Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Elektronikai Technológia Tanszék

Beluzsár János

TERMELÉSÜTEMEZÉSI RENDSZER FEJLESZTÉSE QAD EA INTEGRÁLT VÁLLALATIRÁNYÍTÁSI RENDSZERHEZ

TDK dolgozat

KONZULENS

Dr. Szikora Béla

Iványkovits Zsolt

BUDAPEST, 2013

Tartalomjegyzék

1. Bevezetés

A XXI. század gyorsan változó világában az információ hatalmas kincs. Igaz ez a hétköznapok során is, de az üzleti életben még inkább. Egy nem elég megfontolt ajánlat, egy ad-hoc meghozott döntés vagy egy nem jól előkészített beruházás súlyosan hátráltathatja egy vállalat működését, negatívan befolyásolva annak piaci részesedését, növekedési potenciálját. Éppen ezért napjainkban a cégek igyekeznek egyre több információt összegyűjteni saját belső működésükről, folyamataikról, illetve a többi piaci szereplőről. Ezeket felhasználva pedig olyan felelős döntéseket próbálnak hozni – stratégiai és operatív szinten egyaránt –, melyekkel versenytársaikat megelőzve tudnak eleget tenni a dinamikusan változó vevői igényeknek.

Az említett folyamatban nagyon nagy szerepe van az informatikának. Az adatok gyűjtése és rendszerezése egy bizonyos vállalat méreten túl már nem oldható meg hatékonyan hagyományos eszközökkel, komplex informatikai megoldásokra van szükség. Ehhez nyújtanak támogatást napjainkban az ún. integrált vállalatirányítási rendszerek (Enterprise Resource Planning - ERP), melyek legfőbb feladata a vállalati folyamatok nyomon követése, és a rögzített információk strukturált tárolása, ezáltal a döntéshozatal támogatása. Egy ilyen rendszer helyes használatával egyrészt az adminisztráció csökkenthető, másrészt pedig a rendelkezésre álló információkból kiindulva egyes folyamatok hatékonysága növelhető, a költségek csökkenthetők.

Tipikusan ilyen folyamat a termelő vállalatok esetében az ún. termelésprogramozás. Ennek lényege, hogy a beérkező gyártási megrendeléseket a rendelkezésre álló erőforrásokon ütemezzük. Ugyanakkor adott körülmények között egy ilyen termelési tervet többféleképpen is el lehet készíteni, különböző (olykor egymásnak ellent mondó) szempontokat szem előtt tartva. Mindez indokolttá teszi olyan informatikai megoldások kidolgozását, melyek a rendelkezésre álló információk alapján segítik az optimális termelési terv kialakítását.

Az elmúlt évszázadban rengeteg kutatás foglalkozott a problémával, és igyekezett olyan algoritmusokat találni, amelyek a felmerülő piaci igényeket kielégítő megoldással szolgálnak. A nagy érdeklődésnek köszönhetően gyorsan kialakult a fentebb leírt probléma absztrakt matematikai leírása, amelyet az operációkutatásban job-shop scheduling feladatnak hívnak, ugyanakkor polinom idejű megoldása a mai napig nem ismert.

Dolgozatomban ennek a problémakörnek a legfontosabb sajátosságait mutatom be. Célom az eddigi kutatási irányok, lehetőségek áttekintő bemutatása, értékelése. Ezt követően

kiemelek egy gyakorlati életből vett problémát, és a vizsgált algoritmusok közül kiválasztom a konkrét feladatra várhatóan legjobb megoldást adó algoritmust, és továbbfejlesztem azt az ipari környezetből vett probléma általánosítható követelményei szerint. Végül bemutatom a QAD EA integrált vállalatirányítási rendszer termelésprogramozási modulját, és ismertetem a megoldás alkalmazási lehetőségét a rendszerben.

2. Matematikai modell

A bevezetőben említett job-shop scheduling feladat általános leírásában véges számú n db elvégzendő feladat (job) szerepel ($\{J_i\}_{i=1}^{n}$), és véges számú m db diszkrét erőforrás (machine, $\{M_j\}_{j=1}^{m}$). Minden J_i feladat m_i db műveletből (operation, $\{O_{ik}\}_{k=1}^{m_i}$) áll, melyeket az m db erőforráson egy előre meghatározott sorrendben kell végrehajtani. Összesen tehát $N = \sum_{i=1}^{n} m_i$ műveletet kell ütemezni a megoldás során a rendelkezésre álló erőforrásokon. N értéke a kutatások során többnyire az nm szorzattal egyenlő, ami azt feltételezi, hogy minden műveletet minden rendelkezésre álló erőforráson pontosan egyszer kell elvégezni minden J_i feladat esetében. A továbbiakban én is ezzel az alapvetéssel fogok élni. Ezen kívül egy időben egy erőforráson csak egy művelet hajtható végre, illetve egy művelet végrehajtásához egyszerre egy erőforrás vehető igénybe. O_{ik} a J_i feladat egy ütemezendő művelete, melyet egy meghatározott erőforráson kell végrehajtani. Minden O_{ik} művelethez tartozik egy τ_{ik} műveletvégzési idő, ami előírja, hogy az adott műveletet mennyi idő alatt lehet elvégezni az adott erőforráson megszakítás nélkül. Az összes feladat összes műveletének ütemezés szerinti elvégzési idejét átfutási időnek nevezzük (C_{max}). Az általános job-shop scheduling feladat során a cél az egyes műveletek kezdő időpontjának meghatározása ($t_{ik} \geq 0$) az átfutási idő minimalizálása és a leírt korlátozó feltételek betartása mellett. A bevezetett jelöléseket alkalmazva tehát a cél annak a C_{max}^{*} értéknek a meghatározása, ahol

$$C_{max}^{*} = \min(C_{max}) = \min_{lehetséges\ ütemezések}(max(t_{ik} + \tau_{ik}): \forall J_i\ ).$$

Az előző bekezdésben leírt modell a feladat általános, a kutatások során leggyakrabban használt megfogalmazását tükrözi. Az 1950-es évektől kezdve ez a változat terjedt el széles körben, és a témában végzett kutatások többsége is ezt a leírást tekinti kiinduló pontnak. Éppen ezért dolgozatom során én is ebből a probléma leírásból indulok ki. Ugyanakkor a dolgozat későbbi szakaszában, a gyakorlati életből vett probléma matematikai modelljének megfogalmazása során a fenti definíciót kiegészítem a konkrét feladat általánosítható követelményei szerint.

A bemutatott feladat méretét az n x m értékkel szokás jellemezni, ahol n az elvégzendő feladatok, m pedig a rendelkezésre álló erőforrások száma. Ebből kiindulva egy általános job-shop schedulnig probléma lehetséges megoldásainak számára jó felső korlátot ad az $(n!)^m$ érték, hiszen egy erőforráson n db műveletet kell ütemezni, mivel minden feladatnak pontosan egy olyan művelete van, amit a rendelkezésre álló erőforrások valamelyikén végre kell hajtani.

Egy erőforráson tehát $n!$ féleképpen lehet a műveleteket ütemezni, és mivel m db erőforráson kell mindezt megtenni, ezért a lehetséges ütemezések száma a korábban említett $(n!)^m$ lesz. Az optimális megoldás megtalálásához tehát ezek közül kellene kiválasztani a legkisebb átfutási idejű változatot. Ugyanakkor mivel a felső korlát a bemenet nagyságának exponenciális függvénye, az optimális megoldás kiválasztásához is exponenciálisan sok algoritmikus lépést kell elvégezni, ami miatt az általános job-shop scheduling feladat az NP-nehéz problémák családjába tartozik. [1]

2.1 Eddigi legfontosabb kutatási irányok

A problémával kapcsolatos tudományos kutatások az 1950-es évekig nyúlnak vissza. Akkoriban került megfogalmazásra a feladat fentebb ismertetett formája, és az első megoldási és reprezentálási módszerek is akkor születtek. Ezen kutatások legfőbb célja az optimális megoldást adó algoritmus kifejlesztése volt. Az akkor létrehozott megoldások az általános job-shop scheduling feladat néhány speciális esetére adnak csak optimális megoldást, ugyanakkor az elért eredmények jó kiindulási alapként szolgáltak a későbbiekben.

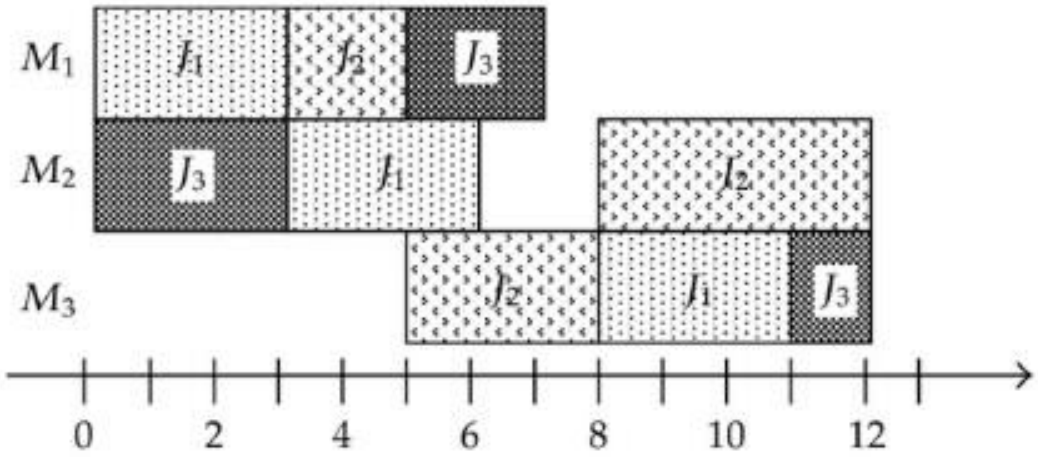
Ezek közül talán legismertebb Jackson 1956-os munkája, aki polinom idejű megoldást adott a probléma m = 2 paraméterű változatára. A megoldás során felhasználta Johnson 1954-es algoritmusát, ami az m = 2 paraméterű job-shop scheduling probléma egy speciális változatával, a 2 gépes flow-shop scheduling problémával foglalkozik. Ez annyiban különbözik a job-shop scheduling feladattól, hogy itt minden feladat esetén a műveletek sorrendje azonos, vagyis minden feladat műveleteit ugyanabban a sorrendben kell végrehajtani a rendelkezésre álló erőforrásokon [2]. A Johnson-módszer a legrövidebb műveleti idejű művelet ütemezésével kezd, és ha azt az adott feladat műveletei közül elsőként kell elvégezni, akkor a feladatot a lehető legkorábbi időpontban hajtja végre, az eddig a végrehajtási sor elejére beütemezett műveletek után. Amennyiben viszont a kiválasztott műveletet az adott feladat műveletei közül másodiknak kell elvégezni, akkor a feladatot az ütemezési lista végére rakja, az eddig hátra rakott feladatok elé. Ezt addig folytatja, amíg az összes feladat be nincs ütemezve. Az optimális végrehajtási sorrend a kapott listában szereplő sorrend lesz, azzal a kikötéssel, hogy egy művelet befejezése után az első erőforráson, rögtön el kell kezdeni a sorrendben következő feladat végrehajtását ugyanazon az erőforráson [3].

Jackson a két gépes job-shop scheduling feladatot Johnson 2 gépes flow-shop scheduling feladatára vezette vissza, és a bemutatott algoritmust is felhasználta. A megoldás során először két csoportra kell osztani az elvégzendő feladatokat:

1. Azon feladatok csoportja, melyeket először az első gépen kell végrehajtani, majd utána a másodikon.
2. Azon feladatok csoportja, melyeket először a második gépen kell végrehajtani, majd utána az elsőn.

A csoportosítást követően mindkét halmazon, egymástól függetlenül végre kell hajtani a Johnson-módszert, így megkapjuk az optimális végrehajtási sorrendet a két esetre. Ez után az első erőforráson előbb az első csoport optimális megoldásának sorrendjében kell ütemezni a műveleteket, majd a második csoport optimális megoldása szerint időben előrefelé haladva. Hasonlóképpen a második erőforráson pedig először a második csoport, majd az első csoport optimális megoldása szerinti sorrendben kell végrehajtani a műveleteket. Így a feladatokat a lehető legkisebb átfutási idővel lehet végrehajtani [4].

A kutatások korai szakaszában az első megoldási javaslatokon kívül kialakultak a feladat máig használt legfontosabb reprezentálási formái is. Ezek közül legelterjedtebb az ún. Gantt-diagramos megjelenés, melyet az 1.1 ábra szemléltet.

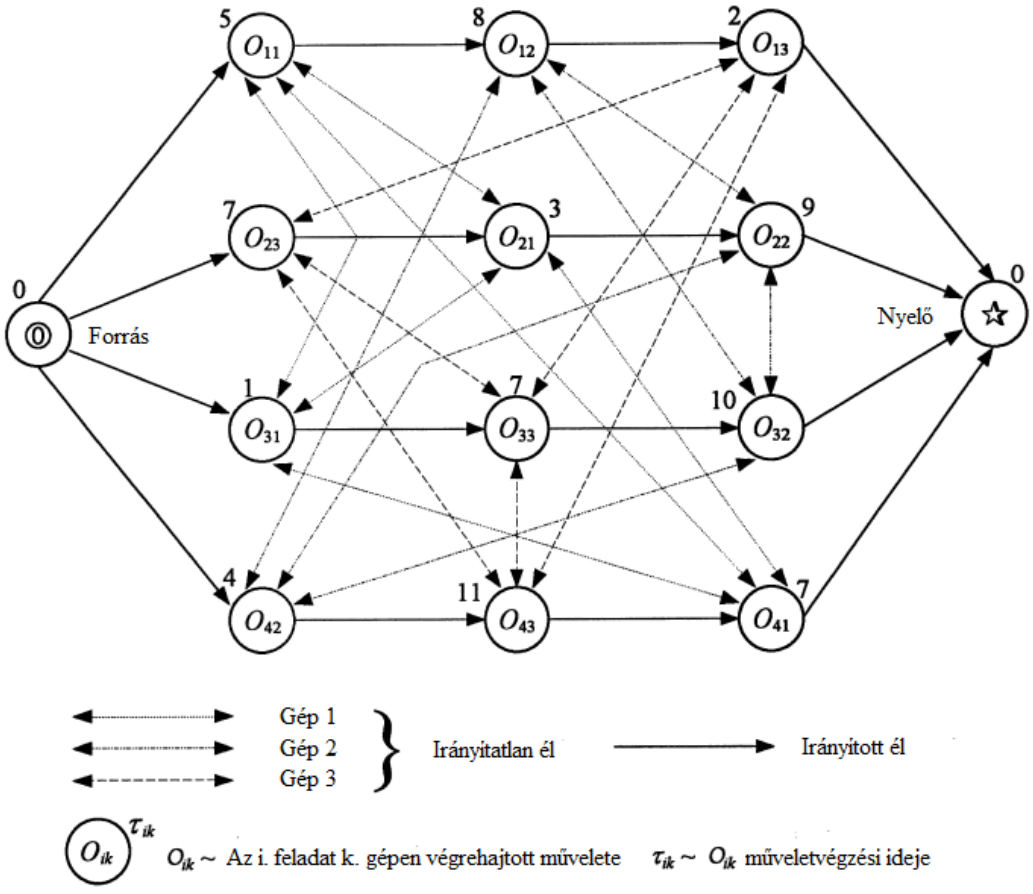


1.1 ábra 3 x 3 – as feladat Gantt-diagramos ábrázolása [10]

Ennek lényege, hogy az ábra minden egyes sora egy-egy gépet reprezentál, a diagram sávjai, pedig az elvégzendő feladatok egy-egy műveletét. A kettő metszetéből leolvasható, hogy az egyes feladatok műveleteit melyik gépen, egymáshoz képest milyen sorrendben kell elvégezni. A műveletek közé szükség esetén berajzolhatók nyilak, melyek a műveleti sorrendet jelképezik, így könnyedén kiszűrhető, ha egy helytelen megoldást kapunk.

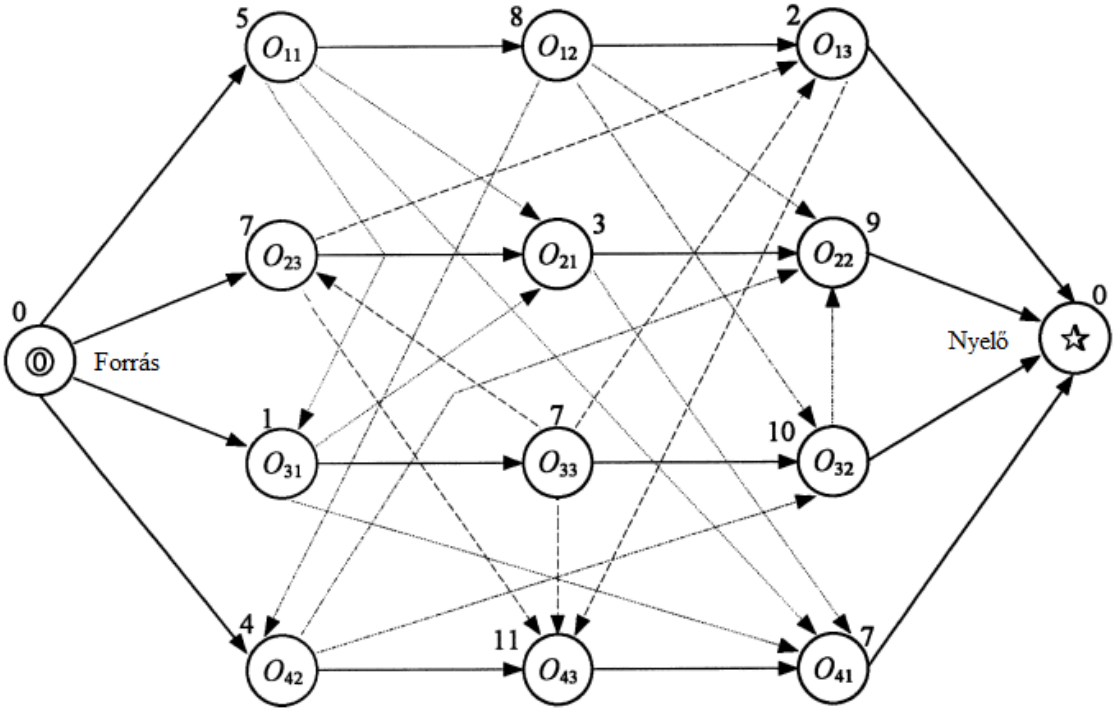
A job-shop scheduling feladatok másik elterjedt megjelenítési formája a Roy és Sussmann által 1964-ben kidolgozott $G = \{N, A, E\}$ ún. diszjunktív gráf modell. Itt a problémát egy gráf reprezentálja, ahol minden feladat minden művelete a gráf egy-egy csúcsa, melyeket az N halmaz tartalmaz. Ezeken kívül ide tartozik még két kiemelt csomópont, a forrás (⓪) és a nyelő (☆). A forrás specialitása, hogy nem mutat bele irányított él, de minden feladat első műveletét reprezentáló csomópontba kiindul belőle egy. Ezzel szemben a nyelőbe minden feladat utolsó műveletéből mutat egy irányított él, viszont egyetlen egy sem indul ki belőle. Minden egyes csúcshoz tartozik egy pozitív súlyérték is, ami az adott művelet τ_{ik} végrehajtási idejét jelenti ($\tau_{⓪} = \tau_{☆} = 0$) [1].

A modellben lévő további két halmaz a gráf éleit tartalmazza. Az A halmaz elemei a feladatokon belüli műveleti sorrendeket reprezentáló irányított élek, míg az E halmazba az egy gépen történő párhuzamos műveletvégzést korlátozó irányítatlan élek tartoznak. Ez utóbbiak olyan i és j műveleteket reprezentáló csomópontok között találhatók, amelyeket ugyanazon az erőforráson kell végrehajtani. Egy job-shop scheduling feladat diszjunktív gráf modell segítségével történő ábrázolását az 1.2 ábra szemlélteti [1].



1.2 ábra 4 x 3 – as feladat diszjunktív gráf ábrázolása [1]

Az 1950-es évek korai eredményei nagy lendületet adtak a problémával foglalkozó kutatások számára. A feladat speciális eseteinek megoldására kidolgozott polinom idejű algoritmusok (Johnson(1954), Jackson(1956), Akers(1956)) bizakodásra adtak okot az általános feladat megoldását illetően. Erre alapozva az 1960-as évekre a kutatások iránya megváltozott. Az eddigi heurisztikus módszerek helyett egzakt, enumerációs algoritmusok keresése került a középpontba. Ezek közül a legnagyobb jelentőségűek a korlátozás és elágazás (Branch and Bound - BB) típusú algoritmusok. Ezek többsége a feladat korábban ismertetett diszjunktív gráf modelljét veszik alapul. Egy lehetséges ütemezés megtalálása a gyakorlatban nem jelent mást, mint az egyes erőforrásokon végrehajtandó műveletek egyértelmű sorrendjének meghatározása a feladatokon belüli műveleti sorrend megtartása mellett. Mindez a diszjunktív gráf modellben annyit jelent, hogy a műveletek közötti összes irányítatlan élet irányított éllé kell átalakítani úgy, hogy a gráf körmentes maradjon. Egy-egy ilyen esetnél a C_{max} értékét a gráf leghosszabb súlyozott útjának hossza a forrás és a nyelő között. Ezek alapján a 2. ábrán bemutatott feladat egy lehetséges megoldásának reprezentációját mutatja a 1.3 ábra [1][5].



1.3 ábra 4 x 3 – as feladat egy megoldásának ábrázolása [1]

A BB eljárás során egy döntési fa kerül felépítésre, melynek minden egyes csúcsához a feladat egy olyan diszjunktív gráfja tartozik, melynek a szülőjéhez képest eggyel több irányított éle van. Mindez azt jelenti, hogy a fa építése során az egyes ágakon egyre szűkül a lehetséges

megoldások száma, míg a fa leveleihez olyan gráfok tartoznak, melyeknek csak irányított éle van. Az algoritmus során a cél az, hogy úgy építsük fel a döntési fát, hogy valamilyen logika alapján lehetőleg már a korai szakaszban kizárjuk azt, hogy az optimális megoldás a fa egy adott részfájában van, így azzal az ággal nem is kell foglalkoznunk, nem kell felépítenünk sem, hiszen biztosra vehetjük, hogy nem ott lesz az optimális megoldás [5].

Annak ellenére, hogy sok éven keresztül a legjobb BB stratégiák megtalálása állt a kutatások középpontjában, az eredmények azt mutatták, hogy csak viszonylag kisszámú művelet (néhány 100 db) esetén találnak elfogadható futási időn belül optimális megoldást az általános job-shop scheduling feladatra. Ennek következtében az 1970-es és ’80-as években a kutatások fókusza ismét áthelyeződött, és a legfőbb cél a probléma komplexitásának meghatározása lett. A kérdésre M. R. Garey, D. S. Johnson és Ravi Sethy 1976-os „THE COMPLEXITY OF FLOWSHOP AND JOBSHOP SCHEDULING” című munkájukban adják meg a végső választ. Ebben sikerült bebizonyítaniuk, hogy azok a job-shop scheduling problémák, ahol m >= 2 (több mint 1 gépre kell az ütemezést végrehajtani) NP-nehéz feladatok, ezért az általános feladat megoldására nem létezik polinom idejű algoritmus (feltételezve, hogy P != NP). Mindez ismét új irányt adott a kutatásoknak, hiszen nyilvánvalóvá vált, hogy az optimális megoldás keresése az általános feladatra nem fog gyakorlatban is használható algoritmust eredményezni [1][6].

A bizonyítás következményeként az 1980-as évektől kezdve a kutatások súlypontja az optimalizáló algoritmusokról áthelyeződött a közelítő (approximációs) algoritmusokra. Ezek lényege, hogy alkalmasan választott stratégiával az optimális megoldáshoz közeli eredményt próbálnak meghatározni úgy, hogy közben a végrehajtási idő polinomiális maradjon. Ezek egyik első megjelenési formái az ún. prioritási szabályokon alapuló ütemezési algoritmusok (priority dispatch rules - pdrs) voltak. Ezek lényege, hogy valamilyen szabályrendszer alapján minden művelethez hozzárendelnek egy prioritás értéket, és a műveleteket ezek sorrendjében ütemezik az erőforrásokon. Az eljárás előnye, hogy könnyen implementálható, és gyors lefutású algoritmusokat lehet létrehozni a segítségével, ugyanakkor a sok megvalósított megoldás azt mutatja, hogy a pdrs eljárások hatékonyságát jelentősen befolyásolja a konkrét megoldandó feladat, valamint az algoritmusok meglehetősen statikusak. Mindez azt jelenti a megoldás megalkotása során nem veszik figyelembe a már ütemezett feladatok által meghatározott pillanatnyi helyzetet, csak a kiinduló állapot alapján dolgoznak [1].

A prds eljárások korlátai miatt hamar felmerült az igény további heurisztikus megközelítésekre, melyek ki tudják küszöbölni az ismertetett hátrányokat, ugyanakkor

továbbra is polinomiális időben lefutnak. Ezen kutatások alapjait Marshall L. Fisher és Alexander H. G. Rinnooy Kan „The Design, Analysis and Implementation of Heuristics” című 1988-as munkája fektette le. A két kutató olyan általános érvényű technikákat határoztak meg a heurisztikák készítéséhez, melyeket a későbbiekben széles körben alkalmaztak nem csak a job-shop scheduling feladat megoldására, hanem további, gyakorlati életben előforduló ütemezési problémák esetén is. Ezen eredmények hatására az 1980-as évek végén, ’90-es évek óriási lendületet vettek a kutatások, és akkoriban született „néhány a leginnovatívabb algoritmusok közül” [1].

Ezen újító megoldások közül az első Joseph Adams, Egon Balas és Daniel Zawack által 1988-ban kidolgozott ún. „shifting bottleneck” eljárás volt. Ennek lényege, hogy az általános feladatot felbontja m db egygépes problémára, és iteratívan minden lépésben egy erőforrás optimális ütemezését hajtja végre, majd a kapott eredmények alapján korrigálja az eddig ütemezett erőforrásokon a sorrendeket. Minden iteráció során az algoritmus először megkeresi a még nem ütemezett gépek közül a szűk keresztmetszetet, vagyis azt, aminek az optimális ütemezése a legjobban hozzájárulna a teljes átfutási idő csökkentéséhez. Ehhez minden még nem ütemezett gépen meg kell határozni az optimális ütemezést úgy, hogy a célfüggvény a maximális késés minimalizálása legyen feltéve, hogy minden művelethez adott a legkorábbi kibocsátási- (release date - r_{ij}) és a legkésőbbi esedékességi (due date – d_{ij}) idő is, melyek a műveletek időadatai alapján meghatározhatók [7].

Ezen értékek alapján gépenként kiszámolható a lokálisan optimális sorrend, valamint a maximális késés. Ezeket összevetve megkapjuk, hogy a maximális késések közül melyik a legnagyobb, és az iterációban az ehhez tartozó erőforráson kell az átfutási időt minimalizáló sorrendet meghatározni, lévén, hogy ez az erőforrás a szűk keresztmetszet. Ehhez Carlier 1982-es algoritmusát használják a szerzők. Végül a már ütemezett erőforrások között minden iteráció végén meg kell próbálni az újraoptimalizálást, a szűk keresztmetszet kereséséhez hasonló módon. Az eljárást addig kell ismételni, míg az összes gépet be nem ütemeztük [7].

Annak ellenére, hogy ez az algoritmus nagy népszerűségnek örvendett a maga idejében, és több későbbi kutatás alapjául is szolgált, viszonylag hamar megmutatkozott legnagyobb gyengepontja, az újraoptimalizálás bonyolultága és nagy számításigénye. Ugyanakkor a megoldás egyes elemei a későbbiekben számos algoritmusban felhasználásra kerültek, melyek közül a legismertebb Balas és társai 1998-as „Job Shop Scheduling with Deadlines” című munkája, melyben sikeresen alkalmazták a módszer módosított változatát olyan esetekre, melyeknél esedékesség is adott a feladatoknál [1].

A ’90-es évek elején tapasztalt nagy lelkesedés a kutatások iránt további ígéretes approximációs algoritmusokat eredményezett az évtized folyamán. Ide tartoznak például a küszöbérték algoritmusok (Threshold Algorithms), melyek lényege, hogy a keresési fa építése során stratégiát váltanak, hogyha az aktuális megoldás és egy szomszédja közötti célfüggvény érték különbség egy adott küszöbérték alá esik. Ezek legfejlettebb változatai nagyon jó eredményeket adnak, ugyanakkor a futási idejük kiábrándító [1].

Az időszak másik ígéretes megoldásai a Tabu Search (TS) algoritmusok, melyek szintén sikeres megoldásokat adtak a job-shop scheduling feladatra. Ezen algoritmusok olyan módszer szerint működnek, ami lehetővé teszi, hogy keresés során ne jussanak el többször olyan vagy hasonló megoldásokhoz, mint amit egyszer már elvetettek. Ezen módszerek közül máig legígéretesebb Nowicki és Smutnicki 1996-os algoritmusa, amely kellően jó megoldásokat képes találni nagyon rövid idő alatt [1].

A 2000-es évek elejére világossá vált, hogy az 1950-es évek óta eltelt időszakban egy teljes ciklus zajlott le a kutatások irányának alakulásában, amely ciklus mind a mai napig megfigyelhető. Napjainkban az 1950-es években népszerű heurisztikus módszerek kutatása a legelterjedtebb terület a job-shop scheduling feladat kapcsán. A kapcsolódó legújabb kutatások ugyanakkor sokat merítenek a természetben megfigyelhető jelenségekből is. Ezek legjobb példái az evolúciós algoritmusok (Evolutionary Algorithms), melyek a természetes kiválasztódás jelenségén alapulnak. Ezen kívül napjaink egyre elterjedtebb megoldásai a hibrid algoritmusok is, melyek már meglévő stratégiák hatékony kombinációjára épülnek, és így próbálnak a korábbiaknál jobb eredményeket elérni [1][12].

2.2 További korlátozó feltételek

Az eddig bemutatott eljárások mindegyike a fejezet elején definiált matematikai modellnek megfelelő feladat megoldásával foglalkozott. Ugyanakkor a gyakorlati életben előforduló ütemezési problémák többsége a leírtaknál sokkal több korlátozó feltételt tartalmaz, amelyeket a megoldás során figyelembe kell venni. Ebben a szakaszban sorra veszek néhány olyan további feltételt, melyek a gyakorlatban használatosak, és amelyekkel az algoritmus megvalósítása során foglalkozni fogok.

2.2.1 Kibocsátási és esedékességi dátumok/idők

A „shifting bottleneck” algoritmus tárgyalása során már említettem, hogy az egyes műveleteknek lehetnek legkorábbi kezdési és legkésőbbi befejezési időpontjai. Akkor azonban ezeket az eljárás során ezek csupán a megadott műveleti időadatok alapján kerültek meghatározásra, és figyelembe vételre a megoldás során. A termelő vállalatoknál viszont jellemző az, hogy az elvégzendő gyártási feladatokhoz hozzárendelnek esedékességi dátumot, amikorra legkésőbb el kell készülnie a terméknek (pl. a vevői megrendelés alapján), illetve kibocsátási dátumot, amikor a munkavégzés leghamarabb elkezdhető (pl. a szükséges alapanyagok akkorra érkeznek be). Minkét időadat kiemelt fontosságú a terv készítése szempontjából, és az egyes műveletek időadatai segítségével nemcsak a teljes elvégzendő feladatra, hanem azok egyes műveleteire is meg lehet határozni a kibocsátási- és esedékességi időpontokat.

2.2.2 Feladatok közötti precedencia

Valós ipari környezetben egyáltalán nem ritka az, hogy nem csak az egy feladaton belüli műveleteknek van egy előre meghatározott sorrendje, hanem az egyes elvégzendő feladatok között is van ilyen megkötés. Ezek többnyire olyankor fordulnak elő, amikor egy gyártási feladathoz felhasznált alkatrész egy másik gyártási feladat eredményeként áll elő, vagy pl. egy beruházási projekt esetén az engedélyezési eljárásnak meg kell előznie a megvalósítási szakaszt. Ezek a korlátozások mind a Gantt-diagramon, mind pedig a diszjunktív gráf modellben jól megjeleníthetők és kezelhetők.

2.2.3 Beállítási- és átállási idők

Gyakran előfordul a termelő vállalatoknál, hogy egy-egy gyártó berendezésen nem lehet azonnal megkezdeni a soron következő művelet végrehajtását az előző művelet befejezése után. Ennek lehet az oka az, hogy pl. egy fröccsöntő gépen más formájú munkadarabokat kell készíteni, mint az előző feladatnál, és ezért a szerszámot ki kell cserélni, vagy a gyártásindítás előtt ellenőrizni kell a gyártósor beállításait. Ezek mind-mind olyan technológiai, ún. beállítási idők, amik önmagukban a műveletvégzésre fordított időtartamot nem befolyásolják, ugyanakkor szerepük lehet az átfutási idők alakulásában, ezért oda kell figyelni rájuk, és számolni kell velük a műveleti idők megadásakor.

A beállítási időktől bonyolultabb esetet képeznek az ún. átállási idők, amelyek szintén technológiai idők, tehát tényleges termelés ezen időtartam alatt sem történik, viszont ezek megléte és hossza nem csak a soron következő művelettől függ, hanem az azt megelőzőtől és magától az erőforrástól is. Tipikusan ilyen időtartamok az élelmiszeriparban a géptisztítási, mosatási idők. Amennyiben egy berendezésen pl. előbb gluténmentes termék készült, majd utána glutén tartalmú, akkor a gépet a két gyártás között egyáltalán nem, vagy csak rövid ideig szükséges takarítani. Ellenben ha glutén tartalmú termék előállításról áll át a gép gluténmentes termékre, akkor egy hosszú, akár több órás takarítási ciklust kell beitatni, mielőtt a következő műveletet meg lehet kezdeni. Ezen időadatok szintén fontos paraméterei egy valós környezetben működő termelésütemező rendszernek, éppen ezért ezeket sem lehet figyelmen kívül hagyni.

2.2.4 Anyagmozgatási idők

Egy nagy üzemcsarnok esetén könnyen előfordulhat, hogy az egyik gépen megmunkált termékeket a következő művelet elvégzésére egy olyan géphez kell szállítani, ami a gyár egy távoli pontján van. A transzportálás időszükséglete szintén fontos, nem elhanyagolható tényező, melyet az ütemterv készítése során is figyelembe kell venni. Ugyanakkor az eddig tárgyalt korlátozásoktól eltérően ez az időadat a gép felhasználható munkaidejét nem csökkenti, vagyis ezen időtartam alatt az adott erőforráson másik művelet végezhető. Tehát ez az időadat csak egy művelet legkorábbi kibocsátási időpontját befolyásolja, de gépidőt nem foglal. Ez a gyakorlatban annyit jelent, hogy az ütemterv elkészítése során arra kell figyelni, hogy egy feladat két egymást követő művelete között elteljen legalább az anyagmozgatási időnek megfelelő időtartam, illetve ennek következtében természetesen az érintett feladat teljes átfutási ideje is növekedni fog.

2.2.5 Nem folyamatos munkavégzés

Valós ipari környezetben viszonylag ritka az olyan eset, amikor a gyártó berendezések megszakítás nélkül, folyamatosan tudnak üzemelni, és egy ütemezett műveletet bármely időpontban végre tudnak hajtani. Általában a gépek nem éjjel-nappal működnek illetve hétvégén és munkaszüneti napokon sem feltétlenül ugyanolyan munkarendben dolgoznak, mint hétköznapokon. Ezen folyamatos munkavégzéstől való eltérések leírására többnyire az adott

géphez hozzárendelt munkarend vagy naptár szolgál, amely tartalmazza, hogy az egyes berendezések a konkrét naptári napokon pontosan mettől meddig üzemel. Az ütemterv elkészítése során számolni kell ezekkel az időadatokkal is, és egy-egy művelet adott gépen adott időpontra történő ütemezése során figyelni kell arra, hogy csak olyan időpontra kerüljön művelet, amikor a kapcsolódó naptár alapján munkaidő van.

2.3 További lehetséges célfüggvények

A fejezet elején bemutatott matematikai modellben a feladat célfüggvényeként a teljes átfutási idő minimalizálását jelöltem meg. Ennek az volt az oka, hogy a bemutatott kutatások, melyek a leginkább előre lendítették a job-shop scheduling probléma megoldását, azok szintén ezt tekintették célnak, ezért az összehasonlításukat egyértelmű objektív szempontok alapján tudtam elvégezni. Ugyanakkor elvonatkoztatva az elméleti kutatásoktól, a gyakorlatban az átfutási idő minimalizálása nem mindig elsődleges fontosságú. A fejezet további részében olyan egyéb célfüggvényekkel foglalkozom, melyek a gyakorlati példák során felmerülhetnek, és melyeket az algoritmus megalkotása során figyelembe veszek.

2.3.1 Késés minimalizálása

Mivel a termelő vállaltok esetében a gyártás valamilyen határidőhöz kötött (esedékességi dátum), amelyek végső soron mindig valamely vevői igény kielégítésére vezethetők vissza, ezért érthető, hogy a határidők be nem tartása komoly hátrányokat, anyagi veszteséget okozhat a cégek számára. Éppen ezért a termelésütemezés során lehet a cél a késések minimalizálása is, vagyis törekedni lehet arra, hogy lehetőleg minél kevesebb alkalommal forduljon elő az, hogy egy feladat az esetékességi dátuma után készül el. Ugyanakkor ennek is több megközelítése lehetséges. Egyrészt amennyiben a késedelmes teljesítésért járó „büntetés” független a késés időtartamától, akkor érdemes lehet a késedelmes feladatok számának minimalizálását kitűzni célul. Másrészt viszont, ha a „büntetés” a késedelem hosszával arányosan nő, akkor elképzelhető, hogy a maximális vagy az átlagos késés minimalizálására kell törekedni.

2.3.2 Készletszintek minimalizálása

Abból eredően, hogy a nagy raktárkészletek tartása magas költségekkel jár a vállalatok számára, érdemes figyelembe venni a termelésprogramozás során annak a lehetőségét, hogy a készletszintek minimalizálása legyen a cél. Ennek lényege, hogy minden feladatot úgy ütemezzünk az erőforrásokon, hogy azok tervezett befejezési időpontja minél közelebb essen az esedékességi dátumhoz, de ne haladja meg azt. Ezzel biztosítható, hogy az előállított termékek a lehető legkevesebb időt töltsék a vállalat raktáraiban, ezáltal a hozzájuk kapcsolódó raktározási- és tőkeköltségek minimalizálhatók. Ugyanakkor a megfelelő egyensúly megtalálása a biztos vevőkiszolgálás és a minimális készletszintek között nem egyszerű feladat.

3. A gyakorlati probléma és a kidolgozott algoritmus

Ebben a fejezetben ismertetem az előzőleg bemutatott ütemezési feladat egy valós ipari követelményeket közelítő változatát. A problémát formalizálom az előző fejezetben bevezetett jelölések figyelembe vételével, majd a prds algoritmusok elvéből kiindulva megfogalmazok egy lehetséges megoldási módszert a feladatra.

3.1 A megoldandó probléma

Az általános job-shop scheduling feladathoz hasonlóan adott n db elvégzendő feladat és m db rendelkezésre álló erőforrás. Minden feladathoz tartozik egy további paraméter is, ami a feladat típusát határozza meg. Így tehát egy feladatot a $J_i(k)$ érték határoz meg ($k \in (1, K)$). Továbbá minden feladathoz adott az r_i kibocsátási- és a d_i esedékességi időpontok, melyek meghatározzák, hogy mikor lehet legkorábban elkezdeni, illetve mikorra kell legkésőbb befejezni az adott feladatot.

Minden $J_i(k)$ feladat m_i db műveletből áll, viszont egy feladatnak tetszőleges számú olyan művelete lehet, melyet ugyanazon az erőforráson kell végrehajtani. Így előfordulhat olyan eset, hogy egy feladatnak több műveletét is ugyanazon a gépen kell elvégezni, illetve az is lehetséges, hogy van olyan gép, amelyen egy adott feladat egyetlen műveletét sem kell végrehajtani. Továbbra is egy időben egy erőforráson csak egy művelet hajtható végre, illetve egy művelet végrehajtásához egyszerre egy erőforrás vehető igénybe. Az O_{ik} műveletekhez tartozó τ_{ik} műveletvégzési idő kiegészül a művelet megkezdése előtt esedékes beállítási idővel, tehát a τ_{ik} megadása során erre is figyelemmel kell lenni.

A célfüggvényként pedig az átfutási idő minimalizálása helyett egy gyakorlati szempontból sokkal praktikusabb megközelítést, a késedelmes feladatok számának minimalizálását választom. Egy $J_i(k)$ feladatot akkor tekintek késedelmesnek, hogyha az ütemterv szerint az utolsó műveletének várható befejezési időpontja (amit jelöljön e_{i,m_i}) nagyobb, mint d_i. Ezek alapján a célfüggvény (L^*_{min}) az alábbiak szerint alakul.

$$L^*_{max} = \min(L_{max}) = \min_{lehetséges\ ütemezések} \left(\sum_{i=1}^{n} L_i \right)$$

$$L_i = \begin{cases} 1, ha\ e_{i,m_i} > d_i \\ 0, ha\ e_{i,m_i} \leq d_i \end{cases}$$

3.2 Az algoritmus

Ahogy azt a fejezet elején említettem, a feladat megoldásához egy prds típusú algoritmust használok. Ezek lényege, hogy a végrehajtandó műveletek között valamilyen heurisztika alapján egy egyértelmű sorrendet állít fel, és ebben a sorrendben hajtja végre az egyes műveleteket. A megoldás során azért alkalmazom ezt a módszert, mert a gyakorlatba könnyen átültethető, alacsony futási idejű, és a végfelhasználók számára is könnyen érthető algoritmust szeretnék készíteni, ami egy elfogadható kiinduló ütemezést készít, amit a termeléstervező az egyedi igények, tapasztalatok, bevált gyakorlatok alapján könnyen módosíthat.

A prioritásokat meghatározó algoritmus az alábbi 4, hierarchikusan egymásra épülő paraméter alapján állapítja meg az elvégzendő feladatok közötti sorrendet.

I. Feladat típusához köthető 1. paraméter
II. Esedékesség
III. Feladat típusához köthető 2. paraméter
IV. Átfutási idő

Ezek az értékek azon gyakorlati tapasztalaton alapulnak, hogy a termelő vállalatok többségénél a feladatok esedékessége az, ami jelentősen befolyásolja azt, hogy az adott feladatot mikor kell elvégezni. Ennek az a magyarázata, hogy egyrészt a vevői igények megfelelő kielégítéséhez hosszútávon létfontosságú, hogy késedelem nélkül lehessen legyártani a termékeket, másrészt viszont az nem megengedhető (és gyakran nem is kivitelezhető, pl.: rövid szavatossági idejű termékek), hogy a késedelem elkerülése érdekében folyamatosan jóval az esedékességet megelőzően fejezzük be a feladatot. Ez utóbbinak az az oka, hogy annak ellenére, hogy a vevői igények garantáltan ki lennének szolgálva, mindez nem kívánt készlet érték (és ezáltal forgótőke és készletezési költség) növekedést eredményezne. Ezt a két, egymásnak ellentmondó követelményt úgy oldom fel a megvalósítás során, hogy arra fogok törekedni, hogy az esedékesség ne csak a feladatok közötti prioritást szabályozza, hanem az ütemezés során az egyes feladatok utolsó műveletének befejezési időpontja lehetőleg minél jobban közelítse is meg azt.

Annak ellenére, hogy az esedékességnek ilyen nagy szerepe van az ütemezés során, mégsem ez áll a paraméter hierarchia csúcsán. Helyette egy nagy szabadságot és testre szabhatóságot biztosító érték az, amit a sorba rendezés során először meg kell vizsgálni. Ezt a

paraméter a feladat típusához rendelem hozzá, és az lesz a szerepe, hogy a csak esedékességi dátum alapján kiinduló sorrend meghatározást eltérítse, és az egyedi igényeknek jobban megfelelő végrehajtási sorrend alakulhasson ki. A gyakorlatban sokszor előfordul, hogy bizonyos feladatok, függetlenül azok esedékességétől fontosabbak más feladatoknál. Ilyen lehet például, hogy a termék, amit gyártunk, az egy kiemelt vevőnek készül, vagy nagyobb értékű gyártott cikkről van szó, amit mindenképpen le kell gyártani időben, mert a késedelem jelentős kötbérfizetési kötelezettséget von maga után. Az ilyen prioritások beállítására alkalmazható a feladat típushoz kötött első paraméter, amivel az algoritmus működése könnyen az alkalmazó cég működéséhez igazítható. Ha viszont nincsen ilyen jellegű testreszabásra igény, akkor minden feladat típusra azonos érték állítható be, így az esedékesség szerinti ütemezést kapjuk.

Az előző bekezdésben leírt tulajdonságok nagy része érvényes a feladat típusához köthető 2. paraméterhez is, ami a hierarchiában a 3. helyen áll. Ez arra szolgál, hogy az azonos esedékességű feladatok között állítson fel egy sorrendet a típushoz rendelt paraméter érték segítségével az egyedi igényeket figyelembe véve. Hasonlóan a típushoz rendelt másik paraméterhez, itt is megtehető, hogy minden típushoz ugyanazt a paraméter értéket rögzítjük, így tulajdonképpen, ha nincs szükség ennek figyelembe vételére, akkor egyszerűen kiiktatható.

Végül, de nem utolsó sorban az utolsó paraméter, amit figyelembe veszek az adott feladat teljes átfutási ideje, vagyis a benne szereplő összes művelet időadatainak összege ($\sum_{k=1}^{m_i} \tau_{ik}$).

A négy paramétert figyelembe véve egy egyértelmű sorrend határozható meg arra vonatkozóan, hogy hogyan kell az ütemezés készítése során a feladatokat feldolgozni. A sorrend meghatározásához célszerűnek tartom a radix rendezési algoritmust használni. Ez alkalmas arra, hogy több, hierarchikus paraméter szerint rakja sorrendbe egy halmaz elemeit, és jelen esetben pont erre van szükség. Ez a módszer a rendezési paraméterek számával megegyező számú (jelen esetben 4 db) ládarendezést hajt végre, kezdve a legutolsó paraméterrel, majd egyre felfelé haladva a listában. A ládarendezés lényege, hogy kiindulásként annyi listát hozunk létre, amennyi a rendezési paraméter lehetséges értékeinek száma lehet, és ezeket a paraméter értékei szerint növekvő sorrendbe helyezzük. Ezt követően minden egyes listába betesszük azokat a rendezendő elemeket, amelyek paramétere megegyezik a lista paraméterével. Ezt követően a listákon növekvő sorrendben végighaladva, és onnan az értékeket a berakás sorrendjében kiolvasva az adott paraméter szerint növekvő sorrendben lesznek az eredeti elemek. A radix rendezés során minden egyes iterációban a hierarchia szerint

következő paraméterrel kell végrehajtani a láda rendezést, és a listákba sorolás alatt mindig az előző rendezés kimenetének sorrendjében kell az elemeket feldolgozni [8].

Az így kapott sorrend ugyanakkor nem csak a feladatok végrehajtási sorrendjét jelentheti, csupán azt mutatja meg, hogy melyik feladat ütemezésével kell korábban foglalkozni. A konkrét ütemezés ezek után többféle elv szerint alakulhat. Jelen algoritmusban azt a megközelítést választom, hogy a kapott sorrendben ütemezem a feladatokat, azon belül pedig a műveleteket növekvő sorszám szerint. Minden műveletet a megfelelő erőforráson arra a legkorábbi pozícióra ütemezek ($t_{i1} \geq r_i$), amikorra éppen nincsen más művelet folyamatban az erőforráson. Ezzel olyan megoldást kapok, ahol a késedelmes feladatok száma viszonylag alacsony, hiszen helyes paraméterezés esetén a korábbi esedékességű feladatok magasabb prioritást kapnak. Ugyanakkor előfordulhatnak olyan feladatok, amelyek az esedékességüknél sokkal hamarabb elkészülnek, így a szükségesnél magasabb raktárkészlet szintet eredményeznek, ami a gyakorlatban nem mindig szerencsés. Éppen ezért az algoritmus alkalmazásánál a 4.6 fejezetben ezt a szemléletet némiképp módosítom, hogy jobban illeszkedjen a gyakorlati igényekhez, és a QAD EA rendszer alapvető logikájához.

4. Termeléstervezés a QAD EA rendszerben

Ebben a fejezetben áttekintem a QAD EA integrált vállalatirányítási rendszer termeléstervezési modulját, az ehhez használt adatokat és funkciókat, majd bemutatom annak egy lehetséges kiegészítését, amivel a termelés finomprogramozást el lehet végezni. Figyelembe véve a rendszer működési logikáját és funkcióit, ismertetem, hogy hogyan lehet az előző fejezetben részletezett algoritmust integrálni a modulba.

4.1 A QAD EA rendszer termeléstervezési modulja

A termelési terület, mint egy vállalat legfőbb átalakító folyamata, lefedi a gyártással kapcsolatos összes tevékenységet. Ennek érdekében rögzíthetők a QAD EA rendszerben az összes előállított és felhasznált termékkel kapcsolatos legfontosabb információk, az ún. törzsadatok. Ezek közé tartoznak többek között az egyes termékek készletezési adatai, beszerzési- és gyártási átfutási idői, rendelési mennyiségei vagy akár gyártási költségei. A gyártott tételekhez kapcsolódóan a felhasznált alkatrészeket és alapanyagokat tartalmazó darabjegyzékek és receptúrák, a gyártási folyamat lépéseit leíró műveletterv és folyamatok, valamint a műveletek során alkalmazott termelő berendezések, az ún. homogén gépcsoportok és gépek is törzsesíthetők. A termelési modul összes funkciója ezekre az alapadatokra fog hivatkozni a folyamatok során. Ezek alapján a gyártott termékre gyártási feladatok, ún. gyártási rendelések (továbbiakban GYR) adhatók ki. Ezek tartalmazzák, hogy milyen termékből, mekkora mennyiséget, milyen alapanyagokból, milyen műveletek elvégzésével és mikorra kell legyártani. Egy ilyen folyamat előrehaladása a műveletterv megadott részletezettséggel végigkövethető, az elvégzett műveletek tényleges elvégzési ideje, a felhasznált alapanyagok és az elkészült termékek tényleges mennyisége lejelenthető, ezáltal a gyártás során felmerült költségek egészen pontosan nyomon követhetők. A gyártás végeztével az elkészült termékeket raktárra lehet venni, és innentől kezdve a logisztikai folyamatok fogják kezelni. Az említett gyártási rendelések létrehozására a kézi bevitelen túl automatizált lehetőség is rendelkezésre áll. Ezt a QAD EA fejlett anyagszükséglet számító és ütemező rendszere (Material Requirement Planning, a továbbiakban MRP) teszi lehetővé, ami így a termeléstervezés kiinduló pontját adja [9].

A termeléstervezés során az egyes értékesített termékekre vonatkozó igényekből fokozatosan előáll egy feladatlista, amely tartalmazza azt, hogy az igények kielégítése

érdekében milyen gyártási rendeléseket kell a vállalatnak teljesítenie. Ezeket a feladatokat kell később a termelésprogramozás során a rendelkezésre álló erőforrásokon ütemezni úgy, hogy az előző fejezetben kitűzött célt, a késedelmes feladatok számának a minimalizálását minél jobban megközelítsük. A teljes folyamatban a kiinduló feladatlista elkészítéséhez ad hatékony támogatást az MRP, ami az egyes értékesített termékekre vonatkozó igények, és a kielégítésükhöz szükséges ellátások közötti egyensúlyt teremti meg mindig arra törekedve, hogy az igények esedékességének időpontjában legyen az igényelt termékből akkora raktárkészlet, hogy a teljesítés megtörténhessen. A QAD EA MRP modulja igényként veszi figyelembe egy tételre vonatkozóan a vevői megrendeléseket, a biztonsági készletszint előírásokat, az előrejelzéseket és szezonális szükségleteket, a telephelyek közötti megrendeléseket valamint azokat a GYR-eket, amelyek az adott tételt beépülőként használják fel. A felsorolt igények kielégítésére pedig ellátásként veszi figyelembe egy tétel vonatkozásában a beszerzési rendeléseket, az MRP hatáskörébe tartozó raktárkészleteket, a telephelyek közötti ellátásokat valamint azokat a GYR-eket, melyek az adott tételt előállítják [9].

Egy tételre vonatkozóan az így meghatározott igények és ellátások közötti különbség az igény esedékességi időpontjában az ún. nettó szükséglet, melynek fedezésére a rendszer gyártott tételek esetén gyártási-, míg beszerzett tételek esetén beszerzési rendeléseket (továbbiakban BR) készít. Mivel a termelésprogramozás során a GYR-ek műveleteit kell ütemezni az egyes erőforrásokon, ezért a továbbiakban az MRP által generált BR-ekkel dolgozatomban nem foglalkozom. Ugyanakkor a termelési terv elkészítése során módosulhatnak ezek a nettó szükségeltek, mert az egyes GYR-eket az időtengelyen el lehet mozgatni. Ezért célszerű a végső ütemezés elkészítése után az MRP-t ismételten lefuttatni, hogy a gyártáshoz szükséges alapanyagok BR-eit a rendszer az elkészített tervhez aktualizálja [9].

Az MRP az egy tételre vonatkozó GYR(ek) létrehozása során elsőként meghatározza az adott tételre felmerülő igények esedékességi időpontjaiban a nettó szükségleteket, majd a rendszerben lévő adatok alapján meghatározza a nettó szükséglet fedezésére szolgáló GYR(ek) adatait. Ehhez az esedékességi dátumból levonja a tételtörzsben megadott gyártási átfutási időt (továbbiakban GYÁI), és az így kapott dátum lesz a GYR(ek) kibocsátási dátuma. A létrehozandó GYR(ek) számát és rendelési mennyiségét a rendszer a tétel törzsadatok között rögzített mennyiségek és a rendelési politika mező függvényében határozza meg, míg a GYR(ek) darabjegyzéke és a műveletterve a tételhez rendelt alapértelmezett darabjegyzék és műveletterv lesz [9].

Az ellátás megtervezését a rendszer rekurzívan folytatja a gyártott tétel beépülőire, méghozzá úgy, hogy a darabjegyzék struktúra felsőbb szintjén álló tételre létrehozott GYR(ek) kezdő dátuma lesz a beépülő tételre felmerülő igény esedékességi dátuma, az igényelt mennyiség pedig a felsőbb szintű GYR(ek) darabjegyzéke és rendelési mennyisége alapján számolt érték. Vagyis ha egy „A" termékre létrehozott GYR rendelési mennyisége 100 db-ra szól, és az „A" termékbe a GYR darabjegyzéke alapján 2 db „B" termék épül be, akkor a „B" termékből igényelt mennyiség 200 db lesz. Az algoritmus egészen addig fut, amíg az összes tételre felmerülő igények ellátását meg nem tervezi [9].

Az MRP által létrehozott GYRek ún. tervezett státuszú (planned – „P") GYR-ek, melyek létrehozását a rendszer javasolja az igények megfelelő kielégítése érdekében. Jóváhagyásukig vagy kézi módosításukig (firm planned – F státusz) ezeket az MRP ismételt futtatása esetén a rendszer szabadon módosíthatja és törölheti [9].

Az MRP-vel előállított gyártási rendelések egy egészen jó hozzávetőleges termelési tervet adnak eredményül, ugyanakkor az algoritmus nem vesz minden gyakorlati szempontból fontos paramétert figyelembe, ezért a kapott terv gyakran nem megvalósítható. Ennek legfőbb oka az, hogy az MRP nem veszi figyelembe az erőforrások kapacitását, tulajdonképpen végtelen kapacitásokkal számol, ami azt eredményezi, hogy a kapott tervben lehetnek olyan műveletek, melyeket ugyanazon az erőforráson ugyanabban az időszakban kell végrehajtani. Ugyanakkor az MRP által meghatározott esedékességek és létrehozott GYR-ek egy nagyon jó kiindulási alapot adnak a termelésprogramozásnak, aminek eredményeként, figyelembe véve a rendelkezésre álló erőforrások kapacitásait, már egy teljesíthető termelési terv áll elő.

4.2 A finomprogramozáshoz felhasznált adatok a QAD EA-ban

A GYR-ek adatait a QAD EA rendszer a wo_mstr adatbázis táblában tárolja el. Itt találhatók azok a legfontosabb adatok, amelyekre a finomprogramozás során szükség van. Ezeket foglalja össze a 4.1 táblázat.

Mezőnév	Típus	Megjegyzés
wo_domain	**karakteres**	**Domain**
wo_nbr	**karakteres**	**GYR száma**
wo_lot	**karakteres**	**GYR azonosítója**
wo_part	karakteres	A gyártott termék tételkódja
wo_qty_ord	decimális	Gyártandó mennyiség
wo_due_date	dátum	Esedékességi dátum

wo_rel_date	dátum	Kibocsátási dátum
wo_status	karakteres	GYR állapota
wo_bom_code	karakteres	GYR darabjegyzék kód
wo_routing	karakteres	GYR műveletterv kód

4.1 táblázat A GYR-ek legfontosabb adatai [11]

A fenti táblázat első három sorában lévő, kiemelt mezők (wo_domain, wo_nbr, wo_lot) a tábla kulcsmezői, vagyis ezek együtt határoznak meg egyértelműen egy GYR-t. A „Domain" mező azért szükséges ide (és a QAD EA szinte minden táblájába), mert a rendszer lehetőséget biztosít arra, hogy egyszerre több, egymástól elkülönült vállalatmodell működjön ugyanabban a fizikai adatbázisban. Ezeket az elkülönült egységeket a „Domain" mező segítségével lehet a rendszerben szeparálni, ezért a továbbiakban is minden kapcsolódó adatbázis táblában fel fogom tüntetni ezt a mezőt.

A táblázatban lévő két dátum mező határozza meg azt, hogy mikortól lehet egy GYR elvégzését megkezdeni (wo_rel_date), illetve meddig kell a folyamatot befejezni ahhoz, hogy a határidőt tartani lehessen (wo_due_date). Ugyanakkor ahhoz, hogy a rendszerben ténylegesen anyagkiadást, bevételezést vagy művelet lejelentést lehessen egy GYR-re végezni, először ki kell azt bocsátani. Ezáltal létrejön a GYR végleges darabjegyzéke és műveletterve, ami már nem módosítható, illetve az állapota (wo_status) kibocsátottra (released – R) változik. Ez a státusz jelöli azt, hogy a gyártás megkezdéséhez minden feltétel rendelkezésre áll.

A GYRek státuszai között található még a lebontott (exploded - E), a foglalt (allocated - A) illetve a lezárt (closed - C). Lebontott állapotra váltás során hozza létre a rendszer a törzsadatok között megadott darabjegyzék és műveletterv alapján a GYR lokális darabjegyzékét és művelettervét, és ezeket befagyasztja, vagyis nem engedi, hogy egy ismételt MRP futtatás ezt módosítsa, viszont a kézi változtatás engedélyezett. Ezt követi sorrendben a foglalt állapot, amikor a GYR lokális darabjegyzéke alapján a beépülő tételek megfelelő mennyiségére tételes foglalás történik, azaz, a raktárban lévő felhasználandó anyagok hozzárendelődnek a kiválasztott GYR-hez, és a rendszer csak ehhez engedi őket felhasználni, egyéb célra nem. Végül a kibocsátott állapotból, ha a gyártási folyamat befejeződött, lezárt állapotúra kell állítani a GYR-t. Ilyenkor további anyagfelhasználás és készletre vétel már nem történhet, viszont a GYR egyes műveleteinek állapotától függően tényidő lejelentés még megengedett.

Figyelembe véve a GYR-ek állapotainak jelentését, a finomprogram létrehozása során csak azokat a GYR-eket célszerű ütemezni, melyek R állapotúak. Ezek ugyanis már végleges

darabjegyzékkel és művelettervvel rendelkeznek, ezáltal minden adott, hogy a végrehajtásuk elkezdődhessen, így a kapacitások elosztása során érdemes velük foglalkozni. A továbbiakban csak az R állapotú GYR-eket fogom figyelembe venni az ütemezés során.

Végül, de nem utolsó sorban a wo_mstr táblában kerül eltárolásra az is, hogy a GYR darabjegyzékét (wo_bom_code) és művelettervét (wo_routing) melyik, a rendszerben törzsadatként tárolt darabjegyzék és műveletterv alapján kell létrehozni. Az ezek alapján létrehozott lokális darabjegyzék és műveletterv a GYR-hez kötötten a wod_det illetve a wr_route táblákban tárolódik el. A finomprogramozás során ez utóbbi tábla adatai alapján kerülnek meghatározásra a GYR műveleteinek legfontosabb adatai, amik a műveletek erőforrásokra történő ütemezését befolyásolják. A wr_route tábla szükséges mezőit a 4.2 táblázat tartalmazza.

Mezőnév	**Típus**	**Megjegyzés**
wr_domain	**karakteres**	**Domain**
wr_nbr	**karakteres**	**GYR szám**
wr_lot	**karakteres**	**GYR azonosító**
wr_op	**egész**	**GYR művelet szám**
wr_wkctr	karakteres	Gépcsoport kód
wr_mch	karakteres	Gép kód
wr_start	dátum	Művelet kezdő dátuma
wr_end	dátum	Művelet befejező dátuma
wr_st_time	egész	Művelet kezdő időpontja
wr_end_time	egész	Művelet befejező időpontja
wr_qty_ord	decimális	Rendelt mennyiség
wr_qty_comp	decimális	Elkészült mennyiség
wr_qty_rjct	decimális	Visszautasított mennyiség
wr_queue	decimális	Sorbanállási idő
wr_setup	decimális	Beállítási idő
wr_run	decimális	Futási idő
wr_wait	decimális	Várakozási idő
wr_move	decimális	Anyagmozgatási idő
wr_sub_act	decimális	Alvállalkozói átfutási idő
wr_tran_qty	decimális	Átfedő egység
wr_status	karakteres	Művelet állapota

4.2 táblázat A GYR művelettervének legfontosabb adatai [11]

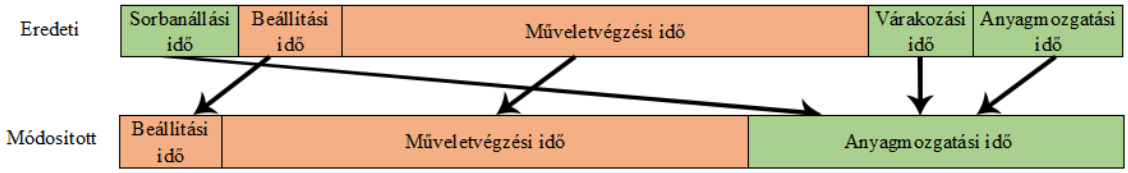
A fenti táblázatból látható, hogy az egyes műveletek hozzá vannak rendelve a kapcsolódó GYR-hez, hiszen a vastaggal jelölt elsődleges kulcsmezők között megtalálható a GYR száma, azonosítója és a művelet száma, valamint a korábban részletezett okok miatt a „Domain” mező is.

Minden művelethez adott, hogy melyik erőforráson és mennyi ideig kell végrehajtani. Ez a két információ lesz a kulcs ahhoz, hogy a finomprogramozást hatékonyan el lehessen végezni. A QAD EA-ban a műveletek definiálása során 6 féle időadat megadására van lehetőség. Ezek mindegyikét figyelembe veszi a finomprogramozó modul az ütemezés elkészítéséhez, de nem mindegyiket úgy, ahogy a rendszerben rögzítve van.

Az időadatok közül a beállítási- és műveletvégzési idők jelentése megegyezik a 2. fejezetben bemutatottakkal, azzal a különbséggel, hogy a rendszerben egy GYR egy műveletének műveletvégzési ideje (wr_run) a gyártott tétel egy egységének megmunkálási idejét jelenti. Ez alapján ahhoz, hogy a teljes futási időt megkapjuk ezt az értéket meg kell szorozni a rendelt mennyiséggel (wr_qty_ord). Az így kapott időtartam kiemelten kezelendő, mert főként ez határozza meg a művelet teljes ütemezendő időtartamát. Abban az esetben, ha a műveletet már elkezdték (wr_qty_comp != 0 vagy wr_qty_rjct != 0), akkor ennek hosszát a következőképpen kell számolni a megjelenítéshez: (wr_qty_ord – wr_qty_comp – wr_qty_rjct) * wr_run.

A sorban állási, várakozási és anyagmozgatási idők termelésszervezési és olyan egyéb technológiai időket jelentenek, melyek nem foglalnak gépidőt, viszont a finomprogram elkészítése során nem szabad megengedni, hogy egy GYR két egymást követő művelete között ezen időtartamok összegénél kevesebb idő teljen el. Ilyen idő lehet pl. a munkadarabok 2 műveletvégző berendezés közötti mozgatásának ideje, vagy egy festési művelet után a száradás ideje. A rendszer ezeket az időket 3 elkülönült mezőben tárolja, viszont a gyakorlatban ritkán szükséges ennyire részletes megbontása az időadatoknak. Ezért a finomprogramozó modul ezeket kezeli olyan módon, hogy csak az anyagmozgatási idő (wr_move) mezőbe írt értéket veszi figyelembe. Ezért ha a másik két időadattal is számolni kell, akkor a műveletterv létrehozásakor az anyagmozgatási idő mezőben kell feltüntetni az adott művelet várakozási és anyagmozgatási idejét, valamint a következő művelet sorban állási idejét. A továbbiakban anyagmozgatási idő alatt minden esetben ezen 3 időadat összegét értem.

A fenti időadatokat és azok általam a továbbiakban használt értelmezését a 4.1 ábra tartalmazza. Az ábrán eltérő színekkel különböztettem meg, hogy melyek azok az időtartamok, amelyek gépidőt foglalnak, és melyek nem.



4.1 ábra A figyelembe vett időadatok

Az utolsó időadat, amivel a rendszer foglalkozik, az az alvállalkozói átfutási idő. Ez valamelyest eltér az eddig bemutatottaktól, ezért a fenti ábrán nem tüntettem fel. Azért van rá szükség, mert egy gyártási folyamat során lehetnek olyan műveletek, amelyeket nem a vállalat a saját erőforrásain végez el, hanem egy alvállalkozó hajtja végre helyette. A standard QAD EA ennek a műveletnek az idejét az alvállalkozói átfutási idő (wr_sub_act) mezőben tárolja napokban, a művelethez tartozó gépcsoport pedig egy technikai gépcsoport. A finomprogramozás során az ilyen alvállalkozói gépcsoporton végrehajtandó műveleteket az egységes működés miatt a rendszer ugyanúgy kezeli, mintha normál műveletek lennének, éppen ezért ezeknél az alvállalkozói átfutási időt nem a műveletterv wr_sub_act mezőjében kell megadni, hanem a beállítási és munkavégzési idő mezőkben, órákban. Abban az esetben, ha szoros a kapcsolat az alvállalkozó és a megbízó között, és a rendelt mennyiségtől függ az átfutási idő, akkor célszerű a műveletvégzési idő mezőben rögzíteni az egységnyi mennyiségre vonatkozó átfutási időt. Ha az átfutási idő nem függ a mennyiségtől, akkor pedig az adatrögzítés során a beállítási idő mezőben kell az értéket rögzíteni a teljes mennyiségre vonatkozóan.

Az alvállalkozói gyártás során ugyanakkor az erőforrás terhelését nem a megrendelő feladata kiegyensúlyozni. Éppen ezért ezeknek a gépeknek a törzsadatai között egy kapcsolóval lehet jelezni, hogy alvállalkozói gépcsoport-e vagy sem, és ha igen, akkor a finomprogram elkészítése során ezeknél a rendszer végtelen kapacitásokat feltételez.

Bár nem explicit időadat, de a műveleti idők számításánál, és az egy GYR-en belüli egymást követő műveltek pozícionálásánál komoly szerepe van az ún. átfedő egységeknek. A gyakorlatban ugyanis előfordulhat olyan eset, hogy egy következő művelet elkezdéséhez nem kell megvárni azt, amíg az előző művelet a teljes gyártott mennyiségen befejeződik. Ilyen esetekben a művelet átfedő egység mezőjében tárolt érték határozza meg, hogy hány munkadarabot kell készre jelenteni ahhoz, hogy a következő műveletet el lehessen kezdeni. Ezen mennyiség és a műveleti idő szorzatából meghatározható, hogy mennyi idő alatt készül el az átfedő egységben definiált mennyiség, és ezt a művelet beállítási idejének végidőpontjához

hozzáadva meghatározható az a legkorábbi időpont, amikor a következő művelet elkezdhető (az anyagmozgatási időt nem számolva).

Az időadatokon kívül fontos paraméter még az is, hogy egy műveleten mekkora mennyiségű munkadarabot kell megmunkálni, és abból mennyi készült már el, illetve mennyi van még hátra. Ezeket határozza meg a rendelt mennyiség (wr_qty_ord), az elkészült mennyiség (wr_qty_comp) és a visszautasított mennyiség (wr_qty_rjct) mezők értéke. A művelet nyitott mennyiségét, vagyis azt, hogy hány munkadarabot kell még megmunkálni a wr_qty_ord − wr_qty_comp − wr_qty_rjct képlettel lehet kiszámolni. A finomprogramozás során ez hasznos információ, hiszen ha a tervben lévő GYR-ek között van olyan, amelynek valamely műveletére már történt tényadat lejelentés, akkor látható, hogy mekkora mennyiség az, amit még meg kell munkálni az adott műveleten, és ezen érték valamint az időadatok alapján a művelet időtartama a valóságnak megfelelően aktualizálható, és így az ütemezés pontosabban elvégezhető.

Végül, de nem utolsó sorban a művelet állapota (wr_status) az, ami még hasznos információval tud szolgálni a feladat készültségéről. A finomprogramozás szempontjából a legfontosabb a „C”-s vagyis lezárt állapot. Ilyen műveletre nem lehet több tényadatot lejelenteni, vagyis a folyamat szempontjából az adott művelet elvégzettnek tekinthető. Éppen ezért csak azokkal a műveletekkel fogok a tervben foglalkozni, amelyek nem „C” állapotúak.

Az eddig említett adatok a végrehajtandó műveleteket írták le, ám arról is fontos szót ejteni, hogy mindezt milyen erőforrásokon kell véghezvinni. Ahogy azt már korábban említettem, minden egyes művelet hivatkozik arra a homogén gépcsoport-gép kombinációra, amelyen azt el kell végezni. A továbbiakban egy homogén gépcsoport-gép kombinációt erőforrásnak fogok hívni, és a műveleteket a finomprogramozás során ezekhez fogom hozzárendelni. A mögöttük álló gépcsoport-gép adatok a standard QAD EA rendszer wc_mstr és shop_cal táblájában tárolódnak.

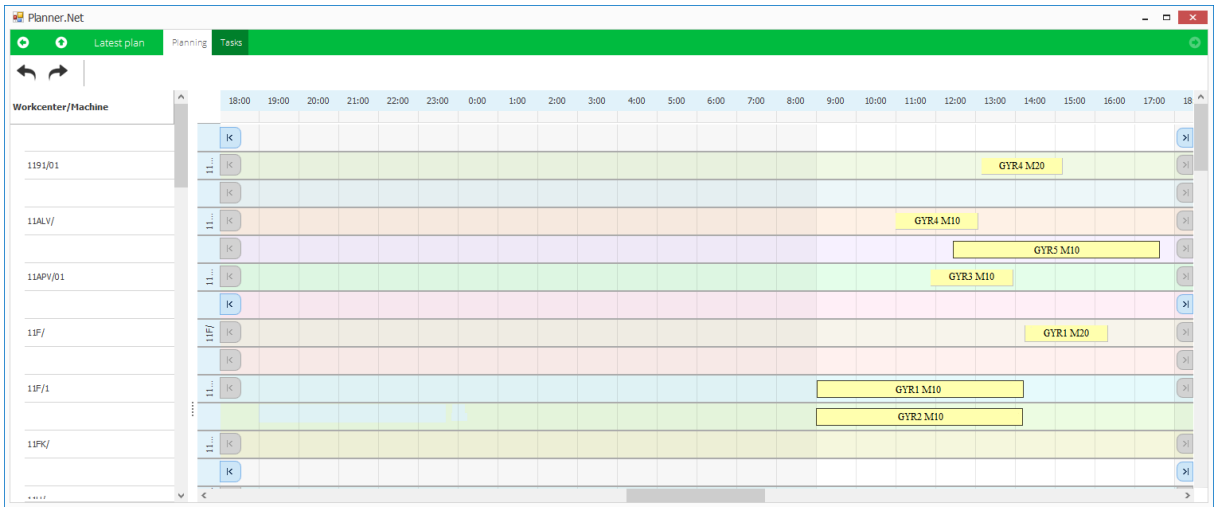
4.3 A finomprogramozó modul funkcionalitása

Ahogy arról már a korábbi fejezetekben részletesen írtam, a finomprogramozás során a GYR-ek műveleteit kell a rendelkezésre álló alkalmas gyártó berendezéseken ütemezni úgy, hogy a feladatok mindegyike határidőre elkészüljön, ezáltal a felmerülő igényeket késés nélkül ki lehessen elégíteni. Azért, hogy mindezt hatékonyan, gyorsan és felhasználóbarát módon

lehessen megtenni, a finomprogramozó modul tartalmaz egy Gantt-diagramon alapuló grafikus tervezőfelületet, ahol a kezelő láthatja, hogy az egyes GYR-ek műveleit mikor, melyik erőforráson szükséges elvégezni. A tervezőasztal X tengelye jelképezi az időtengelyt, az Y pedig a rendelkezésre álló erőforrásokat a standard QAD EA-ban szereplő gépcsoport-gép bontásban. A műveleteket egy-egy vízszintes sáv jelöli a tervezőasztalon, amely a kapcsolódó művelet helyzetétől függően különböző színű. Ha pl. a művelet olyan GYR-hez tartozik, amely utolsó műveletének befejezési időpontja későbbi, mint a GYR esedékességi dátuma (azaz a GYR késedelmes), akkor a műveletet piros színű csíkkal jelenik meg. Amennyiben a GYR nem késedelmes, akkor a művelethez tartozó csík zöld színű. Illetve egy művelet műveletvégzési idejét és beállítási idejét jelző sávokat is jól elkülöníthetőek egymástól.

Annak érdekében, hogy a modul a túlterheléseket jól meg tudja jeleníteni, és ez alapján a felhasználó kézi módosítással, vagy automatizáltan meg tudja ezeket szüntetni, minden erőforráshoz két műveleti sor tartozik, amelyekben a műveletek elhelyezhetők. A pozíciójában magasabban lévő az ún. fő sor. Amennyiben semmilyen, később részletezett kényszert nem sért az elmozgatott művelet a célerőforrás kiválasztott időintervallumában, akkor a fő sorba kerül. Ezen kívül minden erőforráshoz tartozik egy ún. másodlagos sor (a továbbiakban BIN), ami arra szolgál, hogy azokat a műveleteket gyűjtse össze, melyek a mozgatási szabályok bizonyos részét megsértik, de másokat nem. Ebben a másodlagos sorban található műveleteket a speciális, „linearizálás” mozgatás segítségével lehet a kényszerek betartása mellett automatizáltan a fő sorba emelni, ezáltal a termelési tervet kiegyensúlyozni.

A program segítségével a műveletek egyenként vagy csoportosan, különböző logikák szerint áthelyezhetők a tervezőasztalon, de minden esetben csak úgy, hogy az új állapot megfeleljen a műveletek előre meghatározott sorrendjéből származó előírásoknak. Ezen szabályok és ezek alapján a mozgástípusok részletes leírását a 4.4 és 4.5 fejezet tartalmazza. A termelésütemező munkaasztal felépítését a 4.2 ábra szemlélteti.



4.2 ábra A termelésütemező munkaasztal

A tervezőfelületen kívül a modul másik elengedhetetlen elemét azok az adatkinyerést szolgáló keresők jelentik, melyek a standard QAD EA rendszer browse-aihoz hasonlóan, táblázatos formában mutatják meg a tervezés alapjául szolgáló adatokat. Ezek segítségével alapvetően a tervezőasztalon lévő GYR-ek és erőforrások részletező adatait lehet megjeleníteni, ezáltal egyszerűsítve a finomprogramozást. A GYR-ekkel összefüggő keresők a Modell-View-Controller (MVC) architektúrát alkalmazva kapcsolatban állnak a grafikus tervező felülettel, vagyis a felületen egy GYR egy műveletének kijelölésével a támogató keresőkön is az ahhoz kapcsolódó adatok kerülnek megjelenítésre.

4.4 A modul által figyelembe vett kényszerek

A tervezőasztalon a műveletek áthelyezése során többféle mozgatási módszert lehet használni, melyekkel részletesen a 4.5 fejezetben foglalkozom. Közös ezekben az, hogy bármelyiket is választja a felhasználó, a rendszer a módosítás végrehatása előtt minden alkalommal ellenőrzi, hogy bizonyos előírások, ún. kényszerek betartásra kerülnek-e, és ha nem, akkor módosítja a mozgatást, vagy nem is engedélyezi azt. Ezeket a kényszereket két nagy csoportra lehet osztani, melyek között az alapvető eltérés abban van, hogy a csoportba tartozó kényszerek esetén a mozgatást meg kell-e tiltani, vagy elegendő a műveletet a fő sor helyett a BIN sorba helyezni, és ezáltal konfliktust generálni az adott gépen. A kényszerek csoportosítását, és a csoportok elemeinek részletes leírását az alábbi fejezet tartalmazza.

4.4.1 Műveletekből fakadó kényszerek

Ebbe a csoportba azok a kényszerek tartoznak, melyek abból erednek, hogy bizonyos műveletek végrehajtása során szigorú sorrendiséget kell figyelembe venni és betartani. Abban az esetben, ha a mozgatandó művelet új pozíciója ilyen kényszert sért, akkor a mozgatás soha nem engedélyezett.

Az első, és legfontosabb kényszer a kategórián belül az egy GYR-en belüli egymást követő műveletek megfelelő sorrendjének a betartása. Ez azt jelenti, hogy ha egy GYR nagyobb sorszámú műveletét úgy szeretnénk elmozgatni, hogy annak kezdési időpontja korábbra essen, mint ugyanazon GYR előző műveletének befejezési időpontja, és a megelőző műveletnél nincsen átfedő egység megadva, akkor a mozgatás nem engedélyezett. Ugyanez igaz a másik irányban is. Ha egy mozgatandó művelet befejezési időpontja későbbre esne, mint ugyanazon GYR következő műveletének kezdő időpontja, és nincsen átfedő egység megadva, a mozgatást a rendszer akkor is letiltja.

Előfordulhat olyan eset, hogy az időtengelyen visszafelé mozgatott műveletet megelőző műveletnél átfedő egység van rögzítve. Ebben az esetben a mozgatott művelet kezdő időpontja nem lehet korábbi, mint a GYR megelőző műveletének műveletvégzési idejének kezdete + az előző művelet átfedő egységének elvégzéséhez szükséges idő + az előző művelet anyagmozgatási ideje. Továbbá a mozgatott művelet befejezési időpontja sem lehet korábbi, mint az előző művelet befejező időpontja + az előző művelet átfedő egységének elvégzéséhez szükséges idő + az előző művelet anyagmozgatási ideje.

Természetesen az átfedő egység esetén az ellenőrzések elvégzése az időtengelyen előre történő mozgatás során is biztosított, ha a mozgatandó műveletnél átfedő egység be van állítva. Ekkor a mozgatott művelet műveletvégzési idejének kezdete + az átfedő egység elvégzéséhez szükséges idő + az anyagmozgatási idő alapján számolt időpont nem lehet későbbi, mint a követő művelet kezdő időpontja. Továbbá a követő művelet befejezési időpontja sem lehet korábbi, mint a mozgatott művelet befejező időpontja + a mozgatott művelet átfedő egységének elvégzéséhez szükséges idő + a mozgatott művelet anyagmozgatási ideje.

Amennyiben ezen előírások bármelyikét a végrehajtott mozgatás sérti, akkor a mozgás nem engedélyezett.

A következő kényszer a kategóriában az 2.2.2 fejezetben bemutatott feladatok közötti precedenciák megfelelője a finomprogramozási modulban. A GYR-ek között ún. linkekkel lehetőség van megadni azt, hogy egy GYR egy tetszőleges műveletének elkezdését melyik

másik GYR utolsó műveletének befejezése kell, hogy megelőzze. Ezzel előírható, hogy egy GYR egy műveletének beépülő tételeit melyik másik GYR biztosítja. Abban az esetben, ha a mozgatásra kiválasztott művelet új kezdő időpontja korábbra esik, mint a hozzá beállított tetszőleges megelőző GYR (több is lehet) utolsó műveletének befejezési időpontja, akkor a mozgatás nem engedélyezett.

Az előzőhöz hasonlóan ez a kényszer is érvényes az időtengelyen előre felé haladva is. Vagyis ha az adott mozgatással az érintett GYR utolsó műveletének befejezési időpontja, mint a „linkkel" hozzákapcsolt másik GYR kijelölt műveletének kezdő időpontja, akkor a mozgatás szintén tiltott.

Amennyiben egy művelethez „linkkel" hozzárendelt megelőző GYR utolsó műveleténél van definiálva átfedő egység, akkor a követő művelet kezdő időpontja nem lehet korábbi, mint a megelőző GYR utolsó műveletének műveletvégzési idejének kezdő időpontja + az átfedő egység megmunkálásához szükséges idő + az anyagmozgatási idő.

Abban az esetben, hogyha egy műveletet nem csak időben szeretnénk arrébb helyezni, hanem egy másik erőforrásra kívánjuk ütemezni, akkor a rendszer ellenőrzi azt is, hogy a kiválasztott művelet a célerőforráson elvégezhető-e. Ezt hívjuk alternatív műveletterv kényszernek. Ennek érvényesítéséhez a program ellenőrzi, hogy van-e olyan alternatív műveletterv a rendszerben a gyártott tételre, melynek a mozgatni kívánt művelete a cél erőforráson végzendő. Az ellenőrzés során ugyanakkor figyel a rendszer arra is, hogy az alternatív műveletterv csak az adott művelet tekintetében jelentsen változást, az azt megelőző és követő műveletek ne módosuljanak, azaz ne legyen technológia váltás, csupán a műveletvégző erőforrás cserélődjön ki egy azzal hasonló adottságokkal rendelkező másik gépre. Amennyiben van ilyen műveletterv, és egyéb kényszer sem sérül, a GYR műveletterve az alternatív műveletterve módosul.

Végül, de nem utolsó sorban a műveletekből fakadó kényszerek közé tartozik az anyagmozgatási idők betartása is. A GYR műveletterében definiált összes műveletnél megadható egy anyagmozgatási idő, melynek szerepét a 4.2 fejezetben részleteztem. A műveletek mozgatása során a rendszer mindig ellenőrzi, hogy a mozgatandó művelet illetve a GYR-ben azt megelőző illetve követő művelet között eltelik-e a definiált mozgatási időtartam. Amennyiben ezt az előírást a mozgatandó művelet új kezdési és befejezési időpontja bármelyik irányban sérti, és nincsen a mozgatandó és az azt megelőző műveletnél átfedő egység megadva, akkor a mozgatás nem engedélyezett.

Ha a mozgatandó műveletet a GYR-ben megelőző műveletnél be van állítva átfedő egység, akkor az időtengelyen visszafelé történő ütemezés során a mozgatott művelet új kezdő időpontja nem lehet kisebb, mint a GYR megelőző műveletének műveletvégzési idejének kezdete + az előző művelet átfedő egységének elvégzéséhez szükséges idő + az előző művelet anyagmozgatási ideje. Továbbá a mozgatott művelet befejezési időpontja sem lehet korábbi, mint az előző művelet befejező időpontja + az előző művelet átfedő egységének elvégzéséhez szükséges idő + az előző művelet anyagmozgatási ideje.

Ha a mozgatandó műveletnél van beállítva átfedő egység, akkor a mozgatott művelet műveletvégzési idejének kezdete + az átfedő egység elvégzéséhez szükséges idő + az anyagmozgatási idő alapján számolt időpont nem lehet későbbi, mint a követő művelet kezdő időpontja. Továbbá a követő művelet befejezési időpontja sem lehet korábbi, mint a mozgatott művelet befejező időpontja + a mozgatott művelet átfedő egységének elvégzéséhez szükséges idő + a mozgatott művelet anyagmozgatási ideje.

4.4.2 Pozícióból fakadó kényszerek

Ide azok a kényszerek tartoznak, melyek nem szigorú sorrendiséget határoznak meg, csupán azt írják elő, hogy egy időintervallumban egyazon erőforráson több művelet nem hajtható végre, ugyanakkor a műveletek közötti sorrend alapvetően tetszőleges lehet. A mozgatások során azokat a műveleteket, melyek „csak” ilyen kényszereket sértenek, a kívánt pozícióba lehet mozgatni, viszont többnyire nem az erőforrás fősorába, hanem a másodlagos sorba kerülnek.

Első ilyen kényszer a célerőforrás naptárának ellenőrzésére vonatkozik, melyre a 2.2.5 fejezetben már utaltam. A célerőforrás naptárában egyértelműen definiálva van, hogy egy adott időpontban az erőforráson történhet-e gyártás. Ha a mozgatott művelet módosított kezdési és befejezési időpontja között az erőforrás naptárában van olyan időtartam, amikor állásidő van rögzítve, akkor a mozgatandó művelet hossza a művelet időtartamába eső állásidőkkel megnövelésre kerül, és úgy helyezi a program az erőforrás fő sorába.

A következő pozícióból fakadó kényszer az átállási idők ellenőrzésére vonatkozik. Ilyet akkor célszerű definiálni, amikor egy erőforráson két, egymást követő GYR között olyan tevékenységet (pl: mosatás) kell elvégezni, amely egyszerre függ az erőforrástól valamint a két egymást követő gyártott tételtől.

Mindezt figyelembe véve egy művelet mozgatása során a rendszer ellenőrzi, hogy a célerőforráson a mozgatott művelet kezdő időpontja és a megelőző művelet befejezési időpontja között eltelik-e az adott erőforrásra és a megelőző illetve követő művelet GYR-ének tételkódjára definiált átállási időtartam. Ugyanezt ellenőrzi a mozgatandó művelet befejezési időpontjára és a követő művelet kezdési időpontjára is. Amennyiben bármelyik irányban nem telik el a definiált átállási idő, akkor a mozgatandó művelet a cél erőforrás BIN sorába kerül.

Ha a definiált átállási idő mindkét irányban eltelik, akkor a mozgatandó művelet az erőforrás fő sorába kerül, és az átállási idők minden művelet előtt megjelennek.

Az utolsó pozícióból eredő kényszer pedig a párhuzamos művelet végrehajtás ellenőrzése. Ennél a rendszer ellenőrzi azt, hogy a művelet kezdő és befejező időpontjai között van-e a célerőforráson egyéb végrehajtandó művelet. Amennyiben van ilyen művelet, akkor a mozgatandó művelet az erőforrás BIN sorába kerül. Ha ott is van művelet, akkor a bin sor magassága megnő, és a mozgatandó művelet a másik művelet alá, azzal párhuzamos pozícióra kerül, de az előírt kezdő és befejező időpontja megmarad.

4.5 A finomprogramozó modul mozgatási funkciói

Ez a fejezet a finomprogramozó modul által támogatott művelet mozgatási típusok leírását tartalmazza, figyelembe véve a korábban definiált kényszereket. Ezek segítségével a GYRek műveletei mind az időtengelyen mind pedig az erőforrások között áthelyezhetők, kézi módosítással, és automatizáltan egyaránt, így az igényeknek megfelelő termelési terv egyszerűen elkészíthető.

4.5.1 Kézi mozgatás

Ez a legegyszerűbb mozgatási módszer, automatizmusok nélkül. Segítségével a tervezőasztalon drag&drop módszerrel lehet az egyes műveleteket más időintervallumra és erőforrásra áthelyezni, az időtengelyen mind előre mind pedig visszafelé haladva. A kiválasztott művelet új kezdési ideje és erőforrása az egérgomb elengedésekor fennálló kurzorpozíció alapján kerül meghatározásra. Az erőforrást és annak célsorát az Y, a kezdőidőpontot pedig az X koordináta határozza meg. A mozgatás alatt az időtengely különböző felbontásaiból eredő eltérések miatt egy „tooltip” ablakban jelenik meg az aktuális

egérpozíció alapján számolt kezdési és befejezési idő. A mozgatás végrehajtása előtt a rendszer az alábbiak szerint ellenőrzi a korábban definiált kényszereket.

Elsőként az kerül ellenőrzésre, hogy a célerőforrás megegyezik-e a kiindulási erőforrással. Ha nem, akkor az alternatív művelettervre vonatkozó ellenőrzés fut le, és ha az nem létezik, akkor a mozgatás nem engedélyezett. Minden egyéb esetben a következő műveletből eredő kényszer ellenőrzése következik. Ekkor a GYR-beli műveletek növekvő sorrendjét ellenőrzi a program, majd a GYR linkekből eredő sorrendiségeket, végül pedig az anyagmozgatási idők helyességét. Amennyiben ezek közül bármelyik kényszert a tervezett mozgatás megsérti, a mozgatást nem valósul meg.

Amennyiben a műveletekre vonatkozó kényszerek közül egyiket sem sérti meg a mozgatás, akkor az mindenképpen végbemegy. Ezt követően már „csak” az a kérdés, hogy a mozgatandó művelet az erőforrás fő sorába, vagy a BIN sorba kerül. Ezt határozzák meg a pozíciókra vonatkozó kényszerek. Ezek ellenőrzése az alábbi sorrendben történik.

Elsőként a célerőforrás naptárában definiált állásidőkre, majd a GYR-ek közötti átállási időkre, végül pedig az egy időpontban párhuzamosan végzendő műveletekre vonatkozó kényszereket veszi figyelembe a rendszer. Amennyiben a mozgatás csak az állásidőkre definiált kényszert sérti meg, akkor a célerőforrás fő sorába kerül a művelet úgy, hogy időtartama az állásidő hosszával megnő, minden más esetben viszont a BIN sorba kerül. Ebben az esetben, ha az adott időintervallumban már van másik művelet, akkor a BIN sor magasság megnő, és a mozgatandó műveletet a már ott lévő művelet alá kerül.

4.5.2 Tolólap

Ezen mozgatás során egy művelettől elkezdve, „tolólap” szerűen lehet az utána következő műveleteket is eltolni, ezáltal szabad kapacitást „generálva” a mozgatandó művelet erőforrásának fősorán. Ezen mozgatás során minden művelet olyan közel kerül egymáshoz, amilyen közel csak a kényszerek engedik. Ha a mozgatandó művelet új befejezési időpontja eléri

a) a GYR következő műveletének, vagy
b) az erőforrás fő sorában lévő következő műveletnek, vagy
c) a GYR utolsó művelete esetén a GYR-hez „linkelt” következő GYR megfelelő műveletének a kezdő időpontját,

akkor az „utolért” műveletek is a korábbi művelet(ek) előtt tolódnak az erőforrás(ok) fő- és BIN sorában egyaránt.

Az egymás előtt tolt műveletek esetén a műveletekből eredő kényszerek figyelembe vétele a kézi mozgatástól kissé eltérően működik. Mivel ezzel a funkcióval egyetlen műveletet sem lehet másik erőforrásra áthelyezni, és egy erőforrás fő sorában lévő műveletek sorrendjét sem lehet vele módosítani, ezért a műveletekből fakadó kényszerek közül csak az anyagmozgatási idők betartására vonatkozót kell ellenőrizni. Az összes többi kényszer az előző bekezdésben részletezett működési elv miatt automatikusan betartásra kerül a tolólap típusú mozgatás során.

Az egyes műveletek egymás előtt tolása során, amikor bármelyik művelet eléri a GYR-ének következő műveletét egy tetszőleges erőforrás fő sorában, akkor a rendszer megvizsgálja, hogy a két művelet között mekkora az anyagmozgatási idő, és a követő művelet „tolását” ennek betartása mellett kezdi el. Vagyis a megelőző művelet befejezési időpontja és a követő művelet kezdési időpontja között a definiált átállási idő fennmarad, azaz ez az időtartam is „tolódik” a többi művelettel együtt. Ezáltal biztosítható, hogy a mozgatásokból eredő kényszerek mindig betartásra kerüljenek, így tolólap típusú mozgásvégzés esetén soha nem tiltódik le a mozgatás.

A pozíciókból eredő kényszerek közül a mozgástípus sajátosságai miatt csak a GYR-ek közötti átállási időkre vonatkozó kényszert és az erőforrás naptárban definiált állásidőkre vonatkozó kényszereket ellenőrzi a program az erőforrások fő sorában. Az egy időpontban, párhuzamosan történő műveletvégzést nem kell a mozgatás során ellenőrizni, hiszen amikor egy művelet egy erőforrás fő sorában utoléri a következő műveletet, akkor a követő műveletet el kell kezdeni tolni a megelőző művelet előtt, így nem fordulhat elő olyan, hogy két, egymást követő művelet a fősorban „egymásra csússzon”.

A GYR-ek közötti átállási időkre vonatkozó kényszer az anyagmozgatási időkre vonatkozó kényszerhez hasonlóan kerül figyelembe vételre az erőforrások fő során. Vagyis amikor egy erőforráson egy művelet „utolér” egy olyan műveletet, amely egy másik GYR-hez tartozik, akkor ellenőrzi a rendszer, hogy az adott erőforrásra és a két műveletre van-e definiálva átállási idő, és ha van, annak időtartamát a két művelet között kezdi el „tolni”, mintha csak az is egy művelet lenne.

Az erőforrások naptárában definiált állásidőkre vonatkozó kényszer kerül a mozgatás során utoljára ellenőrzésre az erőforrás fő során. A mozgatás érvényesítése előtt a program megvizsgálja, hogy a mozgatással érintett műveletek közül melyek azok, amelyek olyan

időtartamra esnek az erőforrás fő sorában, amikor állásidő van definiálva. Ezen műveletek hosszát az állásidő hosszával megnöveli.

Előfordulhat olyan eset, hogy egy GYR egyik, tolólap mozgatással érintett művelete, amely egy adott erőforrás fő sorában van „utoléri” a GYR következő műveletét, ami viszont egy másik erőforrás BIN sorában van. Ekkor az utolért műveletet is elkezdi tolni a bin sorban tolólapszerűen a rendszer. Viszont ebben az esetben egyetlen fentebb részletezett kényszert sem vesz figyelembe, hanem a mozgástípus alapvető működése szerint az erőforrás BIN sorában a következő műveletet „utolérve” egyszerűen megnöveli a bin sort, és a megnövelt területen tolja tovább a műveletet. Amennyiben egy ilyen, BIN sorban tolt művelet befejezési időpontja eléri az a), b), c) pontokban definiált műveletek valamelyikét, és az utolért művelet egy erőforrás fő sorában van, akkor a mozgástípusra korábban részletezett kényszereket figyelembe véve mozog tovább. Ha viszont az utolért művelet egy erőforrás BIN sorában van, akkor csak a sorrendiség figyelembe vételével tolódik el.

4.5.3 Linearizálás

Mivel a kézi mozgatás során a programnak nem kell figyelnie a kapacitás túlterhelésekre, ezért előfordulhat, hogy egy erőforráson egy időpillanatban több művelet is be van ütemezve. Ebben az esetben a fő sorba „be nem férő” műveletek az erőforrás BIN sorába kerülnek. Ahhoz, hogy ezek a műveltek bekerüljenek az erőforrás fő sorába, és a grafikus felületen a termelési terv „kisimuljon” le kell futtatni az erőforrásra a linearizálás funkciót.

A funkció megkeresi az időtengelyen előrefelé haladva az összes túlterhelést a kiválasztott erőforráson, és megszünteti azokat. Egy túlterheléshez érve megkeresi a legkorábbi kezdő időpontú műveletet, és veszi annak a várható befejezési időpontját. Ezt követően megkeresi az adott túlterheléshez tartozó következő legkorábbi várható kezdési időpontú műveletet a fő sorban, és a tolólap funkcióval annyi „helyet csinál” az adott erőforrás fő sorában, hogy „beférjen” a tervbe a túlterhelésben érintett következő művelet illetve az előtte és utána esedékes átállási idők is.

A tolólap mozgatás úgy történik, mintha kezdő időpontja annak a műveletnek a kezdő időpontja lenne, aminek helyet kell csinálni, végidőpontja pedig az adott művelet, vagy a mozgatandó és az utána következő művelet közé beékelődő átállási idő várható befejezési időpontja lenne, függően attól, hogy van-e a két művelet paramétereinek megfelelően definiált

átállási idő az adott erőforráson. Miután a szükséges szabad kapacitás létrejött, az előkészített helyre a fő sorban a program beemelni a BIN sorból az oda szánt műveletet.

Ezt követően ismét megkeresi a rendszer az erőforrás legkorábbi túlterhelését, és a fent leírt módszerrel azt is megszüntetni. A folyamat akkor ér véget, ha a kiválasztott erőforráson az összes túlterhelést sikerült megszüntetni.

4.5.4 Tömörítés

Ez a mozgatási módszer a kiválasztott erőforrásokon a műveleteket a lehető legnagyobb mértékben „összesűrűsíti”, azaz az erőforrások kapacitását a lehető legnagyobb mértékben igyekszik kihasználni, figyelembe véve a definiált kényszereket. A funkció használatához először ki kell jelölni a tervezőasztal időhorizontján azt az időintervallumot, amelybe eső műveleteket össze kell rántani, valamint ki kell választani az érintett erőforrásokat.

A funkció az összes olyan műveletet kezeli, ami az így meghatározott kezdő- és végidőpontok közé esik a kiválasztott erőforrásokon. Azokat a műveleteket is figyelembe veszi, amelyek az intervallumon belül kezdődnek(végződnek), de azon kívül végződnek(kezdődnek). Ha van olyan művelet, amely belelóg az intervallum kezdetébe, azt érintetlenül hagyja, és csak az utána következő műveleteket mozgatja. Azokat a műveleteket, amelyek teljes egészében beleesnek a kijelölt intervallumba, és az erőforrás fő sorában találhatók, a korábban ismertetett kényszerek figyelembe vételével a lehető legkorábbi időpontra mozgatja át.

Az összerántás során kijelölt erőforrások legkorábbi kezdő időpontú műveletét a rendszer a kijelölt időintervallum legkorábbi olyan időpontjára mozgatja az erőforráson, ahol a módosított pozíció egyik kényszert sem sérti meg. Azaz a műveleteket az aktuális kezdő időpontjuk szerint sorban dolgozza fel, és megpróbálja a kijelölt intervallum elejére pozícionálni az adott erőforráson. Ha a mozgatandó művelet egy erőforrás fő sorában van, és az intervallum elején van az erőforrás fő során végrehajtandó művelet, akkor annak a tervezett befejezési időpontára időzíti a mozgatott művelet kezdő időpontját úgy, hogy a művelet időtartama alatt ne legyen más művelet az adott erőforrás fő sorában. Amennyiben ilyen művelet van, annak a befejezési dátumára időzíti a mozgatott művelet kezdő dátumát, és így tovább egészen addig, amíg a művelet végrehajtásának tervezett időintervallumába nem esik más művelet az adott erőforrás fő sorában. Az így meghatározott várható kezdő- és végidőpontot ezt követően ellenőrzi a program, hogy nem sérti-e valamely definiált kényszert.

A műveletekre vonatkozó kényszerek közül elsőként műveleti sorrendre, majd az anyagmozgatási időre végül pedig a GYR linkekre vonatkozó kényszer kerül ellenőrzésre. Amennyiben bármelyik kényszer sérül, és a konfliktust olyan művelet okozza, amely egy erőforrás fő sorában van, akkor a mozgatandó műveletet arra a legkorábbi időpontra tolja a rendszer az időtengelyen előrefelé haladva, amikor a konfliktus már nem áll fenn. Amennyiben a konfliktusban érintett másik művelet egy erőforrás BIN sorában van, akkor először a rendszer megpróbálja azt az időtengelyen visszafelé haladva az első olyan pozícióra helyezni, amikor a tömörítéssel érintett művelettel nincsen konfliktusban, és egyéb, műveletekre vonatkozó kényszert sem sért. Amennyiben ilyen helyzet nem hozható létre, akkor a BIN sorban lévő műveletet érintetlenül hagyja, és a mozgatott műveletet pozícionálja az időtengelyen előrefelé haladva a legkorábbi olyan helyre, amikor az említett kényszerek nem sérülnek.

Ezt követően a pozíciókból eredő kényszereket vizsgálata következik. Elsőként a definiált átállási időkre, majd a párhuzamos művelet végrehajtásra, végül pedig az állásidőkre vonatkozó. Amennyiben bármelyik kényszer sérül, és a konfliktust olyan művelet okozza, amely egy erőforrás fő sorában van, akkor a mozgatandó művelet arra a legkorábbi időpontra kerül az időtengelyen előrefelé haladva, amikor a konfliktus már nem áll fenn. Amennyiben a konfliktusban érintett másik művelet egy erőforrás BIN sorában van, akkor viszont a pozícióból eredő kényszerek sajátosságai miatt a konfliktussal nem kell törődni, a mozgatást végre kell hajtani.

Ha a mozgatandó művelet egy erőforrás BIN sorában van, akkor is megpróbálja a rendszer a BIN sor legkorábbi olyan időpontjára pozícionálni, ahol semmilyen, műveletből eredő kényszert nem sért. Fontos azonban kiemelni, hogy ezen mozgatási típus esetén művelet soha nem kerülhet át a BIN sorból a fő sorba vagy fordítva.

A feldolgozás során mindig a még nem mozgatott legkorábbi kezdő dátumú műveletet mozgatja következőként a rendszer, így biztosítható, hogy a tömörítés során a műveletek az időtengelyen elfoglalt pozíciójuk szerinti sorrendben kerüljenek feldolgozásra. Ez garantálja azt, hogy egy mozgatott művelet újonnan meghatározott kezdő és esedékességi időpontjának rögzítése után a művelet esedékességi időpontja és az összerántás kezdő időpontja közötti intervallumon az adott gép kihasználtsága a definiált kényszerek figyelembe vételével maximális.

4.6 Az ütemező algoritmus integrálása a finomprogramozó modulba

A 3.2 fejezetben bemutatott prds típusú ütemező algoritmust a finomprogramozó modulban fel lehetne használni arra, hogy a rendszerben lévő GYR-ek adatai alapján gyorsan, egy gombnyomásra létre lehessen hozni egy kiinduló ütemezést. Ez valószínűleg nem lesz optimális, de jó kiindulási alapot szolgál a végleges tervhez, és az előző fejezetben bemutatott mozgatási funkciókkal könnyen átalakítható, módosítható a tervező pillanatnyi információi, tapasztalata és a vállalat célkitűzései szerint.

Az algoritmus által használ 4 paramétert egyértelműen meg lehet feleltetni a QAD EA rendszer bizonyos paramétereinek. Az esedékesség és az átfutási idő megfelel a GYR esedékességének és átfutási idejének annyi kiegészítéssel, hogy az esedékességi időpont a GYR utolsó műveletének erőforrására az esedékességi dátumon definiált utolsó munkaóra. A két, feladat típustól függő paraméternek pedig a gyártási rendelés által előállított tétel törzsadatai között rögzített két érték feleltethető meg. Ez egyrészt jól tükrözi, hogy milyen típusú az elvégzendő feladat, másrészt pedig helyes paraméterezés esetén a beállított értékek jól fogják súlyozni azt, hogy egy adott termék gyártása mennyire fontos a cég termékportfóliójában.

A beállított paraméterek és a rendszerben lévő kibocsátott állapotú GYR-ek alapján az algoritmus a korábban bemutatott módszer szerint egy sorrendet állít fel a tervezőasztalon kijelölt időintervallumba eső GYR-ek között. Amennyiben a teljes időtengelyre vonatkoztatva szeretnénk az algoritmust futtatni, akkor célszerű egy vezérlő mezőben paraméterezhető módon megadni, hogy az aktuális dátumtól számolva hány napra vonatkozó műveleteket ne módosítsa a rendszer. Ez azt segíti elő, hogy a közeljövőbe eső már kialakított ütemezést az automatikus algoritmus ne boríthassa fel. Ezek alapján a folyamat végén kapott sorrend pedig azt mutatja meg, hogy az adott feladat elvégzése mennyire sürgős a vállalat számára ahhoz, hogy az igényeket késedelem nélkül ki tudja elégíteni.

A kiszámított sorrend alapján el lehet kezdeni a műveletek ütemezését az erőforrásokra. Ezt a GYR-ek között felállított sorrend szerint kell végrehajtani, viszont igazodva az MRP terminológiájához, az ütemezést az esedékességi dátumhoz igazítva végzem el. Ez azt jelenti, hogy a meghatározott sorrend szerint kell venni a GYR-eket, és azok utolsó műveletének befejezési dátumaként a GYR esedékességi dátumán az érintett erőforrás utolsó munkaóráját kell megadni, és meg kell próbálni az adott időpontra ütemezni a célerőforrás fősorában. A pozícionálást úgy kell elvégezni, hogy a kihelyezett művelet egyetlen műveletekből fakadó

kényszert se sértsen meg. Ezek közül az anyagmozgatási időre és a GYR linkekre vonatkozó kényszerek azok, amiket ténylegesen ellenőrizni kell, mivel a többi a funkció működésből eredően nem fordulhat elő. A műveleti sorrend kényszer azért nem számít, mert a műveleteket visszafelé ütemezve, a GYR legutolsó műveletével elkezdve folyamatosan kell kipakolni a tervezőasztalra, így a megfelelő sorrend biztosított. Az alternatív művelettervekkel azért nem kell foglalkozni, mert minden műveletet a művelettervben hozzárendelt erőforrásra próbálunk kihelyezni, gépek közötti áthelyezés nincs.

A GYR linkekkel kapcsolatban azt kell csak ellenőrizni, hogy van-e a kihelyezendő műveletre mutató olyan már kihelyezett GYR, aminél az utolsó művelet befejezési időpontja + anyagmozgatási ideje későbbre esik, mint a kihelyezendő művelet kezdő időpontja. Ebben az esetben már kihelyezett műveletet a tolólap funkcióval időben visszafelé haladva el kell tolni úgy, hogy a két művelet között elegendő hely keletkezzen az anyagmozgatási idő teljesítéséhez.

Az éppen kihelyezés alatt lévő GYR műveletei között pedig az anyagmozgatási idők betartására kell még figyelni. Ez azt jelenti, hogy az éppen kihelyezendő és a már kihelyezett művelet közötti anyagmozgatási időt ki kell hagyni a korábbi művelet befejezési és a követő művelet kezdési időpontja között.

Abban az esetben, hogyha a kihelyezendő művelet valamely pozícióból eredő kényszert sért, akkor nem kell elmozdítani a kiszámított helyzetéből, csak nem az erőforrás fősorába, hanem a BIN-be kell helyezni.

Az így előállított finomprogram olyan megoldást ad, ami a késedelmes feladatok minimalizálását szem előtt tartva határozza meg a GYR-ek pozícionálásának sorrendjét, a műveletek kihelyezése viszont az esedékességi dátumból kiindulva visszafelé történik az időtengelyen, ami az MRP működéséhez hasonló módon a készletszintek minimalizálására törekszik minden időpontban. A kapott ütemezésben előfordulhatnak túlterhelések, ugyanakkor ezeket a kezelők a finomprogramozó modul bemutatott mozgatási funkcióival úgy oldhatják fel, ahogy azt az aktuális körülmények leginkább megkövetelik.

5. Összefoglalás

Dolgozatomban áttekintettem egy napjainkban is aktívan kutatott tématerületet, a termelésütemezést. Megvizsgáltam a probléma egy népszerű, általánosított változatát, a job-shop scheduling feladatot. Annak ellenére, hogy az 1950-es évektől rengeteg kutatás foglalkozott ezzel, mind a mai napig nem sikerül általános érvényű, gyakorlatban is jól használható optimalizáló algoritmust találni a feladat megoldására. A munkám elején bemutatott, eddig elért legfontosabb kutatási eredmények viszont mind új megközelítést, szemléletmódot adtak a problémával foglalkozók számára, így nagyon sok jól használható közelítő eredmény született. Mindez azért is fontos, mert egy nagyon is kézzel fogható, a nagyvállalatok életében napi szinten jelentkező feladatra jöttek és jönnek létre napjainkban is olyan megoldások, melyek a mindennapi munkát könnyítik.

Dolgozatom második részében az elméleti áttekintés után inkább a gyakorlatra helyeztem a hangsúlyt, és bemutattam, hogyan lehet alkalmazni a kutatások eredményeit az ipari gyakorlatban. Ehhez a való életből vett szempontok alapján kidolgoztam egy prds típusú algoritmust. Ezt követően bemutattam a QAD EA integrált vállalatirányítási rendszer termeléstervezéshez használt modulját, és bemutattam annak egy kiegészítését, amivel a finomprogramozást el lehet végezni. A megoldás jól mutatja, hogy egyrészt a gyakorlati feladatok esetén gyakran sokkal bonyolultabb feltételrendszerek mellett kell megoldást találni a problémára, sokkal több szempontot szem előtt kell tartani, mint az elméleti kutatások esetén. Másrészt viszont a bemutatott algoritmus alkalmazása a finomprogramozó modulban rámutat arra is, hogy nagyon nagy szükség van a megalapozott algoritmusokra ahhoz, hogy a mindennapi problémákra hatékony megoldásokat találjunk.

Az ismertetett finomprogramozó modul összekapcsolva a QAD EA rendszerrel és az ütemező algoritmussal, kézzel fogható előnyöket ad a termeléstervezők számára. A látványos, könnyen áttekinthető Gantt-diagramos tervezőfelület, a számos, különböző helyzetekben jól használható mozgatási funkció, illetve az automatikus ütemezés mind-mind olyan megoldások, melyek a felhasználók mindennapi munkáját egyszerűsítik. Ezáltal pedig nem csak gyorsabban fogják tudni elvégezni a napi rutin feladataikat, hanem az azonnal rendelkezésre álló információk alapján a váratlan helyzetekre is hatékonyabban fognak tudni reagálni, ami vállalati szinten jelentős költség megtakarításokat eredményezhet. Napjaink dinamikusan változó piaci környezetében pedig nagy szükség van az ilyen jellegű megoldásokra, mert ezek

is nagyban hozzájárulnak a vállalatok sikereihez. A dolgozatban ismertetett megoldásom bevezetésére éles alkalmazásként várhatóan a következő félévben kerül sor.

Irodalomjegyzék

[1] A.s. Jain, S. Meeran: *Deterministic job-shop scheduling: Past, present and future* European Journal of Operational Research, 113 (1999) 390-434

[2] M. L. Pinedo: *Planning and Scheduling in Manufacturing and Services*, Springer, New York, 2005, p506

[3] Dr. Szikora B.: *Termelésinformatika*, oktatási segédanyag, 5.43. változat, BME Elektronikai Technológia Tanszék, 2013, p105

[4] R. W. Conway, W. L. Maxwell, L. W. Miller: *Theory of Scheduling*, Dover Publications Inc., Mineola, 2003, p294

[5] P. Brucker, B. Jurisch, B. Sievers: *A branch and bound algorithm for the job-shop scheduling problem*, Discrete Applied Mathematics 49 (1994) 107-127

[6] M. R. Garey, D. S. Johnson, R. Sethi: *The complexity of flow shop and job-shop scheduling*, Mathematics of Operations Research 1 (1976) 117-129

[7] J. Adams, E. Balas, D. Zawack: *The shifting bottleneck procedure for job shop scheduling*, Managment Science 34 (1988) 391-401

[8] Rónyai L., Ivanyos G., Szabó R.: *Algoritmusok*, Typotex, Budapest, 2008, p349

[9] Dr. Szikora B., Várnai L.: *Vállalatirányítási rendszerek és Termelésinformatika laboratórium*, oktatási segédanyag, 5.42. változat, BME Elektronikai Technológia Tanszék, 2012, p200

[10] http://www.hindawi.com/journals/jam/2012/651310/fig1/ , 2013.10.25.

[11] QAD Inc., *Database Definitions Technical Reference*, 2011. június

[12] T. C. E. Cheng, Bo Peng, Zhipeng Lü: *A hybrid evolutionary algorithm to solve the job shop scheduling problem*, Annals of Operations Research, February 2013, 1-15