Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Elektronikai Technológia Tanszék

Beluzsár János

HIERARCHIKUS, SORRENDFÜGGŐ TERMELÉSÜTEMEZÉSI PROBLÉMA MEGOLDÁSA

TDK dolgozat

KONZULENS

Dr. Szikora Béla

BUDAPEST, 2014

Tartalomjegyzék

1. Bevezetés

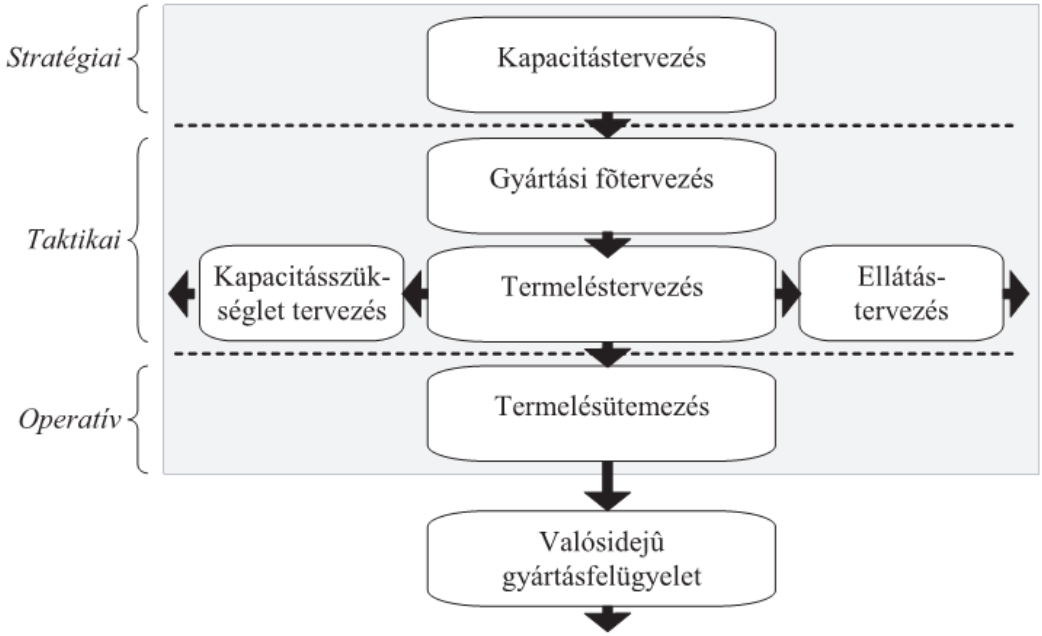
Napjaink nagyvállalatainak termelési folyamatai dinamikusan változnak. A vevői igények gyors módosulása, az egyre rövidülő gyártási átfutási idők, és termék életciklusok, az erős piaci verseny, vagy a növekvő innovációs nyomás következtében a termelő vállalatok számára létfontosságúvá vált a napi gyártási folyamatok egyszerű, gyors és költséghatékony megtervezése. Ehhez nagy segítséget nyújtanak a piacon lévő integrált vállalatirányítási rendszerek (ERP) termelési moduljai, vagy a szigetrendszerként működő termeléstervező modulok. Ugyanakkor ma már sok esetben egy egyszerű anyagszükséglet számító (Material Requirements Planning - MRP) vagy kapacitásszükséglet számító (Capacity Requirements Planning - CRP) algoritmus már messze nem elég a gyártástervezés megbízható és hatékony támogatására.

Napjaink feszített tempójú termelési környezetei sok esetben igénylik az akár percre lebontott, részletes termelésütemezést, ami kulcs szerepet játszhat a megfelelő színvonalú ügyfél-kiszolgálásban és a vállalt határidőre történő maradéktalan teljesítésben. Ezen folyamat során rengeteg körülményt kell figyelembe venni (pl. rendelkezésre álló gépek, termeléshez szükséges dolgozók száma, határidők, tervszerű karbantartások miatti állásidők, alapanyag ellátottság), melyek sok esetben nagyon nehezen követhetők valós időben. Ugyanakkor a döntések meghozatalához ezekre az adatokra mind szükség van, ezért nem hagyhatjuk egyiket sem figyelmen kívül.

Dolgozatomban a termelésütemezés definiálása után egy gyár műanyag alkatrészgyártás ütemezésének legfontosabb feladatait, körülményeit és céljait fogom bemutatni. Felvázolom azokat a szempontokat, melyeket a termelés ütemezőknek mindenképpen szem előtt kell tartaniuk, illetve bemutatom, hogy a folyamatot hogyan lehet célszoftverekkel támogatni. Ezt követően bemutatom a követelmények leírására kifejlesztett matematikai modellemet, melyet egy speciális fejlesztőkörnyezet, az IBM ILOG CPLEX Optimization Studio segítségével implementálok és megoldok. A probléma megoldására többszintű, hierarchikus célfüggvényeket kell alkalmaznom, melyek lehetővé teszik, hogy a különböző fontosságú vállalati célok mentén lehessen automatikusan ütemezni a gyártási feladatokat. Végül a kifejlesztett algoritmus és megoldási módszer teljesítményét egy valós környezetből származó adathalmazon mutatom be.

2. A termelésütemezés általános feladata

A termelésütemezés, vagy finomprogramozás a vállalatok termeléstervezési folyamatainak utolsó lépése, ami minden termelő vállalat életében megjelenik különböző részletezettséggel. Feladata, hogy fokozatosan levezesse a vállalati célokból kiindulva, a rendelkezésre álló erőforrásokat és vevői igényeket figyelembe véve a részletes termék kibocsátási terveket, melyek megadják, hogy egy adott napon mely termékekből, mekkora mennyiséget és milyen erőforrások igénybevételével kell legyártani a célok elérése érdekében. Ez a folyamat többszintű, hierarchikus tervezést igényel, ahol minden szintnek megvannak a saját céljai, határai, és egy alsóbb szint végrehajtása során minden esetben figyelembe kell venni a felsőbb szint eredményeit, korlátait. Az egymásra épülést, és az egyes szinteken megvalósítandó feladatokat mutatja be a 2.1 ábra.



2.1. ábra: A termeléstervezés hierarchikus felépítése [1]

A hierarchia legalsó, operatív szintjén helyezkedik el a termelésütemezés. Ennek feladata, hogy néhány napos időtávon belül a taktikai tervezés eredményeként kapott részletes termék kibocsátási tervekből előállítsa a megvalósítható gyártási tervet az üzem konkrét napi kapacitásait, termelési körülményeit figyelembe véve. Ezen a szinten kell a gyártási rendelések (továbbiakban GYR) műveleteit konkrét gyártó berendezésekre ütemezni, a

rendelések közötti sorrendet felállítani, a szükséges alapanyagokat és munkaerőt egy művelet megkezdésére biztosítani [2].

A finomprogramozás során tehát egy már végleges GYR állományból kell kiindulni, amelynek mindegyik eleme rendelkezik egyedékességi dátummal. Ez határozza meg azt az időpontot, amikorra az adott rendelésen gyártott terméknek legkésőbb el kell készülnie annak érdekében, hogy a kiszállítás is határidőre megtörténhessen. Egy-egy ilyen GYR ugyanakkor egy komplex gyártási folyamatot ír le, amelyhez különböző gyártó berendezéseket különböző időtartamban kell igénybe venni. Éppen ezért a gyártási rendelés tovább bontható elemi műveletekre, melyek meghatározzák, hogy egy munkafázist melyik gépen és mennyi ideig kell végezni. A finomprogramozás alatt tulajdonképpen ezeket a műveleteket kell összerendelni egy adott időintervallumra a megfelelő gyártó berendezésekkel úgy, hogy egy megvalósítható tervet kapjunk. Az összerendelés során ugyanakkor sok korlátozó feltételt kell figyelembe venni ahhoz, hogy jó eredményt érjünk el [3].

Egyrészt mivel a GYR-eken belül a műveletek technológiai sorrendet írnak elő, ezért kötött sorrendjük van, amelyet az ütemezés során figyelembe kell venni (rendszerint a műveleteket sorszámozzák, és az ütemezést úgy készítik el, hogy egy magasabb sorszámú ne kerülhessen egy alacsonyabb sorszámú elé egy GYR-en belül).

Másrészt időadatok nem csak a műveletvégzésre, hanem a műveletközi tevékenységekre is előírhatnak követelményeket, korlátozásokat. Elképzelhető, hogy egy nagy üzemben a gépek fizikai elhelyezkedése miatt számottevő időt kell arra fordítani, hogy a munkadarabokat az egyik műveletvégzési helyről a másikra szállítsák. Ennek eredményeként a tervet is úgy kell tudni elkészíteni, hogy ezen két műveletet ne lehessen közelebbre tervezni egymáshoz, mint az anyagmozgatás időtartama. Hasonlóképpen előfordulhat, hogy egy gépen két egymást követő művelet között van ilyen időtartam előírás (pl. élelmiszeriparban, ha egy gépen glutén tartalmú terméket gyártottak, majd utána gluténmentes terméket fognak, akkor egy nagyon időigényes mosási ciklust kell beiktatni, amivel a tervezés során számolni kell).

Harmadrészt figyelni kell ara, hogy a gyártó berendezések nem rendelkeznek végtelen kapacitásokkal. Az esetek nagy részében egy gépen egy időpillanatban csak egy művelet hajtható végre, amit szintén nem szabad szem elől téveszteni. Továbbá a berendezések jól meghatározott karbantartási ciklusokkal rendelkeznek, melyek időtartamára ki kell vonni azokat a termelésből, így ekkor nem lehet velük az ütemezés kapcsán számolni. Valamint tekintettel kell lenni a szokásos heti munkarendre, az állásidők valamint a műszakok közötti esetleges szünetek betartására.

Végül, de nem utolsó sorban valós ipari környezetben egyáltalán nem ritka az, hogy nem csak GYR-en belüli műveleteknek van egy előre meghatározott sorrendje, hanem az egyes elvégzendő GYR-ek között is van ilyen megkötés. Ezek többnyire olyankor fordulnak elő, amikor egy rendeléshez felhasznált alkatrész egy másik rendelés eredményeként áll elő, vagy pl. egy beruházási projekt esetén az engedélyezési eljárásnak meg kell előznie a megvalósítási szakaszt. Ezeket a precedenciákat, az előző információkhoz hasonlóan szintén ismerni kell, és számolni kell velük az ütemezés során.

A leírtak alapján jól látható, hogy rengeteg korlátozó feltétel mellett kell tudni megalkotni egy megvalósítható gyártási tervet méghozzá úgy, hogy egy előre definiált célnak minél inkább megfeleljen. Ez a konkrét cél többféle lehet, az viszont általánosan elmondható, hogy mindegyik a termelés hatékonyságának növelésével, a költségek vagy a késések csökkentésével áll szoros kapcsolatban.

3. Termelésütemezés a műanyagalkatrész gyártásban

Az előzőekben bemutatott általános termelésütemezési igényekből kiindulva ebben a fejezetben ismertetem, hogy melyek azok a legfontosabb kritériumok és célok, amely mentén egy gyárban (nevezzük Minta Vállalatnak) a műanyagalkatrész gyártás ütemezését végezik, illetve milyen informatikai eszközökkel lehet a folyamatot támogatni. Mindezek figyelembe vételével felvázolom a probléma matematikai formalizációját, amiből kiindulva a későbbiekben elkészítem a feladat modelljét és eljutok a megoldáshoz az IBM ILOG CPLEX Optimization Studio segítségével.

3.1 Az ütemezés célja, bemenő adatai és korlátozó tényezői

A műanyagalkatrész gyártás ütemezéséhez a kiinduló adatok az alkatrészek felhasználásával késztermékeket készítő szerelőműhelyektől érkeznek, mint a gyártmánystruktúra felsőbb szintjeiből eredő, ún. függő igények (dependent demand). Ezek alapján az operatív tervezés megkezdésekor már rendelkezésre áll az az információ, hogy melyik alkatrészekből, mekkora mennyiséget és milyen határidővel kell elkészíteni. Ezen kívül a gyártástechnológia is bizonyos mértékben meghatározásra kerül, ugyanis a termelésütemező részére átadott tervben már az is szerepel, hogy az egyes alkatrészeket milyen szerszám használatával kell legyártani. A szerszámok határozzák meg a fröccsöntés során keletkező alkatrészek alakját, illetve behatárolják azt is, hogy az egyes fröccsöntő gépek közül melyikre lehet az adott szerszámmal készített termék gyártását ütemezni. Ezt a taktikai szintű tervezést végző szakemberek határozzák meg, ugyanis ők azok, akik a szerszámok életútját folyamatosan nyomonkövetik, tervezett megelőző karbantartásaikat, felújításaikat, ellenőrzéseiket ütemezik, így ők tudják azt meghatározni, hogy adott időpontban melyik szerszám lesz elérhető a gyártásban. Mindez azért kiemelten fontos, mivel a szerszámok gyártása és fejújítása nagyon költséges, így arra kell törekedni, hogy minél hosszabb időn keresztül és minél jobb minőségben legyenek képesek az alkatrészek legyártására. Éppen ezért a továbbiakban a szerszámokat használhatónak fogom tekinteni, ezt a technológiai korlátot az ütemezési feladat megfogalmazása és megoldása során figyelmen kívül fogom hagyni.

Szintén a szerszámokhoz kapcsolódik egy másik speciális körülmény, amire az ütemezés során figyelni kell. A szerszámok gyártástechnológiájából, illetve költségcsökkentési okokból eredően lehetnek olyan szerszámok, melyek egy időpillanatban egyszerre több, különböző alkatrész előállítására alkalmasak. Vagyis ha a taktikai szinten a gyártási feladatot egy ilyen szerszámhoz rendelték hozzá, akkor meg kell vizsgálni, hogy van-e olyan gyártási rendelés, ami az adott szerszámmal gyártható másik alkatrészre szól, esedékessége megegyezik a másik gyártási rendeléssel. Ezekben az esetekben a gépfoglalások szempontjából az a helyes eljárás, hogyha csak az egyik ilyen GYR megfelelő művelete kerül be az adott gépre ütemezett feladatok közé, míg a kapcsolódó másik GYR műveletét az ütemezés során nem vesszük figyelembe, csak a folyamat végeztével rendeljük hozzá ugyanazt a gépet és időszakot, mint az ütemezett művelethez. A továbbiakban úgy tekintem, hogy az egyszerre több alkatrészt is előállító szerszámok esetében a bemenő adatok között csak az egyik GYR megfelelő művelete szerepel a kettőből, és az ütemezést ennek az egynek besorolásával kell elvégezni. A kapcsolódó másik GYR vonatkozó műveletének gyártógépre sorolása a kész ütemezés alapján utólag történik meg.

A gyártandó mennyiségek és határidők alapján a termelésütemezőnek a gyártási feladatokat be kell sorolni a kompatibilis gyártógépekre. Ezen folyamat során többségében minden gyártási rendelés esetében egy műveletet, a fröccsöntést kell ütemezni, ugyanakkor speciális alkatrészek, elsősorban külső burkolatok esetében lehetséges, hogy a fröccsöntésen kívül egy tamponozási műveletet is el kell végezni, amely során azonosító adatokat (pl. típus, származási ország, gyártás éve) visznek fel az alkatrészre. Ilyen esetben a műveletek között szigorú precedencia áll fenn, vagyis az elkészült ütemezésben a fröccsöntésnek mindenképpen meg kell előznie a tamponozást.

A műveleti sorrendek figyelembe vételén túl szükség van arra is, hogy a műveleteket úgy rendeljük gyártóberendezésekhez, hogy az adott műveletet az adott gépen engedélyezett legyen elvégezni. Ennek érdekében bemenő információként rendelkezésre áll az, hogy az adott terméket melyik fröccsöntő gépeken lehet gyártani, illetve, hogy a gyártási folyamat mekkora ideig foglalja a gép kapacitását. Ezekből kiindulva tehát az ütemezés során nem csak azt kell eldönteni, hogy egy GYR egy műveletét milyen időszakban kell elvégezni, hanem azt is, hogy melyik gyártóberendezésen.

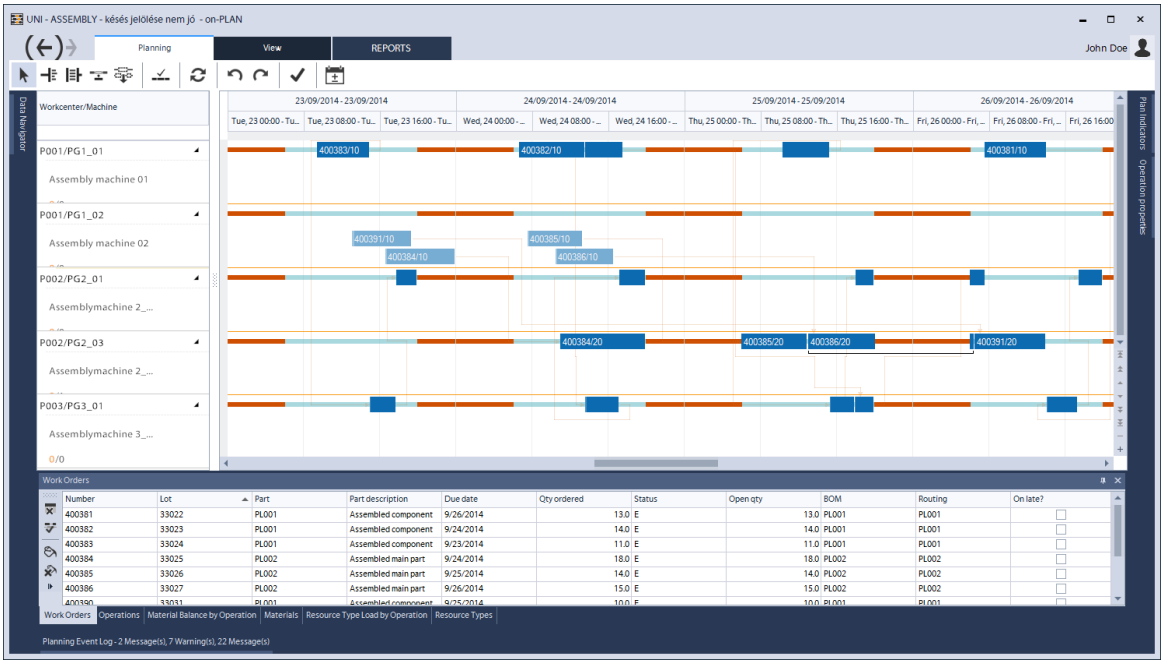
A műveleti sorrendek, és az alternatív gyártó berendezések figyelembe vételén kívül a műanyagalkatrész gyártás ütemezése során tekintettel kell lenni az egy gépen egymás után végzett műveletek között esedékes technológiai időkre is. Ez nem jelent mást, mint az

egymást követő gyártások által igénybe vett eltérő szerszámok cseréjének időtartamát az adott gépen, melynek értéke a gyártó berendezés méretétől függ.

A bemenő adatok és a korlátozó feltételek ismeretében a cél olyan ütemezés készítése, amely lehetővé teszi, hogy gyártási feladatokat, a lehető legkevesebb késedelemmel, azaz esedékességi dátum utáni befejezéssel lehessen elvégezni. Ezen kívül másodlagos célként megjelenik a felesleges gépállásidők kiküszöbölése, ezáltal a termelés folyamatosságának biztosítása. Ez a gyakorlatban azt jelenti, hogy az ütemezés során törekedni kell arra is, hogy a szerszámcserékből eredő állásidők összessége minél kevesebb legyen.

3.2 A folyamat támogatása informatikai eszközökkel

Az előző fejezetben bemutatott ütemezési feladat elvégzésének támogatására napjainkban számos célszoftver kapható. Ezek rendszerint egy integrált vállalatirányítási rendszerhez kapcsolódnak, ami a tervezéshez szükséges bemenő adatokat biztosítja, illetve ahova az elkészült gyártási tervet a célszoftver visszatölti. Ezen kívül többnyire rendelkeznek egy Gantt-diagramon alapuló grafikus tervezőfelülettel is, ahol a gyártási feladatok áttekinthetők időben és a használt gépek tekintetében egyaránt. Itt lehetőség van az ütemterv egyszerűbb algoritmus szerinti optimalizálására, majd kézi módosítására figyelembe véve a véges gépkapacitásokat, illetve a 2. fejezetben ismertetett egyéb korlátoz feltételeket. Egy ilyen termelésütemező szoftver, a QAD EA ERP rendszerhez készült on-PLAN ütemező felületét mutatja az 3.1 ábra [4].



3.1. ábra: Az on-PLAN termelésütemező program tervezőfelülete

Ezen termékek többsége megfelelő támogatást biztosít az ütemezés manuális előállításához, interaktív kezelőfelületük segítségével a műveletek gyorsan és könnyen áthelyezhetők, a kapacitások tervezett kihasználtsága, a GYR-ek átfutási ideje egyszerűen áttekinthető. Ezekre a funkciókra nagy szükség van minden termelő vállalat esetében, hiszen az előre nem látható helyzetekben (pl. váratlan gépmeghibásodás, vevői igények hirtelen megváltozása, késedelmes beszállításból eredő alapanyaghiány) ezek a funkciók biztosítják a tervezők számára a gyors és hatékony beavatkozás lehetőségét.

Ugyanakkor az említett szoftverek többsége egyáltalán nem, vagy nem elég széleskörűen nyújt lehetőséget a terv jóságának az ellenőrzésére, illetve sok esetben hiányoznak belőlük olyan megoldások, amelyek segítségével előre definiált célfüggvény mentén, automatikusan lehetne egy adott körülmények között optimális ütemezést előállítani. Egy ilyen funkció viszont a napi rutin feladatok elvégzését nagyban elő tudja segíteni, mivel minden esetben ugyanolyan szempontok figyelembe vételére van szükség, így a folyamat nagymértékben automatizálhatóvá válik.

3.3 A probléma matematikai modellje

A 3.1 fejezetben ismertetett termelésütemezési feladatot a szakirodalom job-shop scheduling problémaként (JSP) tartja számon. A cél *n* db elvégzendő feladat (job, $J = \{J_i\}_{i=1}^{n}$) műveleteinek (operation, $O = \{O_{ik}\}_{k=1}^{m_i}$) ütemezése az m darab erőforrásra (machine, $M = \{M_l\}_{l=1}^{m}$) valamilyen célfüggvény mentén. Csak olyan megoldás fogadható el, amelynél egy időpontban egy erőforrásra csak egy művelet van ütemezve, illetve egy művelet végrehajtásához csak egy erőforrást vesz igénybe [4]. Az eddig ismertetett paraméterek mindegyike beazonosítható a megoldandó problémában. A feladatok a gyártási rendeléseknek feleltethetők meg, melyek műveleteit kell besorolni a rendelkezésre álló gépekre, melyeket az erőforrások jelképeznek. A műanyagalkatrész gyártás esetében is alapkövetelmény az, hogy egy művelet végrehajtásához egyszerre csak egy gépet vegyünk igénybe, az egy gépen történő párhuzamos munkavégzés tilalmára vonatkozó korlátozást pedig szintén be lehet tartani, annak ellenére, hogy előfordulhat olyan eset, hogy egy időpontban egy gép két különböző alkatrész gyártását is el kell, hogy végezze. Ez úgy lehetséges, hogy a bemenő adatok között azon műveletek közül, melyeket párhuzamosan ugyanazon a gépen kellene végrehajtani, már csak az egyik szerepel, így az ütemezés során már csak azt az egyet kell figyelembe venni.

A 3.1 fejezetben ismertetett feladat egyik sajátossága a klasszikus JSP-hez képest, hogy a megadott bemenő adatok között az O_{ik} műveleteket végrehajtó M_l erőforrások nincsenek kizárólagosan hozzárendelve a művelethez, hanem az ütemezés feladata eldönteni azt is, hogy a kompatibilis gyártógépek közül pontosan melyik az, amelyiken a megmunkálást el kell végezni. A JSP ezen speciális változatát felxible job-shop scheduling problémának (FJSP) nevezik. [5] Az ilyen feladatok megoldásához bemenő adatként szükség van arra, hogy minden O_{ik} művelethez adott legyen egy $\mu_{ik} \subseteq M$ összerendelés, ami azokat az erőforrásokat tartalmazza, amelyen az O_{ik} művelet elvégezhető. Ezen kívül minden $M_l \in \mu_{ik}$ alternatív erőforráshoz adottnak kell lennie egy P_{ikl} időtartamnak, ami meghatározza, hogy mennyi a műveletvégzési idő abban az esetben, ha az O_{ik} művelet az M_l erőforráson kerül végrehajtásra. Ez utóbbi adat erőforrásról erőforrásra eltérhet, hiszen a különböző alternatív műveletvégző gépek típusa, teljesítménye különböző lehet, ami azt eredményezi, hogy eltérő idők alatt tudják elvégezni ugyanazt a műveletet.

A bemutatott feladat másik sajátossága, hogy kötelező technológiai időket kell betartani a szerszámcserék miatt az egyes műveletvégző gépeken attól függően, hogy mely feladatok követik egymást rajta. Ezen időtartamoknak mindenképpen el kell telniük két művelet között, és gépidőt is foglalnak, azaz a megadott intervallumba más művelet nem ütemezhető. A JSP feladatok modellezése során az ilyen jellegű kritériumot sorrendfüggő felkészítési idő korlátnak (sequence-dependent setup time constraint) nevezik [6]. Ahhoz, hogy ezt a probléma megoldása során figyelembe lehessen venni, szükség van arra, hogy tudjuk mekkora a felkészítési ideje az O_{ik} műveletnek az M_l erőforráson abban az esetben, ha közvetlenül megelőzi azt az $O_{op} \in O$ művelet. Ennek jelölésére a továbbiakban az SS_{ikopl} jelölést használom.

Végül, de nem utolsó sorban bemenő információként adott minden J_i feladathoz a D_i esedékességi dátum (due date), amikorra a feladatot mindenképpen be kell fejezni. A gyakorlatban amennyiben a feladat utolsó művelete ezen időpont után fejeződik be, akkor a gyártási rendelést késedelmesnek tekintjük, ami a határidőre történő vevőkiszolgálás sérülését vonhatja maga után. Ezen kívül, a feladat megoldása során a célfüggvénnyel összhangban figyelni kell a késedelmes GYR-eket is. Ugyanakkor mivel nem minden GYR tartalmaz ugyanannyi műveletet, ezért az ütemezés során fontos tudni azt is, hogy egy O_{ik} művelet utolsó művelete-e a J_i feladatnak. Ennek meghatározására bemenő adatként az alábbi bináris változó értékét kell még ismerni.

$$L_{ik} = \begin{cases} 1, ha\ O_{ik}\ a\ J_i\ utolsó\ művelete \\ 0, egyébként \end{cases} \quad (3.1)$$

A feladat megoldásához két döntési változó definiálására van szükség, amelyek számított értéke az ütemezés végeredményét határozza meg. Ezek közül az egyik az S_{ik}, amely az O_{ik} művelet kezdő időpontját adja meg. A másik pedig egy bináris változó, amely az alábbi értékeket veheti fel.

$$SM_{ikl} = \begin{cases} 1, ha\ az\ O_{ik}\ művelet\ az\ M_l\ gépen\ kerül\ végrehajtásra \\ 0, egyébként \end{cases} \quad (3.2)$$

Miután minden bemenő- és döntési változó adott, ezek alapján definiálható a célfüggvény is. A 3.1 fejezetben kétszintű, hierarchikus célrendszert fogalmaztam meg. A Minta Vállalat esetében elsődleges cél az, hogy az ütemezésből eredő összes késés minimális legyen. Ezt az alábbi célfüggvény írja le.

$$\min(\sum_J L_{ik} * \max(0, S_{ik} + \sum_M (SM_{ikl} * P_{ikl}) - D_i)) \quad (3.3)$$

A fenti, elsődleges célfüggvényen túl, ugyanakkor egy másodlagos célt is szem előtt kell tartani, ami nem más, mint a sorrendfüggő átállási idők időtartamainak minimalizálása. Ennek definiálására az ütemterv adataiból származó újabb bináris változóra van szükség, ami az alábbiak szerint kap értéket.

$$X_{ikopl} = \begin{cases} 1, ha\ az\ O_{ik}\ művelet\ az\ M_l\ gépen\ közvetlenül\ megelőzi\ az\ O_{op}\ műveletet \\ 0, egyébként \end{cases} \quad (3.4)$$

A fenti változó segítségével a sorrendfüggő átállási időt minimalizáló célfüggvény a következőképp néz ki.

$$\min(\sum_{O,M} X_{ikopl} * SS_{ikopl}) \quad (3.5)$$

A feladat matematikai modelljének felállításához már csak egyetlen dolog hiányzik, a korlátozó feltételeknek a megadása. Ezek a feladat nagyon fontos részét képezik, és biztosítják azt, hogy a feldolgozás során olyan eredményt kapjunk, ami egy megengedhető megoldás, vagyis nem sért semmilyen, előre definiált körülményt. Ezen korlátozó feltételek közé tartozik a GYR-en belüli műveleti sorrend kényszer a 3.6 egyenlet szerint, a sorrendfüggő felkészítési idők betartását biztosító kényszer a 3.7 egyenlet szerint, valamint az alternatív erőforrások közül műveletenként pontosan egy kiválasztását biztosító kényszer a 3.8 egyenlet szerint.

$$S_{ik} + \sum_M SM_{ikl} * P_{ikl} \leq S_{ik+1} \qquad \forall\ i, k, l \quad (3.6)$$

$$S_{ik} + \sum_M (SM_{ikl} * P_{ikl}) + \sum_M (X_{ikopl} * SS_{ikopl}) \leq S_{op} \qquad \forall i,k,o,p,l \qquad (3.7)$$

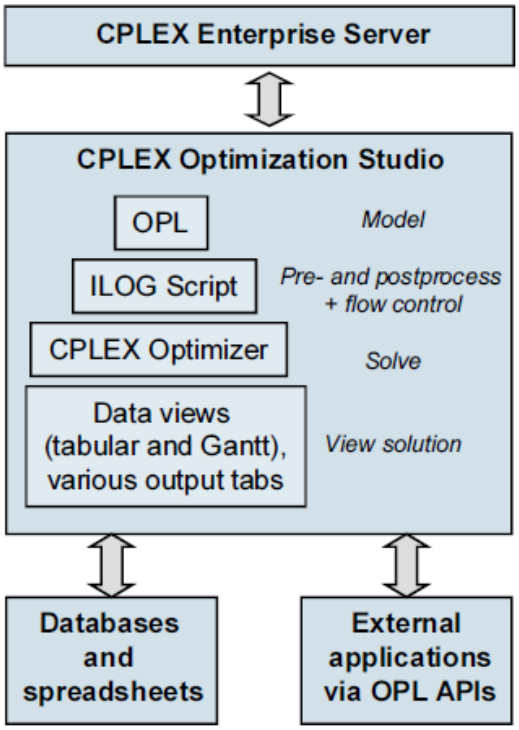
$$\sum_{J_i} SM_{ikl} = 1 \qquad \forall i \qquad (3.8)$$

4. A modell implementálása

A matematikai modell kidolgozása után egy olyan szoftver megoldást kerestem, amely lehetővé teszi, hogy a modellnek megfelelő korlátoptimalizálási feladatot megoldjam a segítségével. A választásom az üzleti döntéstámogató rendszerek között piacvezetőnek számító [7] IBM ILOG CPLEX Optimization Studiora esett, amiben felépítettem a modell vázát. Ezt követően saját C# nyelven készült programommal futásidőben módosítottam azt a hierarchikus célfüggvények követelményeinek megfelelően, majd a külső forrásból származó bemenő adatok feldolgozása után a keretrendszer optimalizáló motorjának segítségével megoldottam a feladatot, a kapott eredményt pedig Gantt-diagramon ábrázoltam. Ebben a fejezetben elsőként a rendszer működési elvét és a megoldás implementálásának legfontosabb lépéseit mutatom be, majd egy valós adathalmazon lefuttatva a programot értékelem a kapott eredményeket.

4.1 Az IBM ILOG CPLEX Optimization Studio [8]

Az IBM ILOG CPLEC Optimization Studio egy integrált fejlesztőkörnyezet, melynek segítségével lineáris programozási feladatok matematikai modelljei építhetők fel, oldhatók meg és ábrázolhatók. A rendszer architektúráját a 4.1 ábra mutatja.



4.1. ábra: A rendszer architektúrája [8]

A szoftver részét képezi az Optimization Programming Language (továbbiakban OPL) amely egy speciális programozási nyelv, ami kifejezetten lineáris programozási feladatokhoz kapcsolódó matematikai modellek egyszerű és gyors implementálására lett kifejlesztve. A program központi eleme a CPLEX Optimizer Engine (továbbiakban COE), ami az OPL nyelven felírt probléma megoldását határozza meg. Ez magában foglalja a lineáris-, egészértékű-, kvadratikus-, vegyes egészértékű- és vegyes kvadratikus programozási feladatok megoldására szolgáló algoritmusokat, úgy, mint a szimplex módszer, korlátoptimalizálási valamint hálózat optimalizálási megoldások. Mindezen túl a COE részét képezi az elsősorban részletes ütemezési feladatok esetén használt Constraint Programming Optimizer (továbbiakban CPO), amely a valós ipari ütemezési környezetekben előforduló korlátozó feltételek (pl. beállítási idők, helyettesítő erőforrások, stb.) egyszerű figyelembe vételét teszi lehetővé a modellépítés valamint a feladatmegoldás során.

A programcsomag a fentieken túl lehetőséget biztosít a bemenő adatok direkt megadásán kívül külső adatállományokból (Excel tábla, adatbázisok ODBC csatolón keresztül) történő importálásra, valamint a kapott eredmények ugyanilyen formában történő exportálására. Az ilyen külső forrásból származó adatok előfeldolgozására, tisztítására valamint a kapott eredmények exportálható formára alakítására szolgál az IBM ILOG Script modul, ami szintén az IDE részét képezi.

Végül, de nem utolsó sorban pedig a rendszer többféle API-n (Application Programming Interface) keresztül lehetőséget biztosít arra, hogy az optimalizáló motort különböző nyelveken megírt (C++, C#, Java, Python) külső alkalmazásokon keresztül érjük el, így annak szolgáltatásait saját fejlesztésű programokba lehet integrálni. A megvalósítás során én is ezen API-k egyikét használom a rendszer funkcióinak saját programomban történő eléréséhez.

Az IDE-vel elkészített modellek különböző adathalmazokon történő lefuttatására szolgál a CPLEX Enterprise Server modul, ami főként nagyvállalati környezetben lehet hasznos. Segítségével a COE szolgáltatásaihoz egyszerre több felhasználó, párhuzamosan tud hozzáférni, és egymástól függetlenül tudnak CPLEX nyelven írt optimalizálási feladatokat nagy számítási teljesítményű központi szerveren megoldani.

4.2 A modell felépítése

A 3.3 fejezetben ismertetett matematikai modellt elsőként OPL nyelven valósítottam meg. Ehhez jól elkülöníthető részekre osztva kellett meghatároznom az egyes építőelemeket.

Elsőként a bemenő változók definiálást végeztem el, melyeket a matematikai modellen túlmutatva az ipari gyakorlatnak megfelelő módon alakítottam ki. A matematikai modell definiálásánál ugyanis még egyszerűen adottnak tekintettem az egyes feladatok erőforrás és műveleti idő szükségleteit, és az alternatív műveletvégző gépeken értendő műveletvégzési időket is közvetlenül a GYR egyes műveleteihez rendelve kezeltem. Ugyanakkor ezt a szemléletet a konkrét megvalósítás során módosítottam annak érdekében, hogy a modell jobban illeszkedjen a gyakorlati igényekhez. Ennek eredményként bemenő adatként nem a konkrét GYR-ek műveleteinek időadatát veszem alapul, hanem a GYR által előállított termék egységnyi mennyiségére meghatározott műveletvégzési idők és a GYR-en gyártott mennyiség alapján határozom meg a rendelések konkrét gépfoglalási időtartamait a különböző gyártógépeken. Ezekhez szükségesek a gyártott tételek műveletterveinek adatai, melyek egy egységnyi termék előállításához szükséges technológiai lépéseket írják le (használt gép és igénybevételi időtartam), valamint a GYR-el előállított termék mennyiség, melyek a mai modern ERP rendszerekből könnyen kinyerhetők.

Bemenő adatként tehát a GYR-ek által előállított termékeket, azok igényelt mennyiségét, illetve az egyes tételek egyes műveleteinek különböző erőforrásokon értendő műveletvégzési időtartamainak az adatait vettem figyelembe. Ezen kívül minden GYR-hez a célfüggvény kezelése miatt rögzítettem az esedékességi dátumot, illetve azt, hogy melyik az utolsó (legnagyobb sorszámú) művelete az adott GYR által előállított tételnek. Ez utóbbira azért volt szükség, hogy tudjam, hogy a konkrét gyártott termék esetében melyik művelet az, amelynek a befejező időpontját kell figyelembe vennem a késés számításánál. Mindezeket az adatokat egy és többdimenziós tömbökben tárolom, és a célfüggvény valamint a korlátozó feltételek megadása során ezen tömbök egyes elemeire hivatkozok majd az összefüggések felírása során. A tömbök indexelésére, kihasználva az OPL nyújtotta lehetőségeket, külön adathalmazokat (ún. `set` adatstruktúrákat) alkalmazok. Ezek alapján a GYR-ek esedékességi dátumait leíró adatokat pl. a 4.2. ábrán látható formában tudom definiálni.

```
15 {string} WOs = ...;
16 int dueDate[WOs] = ...;
```

4.2. ábra: A GYR-ek esedékességi dátumát leíró adatstruktúra

A fenti példában a „`WOs`” nevű `string` típusú elemeket tartalmazó adatstruktúra tárolja az ütemezendő GYR-ek egyedi azonosítóját, az int típusú elemeket tartalmazó `dueDate` tömb pedig azok esedékességi dátumait. A tömb után szögletes zárójelben lévő „`WOs`” érték utal arra, hogy a tömb elemeire a GYR-ek azonosítóinak értékeivel is lehet

hivatkozni. Mindkét változó esetén az egyenlőségjel után három darab pont látható, ami pedig azt jelöli, hogy a modell fájlban csak a változók definiálása történik meg, értékadásra egy külön adatfájlban kerül sor.

Bemenő adatként még egy dolog megadása szükséges, ami nem más, mint az egyes gépeken az egymást követő műveletek közötti sorrendfüggő felkészítési idők. Ezeket szintén az ipari igényeknek megfelelően a matematikai modelltől eltérő módon kezelem, vagyis a megelőző és a követő gyártott tételek között definiálok felkészítési időadatokat minden gépen. Ehhez az OPL nyelv egy speciális adattípusát, a `tuple` szerkezetet használom. Ez az objektum-orientált programozási nyelvekből ismert osztályhoz hasonló összetett típus, ami több, akár különböző típusú változót zár egy csoportba. Jelen esetben a megelőző és követő gyártott tételek azonosítóját és a köztük lévő felkészítési idők hosszát tárolom egy-egy ilyen struktúrában, és egy `set` tömböt képezek belőlük a 4.3 ábrán látható módon.

```
29 tuple seqSetup { int first; int second; int time; }
30 {seqSetup} setup[m in Machines] = ...;
```

4.3. ábra: A felkészítési időket tároló adatszerkezet

A `setup` tömb elemei azokat az átállási időket tartalmazzák, melyek az adott gépen megmunkált tételek között fennállhatnak, méghozzá úgy, hogy ezek egy `set` struktúrába kerülnek összefogásra.

A bemenő változók definiálása után a modell megoldását tároló döntési változókat hoztam létre. Ezek a bemenő adatokból indulnak ki, és értékükre a korlátozó feltételek szabnak határokat. Az ismertetett termelésütemezési feladat estében ezek a döntési változók minden művelethez kapcsolódóan a kezdési időpont és a végrehajtó gép lesznek. Az OPL nyelv az ütemezési feladatok egyszerű kezelése érdekében rendelkezik egy speciális változó típussal, az ún. `interval` lehetőséggel. Ezek egy-egy ütemezendő művelet leképzését jelentik a rendszerben, melyek kezdő és végidőponttal, valamint adott hosszal rendelkeznek. Segítségével minden ütemezendő művelethez hozzárendelhető egy ilyen változó, és a megoldásban ezek adatai jelzik majd, hogy mettől meddig, melyik gépen kell egy-egy műveletet elvégezni. A modell megadása során a 4.4 ábrán látható `interval` típusú döntési változókat definiáltam.

```
31 dvar interval OP[i in WOs][j in Operations];
32 dvar interval MachineAllocation[i in WOs][j in Operations][m in Machines] optional
33              size duration[m][producedPart[i]][j] * orderedQuantities[i];
```

4.4. ábra: A műveleteket reprezentáló adatszerkezetek

Az első OP nevű kétdimenziós tömbben minden GYR minden műveletéhez tartozik egy változó, aminek a kezdő és végidőpontja a feladat megoldása során meghatározásra kerül. A `MachineAllocation` tömbben az előzőektől eltérően ütemezendő műveletenként több döntési változó is eltárolódik, ugyanakkor ezek egy része opcionális, vagyis elképzelhető, hogy nem fog a végső ütemezésben szerepelni. Ennek jelen esetben azért lesz jelentősége, mert így lehetővé válik az alternatív műveletvégző gépek közötti választás az ütemezés során. Ennek mikéntjéről részletesen a korlátozó feltételek bemutatása során írok. Az eddigieken túl a `MachineAllocation` változó esetében minden műveletnek adott a hossza (`size`) is, ami a bemenő adatként megadott `duration` tömbből származik. Ennek meghatározás során figyelembe veszem azt, hogy melyik gépen történik az ütemezés, illetve azt is, hogy mekkora a GYR-el előállított termékmennyiség.

Az eddig bemutatott döntési változók segítségével az alternatív gépek kezelése már megvalósítható, ugyanakkor a gépen belüli átállási idők figyelembe vétele még nem. Ehhez az OPL nyelv egy másik speciális típusú változóját, a `sequence`-et használom a 4.5 ábrán látható formában.

```
37 dvar sequence MachineOrder[m in Machines] in all(i in WOs, j in Operations) MachineAllocation[i][j][m]
38              types all(i in WOs, j in Operations) producedPart[i];
```

4.5. ábra: A gépen belüli műveleti sorrendet tároló adatszerkezet

A `MachineOrder` döntési változó egy tömb, ami gépenként tartalmazza azokat az `interval` típusú döntési változókat, amelyek az adott gépen végrehajtandó műveletekhez tartoznak, hozzárendelve azt is, hogy a művelethez tartozó GYR milyen azonosítójú terméket állít elő. Ennek a felhasználásával a korlátozó feltételek között megadható, hogy minden gépen kerüljenek betartásra az átállási idők is. Ennek módjáról részletesen a korlátozó feltételek bemutatása során írok.

Miután a bemenő- és döntési változók definiálásra kerültek a célfüggvény megadása következik. A 3.3 fejezetben amiatt, hogy többszintű célrendszer megvalósítása szükséges az ütemezés elkészítése során, két célfüggvényt is megfogalmaztam. Első, és legfontosabb az összes késés minimalizálása, amit úgy érhetünk el, hogyha a beütemezett műveletek adatait figyelembe véve minden GYR utolsó művelete és az esedékességi dátuma közötti pozitív értékű különbségek összegének a minimális értékét keressük. Ezt a megvalósítást mutatja a 4.6 ábra, ami jól láthatóan a 3.3 képlet leképezése a modell változóinak megfelelő szintaktikával.

```
44 minimize sum(i in WOs, m in Machines) maxl(0, (startOf(MachineAllocation[i][lastOp[i]][m], 0) +
45                presenceOf(MachineAllocation[i][lastOp[i]][m]) *
46                (duration[m][producedPart[i]][lastOp[i]] * orderedQuantities[i]) - dueDate[i]));
```

4.6. ábra: Az összes késést minimalizáló célfüggvény

A célhierarchia második szintjén az átállási idők minimalizálása áll. Ahhoz, hogy ezt el tudjam végezni egy újabb döntési változóra volt szükségem, ami jelzi azt, hogy `MachineOrder sequence` alapján egy adott műveletet követő műveleten milyen típusú termék készül. Ennek meghatározására a `typeOfNext()` függvényt használtam, ami a `MachineOrder`-ben lévő egy kiválasztott `interval` változó típusát, azaz a gyártott termék azonosítóját adja meg visszatérési értékként. Az elkészített döntési változót a 4.7 ábra szemlélteti.

```
44 dexpr int orders[i in WOs][j in Operations][m in Machines] = typeOfNext(MachineOrder[m], MachineAllocation[i][j][m], 0, 0);
```

4.7. ábra: Az egymást követő műveletek típusait tároló adatszerkezet

Az előbbi kifejezés abban az esetben, hogyha nincsen egy soron következő művelet után több ütemezett művelet az adott gépen, akkor 0 értéket ad vissza. Emiatt a bemenő adatok definiálása során minden alkalommal megadok egy 0-s termékazonosítót is, ami a gépenkénti utolsó műveletek helyes kezelésére szolgál.

A kifejezést felhasználva elkészítettem a második szintű célfüggvényt is, ami az átállási idők minimalizálását végzi. Ebben az ütemezésben szereplő átállási idők összegét veszem alapul, figyelembe véve azt, hogy egy termék gyártását milyen termék gyártása követi az adott gépen, amihez az előző bekezdésben ismertetett `orders` változó értékeit használom.

Végül, de nem utolsó sorban a modell korlátozó feltételeinek definiálását végeztem el. Ide az 3.6, 3.7 és 3.8 képletekkel megfogalmazott összefüggések leképzése tartozik, amik előírják a műveleti sorrendet, az átállási idők betartását és az alternatív gépek közötti választás szabályait. Ezek OPL nyelven történő megfogalmazását mutatja a 4.8 ábra.

```
50 subject to {
51     /* setting up the precedence constraints among the operations in each WO */
52     forall (i in WOs, j in Operations, k in Operations : j < k)
53         endBeforeStart(OP[i][j], OP[i][k]);
54     /* choosing exactly one Machine for each operation from the alternatives */
55     forall (i in WOs, j in Operations)
56         alternative(OP[i][j], all(m in Machines) MachineAllocation[i][j][m]);
57     /* providing no overlapping operations on any machine */
58     forall (m in Machines)
59         noOverlap(MachineOrder[m],setup[m]);
60 }
```

4.8. ábra: A korlátozó feltételek leírása

Ezek közül az első utasítás a GYR-en belüli műveleti sorrendek betartását biztosítja. Az `endBeforeStart()` függvény lehetővé teszi, hogy két művelet között olyan kapcsolatot létesítsek, ami nem engedi meg, hogy az első művelet befejezése előtt a második művelet elkezdődjön.

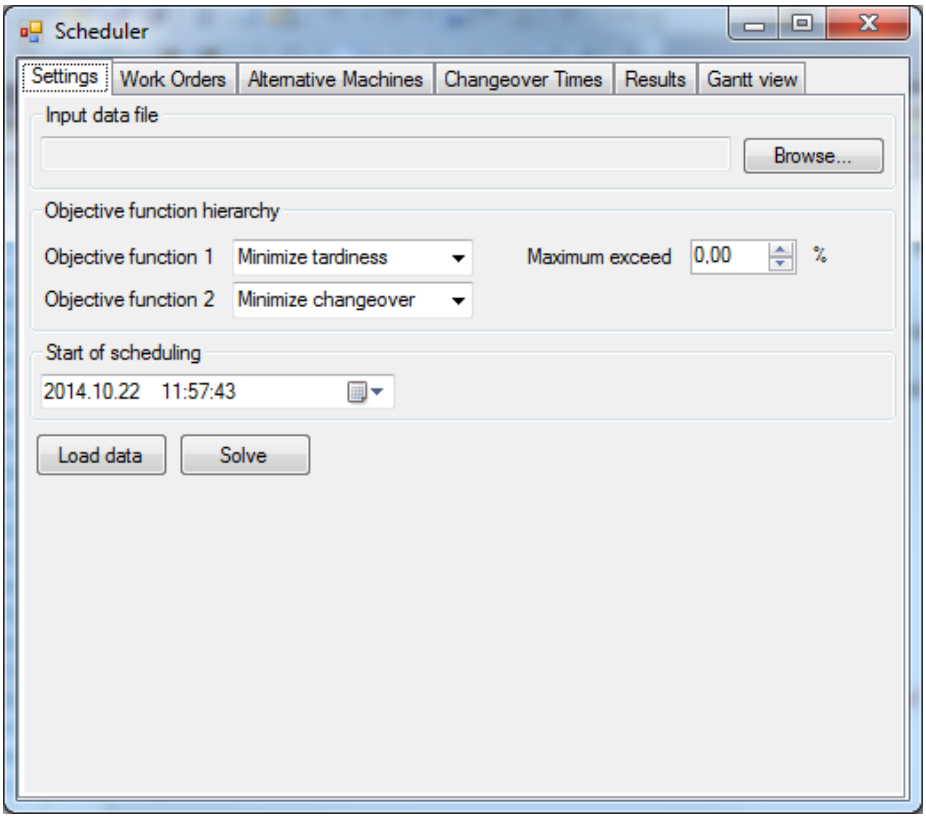
A második utasítás azt hivatott biztosítani, hogy minden művelet pontosan egy gépre kerüljön beütemezésre az alternatívák közül. Ennek megadására az `alternative`() függvényt használtam, ami abban az esetben, hogyha az első paramétereként megadott művelet szerepel az ütemezésben, és meg lett határozva annak a kezdő és végidőpontja, akkor a második paraméterként megadott műveletek közül pontosan egyet enged besorolni az ütemezésbe, és annak kezdő és végidőpontja meg kell, hogy egyezzen az első paraméterként szereplő műveletével.

A harmadik, egyben utolsó utasítás pedig az egy gépen egymást követő műveletek közötti sorrendfüggő átállási idők kezelését végzi. Az OPL nyelven erre a `noOverlap()` függvény szolgál, aminek első paramétere az a korábban definiált `sequence` típusú döntési változó, ami az egyes gépeken a műveleti sorrendeket tárolja, a második paramétere pedig az átállási idők adataiból létrehozott `setup` struktúra.

4.3 Az ütemező szoftver elkészítése

A modell felépítését követően C# nyelven készítettem egy programot, ami lehetővé teszi az ütemezéshez szükséges bemenő adatok Excel táblázatból történő beolvasását, ezek alapján a modell megoldásához szükséges bemenő adatstruktúra előállítását, a modell megoldását hierarchikus célfüggvények figyelembe vételével, a megoldást pedig Gantt-diagramon ábrázolja.

A szoftver egyszerű grafikus felhasználói felülettel rendelkezik, aminek főképernyőjét a 4.9 ábra szemlélteti.



4.9. ábra: Az ütemező szoftver felhasználói felülete

Az első, „Settings” fülön van lehetőség a program bemenő adatainak és futási paramétereinek beállítására. Elsőként a bemenő adatokat tartalmazó Excel fájlt lehet kitallózni a fájlrendszerből, majd a „Load data” gombbal betölteni azt a megfelelő adatstruktúrákba. Ekkor OLEDB csatolón keresztül olvasom ki az adatokat az egyes munkalapokról, melyekhez SQL utasítások segítségével férek hozzá. A bemenő Excel fájl formátuma adott, három munkalapot kell, hogy tartalmazzon, „wo”, „altmachine” és „changeover” néven, mindegyikben adott számú és nevű adatoszloppal. Az adatok kiolvasása után létrehozom azokat az objektumokat, melyek az egyes gyártási rendeléseket, műveleteket, műveletvégzési időket és sorrendfüggő felkészítési időket tárolják a probléma megoldása során.

A feladat megoldása előtt a szoftver még lehetőséget biztosít arra, hogy beállítsuk a hierarchikus célfüggvény egyes szintjein a célokat is. Jelen esetben a 3.3 fejezetben megfogalmazott matematikai modellel összhangban kétszintű hierarchia megadását tettem lehetővé, ugyanakkor most is elvégeztem némi módosítást a gyakorlati igényekkel összhangban. Egyrészt a hierarchia módosítható, vagyis adott a lehetőség, hogy az előre definiált célfüggvényeket különböző sorrend alapján vegyük figyelembe, így a szoftver könnyebben tud alkalmazkodni olyan helyzetekhez, amikor valami miatt az általánosan használt hierarchiát módosítani kell (pl. csak a késést kell minimalizálni, nem számít az

átállás). Ezen kívül megadható az is, hogy a hierarchia első szintjén számított optimális célfüggvény értéktől mekkora mértékben engedélyezett az eltérés akkor, amikor a hierarchia második szintjén lévő célfüggvény optimális értékét számítom. Ez nem jelent mást, minthogy a hierarchia első szintjén számolt optimális célfüggvény érték egy %-os tűrés beiktatása mellett korlátozó feltételként fog megjelenni a modellben a második szintű célfüggvény optimális értékének számítása során.

A fentieken túl definiálhatjuk azt is, hogy mi legyen az ütemezés kezdő időpontja, azaz melyik az a legkorábbi időpont ahova az első műveletet ütemezni lehet. Ennek azért van gyakorlati jelentősége, mert a programot különböző időpontokban, különböző bemenő adatokkal lehet lefuttatni. Ezért, ha pl. ezen a héten készítem el a jövő heti gyártási ütemtervet, akkor nem szeretném azt, hogy a program erre a hétre is ütemezzen olyan feladatokat, amiket csak jövő héten akarok végrehajtani. Ezzel a beállítással biztosítható, hogy egy későbbi időszak terve ne nyúljon át olyan korábbi időszakra, aminek az ütemezését már véglegesítettem, esetleg a gyártás már el is kezdődött.

A bemenő adatok beolvasását és a modell futási paramétereinek beállítását követően a „Solve” gombra kattintva lehet a feladatot megoldását kezdeményezni. Ekkor egyrészt előállítom a beolvasott adatok alapján azt az adatfájlt, amiből az optimalizáló motor be tudja tölteni a bemenő paramétereket. Ennek keretében az esedékességi dátumokhoz kapcsolódóan adatkonverziót is végzek, ugyanis az optimalizáló motor a műveletek kezdő és végidőpontjainak meghatározásakor nem kezel dátum/idő típusú változókat, hanem 0-tól kezdődően egész számokkal jelzi az időskálát. Ezért minden időpontot át kell skáláznom erre a tartományra méghozzá úgy, hogy a 0 időpont a beállítások között megadott kezdő időpont, az egyes esedékességi dátumokat pedig ehhez képest eltelt percek száma legyen. Ez a megoldás maga után vonja azt is, hogy az optimalizáló motor által kezelt egy időegység egy percnek fog megfelelni, így a műveleti időket is percekben szükséges megadni.

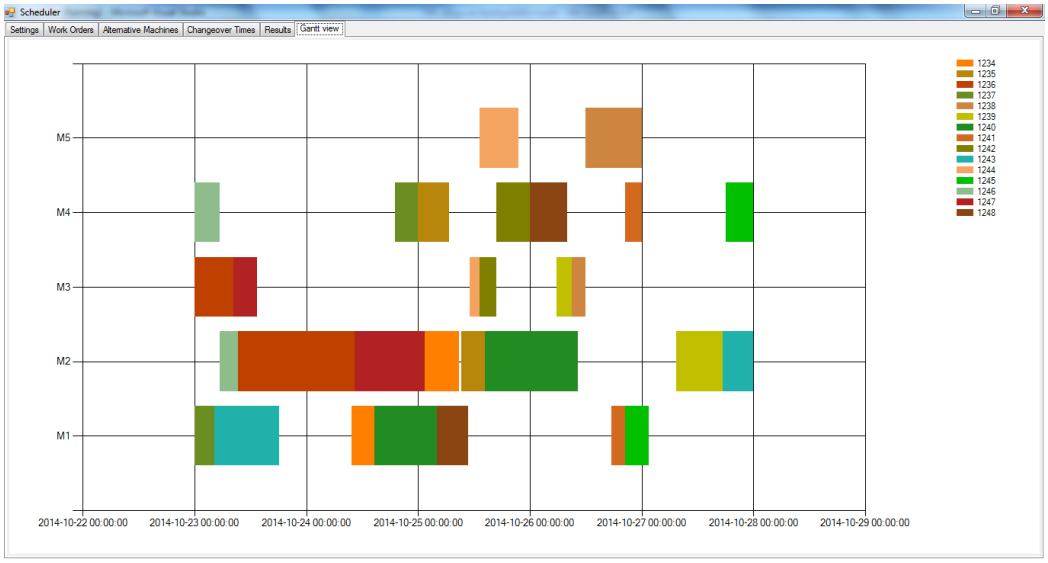
A megoldás előkészítésének másik lépéseként futási időben állítom elő azt a modellt is, ami a megadott célfüggvény hierarchiához illeszkedik. Ehhez a 4.2 fejezetben ismertetett OPL modellnek egy célfüggvényektől mentes változatát használom, mint bemenő fájl, és ezt egészítem ki a célfüggvény hierarchiában beállított sorrendnek megfelelően egy célfüggvénnyel. Ezt követően az előállított modell- és adat fájlokat átadom az optimalizáló motornak, ami megoldja a feladatot, és visszaadja a megoldást a programomnak. Ezt követően attól függően, hogy a célhierarchia hogy van beállítva, előállítok egy újabb modell fájlt amiben a célfüggvény már a hierarchia második szintjén szereplő célnak megfelelő lesz. Ezt

még kiegészítem egy korlátozó feltétellel a visszakapott optimális célfüggvény érték és a megadott maximális túllépés beállított értéke alapján. Az így létrejött modell fájlt és az eredeti adatfájlt ismét átadom az optimalizáló motornak, ami kiszámítja az optimális ütemezést, és visszaadja azt a programomnak. Ezen adatok alapján a műveletek kezdő és végidőpontjait átkonvertálom dátum/idő értékekké, és a gépkiosztásokat is figyelembe véve Gantt-diagramon ábrázolom az eredményt. Ezen kívül a műveletek kezdő és végidőpontjait, valamint az ütemezés szerinti gyártógép azonosítóját táblázatos formában is megjelenítem.

4.4 Tesztelés, alkalmazás

Az elkészült programot több, különböző méretű adathalmazon teszteltem annak érdekében, hogy megbizonyosodjak róla, hogy a megvalósított modell a terveknek megfelelően működik, illetve ellenőrizzem, hogy valós környezetből származó nagyméretű adathalmaz esetén milyen futási időadatokat lehet mérni.

Elsőként az elkészült program megfelelő működését egy kis adathalmazon teszteltem, ahol a kapott megoldáson ránézésre is ellenőrizni lehet, hogy az előírt korlátozó feltételek betartásra kerülnek-e. Az először használt adathalmazba ennek megfelelően mindössze 15 db gyártási rendelést rögzítettem, mindegyikhez 2 db műveletet rendeltem, és ezeket összesen 5db gépen lehetett végrehajtani. Az ütemezés kapott eredményét a 4.11 ábra mutatja.



4.11. ábra: Az ütemezés eredménye Gantt-diagramon ábrázolva

A fenti ábrát és az ütemezés táblázatos eredményét megvizsgálva azt tapasztaltam, hogy a kapott eredményben az összes definiált korlátozó feltétel betartásra került, ezért a programot alkalmasnak találtam arra, hogy további, nagyobb adathalmazokon teszteljem azt.

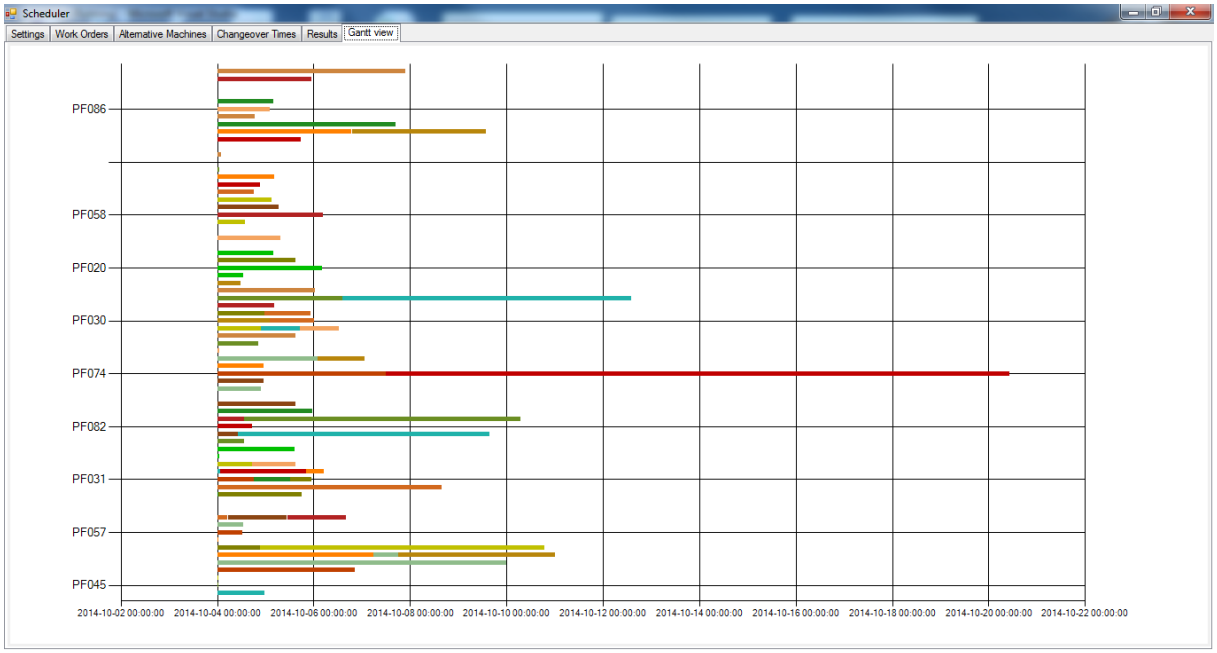
Ehhez fokozatosan elkezdtem növelni a feldolgozandó műveletek, gépek és átállási idők számát, és az ismételt futtatások során folyamatosan ellenőriztem a kapott eredmények helyességét, illetve mértem a különböző méretű adathalmazok esetén a futási időket. A kapott eredményeket a 4.12 ábra szemlélteti.

Műveletek száma	Gépek száma	Gyártott termékek száma	Átállási idők száma	Futási idő (mp)
30	5	5	5	2,26
30	5	10	5	0,14
30	10	10	5	0,18
30	10	10	10	0,12
60	10	10	10	0,29
60	20	10	20	0,78

4.12. ábra: A program futási ideje különböző méretű feladatok esetén

Az előbbi táblázatból jól látható, hogy ugyan a feladat megoldásának ideje függ a feladat méretétől, ugyanakkor mégis minden esetben olyan válaszidővel érkezik meg az eredmény, azaz az optimális ütemezés, ami manuális ütemezés esetén elképzelhetetlen.

A kisebb méretű adathalmazok vizsgálata után a programot lefuttattam egy éles környezetből származó adathalmazon is. Ez egy egy hetes termelési tervet tartalmazott, amiben összesen 86 db ütemezendő művelet, 70 db gép, 82 különböző gyártott tétel és 164 db lehetséges átállási idő szerepelt. A feladat optimális megoldását a program mintegy 45 perc alatt határozta meg, melyet a 4.13 ábrán lévő Gantt-diagram ábrázol.



4.13. ábra: Egy heti optimalizált termelési terv ábrázolása Gantt-diagramon

A futási idő ugyan meglehetősen sok, mégis jelentősen kevesebb, mint a termelésütemezőnek egy heti terv elkészítésére fordított átlagos ideje, nem beszélve arról, hogy az így létrejött megoldás garantáltan optimális lesz a definiált célfüggvény mentén.

A teszteket egy 2,1 GHz-es órajelű Intel Core2Duo processzorral és 4 GB RAM-mal felszerelt régebbi számítógép konfiguráción végeztem, melyen Windows 7 Professional operációs rendszer futott.

5. Összefoglalás

Dolgozatomban bemutattam egy, a gyártó vállalatok életében nagyon fontos üzleti folyamatot, a termelésütemezést. Áttekintettem ennek helyét a cégek termelés előkészítési folyamatában, bemutattam az ezzel kapcsolatban felmerülő leggyakoribb követelményeket és célokat. Ezt követően áttértem egy konkrét iparág, a műanyagalkatrész gyártás termelésütemezési követelményeire, céljaira és megoldási lehetőségeire. Áttekintettem, hogy melyek azok a legfontosabb peremfeltételek, amelyeket egy ilyen tevékenységgel foglalkozó cég elvár a termelésütemezéssel kapcsolatban, és megnéztem, hogy ezekhez milyen konkrét üzleti célok kapcsolódnak.

Az alapvető követelmények és ismeretek áttekintése után elkészítettem egy matematikai modellt a konkrét üzleti célok formalizálásával, majd ezt implementáltam az IBM ILOG CPLEX Optimization Studio segítségével. Az így elkészített modell dinamikus újrafelhasználhatósága és a valós ipari körülményekhez történő hatékonyabb illeszkedés érdekében C# nyelven is készítettem egy alkalmazást, ami a termelésütemezéshez szükséges bemenő adatokat egy Excel táblázatból olvassa be, és a felhasználó által beállított célhierarchiának megfelelően oldja meg a feladatot, felhasználva az ILOG ütemező motorját. Az elkészült programot különböző méretű, éles környezetből származó adathalmazokon teszteltem, és értékeltem a kapott eredményeket.

Az elkészített szoftver segítségével konkrét ipari igényeknek megfelelő ütemezési feladatokat lehet megoldani valós környezetben előforduló adathalmazokon, ami a napi munkavégzést nagyban elő tudja segíteni, lehetővé téve, hogy a termelésütemezők az időigényes kézi ütemezés helyett automatizált eszközök segítségével tudják végezni a napi munkájukat. Ezáltal több időt tudnak fordítani azon váratlan, emberi beavatkozást igénylő helyzetekben, ahol az algoritmus nem tud megoldást találni. Éppen ezért úgy gondolom, hogy egy ilyen program hatékony kiegészítőjévé tud válni az ERP rendszerek termelési moduljainak. A fejlesztés során végig arra törekedtem, hogy az elkészült stand-alone megoldás könnyen hozzáilleszthető legyen egy ERP rendszer adatbázisához, így a jelenlegi Excel alapú adatbevitelt egyszerűen le lehet váltani egy adatbázisból történő lekérdezésre. Ezen kívül a későbbiekben további célfüggvények megtervezése és programba illesztését is indokoltnak érzem annak érdekében, hogy a jelenleg bemutatott igényektől eltérő ütemezést is el lehessen készíteni a programmal.

Irodalomjegyzék

[1] Kovács, A.: *Új modellek és algoritmusok az integrált termeléstervezés és –ütemezésben,* Ph.D értekezés tézisei, BME-VIK MIT, 2005, pp. 4, http://www.sztaki.hu/~akovacs/thesis/booklet_hun.pdf (2014. október)

[2] Dr. Szikora, B.: *Termelésinformatika*, oktatási segédanyag, 5.43. változat, BME Elektronikai Technológia Tanszék, 2013, pp. 58 – 93

[3] M. L. Pinedo: *Planning and Scheduling in Manufacturing and Services*, pp. 4 – 201, Springer, New York, 2005

[4] Beluzsár, J.: *Termelésütemezési rendszer fejlesztése QAD EA integrált vállalatirányítási rendszerhez*, TDK dolgozat, BME VIK, 2013

[5] J. C. Chen, K. H. Chen, J. J. Wu, C. W. Chen: *A study of the flexible job shop scheduling problem with parallel machines and reentrant process,* Int J Adv Manuf Technol (2008) 39, pp. 344–354

[6] R.Moghaddas, M.Houshmand: *Job-Shop Scheduling Problem With Sequence Dependent Setup Times*, IMECS 2008, 19-21 March, 2008, Hong Kong

[7] http://www-01.ibm.com/software/info/ilog/

[8] IBM Industry Solutions oktatási segédanyag, *Model development with IBM ILOG CPLEX Optimization Studio V12.5*, 2012, http://cedric.cnam.fr/~lamberta/MPRO/ECMA/doc/ (2014. szeptember)