makespan minimalizálási algoritmus	8
3. MILP modellek továbbfejlesztése	9
3.1. A precedencia alapú modell gyorsítása	9
3.2. A hibrid time slot alapú modell kiterjesztése	12
3.3. A MILP fejlesztések vizsgálata	14
4. AWS ütemezés az S-gráf keretrendszerrel	17
4.1. AWS ütemezési feladatok modellezése S-gráffal	17
4.1.1. Limited wait taszkok ábrázolása	17
4.1.2. Szemléltető példa	18
4.1.3. Üres robot mozgás ábrázolása	19
4.2. Az S-gráf alapú megoldás implementációja	21
4.2.1. A megoldó architektúrája	21
4.2.2. Az AWS receptgráf létrehozása	22
4.2.3. Limited wait támogatás	23
4.2.4. Üres robot mozgás támogatása	24
5. Az S-gráf alapú módszer gyorsítása	28
5.1. Job segédélek	28
5.2. Módosított élbehúzás egy robot esetén	30
5.3. Kezdeti megoldás keresése JAT heurisztikával	32
5.3.1. Job-at-a-time heurisztika implementálása	32
5.3.2. Előmegoldó interfész	34
6. Összehasonlító vizsgálatok	35
6.1. Üres robot mozgás nélkül	35
6.2. Üres robot mozgás figyelembevételével	37
7. Összefoglalás	40
A. Függelék: Job-at-a-time heurisztika	44
B. Függelék: Makespan minimalizálás	45

1. Bevezetés

Az integrált áramkörök a 60-as években kezdtek elterjedni, és napjainkban megtalálhatóak a mindenki által használt elektronikai eszközökben és az iparban alkalmazott berendezésekben. Emiatt elengedhetetlenné vált a tömeges gyártásuk optimalizálása. Az elkészítés első lépése az alapanyagul szolgáló félvezetőanyag (általában szilícium, ritkábban germánium) megmunkálása. Ez a folyamat lépések bonyolult sorozatából áll, de 3 fő fázisa az anyagfelvitel, a maratás és a litográfia. A maratás egyik típusa, a nedves maratás (wet-etching) az utóbbi években fokozott figyelmet kapott az ütemezési, optimalizálási szakirodalomban.

A nedves maratás célja a félvezető felületéről irányított módon eltávolítani a rétegeket. A mintázat kialakításáról egy maszk gondoskodik, melyet a maratás előtt kell kialakítani a felületen. A félvezető szeleteket különböző maró vegyületeket tartalmazó kádakba merítik, és pontosan meghatározott ideig hagyják benne, majd a következő maratási lépés előtt víz vagy deionizáló folyadék segítségével tisztítják meg.



1.1. ábra. Az AWS felépítése

A marási időket szigorúan be kell tartani, különben a szeletek megsérülnek, ezért az iparban teljesen automatizált módon végzik a folyamatot. Az ilyen rendszerek elterjedt elnevezése az Automated Wet-etch Station (a továbbiakban AWS). Az 1.1. ábrán látható az AWS felépítése: marató és tisztító kádak váltakozó sora és egy vagy több szállító robot található benne. A szeleteket nem egyesével kezelik, hanem egyforma megmunkálásra szánt darabokból álló kötegekben.

Jelen dolgozat célja az AWS ütemezésére használt szakirodalmi módszerek továbbfejlesztésének és egy új módszernek a bemutatása. A 2.1. alfejezet az ütemezési feladatot, míg a 2.2. a szakirodalomban található megoldásokat és az S-gráf keretrendszert mutatja be. A 3. fejezet ismerteti a matematikai programozási módszereken végzett fejlesztéseket. Az S-gráf alapú AWS ütemezési módszer elméleti hátterét, megvalósítását és az alkalmazott gyorsításokat a 4. fejezet tartalmazza. A 6. fejezetben található a módszerek összehasonlító vizsgálata.

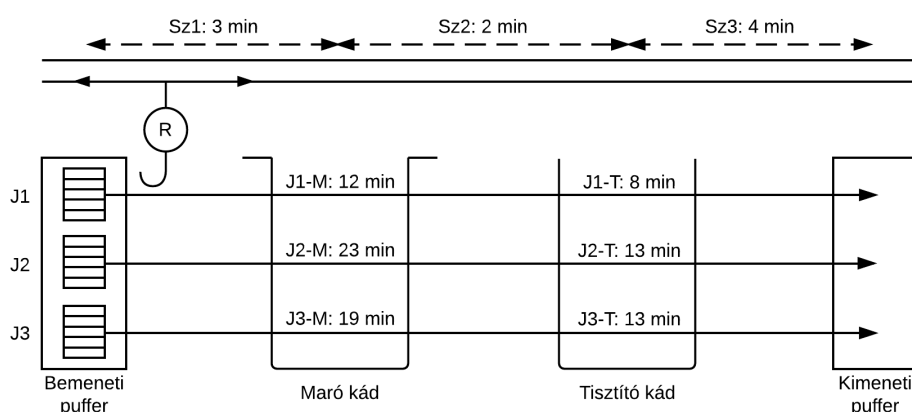
2. Szakirodalmi áttekintés

Az AWS ütemezési feladatot a 2.1. alfejezet mutatja be. A problémára a szakirodalomban heurisztikus, valamint kevert egészértékű lineáris- (MILP – Mixed Integer Linear Programming) és korlátprogramozási (CP – Constraint Programming) módszerek jelentek meg, melyek a 2.2. alfejezetben kerülnek bemutatásra. Az általunk kifejlesztett új módszer a kombinatorikus S-gráf keretrendszerre épül, melyet a 2.3. alfejezet mutat be.

2.1. Feladatleírás

Az AWS ütemezési feladatban adottak az AWS erőforrásainak adatai és a legyártandó termékek paramétereit tartalmazó receptek. Minden köteg egy külön terméknek tekintendő. A bemeneti adatok szemléletes megjelenítése látható a ?? ábrán. A bemenetben szereplő paraméterek listája:

- Termékek száma
- Kádak halmaza
- Kádak távolsága (időben)
- Robotok halmaza
- Robotok mozgási zónája (szakaszokra osztott sínek esetén)
- Termékek receptjei:
 - Gyártási lépések (taszkok) sorozata
 - Az egyes taszkokhoz használható kádak halmaza
 - Taszkok végrehajtási és maximális várakozási ideje

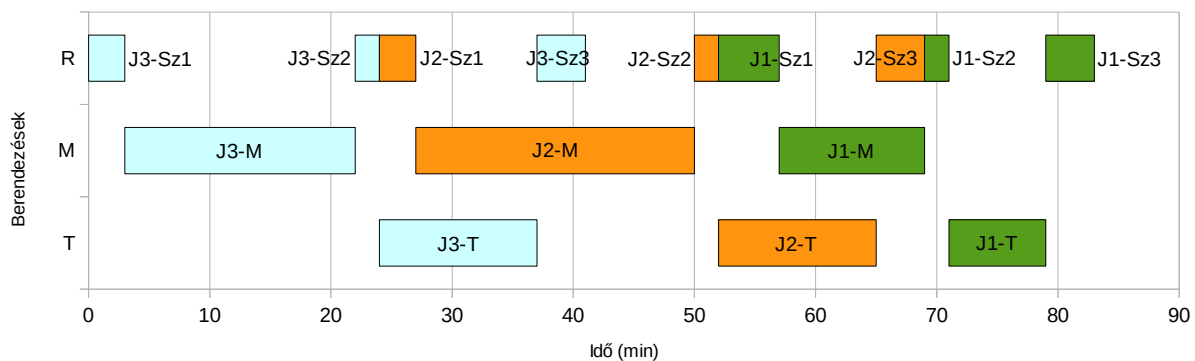


2.1. ábra. A bemenet grafikus reprezentációja

A megoldás során meg kell határozni az ütemezést leíró adatokat:

- Az egyes taszkokat és szállításokat végrehajtó kádakat ill. robotokat
- A taszkok és szállítások kezdési és befejezési időpontját

Az ütemezés célja a teljes gyártási idő (makespan) minimalizálása. A makespan megfelel az utolsó szállítás befejezési idejének. A 4.1. ábrán bemutatott feladat optimális ütemezését a 4.4. ábrán látható Gantt-diagram ábrázolja, amely egy gyakran használt mód ütemezések ábrázolására. A vízszintes tengely az időnek felel meg, a függőleges tengelyen pedig az erőforrások szerepelnek. A téglalapok szemléltetik a taszkokat, azok időtartamát, és az általuk használt erőforrást.



2.2. ábra. A példa optimális ütemezése

Az AWS ütemezési feladatok paramétereik alapján különböző bonyolultságú feladatosztályokba sorolhatók. A szakirodalomban kétféle feladatosztály jelenik meg: a permutation flow-shop (PFS) és az általánosabb hoist scheduling problem (HSP)[1].

A PFS egyszerűbb felépítésű receptjére az alábbiak vonatkoznak:

- A termékek ugyanazokon a taszkokon haladnak végig, csak a végrehajtási időkből különböznek
- Minden taszkot csak egy adott kádban lehet végrehajtani
- A kádak nem ismétlődnek egy recepten belül

A fentiekből következik, hogy a termékek sorrendje minden kád esetén egyforma lesz. Ezt a tulajdonságot több módszer is kihasználja a gyorsabb megoldáshoz. A HSP-ből a fenti korlátozások hiányoznak, így a termékek teljesen különböző útvonalon haladhatnak a rendszerben.

A várakozási idő alapján is két megfogalmazás található a szakirodalomban. A PFS típusú AWS-ekben ZW és UW taszkok váltakoznak a marási és tisztítási fázisoknak megfelelően.

ZW (Zero Wait): Nincs várakozási idő, a végrehajtási idő után azonnal ki kell venni a terméket a kádból

UW (Unlimited Wait): Korlátlan várakozási idő, a termék bármeddig a kádban maradhat

A HSP feladatosztályhoz LW (Limited Wait) korlátozást alkalmaztak. Ez korlátos várakozási időt jelent, amely 0 érték esetén megegyezik a ZW korláttal. Az UW taszkok megvalósítása pedig egy elegendően nagy korlát beállításával történik.

A robotok mozgására is különböző korlátozások léteznek. A PFS feladatokban nincs korlátozva, a robotok párhuzamos síneken mozognak, így megelőzhetik egymást. A közös sín kezelése egy nagyon bonyolult feladat lenne, az AWS ütemezés során ezzel nem foglalkozik a szakirodalom. A HSP feladatokban köztes megoldásként a sítet diszjunkt szakaszokra (robot mozgási zónákra) osztják, és minden robot csak a saját zónájában közlekedhet.

2.2. Szakirodalmi módszerek

Az első megoldási módszert Geiger és társai[2] publikálták. Kezdeti megoldás keresésére két heurisztikát is bemutatnak, a megoldástér bejárását pedig egy Tabu keresésre épülő algoritmussal oldották meg. A módszer még nem volt képes több robot kezelésére.

Bhushan és Karimi[3] új algoritmusokat adtak meg kezdeti megoldás keresésére, a terméksorrendek bejárására és a robot ütemezésére. Módszerük jobb kezdeti megoldást adott, de lassabb volt, ezért nem bizonyult elég hatékonynak. A terméksorrendek bejárását szimulált hűtéssel végezték, az egyes sorrendekre a robot ütemezését egy mohó algoritmussal határozták meg. Job-at-a-time (JAT) heurisztikájukat az S-gráf alapú módszer gyorsítására is felhasználtuk. Az algoritmus előre meghatározott terméksorrend esetén egyetlen robotot feltételezve ad egy mohó módszerrel felépített megvalósítható ütemezést. Az algoritmus leírása megtalálható az A. Függelékben.

Rövidtávú ütemezési feladatok optimális megoldására[4] többféle matematikai programozási módszer is létezik. Az AWS-ek sajátosságait a precedencia és a time slot alapú modellek tudják a legjobban kihasználni. Bhushan és társai[5, 6] time slotokat használtak, melyekhez hozzárendelve a termékeket, eldönthető azok sorrendje. Az egyes kádakban meg kell határozni a time slotok kezdeti és befejezési idejét. Nagyobb méretű példákra a matematikai programozás nem képes belátható időben megoldást adni. Ilyen esetekben érdemes lehet relaxálni a problémát, például a szerzők által bemutatott CURM módszerrel: a robotok figyelmen kívül hagyásával meghatározni az ütemezést, majd az így kapott terméksorrendet rögzítve megoldani a MILP modellt.

Aguirre és társai[7, 8] általános precedencia alapú megközelítést alkalmaztak, valamint több robot kezelésére is képes modellt mutattak be. A kétlépcsős heurisztikát is továbbfejlesztették (rCURM). Az általános precedencia azt jelenti, hogy bináris döntési változók jelölik, hogy két taszk közül melyik kezdődik előbb.

Castro és társai[9] hibrid time slotokra épülve fejlesztettek ki egy új modellt. A módszer a berendezés-specifikus időpontokhoz (unit-specific time points) hasonlít, de mivel permutation flow-shop feladatokra készült, a terméksorrendet csak egyszer kell meghatározni, nem minden berendezésben. A robotok ütemezésénél precedencia változókat is használtak – innen a hibrid jelző. A precedencia alapú modellek relaxált megoldásánál hátrányt jelentő big-M korlátozásait elkerülve, egy robot esetén nagyságrendekkel gyorsabb. Több robot esetén big-M korlátozásokra van szükség, de teljesítménye így is jelentősen meghaladja más módszerekét. Ez volt a motiváció a módszer feladatosztályának kiterjesztésére, melyet a 3.2. alfejezet tartalmaz.

Korlátprogramozási módszerek is megjelentek AWS ütemezésre. Zeballos és társai[10] megadtak egy CP modellt a feladatra, és kifejlesztettek egy keresési stratégiát, ami nagyobb méretű példákra is gyorsan talál megoldást és jól közelíti az optimumot. Novas és Henning[11] szintén CP alapú módszert publikáltak, azonban a fő céljuk a robot üres mozgásának a figyelembevétele volt. Modelljük csak egy robot ütemezésére képes, de azt akár on-line is meg tudja tenni, így akár ipari méretű feladatokat időben részekre bontva is meg lehet vele oldani vagy folyamatos gyártást ütemezni vele hosszútávon.

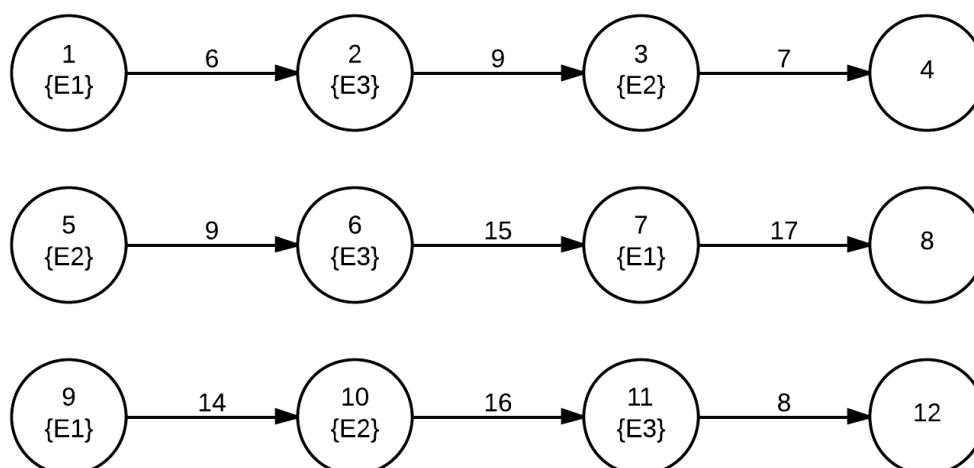
Aguirre és társai[1] kiterjesztették a precedencia alapú MILP modellt a 2.1. fejezetben ismertetett általánosabb feladatosztályra és az üres robot mozgás kezelésére. A megvalósításhoz közvetlen és általános precedenciát is használtak a korlátozásokban. A közvetlen precedencia változók abban különböznek az általánosaktól, hogy értékük csak akkor 1, ha a két taszk *közvetlenül* egymás után kezdődik. A modell részletesebb bemutatása megtalálható a módszer gyorsításával foglalkozó 3.1. fejezetben.

2.3. Az S-gráf módszertan

Az S-gráf keretrendszert[12] szakaszos üzemű gyártórendszerek ütemezésére fejlesztették ki. A feladat sajátosságait kihasználva gyorsabban megtalálhatja az optimális megoldást, mint az általánosabb, MILP modelleket használó módszerek[4]. Hátránya a MILP modellekhez képest az új korlátozások, célfüggvények vagy más funkciók beépítésének nehézsége. A módszertan két fő része az ütemezések modellezésére használható S-gráf és az optimális ütemezés megtalálására szolgáló branch-and-bound algoritmus.

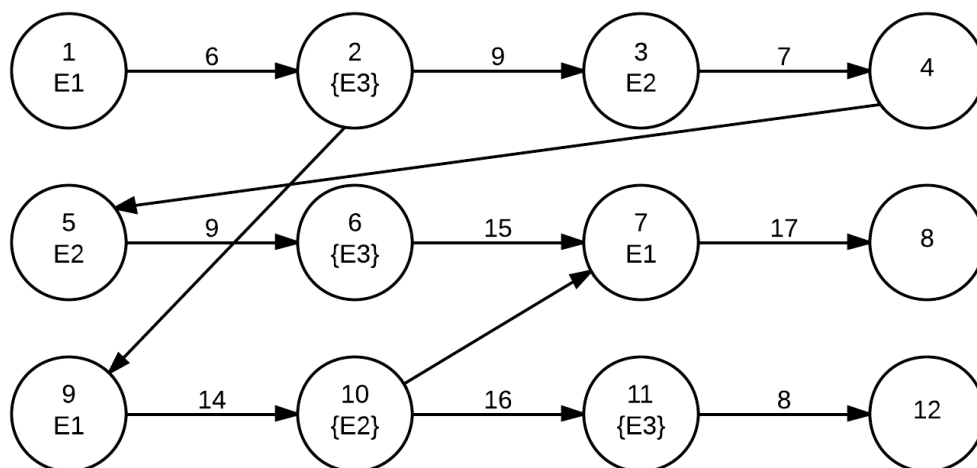
2.3.1. Ütemezések modellezése S-gráfokkal

Az S-gráf egy irányított, súlyozott gráf, ahol a csúcsok a taszkokat és végtermékeket, az ezeket összekötő élek pedig a köztük lévő precedenciát jelölik. Egy él súlya alsó korlát a két végpontja által reprezentált esemény kezdési idejének különbségére. A gyártási receptből előállított kezdeti S-gráfot receptgráfnak, az éleit receptéleknek nevezzük. A receptgráf csúcsaihoz berendezéshalmazok tartoznak, melyek a taszk végrehajtásához használható berendezéseket tartalmazzák. A 2.3. ábra egy receptgráfra mutat példát.

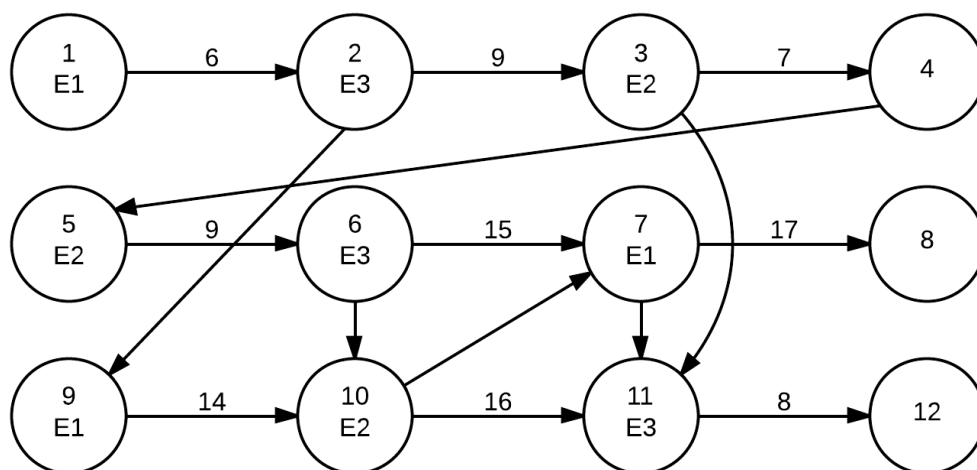


2.3. ábra. Példa receptgráf

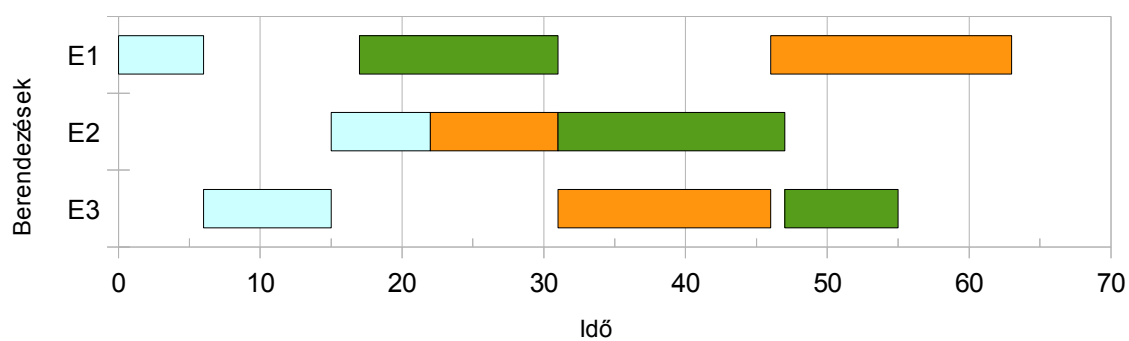
Az ütemezés során minden taszk halmazából ki kell választani egy berendezést, és az azonos berendezéshez tartozó taszkok sorrendjét eldönteni úgynevezett ütemezési élek behúzásával. Ennek módját a köztes tárolóra vonatkozó korlátozások határozzák meg. UIS (Unlimited Intermediate Storage) esetben a két taszk csúcsa között kell behúzni a korábbi taszk végrehajtási idejével (+ átállási idő, ha van) súlyozva. NIS esetén a korábbi taszkot a receptben rákövetkező taszkjának (vagy taszkjainak) csúcsából a későbbi taszkéba kell behúzni 0 (vagy átállási idő) súllyal. A 2.4. ábra egy NIS típusú S-gráfot mutat be. A taszkok között lehet UIS és NIS vegyesen is, ekkor a korábbi taszk tárkorlátja határozza meg az élbehúzás módját. Ha minden taszkra megtörtént a berendezés kiválasztása és az ütemezési élek beállítása, egy ütemezési gráfnak nevezett S-gráf keletkezik, mely egyértelműen megfeleltethető egy ütemezésnek. A szemléltető példa egy (szuboptimális) ütemezési gráfja a 2.5., a belőle generált Gantt-diagram a 2.6. ábrán látható.



2.4. ábra. A példa egy S-gráfja



2.5. ábra. A példa egy ütemezési gráfja



2.6. ábra. Az ütemezés Gantt-diagramja

2.3.2. Makespan minimalizálási algoritmus

A receptgráfból minden ütemezési gráf elérhető ütemezési élek hozzáadásával és a receptélek súlyának esetleges módosításával. Ezt a branch-and-bound algoritmus végzi, amely két részből áll: a BRANCHING és BOUNDING eljárásokból, melyeket a MAIN eljárás vezérel.

Makespan minimalizálás esetén egy részben ütemezett S-gráfban a leghosszabb irányított út össze-súlya alsó korlátja a belőle levezethető ütemezéseknek. A BOUNDING legfontosabb feladata a leghosszabb út megkeresése, de előbb meg kell vizsgálni, hogy körmentes-e a gráf. Amennyiben az S-gráfban irányított kör található, akkor az adott ütemezés nem megvalósítható, így az ilyen gráfok eldobásra kerülnek. A pozitív súlyú kör azt jelenti, hogy egy taszk a kezdési ideje után kell, hogy kezdődjön. A 0 össze-súlyú körök is megvalósíthatatlan ütemezést jeleznek, a NIS tárkorlátnál nem megengedett cross-transzfer: két vagy több berendezés között egyszerre kell helyet cserélniük a bennük lévő anyagoknak, de ez köztes tároló nélkül lehetetlen [13, 14].

A felső korlátot egy megtalált ütemezés makespanje adja, így a szétválasztás és korlátozás keresési módszereknél megszokott módon az ennél nagyobb alsó korláttal rendelkező részütemezéseket sem vizsgálja tovább az algoritmus.

A szétválasztás során az ütemezési döntéseket többféleképpen is meghozhatjuk. Ezek közül a berendezés alapú [12], illetve a taszk alapú [15] algoritmusok jelentek meg a szakirodalomban. A berendezés alapú szétválasztás kiválaszt egy berendezést, aminek a végrehajtási sorának végéhez hozzáfűz egy még nem ütemezett taszkot, amit végre tud hajtani. Minden gyermekben más taszkot fűz hozzá. A taszk alapú algoritmus egy még nem ütemezett taszkot választ ki, amelyet egy berendezés végrehajtási listájába beszúr, minden gyermekben más helyre.

A MAIN és a BRANCHING eljárás berendezés alapú változatának pszeudokódja megtalálható a B. Függelékben.

3. MILP modellek továbbfejlesztése

A matematikai programozási módszereket éveken át tökéletesítették a nagyobb teljesítmény érdekében, így vizsgálatuk elősegítheti az S-gráf alapú módszer gyorsítását is. Különösen a precedencia alapú modelleket érdemes tanulmányozni, melyekben az általános precedencia változók az S-gráf ütemezési éleinek feleltethetők meg. A publikált modellek implementációja után azok továbbfejlesztése is céljaink között szerepelt.

A szakirodalmi módszerek közül két meghatározó fontosságú MILP modellt részletesebben is megvizsgáltunk, és tovább is fejlesztettünk[19]. Az egyik Aguirre és társai[1] precedencia alapú módszere, mely a legáltalánosabb feladatosztály megoldásával emelkedik ki a többi módszer közül. A cél a módszer gyorsítása volt az általános feladatosztály megtartása mellett. A kifejlesztett gyorsítást a 3.1. alfejezet mutatja be.

A másik továbbfejlesztett módszer Castro és társai[9] hibrid time slot alapú modellje volt, amely kizárólag egy szűkebb feladatosztályra alkalmazható, de teljesítménye kimagasló a módszerek között. A cél a modell bővítése volt a számítási igény emelkedésének elkerülésével. A kiterjesztés leírása a 3.2. alfejezetben található.

A fejlesztések hatékonyságát empirikus vizsgálatokkal igazoltuk. Megfigyeléseinket a 3.3. alfejezet mutatja be.

3.1. A precedencia alapú modell gyorsítása

A modell számos módon kiterjesztette a korábbi módszerek által kezelt feladatosztályt.

A szállításokhoz rendelhető robothalmazok és LW végrehajtási idők nem befolyásolták lényegesen a feladat nehézségét. A ZW korlátok lazítása LW-re egyes esetekben könnyebb feladatot eredményezhet.

A több identikus berendezés használata már nagyobb nehézséget jelent, hiszen így nem garantált minden berendezésben ugyanaz a terméksorrend. Ez főleg azon módszereket érinti, ahol kihasználták a feladat ezen tulajdonságát, de a precedencia alapú modell képes kezelni a kiterjesztést.

A nagyobb problémát a robotok üres mozgása jelenti. Ennek időtartama ($\pi_{i,s}^{free}$) attól függ, hogy honnan hova kell mennie, de ezek a helyek előre nem ismertek.

A következő egyenletek a modell időzítésért felelős korlátozásai. A dolgozatban csak a gyorsítás során módosított korlátozások szerepelnek, a teljes modell és a jelölések részletes magyarázata megtalálható az eredeti cikkben[1].

$$Y_{i,i',s,s'} = \begin{cases} 1, & \text{ha } i, s \text{ szállítást } i', s' \text{ után végzi ugyanaz a robot} \\ 0, & \text{különben} \end{cases} \quad (3.1)$$

$$q_{i,s,r} = \begin{cases} 1, & \text{ha } i, s \text{ szállítást } r \text{ robot végzi} \\ 0, & \text{különben} \end{cases} \quad (3.2)$$

$$TS_{i,s} \quad i, s \text{ végrehajtási ideje} \quad (3.3)$$

$$\pi_{i,s}^{load} \quad i, s \text{ szállítási ideje} \quad (3.4)$$

$$TS_{i,s} \geq TS_{i',s'} + \pi_{i,s}^{load} + \pi_{i',s'}^{free} - M_T(1 - Y_{i,i',s,s'}) - M_T(2 - q_{i,s,r} - q_{i',s',r}) \\ \forall i, i' \in I : (i \geq i'), s \in S_i, s' \in S_{i'} : (i, s) \neq (i', s'), r \in R_{i,s} \cap R_{i',s'} \quad (3.5)$$

$$TS_{i',s'} \geq TS_{i,s} + \pi_{i',s'}^{load} + \pi_{i,s}^{free} - M_T(Y_{i,i',s,s'}) - M_T(2 - q_{i,s,r} - q_{i',s',r}) \\ \forall i, i' \in I : (i \geq i'), s \in S_i, s' \in S_{i'} : (i, s) \neq (i', s'), r \in R_{i,s} \cap R_{i',s'} \quad (3.6)$$

A szerzők úgy oldották meg a problémát, hogy az általános precedencia változók ($Y_{i,i',s,s'}$) alapján kiszámolták az egyes szállítások pozícióját ($Pos_{i,s,r}$) a végrehajtási sorban.

$$Pos_{i,s,r} \geq Pos_{i',s',r} + 1 - M_T(1 - Y_{i,i',s,s'}) - M_T(2 - q_{i,s,r} - q_{i',s',r}) \\ \forall i, i' \in I : (i \geq i'), s \in S_i, s' \in S_{i'} : (i, s) \neq (i', s'), r \in R_{i,s} \cap R_{i',s'} \quad (3.7)$$

$$Pos_{i',s',r} \geq Pos_{i,s,r} + 1 - M_T(Y_{i,i',s,s'}) - M_T(2 - q_{i,s,r} - q_{i',s',r}) \\ \forall i, i' \in I : (i \geq i'), s \in S_i, s' \in S_{i'} : (i, s) \neq (i', s'), r \in R_{i,s} \cap R_{i',s'} \quad (3.8)$$

$$Pos_{i,s,r} \leq \sum_{i' \in I} \sum_{s' \in S_{i'}} q_{i',s',r} \quad \forall i \in I, s \in S_i, r \in R_{i,s} \quad (3.9)$$

$$q_{i,s,r} \leq Pos_{i,s,r} \leq M_T \cdot q_{i,s,r} \quad \forall i \in I, s \in S_i, r \in R_{i,s} \quad (3.10)$$

A pozíciók különbségéből egy újabb változót hoztak létre ($K_{i,i',s,s',r}$), amely egyértelműen meghatározta, hogy két szállítást közvetlenül egymás után hajtja-e végre ugyanaz a robot.

$$K_{i,i',s,s',r} = 0 \quad \text{ha } i, s \text{ szállítást közvetlenül } i', s' \text{ szállítás után végzi } r \quad (3.11)$$

$$K_{i,i',s,s',r} = Pos_{i,s,r} - Pos_{i',s',r} - 1 + M_T(1 - Y_{i,i',s,s'}) + M_T(2 - q_{i,s,r} - q_{i',s',r}) \\ \forall i, i' \in I : (i \geq i'), s \in S_i, s' \in S_{i'} : (i, s) \neq (i', s'), r \in R_{i,s} \cap R_{i',s'} \quad (3.12)$$

$$K_{i',i,s',s,r} = Pos_{i',s',r} - Pos_{i,s,r} - 1 + M_T(Y_{i,i',s,s'}) + M_T(2 - q_{i,s,r} - q_{i',s',r}) \\ \forall i, i' \in I : (i \geq i'), s \in S_i, s' \in S_{i'} : (i, s) \neq (i', s'), r \in R_{i,s} \cap R_{i',s'} \quad (3.13)$$

A robot üres mozgása az az idő lesz, ami a két közvetlen egymást követő szállítás közül az első érkezési helye és a második kiindulási helye közti mozgáshoz szükséges ($\pi_{j,j'}^{abs}$). A $\pi_{i,i',s,s'}^{seq-dep}$ a több választható kád miatt szükséges köztes változó.

$$w_{i,s,j} = \begin{cases} 1, & \text{ha } i,s \text{ taszk } j \text{ kádban hajtódik végre} \\ 0, & \text{különben} \end{cases} \quad (3.14)$$

$$\begin{aligned} \pi_{i,s}^{free} &\geq \pi_{i,i',s,s'}^{seq-dep} - M_T(K_{i,i',s,s',r}) - M_T(2 - q_{i,s,r} - q_{i',s',r}) \\ \forall i, i' \in I, s \in S_i, s' \in S_{i'} : (i, s) &\neq (i', s'), r \in R_{i,s} \cap R_{i',s'} \end{aligned} \quad (3.15)$$

$$\begin{aligned} \pi_{i,i',s,s'}^{seq-dep} &\geq \pi_{j,j'}^{abs} - M_T(2 - w_{i,s-1,j} - w_{i',s',j'}) \\ \forall i, i' \in I, s \in S_i : (s > 1), s' \in S_{i'} : (i, s) &\neq (i', s'), j \in J_{i,s-1} \cap J_{i',s'} \end{aligned} \quad (3.16)$$

Az általános precedencia modellből nehéz megállapítani a közvetlen precedencia kapcsolatokat. A segédváltozók bevezetése és a további korlátozások nagymértékben lelassítják a modell megoldását.

A gyorsításra egy olyan megoldás került kifejlesztésre, amely nem igényli új változók vagy korlátozások bevezetését. A robotok üres mozgását az általános precedencia segítségével is meg lehet határozni.

Az eredeti megoldás arra a koncepcióra épült, hogy a mozgási idő a két közvetlen egymás utáni szállítás közti váltáshoz szükséges idő. Ezt csak közvetlen precedencia változókkal lehet meghatározni. A probléma kikerülhető, ha a korlátozást így fogalmazzuk meg:

Bármely két szállítás között eltelt időnek legalább akkorának kell lennie, mint amennyi a korábbi érkezési helyétől a későbbi kiindulási helyéhez vezető mozgáshoz szükséges.

Ez a korlátozás kihasználja az általános precedencia előnyeit és azt a megállapítást, hogy a robotnak akkor is legalább annyi időre van szüksége két szállítás között, ha közben más szállításokat is elvégez, mint ha nem.

Az új korlátozások:

$$\begin{aligned} Ts_{i,s} &\geq Ts_{i',s'} + \pi_{i,s}^{load} + \pi_{j,j'}^{abs} - M_T(1 - Y_{i,i',s,s'}) \\ &\quad - M_T(2 - w_{i,s-1,j} - w_{i',s',j'}) - M_T(2 - q_{i,s,r} - q_{i',s',r}) \\ \forall i, i' \in I : (i > i'), s \in S_i : (s > 1), s' \in S_{i'}, j \in J_{i,s-1}, j' \in J_{i',s'}, r \in R_{i,s} \cap R_{i',s'} \end{aligned} \quad (3.17)$$

$$\begin{aligned} Ts_{i',s'} &\geq Ts_{i,s} + \pi_{i',s'}^{load} + \pi_{j,j'}^{abs} - M_T(Y_{i,i',s,s'}) \\ &\quad - M_T(2 - w_{i,s,j} - w_{i',s'-1,j'}) - M_T(2 - q_{i,s,r} - q_{i',s',r}) \\ \forall i, i' \in I : (i > i'), s \in S_i, s' \in S_{i'} : (s' > 1), j \in J_{i,s}, j' \in J_{i',s'-1}, r \in R_{i,s} \cap R_{i',s'} \end{aligned} \quad (3.18)$$

A gyorsítás mellett a modell javításra is került. Castro és társai[9] a korábbi precedencia alapú modell[7] hibájára hívták fel a figyelmet és közölték javításukat. A hiba az új, általános modellben is megtalálható, viszont az eltérő korlátozások miatt a javítást is újra kellett fogalmazni.

Az eredeti korlátozások:

$$X_{i,i',s,s'} = \begin{cases} 1, & \text{ha } i,s \text{ taszk } i',s' \text{ után következik} \\ 0, & \text{különben} \end{cases} \quad (3.19)$$

$$\begin{aligned} Ts_{i,s} &\geq Tf_{i',s'} + \pi_{i,s}^{load} + \pi_{i',s'+1}^{load} - M_T(1 - X_{i,i',s,s'}) - M_T(2 - w_{i,s,j} - w_{i',s',j}) \\ \forall i, i' \in I : (i > i'), s \in S_i, s' \in S_{i'} : (s' \neq L_{i'}), j \in J_{i,s} \cap J_{i',s'} \end{aligned} \quad (3.20)$$

$$\begin{aligned} Ts_{i',s'} &\geq Tf_{i,s} + \pi_{i',s'}^{load} + \pi_{i,s+1}^{load} - M_T(X_{i,i',s,s'}) - M_T(2 - w_{i,s,j} - w_{i',s',j}) \\ \forall i, i' \in I : (i > i'), s \in S_i : (s \neq L_i), s' \in S_{i'}, j \in J_{i,s} \cap J_{i',s'} \end{aligned} \quad (3.21)$$

A fenti korlátozások az azonos kádban végrehajtott taszkok időzítéséért felelősek. Két taszk közül a korábbi befejezése után az elkészült anyagot a következő kádba kell szállítani, mielőtt a jelenlegi kádba lehet szállítani a következő adagot. Ebből ered, hogy a későbbi taszk legalább a két szállítás idejének összegével a korábbi taszk befejezése után kezdődhet. Egy robot esetén ez a korlátozás megfelelő, hiszen a robotnak először ki kell ürítenie a berendezést, majd betöltenie az új adagot. Több robot esetén azonban a két szállítás párhuzamosan is történhet. A javított korlátozásokban elkülönülnek az esetek, az első kettő (3.22, 3.23) az általános esetre, az utolsó kettő (3.24, 3.25) csak az azonos robothoz rendelt szállításokra vonatkozik.

$$Ts_{i,s} \geq T f_{i',s'} - M_T(1 - X_{i,i',s,s'}) - M_T(2 - w_{i,s,j} - w_{i',s',j})$$

$$\forall i, i' \in I : (i > i'), s \in S_i, s' \in S_{i'} : (s' \neq L_{i'}), j \in J_{i,s} \cap J_{i',s'} \quad (3.22)$$

$$Ts_{i',s'} \geq T f_{i,s} - M_T(X_{i,i',s,s'}) - M_T(2 - w_{i,s,j} - w_{i',s',j})$$

$$\forall i, i' \in I : (i > i'), s \in S_i : (s \neq L_i), s' \in S_{i'}, j \in J_{i,s} \cap J_{i',s'} \quad (3.23)$$

$$Ts_{i,s} \geq T f_{i',s'} + \pi_{i,s}^{load} + \pi_{i',s'+1}^{load} - M_T(1 - X_{i,i',s,s'})$$

$$- M_T(2 - w_{i,s,j} - w_{i',s',j}) - M_T(2 - q_{i,s,r} - q_{i',s'+1,r})$$

$$\forall i, i' \in I : (i > i'), s \in S_i, s' \in S_{i'} : (s' \neq L_{i'}), j \in J_{i,s} \cap J_{i',s'}, r \in R_{i,s} \cap R_{i',s'+1} \quad (3.24)$$

$$Ts_{i',s'} \geq T f_{i,s} + \pi_{i,s}^{load} + \pi_{i',s'+1}^{load} - M_T(X_{i,i',s,s'})$$

$$- M_T(2 - w_{i,s,j} - w_{i',s',j}) - M_T(2 - q_{i,s+1,r} - q_{i',s',r})$$

$$\forall i, i' \in I : (i > i'), s \in S_i : (s \neq L_i), s' \in S_{i'}, j \in J_{i,s} \cap J_{i',s'}, r \in R_{i,s+1} \cap R_{i',s'} \quad (3.25)$$

3.2. A hibrid time slot alapú modell kiterjesztése

A módszer[9] a szűkebb feladatosztály sajátosságait kihasználva képes a korábbi modelleknél nagyságrendekkel nagyobb sebességet elérni. A time slotok a berendezések időrései, melyekhez hozzá kell rendelni az általa végzett taszkokat. Ha minden taszk egyetlen berendezésben hajtható végre, az anyagok sorrendje nem változhat, így a time slotok hozzárendelése minden berendezésre egyforma.

A hibrid jelző arra utal, hogy a robotok ütemezésénél precedencia változók is találhatók a modellben. A time slotok használatával sok big-M korlátozást el lehet kerülni, ami jelentősen növeli a teljesítményt. A szerzők további gyorsulást értek el egy robot esetén, az implikált precedencia változók értékének előzetes beállításával.

A módszer a tesztek alapján gyorsabbnak bizonyult a többinél, de a kezelhető feladatosztály tekintetében elmaradt a korábban bemutatott precedencia alapú modelltől. Nem volt képes kezelni az üres robot mozgást, a robot mozgási zónákat, LW korlátozásokat és több párhuzamos kádat.

Néhány funkció a time slot alapú reprezentáció miatt nem megvalósítható a modell kisebb módosításával. LW időzítéshez az eddig konstans paraméter végrehajtási időt változóvá kellene alakítani, de így a slot hozzárendelést jelölő bináris változóval való szorzás (3.30) miatt a lineáris modell négyzetessé válna. Ezáltal lényegesen megemelkedne a számítási idő, és más módszert is kellene alkalmazni a MILP megoldó helyett.

A több párhuzamos kád kezelése szintén nem lehetséges a modellben, mert az kihasználja a feladat azon tulajdonságát, hogy a terméksorrend nem változhat a folyamatban. Több kád esetén ez az állítás már nem teljesül.

Az üres robot mozgás kezelése viszont megvalósítható, ráadásul a modell komplexitása, a változók száma nem változik. Az eredeti korlátozások:

$$\begin{aligned} T_{t,m} &\geq T_{t',m'} + \pi_m - H(1 - \bar{Y}_{t,m,t',m'}) - H(2 - \bar{W}_{t,m,r} - \bar{W}_{t',m',r}) \\ &\quad \forall t, t' \in T : (t' > t), m, m' \in M : (m \neq m'), r \in R \end{aligned} \quad (3.26)$$

$$\begin{aligned} T_{t',m'} &\geq T_{t,m} + \pi_{m'} - H(\bar{Y}_{t,m,t',m'}) - H(2 - \bar{W}_{t,m,r} - \bar{W}_{t',m',r}) \\ &\quad \forall t, t' \in T : (t' > t), m, m' \in M : (m \neq m'), r \in R \end{aligned} \quad (3.27)$$

Az üres mozgást is kezelő új korlátozások a precedencia alapú modellnél is használt megállapításon alapulnak, és láthatóan csak konstans paraméterekkel bővülnek:

$$\begin{aligned} T_{t,m} &\geq T_{t',m'} + \pi_{m-1,m'}^{abs} + \pi_m - H(1 - \bar{Y}_{t,m,t',m'}) - H(2 - \bar{W}_{t,m,r} - \bar{W}_{t',m',r}) \\ &\quad \forall t, t' \in T : (t' > t), m, m' \in M : (m > 1, m \neq m'), r \in R \end{aligned} \quad (3.28)$$

$$\begin{aligned} T_{t',m'} &\geq T_{t,m} + \pi_{m,m'-1}^{abs} + \pi_{m'} - H(\bar{Y}_{t,m,t',m'}) - H(2 - \bar{W}_{t,m,r} - \bar{W}_{t',m',r}) \\ &\quad \forall t, t' \in T : (t' > t), m, m' \in M : (m' > 1, m \neq m'), r \in R \end{aligned} \quad (3.29)$$

A csak egy robot esetén használt korlátozások hasonlóképp módosíthatók.

Az eredeti egyenletek:

$$T_{t+1,m} \geq T_{t,m} + \sum_{i \in I} N_{i,t} p_{i,m} + \pi_{m+1} + \pi_m \quad \forall m \in ZW, t \in T : (t \neq |T|) \quad (3.30)$$

$$T_{t,m} + \pi_{m+1} + \pi_m \quad \forall m \in LS, t \in T : (t \neq |T|) \quad (3.31)$$

$$T_{t,m} \geq T_{t',m'} + \pi_m - [H(1 - \bar{Y}_{t,m,t',m'})]_{(t,m,t',m') \notin YE1} \quad \forall (t, m, t', m') \in YD0 \quad (3.32)$$

$$T_{t',m'} \geq T_{t,m} + \pi_m - [H(\bar{Y}_{t,m,t',m'})]_{(t,m,t',m') \notin YEO} \quad \forall (t, m, t', m') \in YD1 \quad (3.33)$$

A módosított korlátozások:

$$\begin{aligned} T_{t+1,m} &\geq T_{t,m} + \sum_{i \in I} N_{i,t} p_{i,m} + \pi_{m+1} + \pi_m + \pi_{m+1,m-1}^{abs} \\ &\quad \forall m \in ZW : (m > 1), t \in T : (t \neq |T|) \end{aligned} \quad (3.34)$$

$$T_{t+1,m} \geq T_{t,m} + \pi_{m+1} + \pi_m + \pi_{m+1,m-1}^{abs} \quad \forall m \in LS : (m > 1), t \in T : (t \neq |T|) \quad (3.35)$$

$$\begin{aligned} T_{t,m} &\geq T_{t',m'} + \pi_m + \pi_{m',m-1}^{abs} - [H(1 - \bar{Y}_{t,m,t',m'})]_{(t,m,t',m') \notin YE1} \\ &\quad \forall (t, m, t', m') \in YD0 : (m > 1) \end{aligned} \quad (3.36)$$

$$\begin{aligned} T_{t',m'} &\geq T_{t,m} + \pi_m + \pi_{m,m'-1}^{abs} - [H(\bar{Y}_{t,m,t',m'})]_{(t,m,t',m') \notin YEO} \\ &\quad \forall (t, m, t', m') \in YD1 : (m' > 1) \end{aligned} \quad (3.37)$$

A módosított modell képes a robotok üres mozgásának kezelésére. Az egy robot esetén használt implikált bináris változók beállítása, mely nagyban hozzájárul a módszer teljesítményéhez, továbbra is használható maradt.

A robot mozgási zónákra való kiterjesztés nem igényel sok módosítást: a korlátozásokat $r \in R$ helyett $r \in R_m$ tartományokra kell megfogalmazni, ahol R_m az m kádért felelős robotok halmaza.

3.3. A MILP fejlesztések vizsgálata

A tesztek során Bhushan és Karimi[3] által felírt permutation flow-shop problémájának különböző kád- és termékszámú (M és N) részére vizsgáltuk a módszereket. M és N növekedésével növekszik a probléma komplexitása is, ezt a fajta skálázást gyakran használják a szakirodalomban[6, 11] a megoldómódszerek teljesítményének összehasonlítására. Az időlimit 1000 s volt, ha ezalatt nem fejeződött be a futás, az addig megtalált legjobb megoldás zárójelek között jelenik meg a táblázatokban.

Tesztkörnyezet

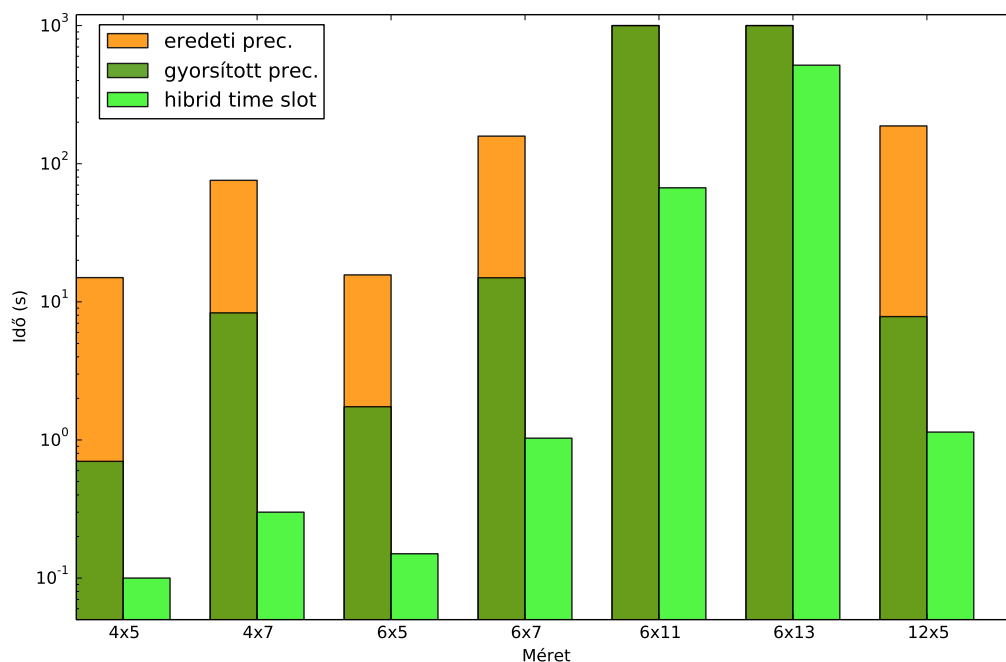
- Intel Xeon E5504 (4 fizikai mag, 2 GHz) CPU
- 8 GiB RAM
- Ubuntu 12.04 operációs rendszer
- MILP megoldó: Gurobi Optimizer 5.5

Az egy robotot tartalmazó, üres robot mozgás nélküli eredményeket a 3.1. táblázat mutatja be, valamint a futási idők könnyebb összehasonlítását segíti a 3.1. ábrán látható diagram. Az oszlopok magassága logaritmikus skálán ábrázolja a megoldási időket, a precedencia alapú módszernél a narancssárga rész a gyorsítás mértéke.

3.1. táblázat. Eredmények egy robottal, üres robot mozgás nélkül

MxN	Eredeti precedencia		Gyorsított precedencia		Eredeti hibrid time slot	
	Idő (s)	Makespan	Idő (s)	Makespan	Idő (s)	Makespan
4x5	14,98	62	0,7	62	0,08	62
4x7	75,81	76,18	8,32	76,18	0,3	76,18
6x5	15,67	72,41	1,74	72,41	0,15	72,41
6x7	158,41	90,83	14,94	90,83	1,03	90,83
6x11	1000	(128,81)	1000	(123,87)	66,94	121,13
6x13	1000	(311,15)	1000	(139,02)	516,89	136,81
12x5	187,79	128,9	7,82	128,9	1,14	128,9

Erre a feladatosztályra még az eredeti hibrid time slot modell is használható volt. A precedencia alapú modell[1] módosítása nagyságrendekkel gyorsította a módszert, de így sem érte utol Castro és társai[9] hibrid time slot alapú módszerét. Ez jól szemlélteti a hibrid time slot modell feladatosztályának kiterjesztésére irányuló motivációt.



3.1. ábra. Futási idők egy robottal, üres robot mozgás nélkül

Az üres robot mozgást is figyelembe vevő tesztek eredményeit a 3.2. táblázat tartalmazza. Megfigyelhető, hogy a hibrid time slot modell kiterjesztése nem lassította jelentősen a megoldást, a módszer továbbra is messze a leggyorsabb. Az eredeti precedencia alapú modell csak a kisebb feladatokat tudta az időkorláton belül megoldani, a vízszintes vonalak azt jelölik, hogy nem talált megoldást 1000 s alatt.

3.2. táblázat. Eredmények egy robottal, üres robot mozgással

MxN	Eredeti precedencia		Gyorsított prec.		Kiterjesztett hibrid time slot	
	Idő (s)	Msp.	Idő (s)	Msp.	Idő (s)	Msp.
4x5	47,86	63,42	0,9	63,42	0,11	63,42
4x7	1000	(78,38)	3,54	78,38	0,4	78,38
6x5	940,07	73,69	2,07	73,69	0,34	73,69
6x7	1000	—	14,92	93,47	1,08	93,47
6x11	1000	—	1000	(130,15)	461,65	126,75
6x13	1000	—	1000	(152,59)	1000	(144,39)
12x5	1000	—	12,4	130,16	2,27	130,16

Több robot esetén a precedencia alapú modell 3.1. alfejezet végén leírt javított változatára volt szükség a helyes megoldás megtalálására. A 3.3. táblázat a 2 robot esetén mért eredményeket mutatja be. A második robot hozzáadása jelentősen megnehezítette a megoldást, különösen az általánosabb, precedencia alapú modell esetén.

3.3. táblázat. Eredmények két robottal, üres robot mozgással

MxN	Javított precedencia		Gyorsított prec.		Kiterjesztett hibrid time slot	
	Idő (s)	Msp.	Idő (s)	Msp.	Idő (s)	Msp.
4x5	287,36	60,58	1,36	60,58	0,29	60,58
4x7	1000	(88,64)	18,48	74,84	4,49	74,84
6x5	716,36	71,15	3,93	71,15	0,95	71,15
6x7	1000	—	24,58	88,75	4,44	88,75
6x11	1000	—	1000	(120,45)	1000	(118,67)
6x13	1000	—	1000	(141,81)	1000	(135,95)
12x5	1000	—	49,13	127,64	7,27	127,64

A MILP tesztek eredményeit összefoglalva, a precedencia alapú modell gyorsítása nagyságrendekkel növelte a hatékonyságot, de a time slot alapú megközelítés az egyszerűbb feladatosztályra továbbra is jobban teljesít, ezért fontos eredmény ennek a modellnek a kiterjesztése is.

4. AWS ütemezés az S-gráf keretrendszerrel

Az AWS feladatok S-gráf alapú megoldásához ki kellett terjeszteni a módszertan modellezési eszköztárát és implementálni a módosított algoritmusokat.

4.1. AWS ütemezési feladatok modellezése S-gráffal

Az AWS feladatok modellezéséhez egy kiterjesztett S-gráf reprezentációra van szükség, mely alkalmas a ZW/LW tárkorlát és a szállítások kezelésére. Az alfejezetben ezen kiterjesztés bemutatása olvasható.

AWS ütemezések modellezéséhez bevezetjük az S-gráf kiterjesztett változatát, az AWS S-gráfot. A taszkok és végtermékek mellett a szállításokat is csúcsok jelölik. A szállítások berendezéshalmaza a robotok halmaza, vagy mozgási zónák esetén azok részhalmaza. Receptélük súlya a minimális szállítási idő. A taszkok berendezései a megfelelő kádak.

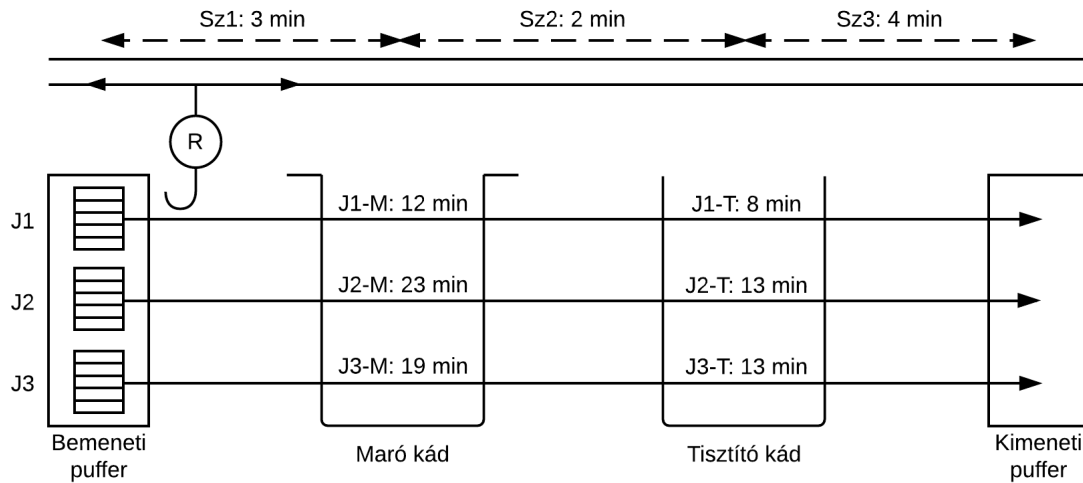
Az AWS ütemezésre használt első MILP modellekkel ellentétben a taszkokhoz használható több berendezés és az eltérő receptű termékek nem jelentenek nagy nehézséget, mert az efféle flexibilitásokat az S-gráf eszközei eredendően támogatják.

4.1.1. Limited wait taszkok ábrázolása

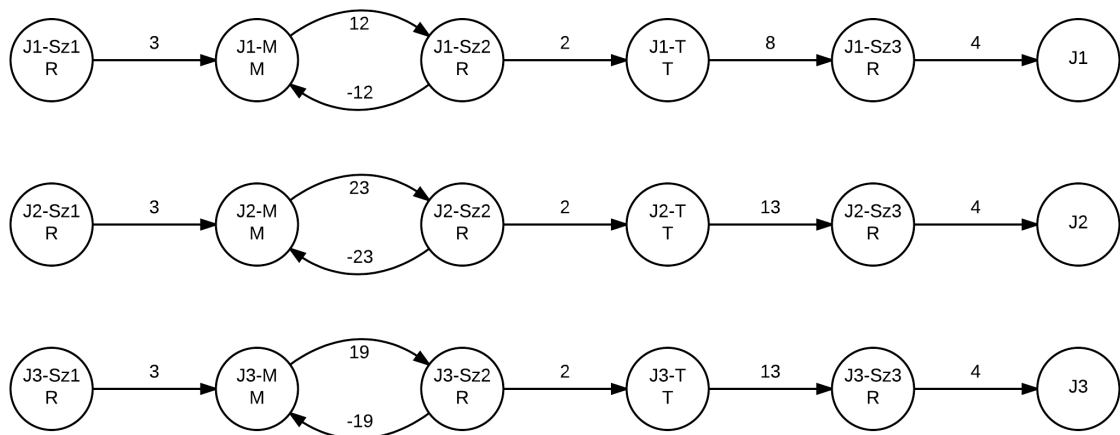
Az AWS feladatok modellezéséhez szükség volt ZW vagy LW taszkok ábrázolására is. Ennek megoldására több módszert is kipróbáltunk, majd a negatív súlyú élek mellett döntöttünk. Ez azt jelenti, hogy a ZW/LW taszkok csúcsaiból kiinduló receptélekkel ellentétes irányban kell behúzni egy-egy élt, melynek súlya a maximális végrehajtási idő ellentettje. Ha az él két végpontja között több idő telik el, mint a végrehajtási és várakozási idő összege, akkor pozitív kör keletkezik, amely már az eredeti módszertan szerint is megvalósíthatatlan ütemezést jelentett. A 0 súlyú körök esetén viszont két esetet kell megkülönböztetni: a negatív súlyú élt is tartalmazó, és a csak 0 súlyú élekből álló körök. Ha negatív súlyú él van benne, az azt jelöli, hogy az LW taszk pontosan annyit várakozik, amennyi a megengedett idő, ezért ez nem jelenti azt, hogy az ütemezés megvalósíthatatlan. A 0 súlyú – általában ütemezési – élekből álló kör viszont továbbra is cross-transfert jelöl[14].

4.1.2. Szemléltető példa

Az ismertetett változtatásokkal már felépíthető egy AWS ütemezési feladat S-gráf alapú modellje. A modellezést egy példán szemléltetjük. A 4.1. ábrán látható a példafeladat. A kész receptgráfot a 4.2. ábra mutatja be.

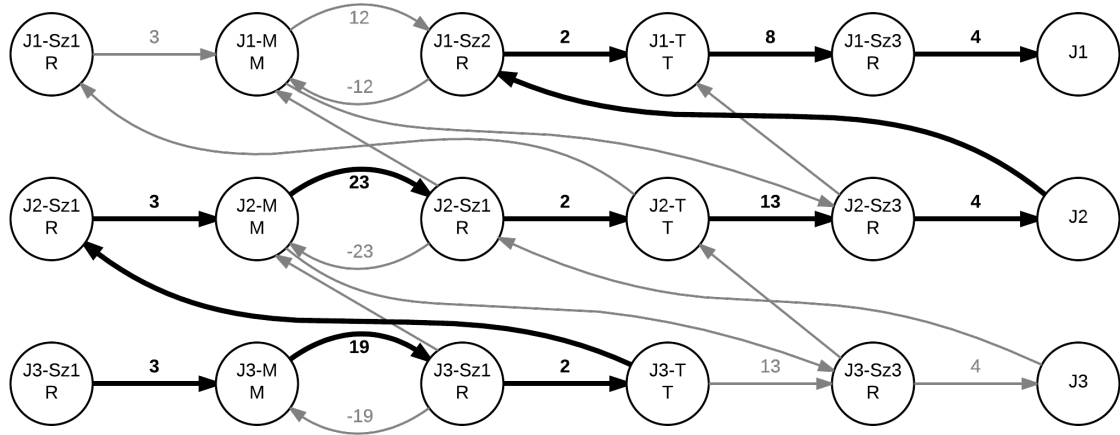


4.1. ábra. AWS példafeladat paramétereinek grafikus reprezentációja

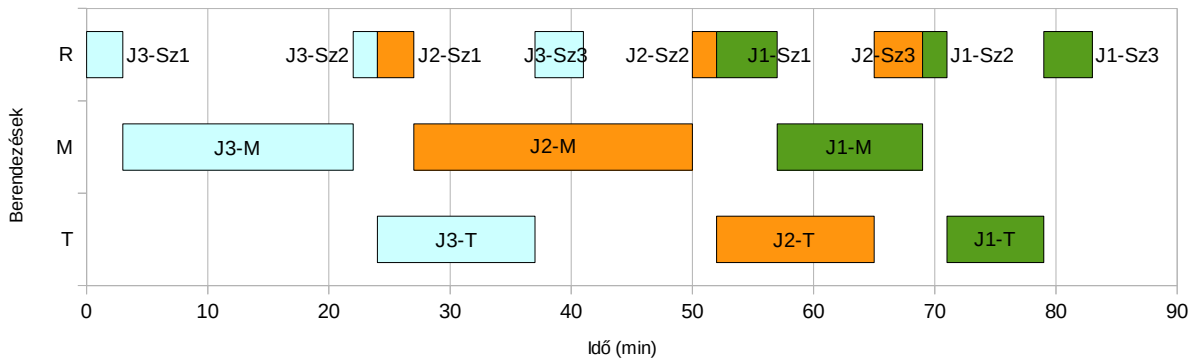


4.2. ábra. A példafeladat receptgráfja

A receptgráf felépítése után az S-gráf megoldó a módosított körkeresési feltétellel már meg tudja oldani a feladatot – az üres robot mozgás figyelmen kívül hagyásával. A megtalált optimális ütemezési gráfot a 4.3. ábra mutatja be a 83 perces makespan jelentő leghosszabb út kiemelésével. Az ütemezési gráf alapján készült Gantt-diagram a 4.4. ábrán látható.



4.3. ábra. A példafeladat optimális ütemezési gráfja



4.4. ábra. A példafeladat optimális ütemezésének Gantt-diagramja

4.1.3. Üres robot mozgás ábrázolása

Az üres robot mozgás az az idő, amire a robotnak szüksége van ahhoz, hogy eljusson egyik kádtól a másikig két szállítás között. Az AWS S-gráfban a robotok ütemezési élek jelölik két szállítás sorrendjét, ezért ennek az élsúlyát kell az üres mozgási időre állítani. Ez az időtartam függ a robot kiindulási és érkezési helyétől. A szállításnál előre ismert, hogy egy kádból melyik másikba kell szállítani a szeleteket. Az üres robot mozgás viszont az ütemezés során meghatározott szállítási sorrendtől függ.

Az S-gráfok és a precedencia alapú modell között sok a hasonlóság. Az ütemezési élek az általános precedencia változókhöz hasonlóan két taszk sorrendjét határozzák meg. Az élsúlyok pedig az időzítésért felelősek. A közvetlen precedencia jelölésére az S-gráf modellben

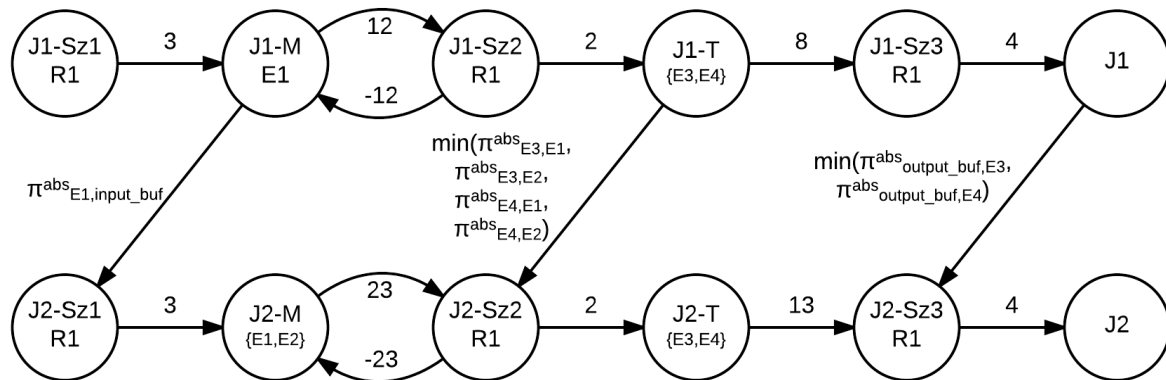
nincs lehetőség, de a 3.1. alfejezet bemutatja, hogy az üres robot mozgás kezeléséhez erre nincs is szükség.

A kádak távolsága ($\pi_{j,j'}^{abs}$) bemeneti paraméter. Ha minden taszkhoz csak egy kád használható, akkor az élsúly egyszerűen meghatározható a csúcsok berendezései alapján.

Ha egy taszkhoz több kád is választható, akkor bonyolulttá válik a mozgási idő kiszámítása. Ez elkerülhető, ha a párhuzamos kádakat távolság szempontjából azonosnak tekintjük, mintha egy helyen lennének. Ilyen AWS a gyakorlatban is létezik: a párhuzamos kádak egymás mellett, a robot sínjének két oldalán helyezkednek el.

A szakirodalomban csak Aguirre és társai[1] korábban ismertetett precedencia alapú módszere képes több párhuzamos kádat kezelni. A modellben csak az üres mozgás számításakor különböztetik meg ezen kádak helyzetét, szállítási időnél a fenti egyszerűsítést használják. A továbbiakban az S-gráf módszer kiterjesztése is ezt az elvet követi, de a szállítási idő kiszámítása is hasonló módon megvalósítható.

A szállítások közti ütemezési élek behúzásakor meg kell vizsgálni a lehetséges indulási és érkezési kádak – vagy pufferek – halmazát, és kiszámolni a létrejövő távolság minimumát. Ez a szám lesz az élsúly, megtartva az S-gráfok azon tulajdonságát, hogy a leghosszabb út össze-súlya alsó korlát a makespanre. Az élsúlyozást a 4.5. ábra szemlélteti. Az első él súlya előre ismert, mert az E1 berendezés már ki lett választva a J1-M taszkhoz. A második élnél két lehetőség van a mozgás kiindulási és érkezési helyére, így összesen 4 távolságnak kell venni a minimumát. A harmadik él esetében a kiindulási hely mindenképp a kimeneti puffer, ezért csak a két lehetséges érkezési helytől való távolság minimumát kell venni.



4.5. ábra. Robot ütemezési élek súlyozása

Ha a két szállítás közé egy újabbat ütemezünk, a köztük lévő él akkor sem jelent problémát, csak kevésbé számít bele a korlátba, mert az újonnan behúzott élek átveszik a szerepét. Két szállítás között mindenképp meg kell tennie a két kád közti utat a robotnak, akkor is, ha közben más szállítást is végez.

4.2. Az S-gráf alapú megoldás implementációja

Az S-gráf alapú megoldás implementációja a már meglévő, tanszéki fejlesztésű, megoldó kibővítésével történt. AWS feladatok kezeléséhez a szoftvernek egyrészt támogatnia kell a ZW tárolási stratégiát, másrészt az AWS feladatok során a szétválasztási eljárás is megváltozik, mivel az ütemezés során változik a robotkarok mozgásához szükséges idő. Az üres robot mozgás kezelése további módosításokat igényel.

4.2.1. A megoldó architektúrája

Az előzőekben leírt módszertanhoz készült egy megoldószoftver, amely C++ programozási nyelven íródott. A nyelv népszerű választás olyan tudományos szoftverek fejlesztéséhez, ahol különösen fontos a nagy teljesítmény. A C++-hoz tartozó Standard Template Library (STL) adattípustól független tárolókat, illetve algoritmusokat biztosít. A megoldó nagymértékben támaszkodik az STL tárolókra, illetve számos helyen alkalmazza az STL algoritmusait. A szoftver elsősorban az objektum orientált programozási paradigmát követi.

A megoldó a branch-and-bound keretalgoritmust használja a megoldás során. A főbb lépések a korlátozás és a szétválasztás. A szoftver felépítésében is megjelennek ezek a lépések. A korlátozásért és a döntések meghozásáért külön osztályok felelősek, valamint a megoldások fáját is egy másik osztály tartja karban. A döntések ebben az esetben azt jelentik, hogy melyik taszkot melyik berendezés hajtja végre, illetve, hogy az adott taszkot melyik másik taszk előzi meg. Ezeket a döntéseket több szempont alapján is meg lehet hozni, ezért fontos a moduláris felépítés, hogy a különböző algoritmusok egyszerűen cserélhetőek legyenek.

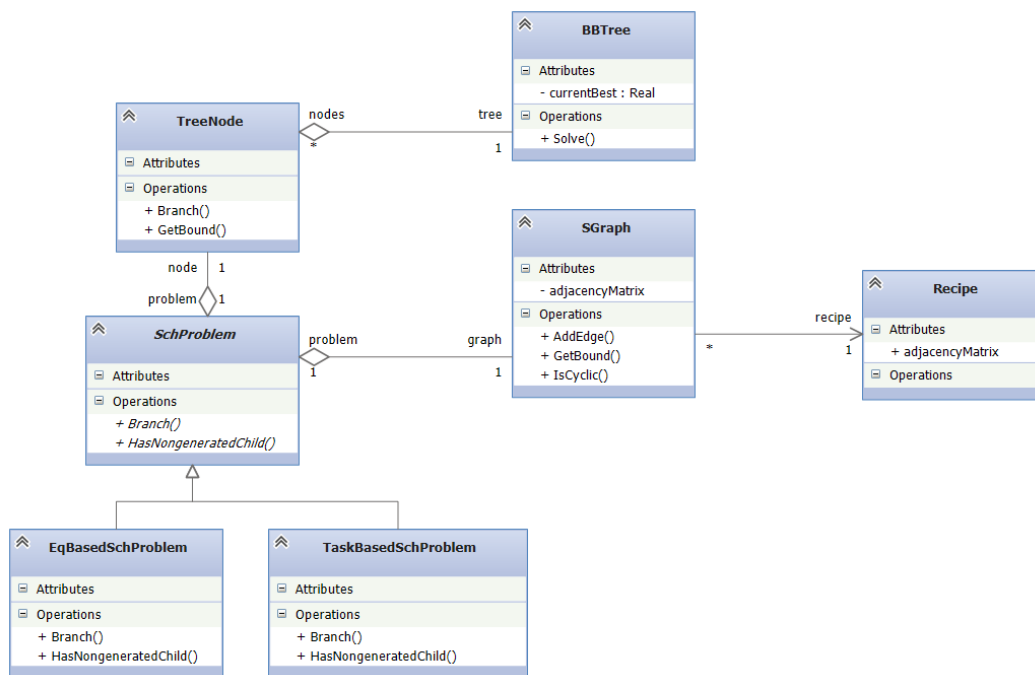
Sanmartí és társai[12] algoritmusára Smidla és társa[16] adott egy implementációt, mely képes az ütemezések fáját párhuzamosan bejárni – ezzel többmagos környezetben nagymértékben lerövidülhet a futási idő. A párhuzamosítás lényegében csak a fa bejárását érinti, az egyes részütemezéseket már csak egyetlen szál kezeli, így ezeknek nem kell szálbiztosnak (thread-safe) lenniük. Jelen dolgozatban tárgyalt szoftver ebből az implementációból indul ki.

Az S-gráffal kapcsolatos műveleteket az *SGraph* osztály valósítja meg. Az osztály segítségével például ütemezhető egy adott taszk egy adott berendezésre. Ezen műveletek előtt vagy után különböző ellenőrzéseket kell végezni. Például egy él behúzása, vagy módosítása után meg kell vizsgálni, hogy nem vált-e megvalósíthatatlanná az ütemezés. Mindezek miatt ez az osztály a szoftver egyik legfontosabb része. Minden *SGraph* tartalmaz egy mutatót egy *Recipe* példányra, amely a feladat receptgráfját írja le. A *Recipe* osztály segítségével objektum orientáltan definiálhatóak ütemezési feladatok. Lehetőséget biztosít termékek, taszkok, illetve berendezések hozzáadására, illetve kapcsolatuk ábrázolására.

A megoldás menetének rövid leírása a következő: ameddig a fa nem üres, addig kiválaszt egy nyitott részproblémákkal rendelkező gyereket és meghívja a szétválasztás eljárást, ami generál egy újabb részütemezést. Ha az újonnan generált ütemezés teljes, akkor eltárolja, mint megoldás, és amennyiben a megoldás célfüggvényének értéke alacsonyabb, mint az eddigi legjobb megoldásé, akkor a csökkenti a keresett problémák felső korlátját. A folyamatot megvalósító osztályok, és kapcsolatuk a 4.6. ábrán láthatóak.

A B&B fát a egy külön osztály, a *BBTree*, tartja karban, mely tartalmazza a fa elemeit, amelyben megtalálható többek között maga az S-gráf is. A különböző szétválasztási eljárásokat különböző osztályok valósítják meg – ez egyszerűvé teszi az új branching algoritmusok implementációját. A megoldó leegyszerűsített osztálydiagramját a 4.6. ábra szemlélteti.

A fa bejárása a következőképp történik: az algoritmus minden lépésben egy olyan csúcsot



4.6. ábra. A B&B keretalgoritmust megvalósító osztályok

keres, amelynek korlátja nem rosszabb, mint az eddigi legjobb megoldás. Először megvizsgálja az előzőleg szétválasztott részproblémát, és amennyiben van még nem generált gyereke, akkor azt választja. Egyébként feljebb lép a fában. Ha nem sikerül a feltételnek megfelelő részproblémát találni, akkor a keresés leáll, és így a megoldás is véget ér.

4.2.2. Az AWS receptgráf létrehozása

Ahhoz, hogy AWS feladatokat S-gráf segítségével oldjunk meg, először a kérdéses feladatot fel kell írni S-gráfként.

Első lépésben a joboknak megfelelő számú terméket adunk a recepthez, melyek neve J_i , ahol $i \in \{0, \dots, jobNum - 1\}$. Ezt követően a robotok számának megfelelő robotot adunk a berendezésekhez, melyek neve R_i , ahol $i \in \{0, \dots, robotNum - 1\}$. A kádak is berendezésként kerülnek a receptbe – nevük B_i , ahol $i \in \{0, \dots, bathNum - 1\}$. Ezek után a transzfer mátrix kerül kitöltésre, mely bármely két kád közötti mozgatási időt tartalmazza.

Ezt követően az egyes taszkok létrehozása következik. A termék kádban való kezelése és a kádba való szállítás is egy-egy taszk, így gyártási lépésenként két taszk kerül létrehozásra, illetve jobonként még egy transzfer taszk, amely a kimeneti pufferbe (outlet buffer) szállítja az immár kész terméket.

Ha mindez megtörtént, akkor következik az élek behúzása. Az egyes jobokhoz tartozó taszkok között egyszerűen meghatározható precedenciát írják le a receptélek, vagyis minden kádban végrehajtandó kezelést megelőző egy transzfer taszk. Az utolsó kezelést követően pedig egy transzfer taszk az outlet pufferbe szállítja a terméket, ezt követően pedig a termék csúcs következik.

4.2.3. Limited wait támogatás

Automated wet-etch station feladatok megoldásához elengedhetetlen, hogy a megoldószoftver képes legyen kezelni a zero wait (ZW) tárolási stratégiát.

Munkánk során LW kezeléssel bővítettük ki a megoldót. A módszer lényege, hogy ellentétes irányban behúzz egy élt, melynek súlya $-pt - wait_time$, ahol pt a végrehajtási idő, max_wait pedig a maximális várakozási idő. Amennyiben több berendezés is végre tudja hajtani az adott taszkot, akkor a receptgráfban pt a lehetséges végrehajtási idők minimuma lesz. Azonban az ellentétes irányban behúzott élnél ezek maximumát kell tekintenünk, hogy a tárolási limit teljesüljön. Ez azzal jár, hogy az ütemezés során az élek súlya nem csökkenhet, amely tulajdonság eddig is fent állt, és több algoritmus, köztük a körkeresés is, felhasználta. Az ellentétes irányú élek behúzását az *SGraph::AddLwAuxEdges()* metódus végzi (4.1. forráskód).

Forráskód 4.1. SGraph::AddLwAuxEdges()

```
1 void SGraph::AddLwAuxEdges() {
2     for(uint i = 0; i < nodecount; i++) {
3         if(GetTaskIdOfNode(i) != -1) {
4             const Task& cTask = GetTask(GetTaskIdOfNode(i));
5             ushort policy = cTask.GetPolicy();
6             if(policy & LW || policy & ZW) {
7                 for(uint j = 0; j < nodecount; j++) {
8                     double weight = matrix->Get(i, j);
9                     if(weight != SGraphMatrix::nothing && weight > 0) {
10                        double newWeight = -GetMaxProcTime(i) - cTask.
11                        GetMaxWaitTime();
12                        matrix->Set(j, i, newWeight);
13                        recipe->_matrix->Set(j, i, newWeight);
14                        SetLwAuxEdge(j, i, true);
15                    }
16                }
17            }
18        }
19    }
```

Forráskód 4.2. SGraph::ModifyRecipeEdge()

```
1 void SGraph::ModifyRecipeEdge(uint taskid, double time) {
2     const Task& task = GetTask(taskid);
3     uint nodeid = task.GetNodeId();
4     for (uint i = 0; i < nodecount; i++) {
5         if (IsRecipeEdge(nodeid, i) && !IsLwAuxEdge(nodeid, i)) {
6             if(IsLwAuxEdge(i, nodeid))
7                 SetWeight(i, nodeid, -time-task.GetMaxWaitTime());
8             SetWeight(nodeid, i, time);
9         }
10    }
11 }
```

Az ütemezés során egyértelműen meghatározásra kerül, hogy melyik berendezés hajtja végre az adott taszkot, így ilyenkor módosulhat a kezdetben megadott pt , és ezzel együtt lényegében a receptél is. Ezért ilyenkor az *AwsSGraph* osztály *ModifyRecipeEdge()* metódusa fut le, amelyet úgy kellett kialakítani, hogy az ellentétes irányú negatív éleket is frissítse – ez látható a 4.2. forráskódon. Ahogy az a forráskódokon is látható, minden élről tárolásra kerül, hogy azok valójában LW segédélek-e. Ez ronthatja a teljesítményt, mivel a branchelés során több adatot kell másolni, illetve a lefoglalt memória mérete is nagyobb.

Módosított körkeresés

A célfüggvény értékének meghatározásához szükség van az adott részütemezésben található leghosszabb út hosszára. Mivel erre a megoldás során minden korlátozás során szükség van, ezért a bármely két csúcs közötti leghosszabb utakat a szoftver folyamatosan karbantartja – ezt megkönnyíti, hogy a behúzott élek súlya nem csökkenhet. Ezt a mátrixot azonban használhatjuk körkereséshez is. Ebben az esetben, új él behúzásakor azt kell megvizsgálni, hogy ellentétes irányban létezik-e út. Mivel a művelet során csupán egyetlen értéket olvasunk be a mátrixból – melynek komplexitása $O(1)$ –, ezért ez a módszer lényegesen gyorsabb, mint bármely körkereső algoritmus. Az implementációhoz a szoftver ezen metódusát módosítani kellett a 4.1.1. alfejezetnek megfelelően. A körkeresés során figyelembe kellett venni, hogy a gráfban két csúcs, vagyis két taszk között létezhet negatív súlyú él is. Ebben az esetben ellentétes irányban egy pozitív súlyú él húzódik.

Az implementáció során különös gondot kellett fordítani a 0 hosszú körök kezelésére is, amelyet tovább nehezített az általánosan elterjedt IEEE 754 szabványú[17] lebegőpontos számábrázolás sajátossága. Ennek értelmében egy lebegőpontos szám nem veheti fel a 0 értéket, csupán annak ϵ sugarú körében tartózkodhat. Mindezek miatt szükség volt egy megengedőbb összehasonlításra (4.1. egyenlet).

$$\left| \left| \frac{a}{b} \right| - 1 \right| < \epsilon, \text{ ahol } a \text{ és } b \text{ a két összehasonlítandó szám} \quad (4.1)$$

Az ϵ minden művelet (összeadás, szorzás, stb.) után növekszik, azonban a leghosszabb utakat karbantartó algoritmusban ezek száma alacsony, így ϵ is alacsonyan tartható. Az elvégzett tesztek alapján a jelenlegi megoldás kiküszöböli ezt a problémát.

A számábrázolásból eredő nehézségek mellett további gondot okoz az is, hogy a 0 hosszú körök bizonyos esetben megvalósíthatatlan ütemezést jelentenek, más esetben azonban nem. Ennek eldöntéséhez bármely két csúcs között tárolni kell, hogy a létezik-e közöttük csak ütemezés élekből álló út. Amennyiben nem, akkor a 0 hosszú kör megengedhető, egyébként nem az. A problémát egyszerűsíti, hogy a megoldás során a behúzott élek súlya nem csökkenhet. Tehát, amennyiben két csúcs közötti leghosszabb út növekszik, akkor rendelkezésünkre áll, hogy az adott út tartalmaz-e ütemezési élet vagy sem.

Több, erre kiélezett irodalmi példával[18, 13] tesztelésre került az implementáció, es egyetlen esetben sem mutatott logikai vagy numerikus okokból eredő hibát.

4.2.4. Üres robot mozgás támogatása

A szállítások közé újonnan behúzott ütemezési él súlyát az 1. algoritmusban található GETWEIGHT függvény végzi. A korábbi szállítás érkezési helye az ütemezési él kezdőpontjához rendelt kád. Ha a csúcshoz nem lehet kádat rendelni, mert termékcsúcs, az azt jelenti, hogy a kimeneti pufferben ért véget a szállítás. Ha a csúcs egy taszk, de még nincs hozzárendelve kád, a távolság számításakor az összes választható kádat figyelembe veszi. A későbbre ütemezett szállítás kiindulási pontját annak csúcsát megelőző taszkcsúcs segítségével számítja ki. Ha nincs megelőző csúcs, akkor a szállítás a bemeneti pufferben kezdődik. Ha a csúcshoz nincs kád rendelve, akkor az előbbi módszerhez hasonlóan minden lehetséges kádra kiszámolja a távolságot. Ha különböző távolságokat számolt, akkor azok minimumára állítja az élsúlyt. A kádak ütemezésekor a taszk hozzárendelésénél szükséges lehet a taszkhoz való beszállítást megelőző és kiszállítást követő robot mozgások élsúlyának frissítésére. Ez a 2. algoritmusban bemutatott új BRANCHING eljárás 11-17. sorában látható ciklusban történik meg. Megvizsgálja, hogy van-e olyan ütemezési él, amit frissíteni kell, majd a GETWEIGHT függvényhez hasonló,

1. algoritmus A robot ütemezési élek súlyát meghatározó függvény

function GETWEIGHT($(l, k), EQ, A_1, last_node, eq_task_pairs$) $start := \emptyset$ $finish := \emptyset$ **if** l is a product node **then** $start := \{output_buf\}$ **else****if** $\exists e (e, l) \in eq_task_pairs$ **then** $start := \{e\}$ **else** $start := \bigcup_{\forall e: l \in N_e, (e, 0) \notin last_node} e$ **end if****end if****if** $\exists (j, k) \in A_1$ **then****for all** $(j, k) \in A_1$ **do****if** $\exists e (e, j) \in eq_task_pairs$ **then** $finish := \{e\}$ **else** $finish := \bigcup_{\forall e: j \in N_e, (e, 0) \notin last_node} e$ **end if****end for****else** $finish := \{input_buf\}$ **end if****return** $\min_{e \in start, e' \in finish} (\pi_{e, e'}^{abs})$ **end function**

de, mivel a mozgás egyik végpontja már biztosan ismert, annál egyszerűbb módon kiszámolja az új súlyt.

Az eredeti berendezés alapú szétválasztás során elegendő volt minden berendezéshez csak az utoljára ütemezett taszkját eltárolni. A mozgási idők kiszámításához azonban tudni kell, hogy az egyes taszkok mely kádákhoz lettek hozzárendelve vagy azt, hogy még egyikhez sem. A hozzárendeléseket az eq_task_pairs halmaz tárolja, így ez is a részfeladat eleme lesz.

2. algoritmus A módosított szétválasztási eljárás

```

procedure BRANCHING( $G(N, A_1, A_2)$ ,  $bound$ ,  $last\_node$ ,  $eq\_task\_pairs$ ,  $SOUN$ )
  Let  $EQ$  be a unit such that  $N_{EQ} \cap SOUN \neq \emptyset$  and  $(EQ, 0) \notin last\_node$ 
  for all  $k \in N_{EQ} \cap SOUN$  do
     $G^k(N, A_1^k, A_2^k) := G(N, A_1, A_2)$ 
     $last\_node^k := last\_node$ 
     $eq\_task\_pairs^k := eq\_task\_pairs \cup \{(EQ, k)\}$ 
     $SOUN^k := SOUN \setminus \{k\}$ 
    for all  $(k, l) \in A_1^k$  do
      UPDATE( $c(k, l)$ ) in  $A_1^k$ 
      if  $EQ \notin R$  then
        for all  $(j, l) \in A_2^k$  do
          if  $\exists EQ' (EQ', j) \in eq\_task\_pairs^k$  then
             $c(j, l)$  in  $A_2^k := \pi_{EQ, EQ'}^{abs}$ 
          else
             $c(j, l)$  in  $A_2^k := \min_{e|j \in N_e} (\pi_{EQ, e}^{abs})$ 
          end if
        end for
      end if
    end for
    if  $\exists j (EQ, j) \in last\_node$  then
      for all  $(j, l) \in A_1^k$   $c(j, l) \geq 0$  do
         $A_2^k := A_2^k \cup \{(l, k)\}$ 
        if  $EQ \in R$  then ▷  $R$  is the set of robots
           $c(l, k)$  in  $A_2^k := \text{GETWEIGHT}((l, k), EQ, A_1^k, last\_node, eq\_task\_pairs^k)$ 
        end if
      end for
       $last\_node^k := last\_node^k \setminus \{(EQ, j)\}$ 
    end if
     $bound^k := \text{bounding}(G^k(N, A_1^k, A_2^k))$ 
    if  $bound^k < current\_best$  then
      if  $SOUN^k = \emptyset$  then
         $solution := G^k(N, A_1^k, A_2^k)$ 
         $current\_best := bound^k$ 
      else
         $SET := SET \cup \{(G^k(N, A_1^k, A_2^k), bound^k, last\_node^k, eq\_task\_pairs^k,$ 
 $SOUN^k)\}$ 
      end if
    end if
  end for

```

```

if  $\forall k \in N_{EQ} \cap SOUN \exists EQ' \neq EQ \ k \in N_{EQ'}, (EQ', 0) \notin last\_node$  then
   $G^*(N, A_1^*, A_2^*) := G(N, A_1, A_2)$ 
   $last\_node^* := last\_node$ 
  for all  $k \in N_{EQ}$  do
    for all  $(k, l) \in A_1^*$  do
       $UPDATE(c(k, l))$  in  $A_1^*$ 
    end for
  end for
  if  $\exists (EQ, j) \in last\_node$  then
     $last\_node^* := last\_node^* \setminus \{(EQ, j)\}$ 
  end if
   $last\_node^* := last\_node^* \cup \{(EQ, 0)\}$ 
   $bound^* := BOUNDING(G^*(N, A_1^*, A_2^*))$ 
  if  $bound^* < current\_best$  then
     $SET := SET \cup \{(G^*(N, A_1^*, A_2^*), bound^*, last\_node^*, eq\_task\_pairs, SOUN)\}$ 
  end if
end if
end procedure

```

5. Az S-gráf alapú módszer gyorsítása

Az AWS S-gráf modell segítségével az eredeti branch-and-bound algoritmus a fenti módosításokkal képes AWS ütemezési feladatok megoldására. A teljesítmény növeléséhez azonban olyan gyorsítási technikákra van szükség, amik kihasználják a feladat sajátosságait.

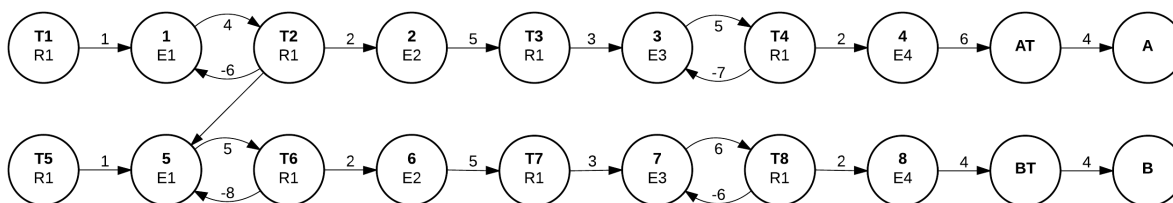
A permutation flow-shop feladatokban a termékek gyártási folyamata megegyezik, és minden lépésben csak egy kád használható, így minden kádban egyforma a terméksorrend. Az eredeti megoldó módszer ezt a redundanciát nem vette figyelembe. Az 5.1. alfejezetben bemutatott gyorsítás erre a problémára ad megoldást.

Ha csak egy robotot kell ütemezni, további egyszerűsítések alkalmazhatók, így ezt az esetet a szakirodalmi módszerek[9] is külön kezelik. Az 5.2. alfejezetben is erre az esetre adunk meg egy gyorsítási módszert.

Az 5.3.1. alfejezetben a Bhushan és társai[3] által publikált heurisztika implementációját és felhasználását mutatjuk be.

5.1. Job segédélek

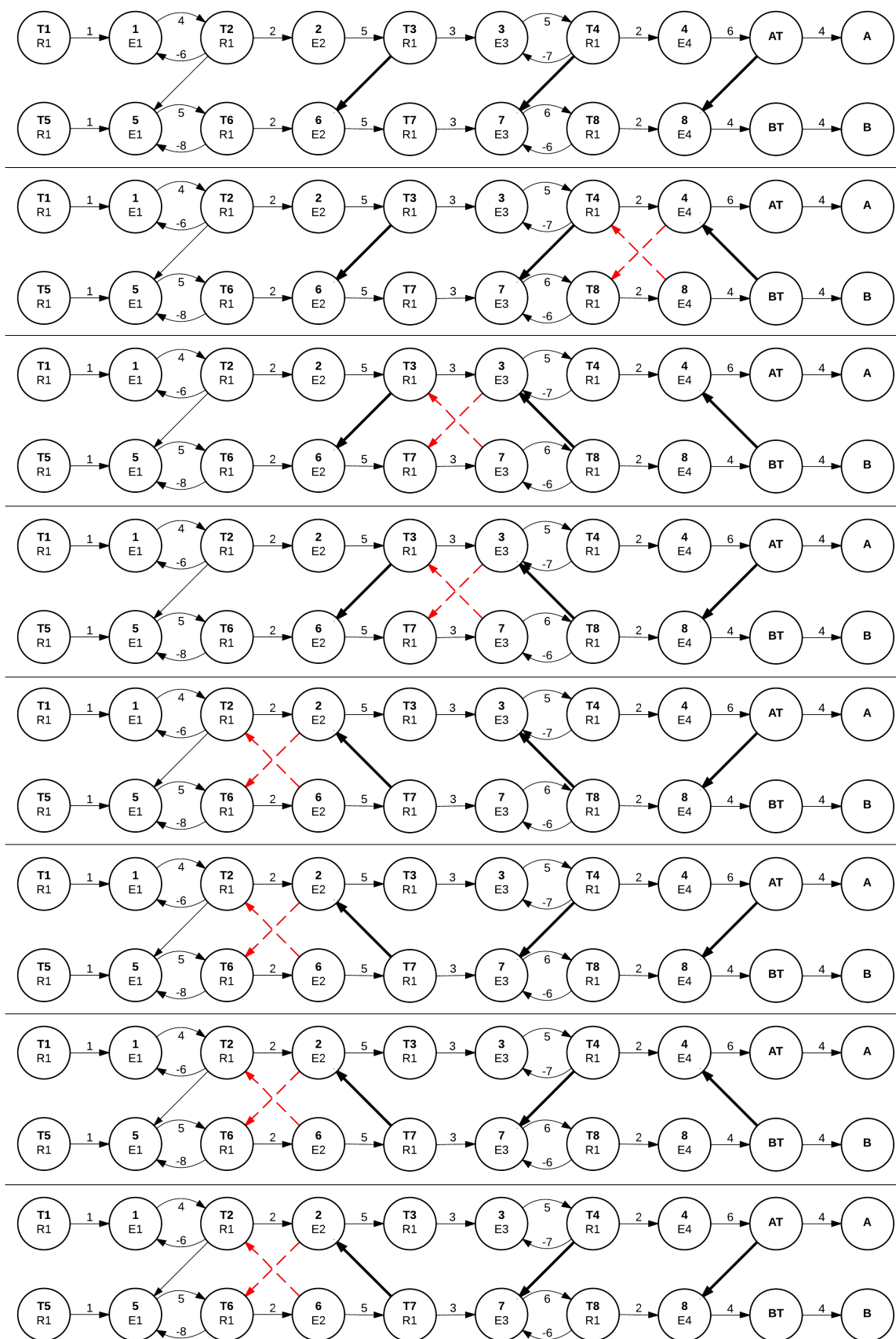
A jelenséget a következő példa szemlélteti. Az 5.1. ábrán látható egy S-gráf, ahol még csak az első kád (E1) van ütemezve. Először az A, majd a B termék első taszkja (1 és 5 csúcs) kerül végrehajtásra. A maradék 3 kádnak 8-féle ütemezése lehetséges, melyeket az 5.2. ábra mutat be. Ezek közül csak az első vezet megvalósítható ütemezéshez. A többiben sincs kör, de a robot ütemezési élek behúzásával keletkezni fog, ahogy azt a szaggatott élek mutatják – a két él bármelyike kört hoz létre. Tehát már korábban garantált, hogy a többi ág részfaiban nincs megvalósítható megoldás, de ez csak a részfák alsóbb szintjén derül ki, így addig több részproblémát is feleslegesen megvizsgálunk.



5.1. ábra. A példa kiindulási S-gráfja

A fenti probléma kiküszöbölhető úgy, hogy egy szétválasztási lépésben nem csak egy taszkot ütemezünk, hanem a termék összes taszkját, azaz egy teljes jobot.

A példára vetítve ez azt jelenti, hogy az első kád ütemezésekor az 5.2. ábra első gráfjának mind a 4 ütemezési élet behúzzuk. A gyorsítás ezáltal egyetlen szétválasztással olyan S-gráfokat



5.2. ábra. A kádak ütemezése utáni S-gráfok

hoz létre, amelyeket az eredeti módszer több döntés meghozatalával érne el, miközben több olyan gráfot is generálna, amelyek később kört eredményeznének. A keresési tér és a fa magasságának csökkentése jelentősen növelheti a megoldás sebességét.

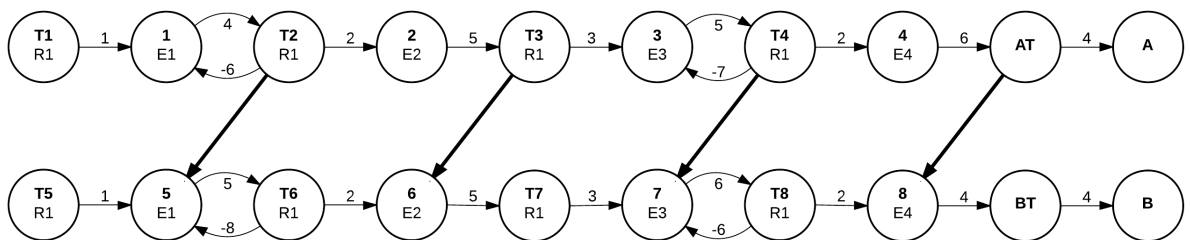
Az eredeti taszk alapú szétválasztás egy taszkot választ ki, melyet a használható berendezések végrehajtási listáinak összes lehetséges pontjára beszúrja, így hozza létre egy részprobléma gyerekeit. A gyorsítással módosított szétválasztás abban különbözik, hogy amikor egy taszkot beütemez egy kádra, ugyanannak a jobbnak a többi taszkját is beszúrja a neki megfelelő kád végrehajtási listájába, ugyanabba a pozícióba.

A berendezés alapú szétválasztásba hasonló módon integrálható a gyorsítás. Itt egy berendezés kerül kiválasztásra, és minden gyerekben a berendezés által végrehajtható taszkokból egy-egy kerül beütemezésre a végrehajtási sor végére. Ha a választott berendezés egy kád, akkor a módosított szétválasztás minden gyerekben egy-egy job taszkjait beütemezi a megfelelő kádak végrehajtási sorának végére.

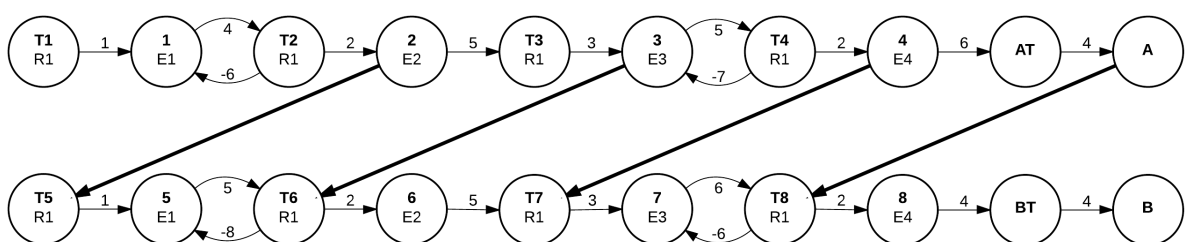
5.2. Módosított élbehúzás egy robot esetén

Ha a permutation flow-shop feladatban csak egy robot áll rendelkezésre, az tovább egyszerűsíti a problémát.

Eddig az ütemezési élek azt jelentették, hogy egy kád akkor kezdheti a következő taszkot, ha az előzőt befejezte. A korábbi taszk termékét tovább kell szállítani, a későbbi kezdéséhez pedig a kádba kell vinni a következő anyagot. Több robot segítségével a két szállítás párhuzamosan is történhet, viszont ha csak egy robot áll rendelkezésre, akkor az csak azután tudja megkezdeni a beszállítást, ha a kiszállítást már véget ért. Ezt felhasználva módosítható az élbehúzás, hogy a taszkok közti ütemezési éleket ne a következő csúcsból (5.3. ábra), hanem az azutániból a későbbi taszk csúcsát megelőzőbe (az 5.4. ábra) húzza be az algoritmus.



5.3. ábra. Az eredeti ütemezési élek



5.4. ábra. Az új ütemezési élek

Megfigyelhető, hogy ezek megfelelnek a robot néhány ütemezési élének. A gyorsítás tehát lényegében együtt ütemezi a robotot a kádakkal. A többi robot ütemezési él behúzása a hagyományos módon történik.

A módosított élbehúzás élesebb korlátot ad, ezért hamarabb kivágásra kerülnek a szuboptimális részütemezések. A kisebb keresési fa pedig gyorsabb megoldáshoz vezethet.

A módszer használható a job segédélekkel együtt is, a berendezés alapú és a taszk alapú szétválasztási eljárásoknál is.

Az új élbehúzást az SGraph osztályból származó AwsSGraph osztály GetOneRobotScheduleEdge() függvénye valósítja meg (5.1. forráskód). A függvényt az AwsSGraph::AddSchedulingEdges() virtuális függvény hívja meg – ha a gyorsítás be van kapcsolva és a feladat megfelel a követelményeknek –, majd a visszakapott Edge vektorban lévő éleket hozzáadja a gráfhoz.

A paraméterként kapott prevtask és taskid a két egymás után ütemezett taszk, melyek nem szállítások. A hozzájuk tartozó csúcsok a prevtasknode és tasknode. A 6. sorban lévő for ciklus bejárja a gráf csúcsait, a 8. sorban pedig megvizsgálja, hogy vezet-e oda receptél a prevtasknode-ból. A startnode fogja tárolni a prevtask-ot követő szállítást. A 9-11. sorokban hasonló módon keresi meg a taskid-t megelőző szállítás csúcsát, és tárolja el az endnode-ban. Végül a GetScheduleEdgeNIS() függvényt hívja meg, ami a szokásos módon visszaadja a startnode-ot követő csúcsból az endnode-ba húzandó élek vektorát.

Forráskód 5.1. AwsSGraph::GetOneRobotScheduleEdge()

```

1  vector<Edge> AwsSGraph::GetOneRobotScheduleEdge(int prevtask, uint taskid, uint
    eqid) {
2      vector<Edge> edges;
3      if (prevtask != -1) {
4          uint prevtasknode = GetRecipe()->GetTask(prevtask).GetNodeId();
5          uint tasknode = GetRecipe()->GetTask(taskid).GetNodeId();
6          for (uint start = 0; start < GetRecipe()->GetTaskCount(); start++) {
7              uint startnode = GetRecipe()->GetTask(start).GetNodeId();
8              if (GetRecipe()->_matrix->Get(prevtasknode, startnode) !=
                SGraphMatrix::nothing) {
9                  for (uint end = 0; end < GetRecipe()->GetTaskCount(); end++) {
10                     uint endnode = GetRecipe()->GetTask(end).GetNodeId();
11                     if (GetRecipe()->_matrix->Get(endnode, tasknode) !=
                        SGraphMatrix::nothing) {
12                         vector<Edge> newedges = GetScheduleEdgeNIS(start, end,
                            eqid);
13                         edges.insert(edges.end(), newedges.begin(), newedges.
                            end());
14                     }
15                 }
16             }
17         }
18     }
19     return edges;
20 }

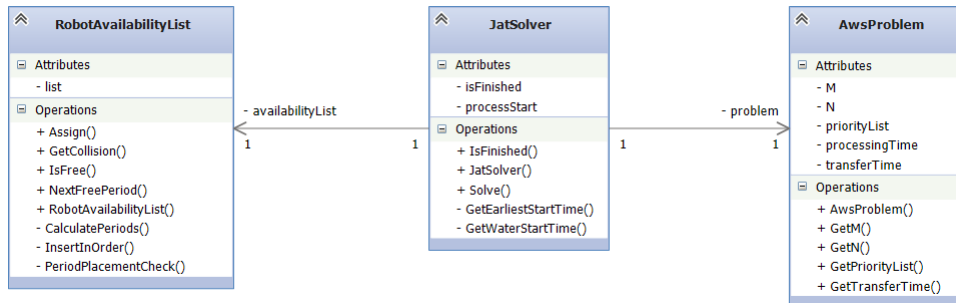
```

5.3. Kezdeti megoldás keresése JAT heurisztikával

Különösen nagy feladatok esetén fordul elő, hogy a szoftver nem talál megoldást belátható időn belül. Mivel az optimalizáció során diszkrét döntéseket kell meghozni, a kombinatorikus robbanás miatt egy bizonyos méretű feladat felett ez nem is lehet reális cél. Azonban számos heurisztika létezik, melyek nagyméretű feladatok esetén is gyorsan jutnak eredményre. Ez felhasználható a megoldás megkezdése előtt, hiszen már a B&B fa bejárásának kezdetén egy felső korlátot ad. Így a bejárt részproblémák száma csökkenhet, ami nagy feladatoknál jobb célfüggvény értékű ütemezéshez vezethet, kisebb feladatok esetén pedig rövidebb futási időt eredményez.

5.3.1. Job-at-a-time heurisztika implementálása

A Bhushan és társa[3] által bemutatott job-at-a-time (JAT) heurisztika (A. függelék) olyan AWS problémákra alkalmazható, ahol a végrehajtandó jobok sorrendje előre megadott, és egyetlen robotkart kell ütemezni. Kevésbé szigorú megkötéseket alkalmazó feladatra is megvalósítható eredményt ad, azonban az eredmény várhatóan messzebb lesz az optimálistól. Ez azonban előmegoldóként való felhasználását nem akadályozza meg, így az algoritmust bármely AWS feladatra alkalmazhatjuk, ahol nem vesszük figyelembe az üres robot mozgást. Bár a heurisztika implementálása nem tűnik bonyolultnak köszönhetően egyszerű felépítésének, a szerzők nem adtak leírást a legkorábbi szabad intervallum kereséséhez, amely kulcsfontosságú a jó megoldások megtalálásához. Mivel valós intervallumokról van szó, ezért a feladat nem triviális.



5.5. ábra. A JAT heurisztika implementációjának osztálydiagramja

A megoldó által kezelt feladatok esetén a jobok sorrendje nincs előre megadva, ezért az egyszerűsítést minden bemenet esetén végre kell hajtani. Jelenleg a jobok sorrendje a megadásuk sorrendje szerint kerül rögzítésre – ezt a jövőben heurisztikák határozhatják meg. Magát az algoritmust a *JatSolver* osztály *Solve()* metódusa tartalmazza. Az osztály tartalmaz egy *RobotAvailabilityList*-et, amely felelős az intervallumok karbantartásáért. Az intervallumokat a kezdő- illetve végpontjukkal jellemzi, és egy *vector*-ban tárolja őket időrendi sorrendben. Ez megkönnyíti a beillesztést.

Az osztály az *Assign()* és a *NextFreePeriods()* metódusokon keresztül használható. Az *Assign()* egyetlen intervallumot vár, amelyet a lehető legkorábbra igyekszik behelyezni a listába, vagyis célja, hogy a megadott kezdőidőponttól való eltérés a legkisebb legyen. Azonban a JAT algoritmus működéséhez ez még nem elég, ugyanis az AWS feladat ZW tárolási stratégiája miatt a következő robotmozgásnak pontosan $t_{(k,i)}$ idő múlva kell megtörténnie. Ez azt jelenti,

hogy egyszerre két intervallumot kell behelyeznünk a listában úgy, hogy a köztük lévő távolság nem változhat. Ebben segít a 4. algoritmus, amely két intervallumot vár, és két intervallumot ad vissza. A visszaadott intervallumok a bemenet eltoltjai.

Az algoritmus lényege, hogy a két intervallumot egyszerre próbálja beilleszteni a listába, és ha ütközést észlel, akkor megpróbálja az ütköző intervallum végére mozgatni az intervallumokat. Mindezt addig végzi, amíg nem lesz szabad mindkét intervallum. Ez mindenképp megtörténik, hiszen véges számú robotmozgást tartalmaz az intervallumlista.

3. algoritmus Adott intervallummal ütköző intervallum keresése

Require: ISFREE($(start, stop)$) is false

```

function GETCOLLISION( $start, stop$ )
  for all  $i = \{1, \dots, len(list)\}$  do
    if  $start < list_{i,2}$  and  $stop > list_{i,1}$  then return  $list_i$ 
    end if
  end for
end function

```

4. algoritmus Két szabad intervallum keresése – a köztük lévő távolság adott

```

function NEXTFREEPERIODS( $load, unload$ )
  if ISFREE( $load$ ) and ISFREE( $unload$ ) then return ( $load, unload$ )
  else
     $loadLen \leftarrow load_2 - load_1$ 
     $unloadLen \leftarrow unload_2 - unload_1$ 
     $waitLen \leftarrow unload_1 - load_2$ 
     $curr \leftarrow (load, unload)$ 
    while true do
       $currStart \leftarrow curr_{1,1}$ 
       $\Delta \leftarrow 0$ 
      if ISFREE( $curr_1$ ) then
        if ISFREE( $curr_2$ ) then return  $curr$ 
        else
           $j \leftarrow GETCOLLISION(curr_2)$ 
           $\Delta \leftarrow j_2 - curr_{2,1}$ 
        end if
      else
         $j \leftarrow GETCOLLISION(curr_1)$ 
         $\Delta \leftarrow j_2 - curr_{1,1}$ 
      end if
       $curr \leftarrow ((currStart + \Delta, currStart + \Delta + loadLen), (currStart + \Delta + loadLen + waitLen, currStart + \Delta + loadLen + waitLen + unloadLen))$ 
    end while
  end if
end function

```

Az implementált intervallum kereső algoritmus feltehetően nem egyezik meg a szerzők által választottal, mivel, bár megvalósítható, de a cikkben publikáltaktól eltérő megoldásokat ad.

5.3.2. Előmegoldó interfész

A heurisztikus algoritmusok beillesztéséhez szükség van egy egyszerű interfészre, amelyen keresztül hozzáférnek a feladathoz, és visszaadhatják a kapott eredményt. Alapvetően két részre oszthatók ezek az eljárások. Léteznek olyan algoritmusok, amelyek csupán a felső korlátot határozzák meg, míg más módszerek konkrét megoldást is adnak. Ennek megfelelően két absztrakt osztály szolgál e két típus őséül. A *BoundPresolver* esetén csak egy *double* típusú visszatérési értékkel rendelkező *CalculateBound()* metódust kell felüldefiniálni, míg a *SolutionPresolver*-ből származtatva a *GetSolution()* metódus felülírása szükséges. A heurisztikákra jellemző, hogy a fellelhető problémaosztályoknak csak egy részére képesek megoldást találni, ezért minden esetben a futás előtt el kell dönteni, hogy a bemenet megfelelő-e. Erre szolgál az *IsSolvable()* metódus, amelyet minden leszármazottnak felül kell definiálnia. Az algoritmusok a gyökér részproblémát kapják bemenetként, *TreeNode* típusú objektumként. Ezen változtathatnak is, amennyiben egyszerűsíteni tudják anélkül, hogy optimális megoldást zárnának ki a keresésből. A heurisztikus eljárásokat a fa bejárása előtt futtatjuk le. Az interfészre épülve a kutatócsoport többi tagja által több heurisztikus algoritmus is implementálásra került, amelyek szakaszos üzemű termelőrendszerek ütemezésének egy-egy csoportjára képesek megoldást adni, azonban ezek nem alkalmazhatóak AWS feladatokra.

6. Összehasonlító vizsgálatok

A tesztkörnyezet a 3.3. alfejezetben leírtakkal megegyezik. Az eredmények könnyebb megértéséért és a módszerek képességeinek összefoglalásaként a 6.1. táblázat bemutatja az egyes módszerek által kezelt feladatosztályok jellemzőit. Látható, hogy az S-gráf megoldó és a precedencia alapú módszer képes a legáltalánosabb feladatosztály kezelésére, így ezeket kell egymással összehasonlítani; a time slot alapú módszertől elvárható, hogy az egyszerűbb feladatosztályon jobban teljesítsen.

6.1. táblázat. Módszerek kezelt feladat-tulajdonságai

Tulajdonság	Eredeti time slot	Kiterjesztett time slot	Precedencia alapú	S-gráf megoldó
Permutation flow-shop 1 robottal	✓	✓	✓	✓ + job segédélek + új élbehúzás
Több robot kezelése	✓	✓	✓	✓ + job segédélek
Üres robot mozgás		✓	✓	✓
Robot mozgási zónák		✓	✓	✓
Párhuzamos kádak			✓	✓
LW korlátozás			✓	✓
Ismétlődő kádak			✓	✓

A feladatosztályok nehézségét két tulajdonság befolyásolja nagymértékben: az üres robot mozgás kezelése és a robotok száma. Az üres robot mozgással foglalkoztunk dolgozatunkban, és a következőkben a módszerek összehasonlításra kerülnek az üres robot mozgás figyelmen kívül hagyásával és figyelembevételével is. A több robotot tartalmazó feladatok hatékony megoldása viszont további kutatást igényel. Az eredeti S-gráf módszertan képes kezelni több azonos berendezést, de mivel a robotoknak sok taszkjuk van, számtalan redundáns ütemezés keletkezik, melyek csak annyiban térnek el, hogy egy-egy szállítást másik robot végez. Ezen redundancia csökkentése a jövőbeni terveink között szerepel.

6.1. Üres robot mozgás nélkül

A berendezés alapú szétválasztás eredményei az egyes gyorsításokkal a 6.2. táblázatban találhatóak. A jobb oldali oszlopban látható, hogy a legjobb teljesítmény úgy érhető el, ha mindkét gyorsítást alkalmazzuk. Külön-külön vizsgálva a gyorsításokat, megfigyelhető, hogy berendezés alapú szétválasztásnál a módosított élbehúzás eredményezett nagyobb gyorsulást.

6.2. táblázat. Berendezés alapú szétválasztás üres robot mozgás nélkül

	Eredeti		Módosított élek		Job segédélek		Mindkettő	
MxN	Idő (s)	Makespan	Idő (s)	Msp.	Idő (s)	Msp.	Idő (s)	Msp.
4x4	0,42	50,67	0,05	50,67	0,08	50,67	0,03	50,67
4x5	42,24	62,00	0,21	62,00	1,38	62,00	0,08	62,00
4x6	1000	(69,15)	1,07	69,15	31,59	69,15	0,38	69,15
4x7	1000	(83,60)	17,65	76,18	517,07	76,18	2,36	76,18
6x4	1,07	61,08	0,16	61,08	0,26	61,08	0,1	61,08
6x5	114,81	72,41	0,96	72,41	7,5	72,41	0,47	72,41
6x7	1000	(102,92)	86,2	90,83	1000	(94,17)	9,46	90,83
12x5	1000	(136,84)	21,1	128,90	1000	(136,84)	21,1	128,90

A taszk alapú szétválasztással mért eredményeket a 6.3. táblázat mutatja be. A módosított élbehúzás nem befolyásolta lényegesen a teljesítményt. Ez talán annak köszönhető, hogy az új élek egybeesnek a robot ütemezési éleivel, így a kádak és a robot ütemezése között redundancia lép fel. A job segédélek viszont jelentős gyorsulást értek el, és a legjobb teljesítményt itt is a két gyorsítás együttes használata érte el.

6.3. táblázat. Taszk alapú szétválasztás üres robot mozgás nélkül

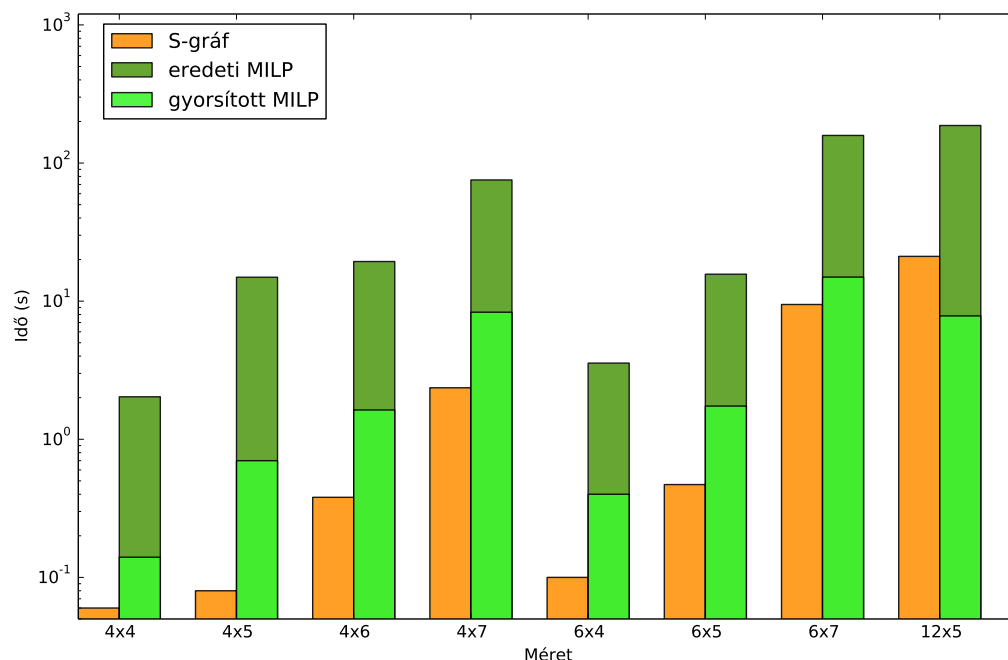
	Eredeti		Módosított élek		Job segédélek		Mindkettő	
MxN	Idő (s)	Makespan	Idő (s)	Msp.	Idő (s)	Msp.	Idő (s)	Msp.
4x4	0,18	50,67	0,18	50,67	0,1	50,67	0,1	50,67
4x5	15,27	62,00	15,38	62,00	7,74	62,00	7,46	62,00
4x6	196,65	69,15	195,04	69,15	94,77	69,15	86,01	69,15
4x7	1000	(76,18)	1000	(76,18)	1000	(76,18)	1000	(76,18)
6x4	3	61,08	3,08	61,08	1,86	61,08	1,85	61,08
6x5	1000	(72,41)	1000	(72,41)	819,6	72,41	808,92	72,41
6x7	1000	(97,98)	1000	(97,98)	1000	(97,98)	1000	(97,98)
12x5	1000	(132,48)	1000	(132,48)	1000	(134,8)	1000	(133,36)

Mindkét szétválasztás a gyorsítások együttes használatával érte el a legjobb teljesítményt, ezért ezeket az eredményeket hasonlítottuk össze a MILP módszerekével a 6.4. táblázatban. A legtöbb esetben a berendezés alapú szétválasztás volt a leggyorsabb, de túl sok kád esetén lecsökken a teljesítménye. A 12 kádat tartalmazó esetben az általunk továbbfejlesztett precedencia volt a legjobb.

A gyorsításokkal javított berendezés alapú szétválasztás és a precedencia alapú módszer könnyebb összehasonlítását segíti a 6.1. ábrán látható diagram. A futási idők a nagy különbségek miatt logaritmikus skálán vannak ábrázolva.

6.4. táblázat. Üres robot mozgás nélküli eredmények különböző módszerekkel

	S-gráf				MILP			
	Berendezés alapú		Taszk alapú		Eredeti precedencia		Gyorsított prec.	
MxN	Idő (s)	Makespan	Idő (s)	Msp.	Idő (s)	Msp.	Idő (s)	Msp.
4x4	0,03	50,67	0,1	50,67	2,03	50,42	0,14	50,67
4x5	0,08	62,00	7,46	62,00	14,9	62,00	0,7	62,00
4x6	0,38	69,15	86,01	69,15	19,36	69,15	1,63	69,15
4x7	2,36	76,18	1000	76,18	75,41	76,18	8,32	76,18
6x4	0,1	61,08	1,85	61,08	3,56	61,08	0,4	61,08
6x5	0,47	72,41	808,92	72,41	15,67	72,41	1,74	72,41
6x7	9,46	90,83	1000	(97,98)	158,41	90,83	14,94	90,83
12x5	21,1	128,90	1000	(132,48)	186,96	128,90	7,82	128,90



6.1. ábra. A leggyorsabb módszerek összehasonlítása üres robot mozgás nélkül

6.2. Üres robot mozgás figyelembevételével

Míg a MILP módszerek jellemzően lassabbak voltak ha figyelembe vették az üres robot mozgást, az S-gráf megoldó egyes példákra gyorsabb, másokra lassabb volt. A robotok ütemezései éleinek súlyozása jobb korlátozást eredményezhet és javíthatja a branch-and-bound algoritmus hatékonyságát.

A berendezés alapú szétválasztás gyorsítás nélkül meglepően rosszul teljesített, de mindkét gyorsítás jelentős javulást eredményezett külön-külön is, de a legjobb eredményt együttes használatuk jelentette. Az eredményeket a 6.5. táblázat mutatja be.

Taszk alapú szétválasztásnál ismét a job segédélek eredményeztek nagyobb gyorsulást, de itt is a két gyorsítás együtt érte el a legjobb teljesítményt, ahogy a 6.6. táblázaton látható.

6.5. táblázat. Berendezés alapú szétválasztás üres robotmozgással

	Eredeti		Módosított élek		Job segédélek		Mindkettő	
MxN	Idő (s)	Makespan	Idő (s)	Msp.	Idő (s)	Msp.	Idő (s)	Msp.
4x4	5,42	51,76	0,06	51,76	0,12	51,76	0,05	51,76
4x5	441,81	63,42	0,2	63,42	1,21	63,42	0,18	63,42
4x6	1000	—	0,52	70,92	23,72	70,92	0,29	70,92
4x7	1000	—	7,01	78,38	389,9	78,38	1,62	78,38
6x4	1000	—	0,11	62,03	0,24	62,03	0,09	62,03
6x5	1000	—	0,6	73,69	2,66	73,69	0,42	73,69
6x7	1000	—	39,04	93,47	1000	(93,47)	5,63	93,47
12x5	1000	—	19,86	130,16	1000	(136,36)	29,04	130,16

6.6. táblázat. Taszk alapú szétválasztás üres robotmozgással

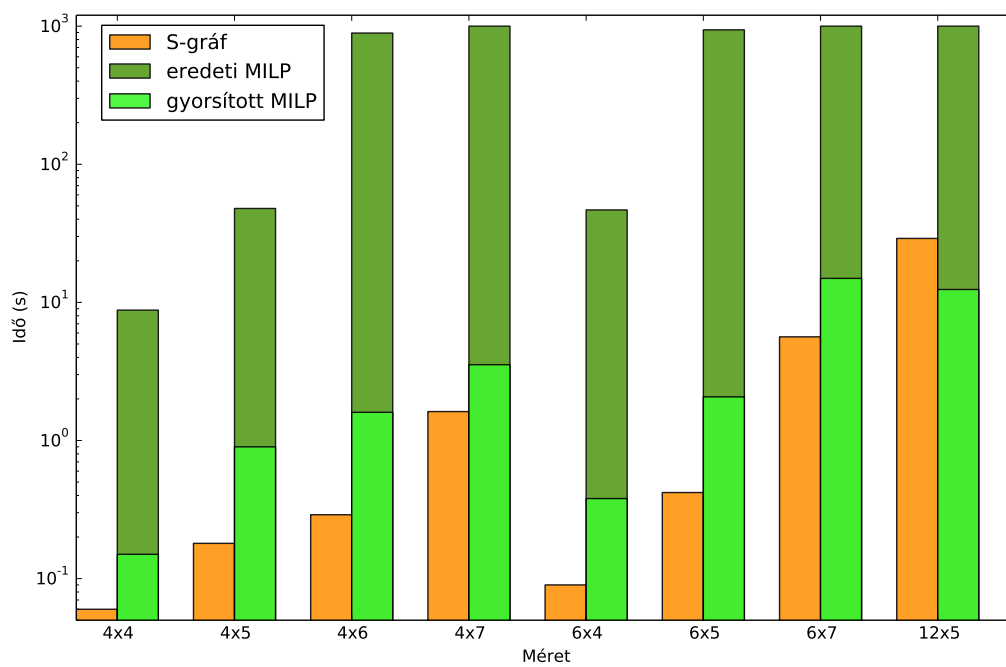
	Eredeti		Módosított élek		Job segédélek		Mindkettő	
MxN	Idő (s)	Makespan	Idő (s)	Msp.	Idő (s)	Msp.	Idő (s)	Msp.
4x4	0,14	51,76	0,14	51,76	0,09	51,76	0,08	51,76
4x5	10,81	63,42	10,23	63,42	5,8	63,42	5,25	63,42
4x6	142,74	70,92	136,48	70,92	73,13	70,92	62,18	70,92
4x7	1000	(78,38)	1000	(78,38)	1000	(78,38)	1000	(78,38)
6x4	1,74	62,03	1,71	62,03	1,11	62,03	1,03	62,03
6x5	612,34	73,69	564	73,69	315,81	73,69	288,66	73,69
6x7	1000	(100,01)	1000	(100,01)	1000	(100,01)	1000	(100,01)
12x5	1000	(135,22)	1000	(135,22)	1000	(134,84)	1000	(134,84)

A módszerek sorrendje üres robot mozgás esetén megegyezik az üres robot mozgást nélkülöző teszteknel megfigyelt eredménnyel. A berendezés alapú szétválasztás egy eset kivételével a leggyorsabb volt, azt követi a gyorsított precedencia, majd a taszk alapú szétválasztás, végül az eredeti precedencia. A pontos teszteredményeket a 6.7. táblázat mutatja be, a 6.2. ábra diagramja pedig a berendezés alapú szétválasztás és a precedencia alapú MILP módszerek összehasonlítását segíti.

6.7. táblázat. Különböző módszerek eredményei üres robot mozgással

	S-gráf				MILP			
	Berendezés alapú		Taszk alapú		Eredeti precedencia		Gyorsított prec.	
MxN	Idő (s)	Makespan	Idő (s)	Msp.	Idő (s)	Msp.	Idő (s)	Msp.
4x4	0,05	51,76	0,08	51,76	8,79	51,76	0,15	51,76
4x5	0,18	63,42	5,25	63,42	47,86	63,42	0,9	63,42
4x6	0,29	70,92	62,18	70,92	890,96	70,92	1,6	70,92
4x7	1,62	78,38	1000	(78,38)	1000	(78,38)	3,54	78,38
6x4	0,09	62,03	1,03	62,03	46,7	62,03	0,38	62,03
6x5	0,42	73,69	288,66	73,69	940,07	73,69	2,07	73,69
6x7	5,63	93,47	1000	(100,01)	1000	—	14,92	93,47
12x5	29,04	130,16	1000	(134,84)	1000	—	12,4	130,16

Összegezve az eredményeket, az elvégzett tesztek alapján elmondható, hogy az S-gráf alapú megoldás gyorsításai sikeresek voltak, és egy robot esetén a szakirodalmi módszerekkel összemérhető teljesítményre képesek.



6.2. ábra. A leggyorsabb módszerek összehasonlítása üres robot mozgással

7. Összefoglalás

A munka során megvizsgáltuk az AWS ütemezési feladat szakirodalmát. A megismert módszerek közül két meghatározó fontosságú MILP modellt implementáltunk és tovább is fejlesztettünk. Az általánosabb feladatokat is kezelő precedencia alapú módszerhez hatékony gyorsítás készült, míg a hibrid time slot modell által kezelt speciálisabb feladatosztály bővítésre került. Az S-gráf modellt kiterjesztettük AWS ütemezések leírására. A megoldó szoftverbe is integráltuk az AWS feladatok kezelését, és a hatékonyság növelésére több módszert is kifejlesztettünk. A gyorsítások hatékonyságát szakirodalmi példákon végzett tesztekkel igazoltuk. Az eredmények azt mutatják, hogy a kombinatorikus megközelítés hatékonysága több esetben felülmúlta a MILP módszereket.

Munkánk eredményei a Chemical Engineering Transactions folyóiratban is megjelentek[19]. A téma számos további kutatási lehetőséget kínál. A több robotot tartalmazó feladatok hatékony megoldásához új algoritmusokat kell kifejleszteni az S-gráf megoldóhoz. A berendezés alapú szétválasztás teljesítménye nagyban függ a berendezések kiválasztási sorrendjétől, ezért a jobb sorrendet előzetesen meghatározó heurisztikák jelentősen gyorsíthatják a megoldást. Ezek kutatása jövőbeni terveink között szerepel.

Támogatás elismerése

A publikáció az Európai Unió, Magyarország és az Európai Szociális Alap társfinanszírozása által biztosított forrásból a TÁMOP-4.2.2.C-11/1/KONV-2012-0004 azonosítójú „Nemzeti kutatóközpont fejlett infokommunikációs technológiák kidolgozására és piaci bevezetésére” című projekt támogatásával jött létre.

Irodalomjegyzék

- [1] **Adrián M. Aguirre, Carlos A. Méndez, Álvaro García-Sánchez, Miguel Ortega-Mier, and Pedro M. Castro.** “General Framework for Automated Manufacturing Systems: Multiple Hoists Scheduling Solution”. In: *Chemical Engineering Transactions* 32 (2013), pp. 1381–1386. DOI: 10.3303/CET1332231.
- [2] **Christopher D. Geiger, Karl G. Kempf, and Reha Uzsoy.** “A Tabu search approach to scheduling an automated wet etch station”. In: *Journal of Manufacturing Systems* 16.2 (1997), pp. 102–116. DOI: 10.1016/S0278-6125(97)85674-9.
- [3] **Swarnendu Bhushan and I.A. Karimi.** “Heuristic algorithms for scheduling an automated wet-etch station”. In: *Computers & Chemical Engineering* 28.3 (2004), pp. 363–379. DOI: 10.1016/S0098-1354(03)00192-3.
- [4] **Carlos A. Méndez, Jaime Cerdá, Ignacio E. Grossmann, Iiro Harjunkoski, and Marco Fahl.** “State-of-the-art review of optimization methods for short-term scheduling of batch processes”. In: *Computers & Chemical Engineering* 30.6-7 (2006), pp. 913–946. DOI: <http://dx.doi.org/10.1016/j.compchemeng.2006.02.008>.
- [5] **Swarnendu Bhushan and I. A. Karimi.** “An MILP Approach to Automated Wet-Etch Station Scheduling”. In: *Industrial & Engineering Chemistry Research* 42.7 (2003), pp. 1391–1399. DOI: 10.1021/ie020296c. eprint: <http://pubs.acs.org/doi/pdf/10.1021/ie020296c>.
- [6] **I.A. Karimi, Zerlinda Y.L. Tan, and Swarnendu Bhushan.** “An improved formulation for scheduling an automated wet-etch station”. In: *Computers & Chemical Engineering* 29.1 (2004), pp. 217–224. DOI: 10.1016/j.compchemeng.2004.07.007.
- [7] **Adrián M. Aguirre, Carlos A. Méndez, and Pedro M. Castro.** “A novel optimization method to automated wet-etch station scheduling in semiconductor manufacturing systems”. In: *Computers & Chemical Engineering* 35.12 (2011), pp. 2960–2972. DOI: 10.1016/j.compchemeng.2011.02.014.
- [8] **Adrián M. Aguirre, Carlos A. Méndez, Gloria Gutierrez, and Cesar De Prada.** “An improvement-based MILP optimization approach to complex AWS scheduling”. In: *Computers & Chemical Engineering* 47 (2012), pp. 217–226. DOI: 10.1016/j.compchemeng.2012.06.036.
- [9] **Pedro M. Castro, Luis J. Zeballos, and Carlos A. Méndez.** “Hybrid time slots sequencing model for a class of scheduling problems”. In: *AIChE Journal* 58.3 (2012), pp. 789–800. DOI: 10.1002/aic.12609.

- [10] **Luis J. Zeballos, Pedro M. Castro, and Carlos A. Méndez.** “Integrated Constraint Programming Scheduling Approach for Automated Wet-Etch Stations in Semiconductor Manufacturing”. In: *Industrial & Engineering Chemistry Research* 50.3 (2011), pp. 1705–1715. DOI: 10.1021/ie1016199. eprint: <http://pubs.acs.org/doi/pdf/10.1021/ie1016199>.
- [11] **Juan M. Novas and Gabriela P. Henning.** “A comprehensive constraint programming approach for the rolling horizon-based scheduling of automated wet-etch stations”. In: *Computers & Chemical Engineering* 42 (2012), pp. 189–205. DOI: 10.1016/j.compchemeng.2012.01.005.
- [12] **E. Sanmarti, T. Holczinger, L. Puigjaner, and F. Friedler.** “Combinatorial framework for effective scheduling of multipurpose batch plants”. In: *AIChE Journal* 48.11 (2002), pp. 2557–2570.
- [13] **S. Ferrer-Nadal, E. Capón-García, C. A. Méndez, and L. Puigjaner.** “Material transfer operations in batch scheduling. A critical modeling issue”. In: *Industrial & Engineering Chemistry Research* 47 (2008), pp. 7721–7732.
- [14] **M. Hegyháti, T. Majozi, T. Holczinger, and F. Friedler.** “Practical infeasibility of cross-transfer in batch plants with complex recipes: S-graph vs MILP methods”. In: *Chemical Engineering Science* 64.3 (Feb. 2009), pp. 605–610. DOI: 10.1016/j.ces.2008.10.018.
- [15] **R. Adonyi.** “Batch process scheduling with the extensions of the S-graph framework”. PhD thesis. Doctoral School of Information Science and Technology - University of Veszprem, 2008.
- [16] **J. Smidla and I. Heckl.** “S-graph based parallel algorithm to the scheduling of multipurpose batch plants”. In: *Chemical Engineering Transactions* 21 (2010), pp. 937–942. DOI: 10.3303/CET1021157.
- [17] **Dan Zuras et al.** *IEEE Standard for Floating-Point Arithmetic*. Tech. rep. IEEE, Aug. 2008, pp. 1–70. DOI: 10.1109/ieeestd.2008.4610935.
- [18] **M. Hegyháti, T. Holczinger, A. Szoldatics, and F. Friedler.** “Combinatorial Approach to Address Batch Scheduling Problems with Limited Storage Time”. In: *Chemical Engineering Transactions* 25 (2011), pp. 495–500. DOI: 10.3303/CET1125083.
- [19] **M. Hegyháti, O. Ósz, B. Kovács, and F. Friedler.** “Scheduling of Automated Wet-Etch Stations”. In: *Chemical Engineering Transactions*, 39 (2014), pp. 433–438. DOI: 10.3303/CET1439073.

A. Függelék: Job-at-a-time heurisztika

A használt változók listája:

- Θ : szállítási időintervallumok halmaza
- $TS_{k,j}$: k . termék j . taszkjának kezdete

A paraméterek listája:

- π_j : $j - 1$ -edik és j -edik berendezés közötti szállítás ideje
- $t_{(k,j)}$: k . termék j . taszkjának végrehajtási ideje

$$TS_1 = \max(TS_{k,j-1} + t_{k,j-1}, \pi_{j-1}, TS_{k-1,j+1} + \pi_{j-1}, TS_{k-1,j+2} - t_{k,j}) \quad (A.1)$$

$$TS_2 = TS_1 + t_{k,j} + \pi_j \quad (A.2)$$

Az algoritmus lépései:

1. Inicializálás: Θ -t és a $TS_{0,j}$, $j = 1 - (M + 1)$
2. $k := j := 1$
3. $TS_{k,j}$ és $TS_{k,j+1}$ kiszámítása A.1 és A.2 egyenletekből
4. Ha a robot szabad a $[TS_{k,j} - \pi_j, TS_{k,j}]$, illetve $[TS_{k,j+1} - \pi_j, TS_{k,j+1}]$ intervallumokban, akkor ezeket hozzáadja Θ -hoz, ha nem, akkor megkeresi a legkorábbi¹ szabad intervallumot, és azt adja hozzá
5. Ha $j < M$, akkor $j := j + 2$, és vissza a 3. lépéshez
6. Ha $k < N$, akkor $k := k + 1$, $j := 1$, és vissza a 3. lépéshez
7. Eredmény: $TS_{N(M+1)}$ a makespan

¹A szerzők dolgozata nem nyújt útmutatást ahhoz, hogy hogyan állapítható meg a legkorábbi szabad intervallum a mozgathatósághoz.

B. Függelék: Makespan minimalizálás

A formális leírásához egy S-gráf megadható $G(N, A_1, A_2)$ alakban, ahol N a csúcsok, A_1 a receptélek, A_2 pedig az ütemezési élek halmaza. Jelölje c_{kl} a k és l csúcsok közti él súlyát, egy EQ berendezés által végrehajtható taszkok halmazát pedig N_{EQ} . A MAIN eljárást az 5. algoritmus mutatja be.

A változók elnevezése a következő:

solution: az eddig megtalált legjobb ütemezés gráfját tárolja

current_best: a *solution* makespanje

SOUN: még nem ütemezett taszkok halmaza

SET: nyitott részproblémák halmaza

PP: aktuális részprobléma

bound: a részprobléma alsó korlátja

last_node: $\left\{ (EQ, t) \mid \begin{array}{l} t \in N_{EQ} \text{ az } EQ \text{ berendezésbe utolsóként ütemezett taszk,} \\ \text{vagy } t = 0, \text{ ha } EQ \text{ nem hajt végre több taszkot} \end{array} \right\}$

A MAIN eljárás bemenete egy receptgráf ($A_2 = \emptyset$) a berendezések N_i halmazaival, kimenete egy optimális ütemezési gráf – ha létezik. A SELECT eljárás a nyílt részproblémákból választ egyet, megvalósítása többféleképpen lehetséges.

Az algoritmus legfontosabb része a szétválasztásért felelős BRANCHING eljárás. A szétválasztást megvalósító eljárás bemenete egy részprobléma, mely áll egy S-gráfból, a korábban kiszámolt alsó korlátból, valamint a *last_node* és *SOUN* halmazokból. Az eljárás generálja a részprobléma gyerekeit, és hozzáadja őket a *SET* halmazhoz. A berendezés alapú szétválasztás pszeudokódja megtalálható a 6. algoritmusban.

5. algoritmus A MAIN eljárás

```
procedure MAIN( $G(N, A_1, \emptyset), N_i(i = 1, 2, \dots, n)$ )  
   $solution := G(N, A_1, \emptyset)$   
   $bound := 0$   
   $SOUN := \bigcup_{i=1}^n N_i$   
   $last\_node := \emptyset$   
   $current\_best := \infty$   
   $SET := \{(G(N, A_1, \emptyset), bound, last\_node, SOUN)\}$   
  while  $SET \neq \emptyset$  do  
     $PP := \text{SELECT}(SET)$   
     $SET := SET \setminus \{PP\}$   
    BRANCHING( $PP$ )  
  end while  
  if  $current\_best < \infty$  then  
    return  $solution$   
  end if  
end procedure
```

6. algoritmus Berendezés alapú szétválasztási eljárás

procedure BRANCHING($G(N, A_1, A_2)$, $bound$, $last_node$, $SOUN$)

Let EQ be a unit such that $N_{EQ} \cap SOUN \neq \emptyset$ and $(EQ, 0) \notin last_node$

for all $k \in N_{EQ} \cap SOUN$ **do**

$G^k(N, A_1^k, A_2^k) := G(N, A_1, A_2)$

$last_node^k := last_node$

$SOUN^k := SOUN \setminus \{k\}$

for all $(k, l) \in A_1^k$ **do**

 UPDATE($c(k, l)$) in A_1^k

end for

if $\exists j (EQ, j) \in last_node$ **then**

for all $(j, l) \in A_1^k, l \neq k$ **do**

$A_2^k := A_2^k \cup \{(l, k)\}$

end for

$last_node^k := last_node^k \setminus \{(EQ, j)\}$

end if

$bound^k := BOUNDING(G^k(N, A_1^k, A_2^k))$

if $bound^k < current_best$ **then**

if $SOUN^k = \emptyset$ **then**

$solution := G^k(N, A_1^k, A_2^k)$

$current_best := bound^k$

else

$SET := SET \cup \{(G^k(N, A_1^k, A_2^k), bound^k, last_node^k, eq_task_pairs^k, SOUN^k)\}$

end if

end if

end for

if $\forall k \in N_{EQ} \cap SOUN \exists EQ' \neq EQ, k \in N_{EQ'}, (EQ', 0) \notin last_node$ **then**

$G^*(N, A_1^*, A_2^*) := G(N, A_1, A_2)$

$last_node^* := last_node$

for all $k \in N_{EQ}$ **do**

 UPDATE(k, l) in A_1^*

end for

if $\exists (EQ, j) \in last_node$ **then**

$last_node^* := last_node^* \setminus \{(EQ, j)\}$

end if

$last_node^* := last_node^* \cup \{(EQ, 0)\}$

$bound^* := BOUNDING(G^*(N, A_1^*, A_2^*))$

if $bound^* < current_best$ **then**

$SET := SET \cup \{(G^*(N, A_1^*, A_2^*), bound^*, last_node^*, eq_task_pairs, SOUN)\}$

end if

end if

end procedure
