MISKOLCI EGYETEM


GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR

HATVANY JÓZSEF INFORMATIKAI TUDOMÁNYOK DOKTORI ISKOLA

# ÜTEMEZÉSI MODELL ÉS HEURISZTIKUS MÓDSZEREK AZ IGÉNY SZERINTI TÖMEGGYÁRTÁS FINOMPROGRAMOZÁSÁNAK TÁMOGATÁSÁRA

PHD ÉRTEKEZÉS

KÉSZÍTETTE:

KULCSÁR GYULA

OKLEVELES MÉRNÖK-INFORMATIKUS

TÉMAVEZETŐ:

DR. ERDÉLYI FERENC

A MŰSZAKI TUDOMÁNY KANDIDÁTUSA

DOKTORI ISKOLA VEZETŐ:

PROF. DR. TÓTH TIBOR

A MŰSZAKI TUDOMÁNY DOKTORA

MISKOLC, 2007

# ÖSSZEFOGLALÁS

Az értekezésben a rövid távú, műhelyszintű termelésprogramozás területén végzett kutatómunkám legfontosabb eredményeit foglalom össze. Bemutatok egy új, kiterjesztetett rugalmas Flow Shop ütemezési modellt, valamint olyan megoldási módszereket ismertetek, amelyek a termelés változó feltételeihez és igényeihez igazodva, több kritérium együttes figyelembevételével optimumközeli, végrehajtható termelési finomprogramot állítanak elő. A módszer alkalmas nagyméretű feladatok megoldására a gyakorlatban elfogadhatónak számító tervezési (futási) időkorlátok betartásával. Bemutatok egy új, kombinált megoldási koncepciót megvalósító termelésprogramozó szoftvert, amely alkalmas a termelés rövid távú ütemezésére, valamint a termelés újraütemezésével a termelési bizonytalanságokból és zavarokból keletkező hibák csökkentésére.

Az értekezés felépítése a következő: Az 1. fejezetben ismertetem a kutatómunka témáját és a kitűzött tudományos célokat. A 2. fejezetben összefoglalom a témához kapcsolódó fontosabb ismereteket. Vázolom az alap és a rugalmas Flow Shop ütemezési modelleket, azok megoldási módszereit, alkalmazásuk lehetőségeit és korlátait. Áttekintem a korszerű számítógépes gyártásirányítás szerepét és feladatait. A 3. fejezetben összefoglalom az igény szerinti tömeggyártást folytató vállalatok diszkrét gyártó-szerelő folyamatainak a termelésprogramozás szempontjából fontos általánosítható jellemzőit. A szakirodalomban használt szimbolikus jelölésrendszerre támaszkodva definiálok egy új ütemezési modellt. A modell építőelemeinek számítógépi realizálására egy speciális indexelt adatmodellt javaslok. A 4. fejezetben a termelési finomprogramok elkészítésére egy végrehajtás-szemléletű, szimulációra és kombinált heurisztikus módszerekre alapozott többcélú megoldási koncepciót javaslok. A megoldási módszer az ütemezési feladat egyes részfeladataihoz kapcsolódó döntéseket integrált módon, egyszerre hozza meg. Az 5. fejezetben az újraütemezési feladatok megfogalmazására és megoldására egy kibővített ütemezési modellt és továbbfejlesztett módszereket javaslok. Az értekezés 6. fejezetében egy integrált termelésprogramozási koncepciót ismertetek. Erre alapozva bemutatok egy megvalósított szoftverprototípust, amely egy gyártásirányító rendszer szerves részeként működve támogathatja a dinamikus irányító funkciókat, beleértve a termelési folyamatok minősítésének, a korrekciós újraütemezéseknek, a termelési célok, prioritások és paraméterek megváltoztatásának, valamint a bizonytalanságok menedzselésének igényeit is. Az értekezés további fejezetei az új tudományos eredményeket, azok alkalmazhatóságát, a továbbfejlesztési lehetőségeket, valamint a témában készített saját tudományos közleményeket foglalják össze.

# TARTALOMJEGYZÉK

# RÖVIDÍTÉSEK LISTÁJA

AI – Artificial Intelligence – mesterséges intelligencia
ALA – Adaptive Learning Approach – alkalmazkodó tanuláson alapuló megközelítés
BDE – Borland Database Engine – Borland fejlesztésű adatbáziskezelő motor
CAD – Computer Aided Design – számítógéppel segített konstrukciós tervezés
CAD/CAM – számítógéppel segített gyártáselőkészítés
CAE – Computer Aided Engineering – számítógéppel segített mérnöki tervezés
CAL – Computer Aided Logistics – számítógéppel segített logosztika
CAM – Computer Aided Manufacturing – számítógéppel segített gyártás
CAPC – Computer Aided Production Control – számítógéppel segített termelésirányítás
CAPP – Computer Aided Process Planning – számítógéppel segített folyamattervezés
CAQA – Computer Aided Quality Assurance – számítógéppel segített minőségbiztosítás
CAQP – Computer Aided Quality Planning – számítógéppel segített minőségtervezés
CC – Cell Controller – cellavezérlő
CDS – Campbell-Dudek-Smith algorithm – a szerzőiről elnevezett ütemező algoritmus
CNC – Computer Numerical Control – számítógépes számjegyvezérlés
CRM – Customer Relationship Management – ügyfélkapcsolati-menedzsment
DBMS – Database Manegement System– adatbáziskezelő rendszer
DCS – Distributed Control System – elosztott vezérlőrendszer
ERP – Enterprise Resources Planning – integrált vállalatirányítási rendszer
FFL – Flexible Flow Line – rugalmas gyártósor (ütemezési modell)
FFS – Flexible Flow Shop – egyutas párhuzamos gépes (ütemezési modell)
FIFO – First In First Out – érkezési sorrenben (prioritási szabály)
FMSC – Flexible Manufacturing System Conroller – rugalmas gyártórendszer vezérlője
FS – Flow Shop – egyutas kötött műveleti sorrendű (ütemezési modell)
FSMP – Flow Shop with Multipe Processors – egyutas párhuzamos gépes (ütemezési modell)
FSPM – Flow Shop with Paralell Machines – egyutas párhuzamos gépes (ütemezési modell)
GA – Genetic Algorithm – genetikus algoritmus
GUI – Graphical User Interface – grafikus felhasználói felület
HFS – Hybrid Flow Shop – egyutas párhuzamos gépes (ütemezési modell)
IC – Inventory Control – készletgazdálkodás és -irányítás
LPT – Longest Processing Time – leghosszabb műveleti idejű munka előre (prioritási szabály)
MA – Manufacturing Automation – gyártásautomatizálás
MES – Manufacturing Execution System – számítógépes gyártásirányító rendszer
MFS – Multiprocessor Flow Shop – egyutas párhuzamos gépes (ütemezési modell)
MIS – Management Information System – vezetői információs rendszer
MMC – Measuring Machine Controller –mérőgépvezérlő
MSF – Minimum Slack First – legkisebb tartalékú munka előre (prioritási szabály)
MSFS – Multi-Stage Flow Shop – egyutas, párhuzamos gépes (ütemezési modell)
NEH – Nawaz-Enscore-Ham – a szerzőiről elnevezett ütemező algoritmus
NP – Non-Polinomial – nem polinomiális (megoldási idejű feladat)
PLC – Programmable Logic Controller – programozható logikai vezérlő
PPS – Production Planning and Scheduling – Termeléstervezés és -ütemezés
ROC – Robot Control – robotvezérlés
SA – Simulated Annealing – szimulált hűtés

SCM – Supply Chain Management – ellátási lánc menedzsment
SFC – Shop Floor Control – műhelyszintű gyártásirányítás
SPC – Statistical Process Control – statisztikai folyamatellenőrzés
SQL – Structured Query Language – strukturált lekérdezőnyelv
TS – Tabu Search – tabukeresés
VCL – Visual Component Library – vizuális komponensgyűjtemény
WSPT – Weighted Shortest Processing Time – súlyozott legkisebb műveleti idejű munka előre (prioritási szabály)

# JELÖLÉSEK JEGYZÉKE

*BWBA* – alapvető terheléskiegyensúlyozó algoritmus
*DF_Pri[dk]* – a *dk*-adik újraütemezési célfüggvény prioritásértéke
*dk* – az újraütemezési célfüggvények azonosítója *(dk = 1,…, $N_{DK}$)*
*DSCH* – részletes termelési finomprogram
*E_O* – a gyártásifolyamat-típusokat és a megrendeléseket összerendelő kapcsolótömb
*EFFS* – kiterjesztett rugalmas flow shop (ütemezési modell)
*EHIA* – kiterjesztett heurisztikus beszúró algoritmus
*es* – végrehajtási lépések azonosítója *(es= 1,…, $N_{ES}$)*
*F_Pri[k]* – a *k*-adik célfüggvény fontosságát kifejező prioritásérték
*FT* – zárolási időpont
*HEFCA* – prioritási szabályokat kombinált módon alkalmazó heurisztikus algoritmus
*HIA* – heurisztikus beszúró algoritmus
*i* – a munkák azonosítója *(i = 1,…, $N_J$)*
*J[i].FJ* – az *i* munka zárolásának módját definiáló objektum
*J[i].FJ.enabled_MG_M[mg][am]* – az *i* zárolt munka teljesítéséhez az *mg* gépcsoportból használható *am*-edik párhuzamos gép
*J[i].FJ.enabled_R[ar]* – az *i* zárolt munka teljesítéséhez használható *ar*-edik végrehajtási útvonal
*J[i].LFS* – az *i* munka utolsó zárolt végrehajtási lépésének sorszáma
*J[i].OID* – az *i* munka megrendelésének azonosítója
*J[i].RET* – az *i* munka befejezési időpontja
*J[i].RST* – az *i* munka kezdési időpontja
*J_T[i][am]* – az *i* munka *am*-edik végrehajtási lépéséhez rendelt gép
*J_Tp[i][am]* –az *i* munka *am*-edik gyártási feladatának pozíciója a hozzá rendelt gépen
*k* – a célfüggvények azonosítója *(k = 1,…, $N_K$)*
*m* – a gépek azonosítója *(m=1,…, $N_M$)*
*M[m].CAL[ci]* – az *m* géphez tartozó *ci*-edik rendelkezésre állási időintervallum
*M[m].CAL[ci].ET* – az *m* gép *ci*-edik rendelkezésre állási időintervallumának végét jelentő időpont
*M[m].CAL[ci].ST* – az *m* gép *ci*-edik rendelkezésre állási intervallumának kezdetét jelentő időpont
*M[m].ENGAGED* – az *m* gép lefoglaltságának végét jelentő időpont
*M[m].ES* – az *m* gép műveletvégző képességét kifejező végrehajtási lépés típusa
*M[m].LFP* – az *m* gép végrehajtási sorában az utolsó zárolt munka sorszáma
*M[m].NCAL* – az *m* géphez tartozó rendelkezésre állási időintervallumok aktuális száma
*M[m].PR[p]* – az *m* gépen időegység alatt megmunkálható *p* típusú termékek darabszáma
*M[m].ST[p1][p2]* – az *m* gép átállításának időtartama *p1* terméktípus gyártásáról *p2* gyártására
*M_STET[m][ai].ET* – az *m* gép *ai*-edik munkájának befejezési időpontja
*M_STET[m][ai].ProcT* – az *m* gép *ai*-edik munkájának műveleti időtartama

*M_STET[m][ai].SetT* – az *m* gép *ai*-edik munkájához tartozó gépátállítás időtartama
*M_STET[m][ai].ST* – az *m* gép *ai*-edik munkájának kezdési időpontja
*M_T[m][ai]* – az *m* gép munkasorozatának *ai*-edik munkája
*M_Tp[m][ai]* – az *m* gép *ai*-edik gyártási feladatához tartozó munka végrehajtási lépésének sorszáma
*mg* – a gépcsoportok azonosítója *(mg = 1,..., $N_{MG}$)*
*MG_M* – a gépcsoportokba tartozó gépeket leíró kapcsolótömb
$N_{ES}$ – a végrehajtásilépés-típusok darabszáma
$N_{EXE}$ – a gyártásifolyamat-típusok darabszáma
$N_J$ – a munkák darabszáma
$N_K$ – a célfüggvények darabszáma
$N_M$ – a gépek darabszáma
$N_{MG}$ – a gépcsoportok darabszáma
$N_O$ – a belső rendelések darabszáma
$N_P$ – a terméktípusok darabszáma
$N_R$ – a végrehajtásiútvonal-típusok darabszáma
$N_{TS}$ – a technológiailépés-típusok darabszáma
*o* – a belső rendelések azonosítója *(o = 1,..., $N_O$)*
*O[o].CET* – az *o* megrendelés befejezési határideje
*O[o].CST* – az *o* megrendelés legkorábbi kezdési időpontja
*O[o].FJ* – az *o* rendeléshez tartozó legkisebb sorszámú munka azonosítója
*O[o].FO* – az *o* megrendelés zároltságát jelző érték
*O[o].MG_M[mg][am]* – az *o* megrendelés teljesítéséhez az *mg* gépcsoportból használható *am*-edik gép
*O[o].NJ* – az *o* megrendeléshez tartozó munkák darabszáma
*O[o].NP* – az *o* megrendelésben előírt gyártandó termék darabszáma
*O[o].P* – az *o* megrendelésben előírt gyártandó termék típusa
*O[o].R[ar]* – az *o* megrendelés teljesítésére alkalmas *ar*-edik végrehajtási útvonal típusa
*O[o].RET* – az *o* rendelés teljesítésének befejezési időpontja
*O[o].RST* – az *o* rendelés teljesítésének kezdési időpontja
*OBJ_VAL.DF[dk]* – a *dk*-adik újraütemezési célfüggvény értéke
*OBJ_VAL.F[k]* – a *k*-adik célfüggvény értéke
*p* – a gyártható termékek azonosítója *(p = 1,..., $N_P$)*
*P[p].BOM* – a *p* termék darabjegyzék-azonosítója
*P[p].EXE* – a *p* termék gyártási folyamatának azonosítója
*P[p].MG_M [mg][am]* – a *p* termék szempontjából az *mg* gépcsoporthoz tartozó *am*-edik gép
*P[p].Q* – a *p* termék legkisebb gyártható darabszáma (logisztikai egység mérete)
*P[p].R[ar]* – a *p* termék gyártására alkalmas *ar*-edik végrehajtási útvonal
*P[p].S* – a *p* termék gyártásához előírt gépbeállítás-típus azonosítója
*PRI_MG* –prioritásértékeket és gépcsoportokat összekapcsoló tömb
*r* –a végrehajtási útvonalak azonosítója *(r = 1,...,$N_R$)*
*R_MG* – a végrehajtási útvonalakhoz tartozó gépcsoportokat leíró kapcsolótömb
*RT* – az újraütemezés megkezdésének időpontja
*SCH* – termelési ütemterv
*SHOP.DSCH* – a gyártórendszerben végrehajtásra kiadott termelési finomprogram vagy megvalósított termelési finomprogram
*SHOP.J* – a gyártórendszer munkaállománya
*SHOP.M* – a gyártórendszer gépállománya
*SHOP.O* – a gyártórendszer rendelésállománya
*SHOP.OBJ_VAL* – a gyártási folyamat tervezett teljesítménymutatói vagy tényleges teljesítménymutatói
*ts* – technológiai lépések azonosítója *(ts =1,..., $N_{TS}$)*

# 1. BEVEZETÉS

Az alkalmazott informatika egyik nagyfontosságú és gyorsan fejlődő területe a termelési rendszerek informatikai támogatása. Napjainkban a termelési rendszerek minden hierarchiai szintjén rendelkezésre állnak már a tervezést, az irányítást, a megfigyelést, a végrehajtást és a döntéstámogatást segítő számítógépes alkalmazások. Kialakultak olyan nagy, integrált alkalmazási rendszerek (CAD/CAM, ERP, MES), amelyek komponenseit, moduljait a termelő és szolgáltató rendszerekre illesztve hatékony megoldások nyerhetők. A tapasztalatok szerint azonban a termelési folyamatok sokszínűsége, bonyolultsága, valamint a piaci, üzleti környezet változékonysága állandóan újratermeli az alkalmazott vagy alkalmazható modellek és eljárások felülvizsgálatát, fejlesztését és az alkalmazások integrációjának igényét.

Napjainkban a tömeggyártással foglalkozó vállalatok tekintélyes része is kénytelen a vevők igényeinek közvetlen kiszolgálására (pl. tömegcikkek, világítótestek, háztartási gépek stb. gyártása). A versenyképesség növelése érdekében folyamatosan és rugalmasan alkalmazkodniuk kell a piaci körülmények gyors változásaihoz. Ez megköveteli a gyártási hatékonyság és a szállítókészség egyidejű javítását. Az előbbit például az erőforrások minél jobb kihasználásával, alacsony gyártási költségekkel, az utóbbit jól ütemezett termeléssel, a rendelésre és a raktárra való gyártás kombinálásával lehet elérni. A vállalatok sikeressége számos esetben a megrendelők határidő-igényeinek magas szintű kielégítésén múlik. Mindez egyre fontosabbá teszi a hosszú, közép és rövid távú termeléstervezési és -ütemezési feladatok hibátlan megoldását.

A termelés finomprogramozása (folyamatközeli ütemezés) a MES (Manufacturing Execution System) néven ismert gyártásirányítás rövid távú – esetenként valós idejű – tervezési feladata. Ismertek a termelési rendszer korlátozásai, a véges technológiai kapacitások, az erőforrások rendelkezésre állásai, a műveletek sorrendi előírásai stb. Az ütemezésnél a gyártásra kiadott, belső rendeléseken alapuló munkák és műveletek elvégzéséhez gyártási erőforrásokat (gépek, eszközök stb.), valamint indítási és befejezési időpontokat kell tervezni úgy, hogy a korlátozások teljesüljenek, és a termelés teljesítményét mérő mutatók optimálisak legyenek, azaz a menedzsment magasabb szintjén megfogalmazott célok megvalósuljanak.

A diszkrét termelési folyamatokban a termékek elkészítése gyártási sorozatokban, kötegekben történik. A sorozatok nagysága nagyon változó lehet, egyetlen darabtól (pl.: nagy értékű alkatrészek vagy egyedi gépek, berendezések gyártása) több ezer, sőt több millió darab termékig is terjedhet (pl.: tömegcikkek, komponensek, alkatrészek tömeggyártása esetén). A diszkrét gyártó-szerelő rendszerekben a műveletek végrehajtása elkülönült gépeken (esetleg kézi munkahelyeken) történik. A gyártási

folyamat rendszerint több, térben és időben egymás után (sorba) kapcsolt részfolyamatból, műveletből, technológiai lépésből áll. A műveleteknek és ezek sorrendjének alternatívái is lehetnek. A sorba kapcsolt műveletek száma néhánytól (2-5) akár 100-ig is terjedhet. Tömeggyártásnál nagyszámú termék ugyanazt a technológiai utat járja be. Az ilyen gyártási folyamatot soros felépítésű, folyamszerű „Flow Shop" gyártásnak nevezik.

A soros gyártási struktúrához és homogén technológiai tervekhez kapcsolódó modellben (Flow Shop, FS) a rendelésekben megadott számú munkadarab adott számú különböző gépen, a technológiai sorrendnek megfelelően, egymás után kerül megmunkálásra. Ha megengedett az, hogy az egyes gépeken a munkák sorrendje eltérő legyen, akkor előzéses, ellenkező esetben előzésmentes modellről van szó. A termelés során tehát munkadarab sorozatok (kötegek) gyártásáról kell gondoskodni. A rendelési sorozatok független bemeneti adatok a gyártás számára. A gyártási és logisztikai sorozatnagyságok azonban termelésirányítási döntések tárgyai lehetnek. Ez az értekezés az igény szerinti tömeggyártás finomprogramozásának támogatására szolgáló számítógépes modellek és ütemezési módszerek továbbfejlesztése terén szándékozik új eredményeket bemutatni.

## 1.1. Célkitűzés

Napjainkban – a globalizációs kihívásokra válaszolva – számos nagyvállalat egyedi megrendelésekhez, a kereskedelmi és logisztikai központok egyedi igényeihez igazodó tömegtermelést kénytelen folytatni (Customized Mass Production). Az ilyen termelési folyamatok közös jellemzője a terméktulajdonságok, specifikációk, technológiai, valamint erőforrás-alternatívák sokfélesége, a megrendelések változatossága, az igények nehéz előrejelezhetősége, és a vevők kívánságaihoz való magasfokú alkalmazkodás igénye (pl. egyedi kiszerelés, szállítási időpontok) a nagysorozat- és tömeggyártás technológiai körülményei között.

A kutatás alapvető célja az igény szerinti tömeggyártásban fellépő rövid távú, műhelyszintű termelésprogramozási feladatok modelljének továbbfejlesztése a gyártási feladatok és sorozatnagyságok maghatározásának, a gépek allokálásának és a munkák időbeli ütemezésének szempontjából. További célja olyan új megoldási módszerek és algoritmusok kifejlesztése, amelyek a termelés változó feltételeihez és igényeihez igazodva egy vagy több értelemben optimumközeli, végrehajtható termelési finomprogramot állítanak elő a gyakorlatban elfogadhatónak számító tervezési (futási) időkorlátok betartásával.

A rövid időhorizontú tervezési modell és ütemező egy gyártásirányító (MES) alkalmazás szerves része kell legyen, amelynek támogatnia kell a MES dinamikus gyártásirányító funkcióit is, beleértve a termelési folyamatok minősítésének, a korrekciós újraütemezéseknek, a termelési célok, prioritások és paraméterek megváltoztatásának, valamint a bizonytalanságok menedzselésének igényeit is.

Egy sikeres ütemezési koncepció kidolgozásához szükséges a témához kapcsolódó ismert fontosabb ütemezési modellek és azok megoldási módszereinek elemzése, alkalmazhatóságuk vizsgálata, a lehetőségek és a követelmények

összevetése. Ezek ismeretében lehet új, hatékonyabb ütemezési, döntéstámogatási modelleket és algoritmusokat kifejleszteni, implementálni, tesztelni és értékelni.

Az értekezésben összefoglalt kutatás az MTA SZTAKI vezetésével folyó *„Valós idejű, kooperatív vállalatok”* (VITAL) c. projekthez kapcsolódik. A funkcionális követelmények megfogalmazásánál a VITAL kutatási beszámolókban megfogalmazott specifikációkat is figyelembe vettem.

# 2. TUDOMÁNYOS ELŐZMÉNYEK

## 2.1. Flow Shop ütemezési modellek

### 2.1.1. Alap Flow Shop ütemezési modellek

A „Flow Shop” típusú ütemezési feladatok leírása és a megoldás lehetőségeinek vizsgálata már több mint ötven éves, napjainkban is számos termelésirányítási és ütemezési tankönyvben vagy monográfiában megtalálható (pl.: [34], [128], [48], [26]).

A formális alapmodellben ismert a gépek (erőforrások) halmaza: $M = \{m_j\}$, $j = 1,...,m$, $m \in Z^+$; a munkák halmaza: $J = \{J_i\}$, $i = 1,...,n$, $n \in Z^+$, minden munka sorba kapcsolt műveletekből áll: $O = \{o_{i,j}\}$, minden művelet $\tau_{ij}$ időt vesz igénybe, $\tau_{ij} \in \Re$. A műveletek homogén sorrendje: $j = 1 \to 2 \to ... \to m$. A raktárra gyártás (*make to stock*) teljesítményt mérő célfüggvénye legtöbbször a „*makespan*”: $f = min[max(T_{Ci}) - min(T_{Ri})]$, ahol $T_{Ci}$ az $i$. munka befejezési időpontja, és $T_{Ri}$ az $i$. munka indítási időpontja. Rendelésre gyártás esetén a célfüggvény inkább a késések összes idejének minimalizálása: $f = min \sum_{i=1}^{n} max(0, T_{Ci} - T_{Di})$, ahol $T_{Di}$ az $i$. munka határideje.

A klasszikus (egyutas, előzésmentes) feladatnak a teljes átfutási idő minimalizálására csak $m \le 2$ esetben, valamint bizonyos megszorításokkal $m = 3$ esetben van polinomiális futási idejű megoldási algoritmusa. Ez a nevezetes *Johnson-algoritmus* [73]. Ennek főként elméleti jelentősége van, mert a gyakorlati feladatokban általában $m > 2$. Ezekre az esetekre a feladat NP-nehéz, sőt NP-teljes [52]. Gyors processzorokkal a leszámlálásos triviális tervezési algoritmus előzésmentes esetben kb. $n < 10$-re ad elfogadható tervezési időt.

Az évek során a különböző méretű és komplexitású FS feladatok megoldására sokféle egzakt módszer, approximáció, heurisztika, metaheurisztika és más alapú megoldás született.

Számos kutató az alap FS ütemezési feladat permutációs változatának olyan megoldási módszereit vizsgálta, amelyek a legkésőbbi befejezési időpont minimalizálását célozták meg. Ismerve Garey és társai (1976) munkássága [52] következtében a probléma NP-nehéz jellegét, a megoldások többsége heurisztikus megközelítést alkalmazott a munkák optimumhoz közeli sorrendjének meghatározására, annak érdekében, hogy az eredmény elfogadható időn belül előálljon.

Legkorábban Johnson (1954) mutatott be olyan heurisztikus megoldást, amely két gépes, egyutas, előzésmentes esetben adott számú munka végrehajtási sorrendjét úgy határozta meg, hogy az utolsó munka befejezési időpontja szempontjából az ütemterv optimális [73].

Lomincki (1965) szétválasztás és korlátozás alapú megközelítést alkalmazott az optimális munkasorrend megtalálására [93].

Palmer (1965) javasolt egy hatékony algoritmust „slope" mutató számítására alapozva [104]. Majd Gupta (1971) egy alternatív számítási módszert használt a „slope" index meghatározására [55].

Campbell és társai CDS néven ismert heurisztikus algoritmusukat 1970-ben mutatták be [36]. Gupta (1979) lexikografikus heurisztikát fejlesztett ki a feladat megoldására [56].

Nawaz és társai (1983) felépítő heurisztikát dolgoztak ki [99], amely NEH néven vált ismertté, és az újabb algoritmusok minősítésénél ezt a megoldási módszert a mai napig összehasonlítási alapnak tekintik.

Gupta és Darrow (1986) olyan kétgépes modellt készített, amelyben a munkák sorrendjétől függő gépátállítási időket is figyelembe vették [58].

Rajendran és Chaudhuri (1993) továbbfejlesztették a NEH algoritmust, bevezettek egy súlyozott összeg képzésén alapuló számítási módszert a munkák rangsorolására [110].

Taillard (1990) tesztadatokat definiált, és ezekre vonatkozó futási eredményekkel igazolta, hogy a vizsgált felépítő jellegű heurisztikus algoritmusok közül a NEH változatok a leghatékonyabbak [120]. Framina és társai (2003) szintén NEH algoritmuson alapuló megoldási variánsokat mutattak be [51].

Jelentős fejlődést hozott az iteratívan javító heurisztikus megközelítési módok megjelenése, és egyre gyakoribb alkalmazása.

Egyutas, előzésmentes feladatok legkésőbbi befejezési időpontjának minimalizálására szimulált hűtésen alapuló megoldási módszert javasolt Osman és Potts (1989) [103], valamint Ogbu és Smith (1990) [102]. Stefán (2003) megerősített tanulás és szimulált hűtés kombinált alkalmazásával robusztus ütemezési módszert dolgozott ki [118].

Tabukeresés elvén működő ütemező algoritmust fejlesztett ki többek között Nowicki és Smutnicki (1996) [100] valamint Daya és Al-Fawzan (1998) [47]. Grabowski és Wodecki (2002) szintén tabukeresési technikán alapuló hatékony algoritmust fejlesztett ki [54].

Genetikus algoritmuson alapuló megoldási módszereket dolgoztak ki Chen és társai (1995) [38], Reeves és Yamada (1998) [112]. Alcaraz és társai (2006) további két robusztus genetikus algoritmus variációt mutattak be [20], és munkájukban tematikus összefoglalót adtak a permutációs egyutas ütemezési feladatok megoldásairól.

Ying és Liao (2004) „ant-colony” megközelítési módot alkalmazott, ezáltal gráfon értelmezett speciális útkeresési feladat megoldási technikájához hasonló módon kezelte az előzésmentes, egyutas ütemezési feladatot [138].

Agarwal és társai (2006) a [18] dolgozatban ismertettek egy adaptív tanuláson (Adaptive Learning Approach, ALA) alapuló heurisztikus megoldást, szintén permutációs egyutas ütemezési feladatok megoldására. A módszer lényege, hogy felépítő jellegű heurisztikát használt egy kiindulási megoldás előállítására, majd nem-determinisztikus lokális szomszédsági keresést alkalmazott, melyben súlyfaktorokon alapuló adaptív vezérlési stratégiát használt. A módszer előnyös tulajdonságait ismert tesztproblémákra vonatkozó futási eredményekkel támasztották alá a Palmer, CDS, NEH módszerekhez képest.

Blazewicz és társai (2005) az alap egyutas ütemezési feladatnak olyan esetét vizsgálták, amelyben csak két dedikált gép szerepel, viszont minden munkára határidő van előírva, amely minden munkára azonos. Célfüggvényként a közös határidő letelte után végzett műveletek időtartamának súlyozott összege van megjelölve [30]. Kisméretű feladatokon dinamikus programozási megközelítést, leszámlálásra alapozott módszereket, valamint heurisztikus listás ütemezési módszereket hasonlítottak össze. Megállapították, hogy nem lehet egyértelműen kiválasztani a vizsgált módszerek közül a legjobbat, mert mindegyiknek egyaránt van előnye és hátránya is. Megkötés nélküli műveleti sorrendű (Open Shop) környezetben is hasonló vizsgálatokat végeztek ugyanazzal az általuk bevezetett célfüggvénnyel [29].

Az egyutas, előzéses ütemezési feladatok megoldására – hasonlóan az előzésmentes feladatokhoz – számos esetben keresési metaheurisztikákon alapuló módszereket fejlesztettek ki. Taillard (1990) tabukeresésen alapuló megoldást dolgozott ki [120]. Ponnanbalan és társai (2001) genetikus algoritmus változatokat fejlesztettek ki alapvetően a legkésőbbi befejezési időpont csökkentésére törekedve. Mindkét esetben a javasolt módszerek hatékonyságát jól ismert (Palmer, CDS, NEH stb.) megoldási módszerekhez hasonlítva mutatták be.

Az alap kétgépes FS modellnek olyan kiterjesztését, amely figyelembe veszi a gépek korlátozott rendelkezésre állását, elsőként Lee (1997) tette meg. Kimutatta a feladat NP-nehéz jellegét egyetlen intervallummegszakítás esetére is. A legkésőbbi befejezési időpont minimalizálására olyan dinamikus programozási módszert javasolt, amely garantálja az optimum elérését [86].

Cheng és Wang (2000) olyan kétgépes FS modellt vizsgáltak, amelyben csak az első géphez rendeltek előre ismert rendelkezésre állási intervallumokat. A munkák teljes átfutási idejének minimalizálására heurisztikus algoritmust fejlesztettek ki [40]. Blazewicz és társai (2001) szintén erre a célfüggvényre koncentrálva vizsgálták a kétgépes FS modellt úgy, hogy általánosan felírt rendelkezésre állási időintervallomokat is figyelembe vettek. Felépítő jellegű heurisztikát és lokális keresésen alapuló módszereket dolgoztak ki [28]. Az időben korlátozottan elérhető gépeket tartalmazó egygépes, párhuzamos gépes, és FS modellekről, valamint a javasolt megoldási módszerekről Schmidt (2000) részletes áttekintést készített [113]. Yamada (2003) PhD értekezésében szintén megjelentek a különböző metaheurisztikák és azok alkalmazása alap FS ütemezési feladatok megoldására [136].

Megállapítható, hogy mivel mind az előzésmentes, mind az előzéses FS ütemezési feladatok NP-teljesek, ezért megoldásukra többségében heurisztikus

megoldásokat alkalmaznak. Ezeket a heurisztikákat három csoportba szokás sorolni. Az első csoportba tartoznak az egyszerű (single-pass) heurisztikák – amilyen például a Palmer-szabály is [104] – melyek valamilyen egyszerűen számolható index alapján rendezik sorba a munkákat. A második csoportba tartoznak a felépítő jellegű heurisztikák, amelyek már összetettebb számítási eljárással, több szempontot figyelembe véve határozzák meg a munkákhoz tartozó mutatókat. Majd ezek alapján veszik egymás után sorba a soron következő munkát, és hozzák meg a végleges döntést az adott munkára vonatkozóan. Tipikus példa a konstruktív heurisztikákra a CDS [36] és a NEH [99] heurisztika. A harmadik csoportba tartoznak az iteratív javításon alapuló módszerek. Ebbe a kategóriába tartoznak például a genetikus algoritmusok (GA), a szimulált hűtés (SA), a tabukeresés (TS) továbbá ezek kombinált, továbbfejlesztett megoldásai is. Ezek más metaheurisztikákhoz hasonlóan alapjában véve determinisztikus vagy nemdeterminisztikus keresési technikák, amelyekben valamilyen stratégiát követve a keresési tér fokozatosan, lokális tartományokon keresztül kerül feltérképezésre. Determinisztikus keresés során egy vizsgált közbenső állapotból a továbblépés meghatározott szabály alapján történik, pl.: hegymászó vagy gradiens keresésnél a legjobb szomszédos megoldás irányába. Nemdeterminisztikus keresés esetén a kiterjesztéskor véletlenszerűen választott értékek is befolyásolják a döntést, pl.: genetikus algoritmusban a mutációs operátor, szomszédsági keresésnél a módosító-operátor stb. Az egyik legfontosabb tulajdonsága ezeknek a módszereknek az, hogy az adott problémára jellemző speciális ismeretek beépíthetők a keresési folyamatba, gyorsítva ezáltal a megoldási folyamatot.

Az alap FS modellekről további részletes áttekintést, összefoglalót adnak például a [34], [128], [20], [50], [39] munkák. A weben is számos gyűjtemény található az ütemezési feladatok megoldási algoritmusairól (pl. [42], [107], [35]).

### 2.1.2. Párhuzamos gépes Flow Shop ütemezési modellek

A gyártásütemezéssel kapcsolatos kutatások fontos feladata a klasszikus FS feladat bővítése, kiterjesztése olyan irányokban, melyek megfelelnek a diszkrét gyártásirányítás növekvő számítógépes támogatási igényeinek.

Az alapmodell párhuzamos gépekkel történő kibővítéseként ismert a rugalmas, egyutas (Flexible Flow Shop, FFS) modell. Az ilyen modellekben összetett munkahelyek vannak definiálva. Minden egyes munkahelyen adott számú, azonos feladat végrehajtására alkalmas, egymással párhuzamosan működő egyenértékű, részben azonos vagy különböző intenzitás- és más paraméterértékekkel rendelkező gépek találhatók. A párhuzamosan működő gépek száma munkahelyenként eltérő lehet. Az egymást követő munkahelyek között átmeneti tárolók kaphatnak helyet. A munkadarabokat minden munkahelyen – annak egy kiválasztott gépén – kell megmunkálni. Ezáltal a modellekben megjelenik a gépválasztás (hozzárendelés) feladata is, ugyanakkor továbbra is alapvető szerepet játszik a munkák gépenkénti sorrendjének és indítási időpontjának meghatározása.

Az alap FS feladat ilyen irányú kibővítését Arthanari és Ramamurthy (1971) az elsők között tették meg [25]. Olyan általánosított egyutas ütemezési modellt állítottak

fel, amelyben munkahelyet definiáltak. Ennek egy speciális esete a klasszikus FS modell, amelyben minden egyes munkahelyen pontosan egy gép lehet. A hagyományos P modell szintén az FFS egy speciális esete, melyben csak egy munkahely van párhuzamos gépekkel. Azóta nagyon sok kutatató foglalkozott a rugalmas, párhuzamos gépes, egyutas ütemezési feladatokkal. Ennek megfelelően számos eredmény született ebben a témában.

A nagy érdeklődésnek több oka is van. Egyik fontos oka az lehet, hogy ezek az ütemezési feladatok nehezen megoldható feladatok. Ullman (1975) megállapította, hogy a párhuzamosan működtetett gépeket tartalmazó ütemezési problémák NP-teljes feladatok [123]. Később Gupta (1988) majd Hoogeveen és társai (1996) bemutatták, hogy már két munkahely és legalább az egyik munkahelyen két teljesen egyenértékű, párhuzanosan kapcsolt gép esetében is a feladat NP-nehéz feladatosztályba tartozik [57], [67].

Egy másik fontos oka az lehet, hogy a rugalmas egyutas ütemezési feladatok különböző formái sokszor ipari feladatok alapját képezik. Az FFS feladatok megjelenése a gépgyártás ütemezési feladataira vezethető vissza, azonban számos más területen is előfordul (pl.: többprocesszoros számítógépek taszkjainak ütemezése, számítógépes hálózati kommunikáció, projekttervezés, elektronikai cikkek és gépjárművek összeszerelésének ütemezése, textilipari valamint húsipari műveletek ütemezése stb.). Ezáltal a gyakorlatban fellépő igények kielégítése során születnek új eredmények.

A kutatók széles körben, különböző jelzőket használva tárgyalják ezeket a modelleket. Leggyakoribbak a rugalmas (Flexible Flow Shop: FFS), a hibrid (Hybrid Flow Shop: HFS), a párhuzamos gépes (Flow Shop with Paralell Machines: FSPM), az összetett munkahelyes (Multi-Stage Flow Shop: MSFS) és a többgépes (Multiprocessor Flow Shop: MFS, vagy Flow Shop with Multipe Processors: FSMP) elnevezésű modellek. Ezek közé sorolhatók még a rugalmas gyártósor (Flexible Flow Line: FFL) néven ismert modellek is.

A probléma komplexitása és a konkrét alkalmazási területek eltérő volta következtében többféle megoldási megközelítés használatos. Alapvetően három nagyobb irányzat különböztethető meg:

- Optimális megoldást adó módszerek (optimum approach).
- Heurisztikus megoldási módszerek (heuristic approach).
- Keresési technikákra alapozott módszerek (AI search approach).

#### 2.1.2.1. Egzakt megoldási módszerek

A kezdeti kutatások csak két munkahelyes FFS környezet vizsgálatára terjedtek ki, és főként a legkésőbbi befejezési időpontot alkalmazták az ütemtervek minősítésére. Leggyakrabban szétválasztás és korlátozás alapú egzakt algoritmusokat fejlesztettek ki, ahogyan azt például Arthanari és Ramamurthy (1971) [25] munkája is mutatja. Egyre jobb alsó korlátokat határoztak meg, azonban nagy feladatok esetében az egzakt módszerek futási ideje túlságosan nagy, ezért csak kis méretű feladatokon alkalmazhatók hatékonyan.

Brah és Hunsucker (1991) szétválasztás és korlátozás elvén alapuló megközelítést használt az FFS feladatok olyan osztályára, amelyben különböző képességű párhuzamos gépek állnak rendelkezésre a definiált munkahelyeken és a munkák legkésőbbi befejezési időpontjának minimalizálása a cél [32]. Hasonlóan Rajendran és Chaudhuri (1992) is a szétválasztás és korlátozás elvét alkalmazták a párhuzamos gépes esetre, de csak előzésmentes ütemtervek előállításával vizsgálták a legkésőbbi befejezési határidő csökkentésének lehetőségeit [109].

Gépek korlátozott rendelkezésre állását vizsgálták HFS modellben Allaoui és Artiba (2006) [24]. Modelljükben két munkahelyet definiáltak úgy, hogy az elsőn csak egyetlen gép, míg a másodikon több párhuzamos gép szerepel. Szétválasztás és korlátozás alapú megoldási módszert javasoltak a legkésőbbi befejezési időpont minimalizálására.

Kis és Pesch (2005) részletes áttekintést adnak a rugalmas egyutas ütemezési feladatok megszakítás nélküli változatairól, és a legkésőbbi befejezési időponttal valamint átfutási idővel kapcsolatos célfüggvények minimalizálására kifejlesztett egzakt megoldási módszerekről [79].

#### 2.1.2.2. Heurisztikus megoldási módszerek

Az egzakt megoldási technikák alkalmazása NP-teljes feladatok esetében csak szűk mérettartományban alkalmazható a nagy futási idő (költség) miatt. Fokozatosan előtérbe kerültek a heurisztikus megközelítések, amelyek gyors megoldást adnak ugyan, azonban az optimum elérését nem tudják garantálni. Ugyanakkor a heurisztikus megoldások lehetővé teszik, hogy a feladatok általánosabb formában, nagyobb méretekben is vizsgálhatók legyenek.

Wittrock (1985) az FFS feladatok megoldására periodikus heurisztikát fejlesztett ki [134]. Egy olyan módszert javasolt a feladat megoldására, amelyben a hangsúly a teljes átfutási időnek és a folyamatban lévő munkák átlagos számának csökkentésére került. Majd (1988) háromfázisú, nemperiodikus algoritmust javasolt [135]. Az első fázisban meghatározásra kerültek a munka-gép összerendelések minden munkahelyen, a második fázisban a munkák gépenkénti végrehajtási sorrendjét rögzítették, végül a harmadik fázisban hozták meg a konkrét indítási időpontokra vonatkozó döntéseket.

Kochar és Morris (1987) az FFL feladatnak kétgépes esetét vizsgálták. Modelljükben figyelembe vették az átállítási időket is, és kétfázisú heurisztikus megoldást javasoltak [81]. Az első fázisban a munkáknak az indítási sorrendjét határozták meg lokális keresési technikával, majd a második fázisban az egyes munkákhoz tartozó gépátállítási időknek, továbbá a gépek előtt elhelyezett átmeneti tárolók méretének együttes figyelembevételével szabályalapú géphozzárendelést alkalmaztak.

Gupta és társai (1991, 1997) olyan kétmunkahelyes FFS modellt vizsgáltak, amelyben az első munkahelyen csak egyetlen gép található. A cél a legkésőbbi befejezési időpont minimalizálása volt [61], [59]. Vizsgálatukat kiterjesztették olyan esetekre is, ahol mindkét munkahelyen több, teljesen azonos gép dolgozhat párhuzamosan, és a gépátállítási időtartamok önállóan jelennek meg elkülönítve a műveleti időktől [62]. Főként az egyszerű prioritásos diszpécserszabályokra (pl.:

leghosszabb műveleti idejű munka előre, LPT) valamint a munkák heurisztikus beszúrására alapozott algoritmusokra építették megoldómódszereiket. Az előzőekhez hasonló módon kettőnél több munkahelyes modellt vizsgáltak. A munkákhoz önálló (nem egységes) határidőket rendeltek, és az azokkal kapcsolatos célfüggvényeket külün-külön vizsgálták [60].

Az átfutási időkre vonatkozó kutatásokhoz viszonyítva a munkák késésével kapcsolatos célfüggvényeket jóval kevesebben próbálták minimalizálni. Chang és Liao (1994) átállítási idők figyelembe vétele nélkül a késések súlyozott összegének minimalizálására dolgoztak ki heurisztikus közelítő megoldási módszert. Ennek alapját egy gráfon értelmezett, minimális költségű átfolyási feladat megoldási módszere jelentette [37].

Liu és Chang (2000) hasonló megközelítésben – azonban már sorrendfüggő átállási időket is beépítve a modellbe – vizsgálták az FFS feladatokat [92]. Yang és társai (2000) dekompozíciós heurisztikát és lokális keresési technikát kombináló módszert mutattak be az FFS feladatok megoldására, szintén a súlyozott késések összegét tekintve minősítési kritériumnak [137]. Heurisztikájuk lényege az, hogy egyetlen munkahelyes, párhuzamos gépes részmodellekre vezetik vissza az FFS modellt. A WSPT (súlyozott legkisebb műveleti idejű előre) szabály és a MSF (legkisebb tartalékú előre) szabály kombinált alkalmazásával a késések várható költségére alapozták a megoldási módszert. A szimulált hűtés elvéhez hasonló lokális keresési technikát használtak az eredmények javítására.

Botta-Genoulaz (2000) az FFS feladatok megoldása során munkákra vonatkozó sorrendi korlátozásokat, időablakokat vett figyelembe, továbbá szétválasztotta a gépátállítási és megmunkálási időket. Az ütemezés egyetlen céljának a legnagyobb késés minimalizálását tekintette [31].

Hong és társai (1999, 2000, 2001) fuzzy koncepción alapuló módszereket alkalmaztak az FFS feladatok megoldására. Az ütemezendő munkákat fuzzy halmazokkal írták le, speciális tagsági függvényeket definiáltak a gépek allokálására (fuzzy LPT) és a munkák indítási sorrendjének meghatározására (fuzzy Palmer) [64], [65], [66].

Brah és Loo (1999) a megoldandó FFS feladat jellemzőinek (pl.: munkák száma, munkahelyek száma, munkahelyeken lévő gépek száma stb.) és az alkalmazott megoldó heurisztikáknak az eredményre gyakorolt hatását vizsgálták [33]. Tanulmányukban bemutatták, hogy a feladat jellemzői nagymértékben befolyásolják az elérhető eredményt. Példaként kiemelhető, hogy a legkésőbbi befejezési időpont és az átlagos átfutási idő lineárisan nő a munkák vagy a munkahelyek számának növelésével. Az átfutási idők jelentősen csökkennek az adott munkahelyeken elhelyezett párhuzamos gépek számának növelésével.

#### 2.1.2.3. AI keresési technikákra alapozott megoldási módszerek

A speciális feladatváltozatokra kifejlesztett heurisztikus módszerek alkalmazhatósága más célfüggvényekre erősen korlátozott, részletesebb modellekre pedig még komoly módosításokkal sem igazán alkalmazhatók. A kutatások fő vonulata olyan irányba terelődött, amelyben az adaptálható módszerek kerültek a középpontba.

Az utóbbi két évtizedben előszeretettel alkalmaztak általános AI keresési technikákat – más típusú optimalizálási feladatokhoz hasonlóan – az ütemezési feladatok megoldására. Az FFS feladatok megoldására főként a genetikus algoritmusokat, a tabulistás kereséseket, a szimulált hűtés elvére alapozott algoritmusokat és a mesterséges neurálishálókat alkalmazó módszereket alkalmazták sikeresen.

Nowicki és Smutnicki (1998) tabukeresési megközelítést alkalmaztak az FFS feladatok legkésőbbi befejezési időpontjának minimalizálására [101]. Módszerük alapját a kritikus útvonal fogalmára és a munkákból szervezett blokkokra vonatkoztatott speciális szomszédsági reláció jelentette.

Lee és társai (1997) egy genetikus algoritmus változatot javasoltak az FFS feladatok esetében a legkésőbbi befejezési időpont minimalizálására [87]. Modelljükben fontos szerepet kapott a sorozatnagyság, és ennek megfelelően a munkasorozatokat egyesítő/szétválasztó rekombinációs operátor is. A kifejlesztett algoritmusuk hatékonyságát más AI keresési technikákkal összehasonlítva, futási eredményekkel támasztották alá.

Wang és Li (2002) szintén genetikus algoritmust használtak az FFS feladatok megoldása során a munkák végrehajtási sorrendjének meghatározására [131]. A legkésőbbi befejezési időpont minimalizálására törekedtek, egyszerű modellt javasoltak a megoldások reprezentálására, azok értékelésére és kiválasztására, valamint könnyen kezelhető operátorokat a keresztezés és a mutáció megvalósítására. Módszerük hatékonyságát Wittrock (1988) [135] heurisztikájával hasonlították össze.

Leon és Ramamoorthy (1997) rugalmas gyártósorok esetében, reguláris célfüggvények figyelembevételével vizsgálták a keresési technikákat [89]. Az FFL modellt az alap FS modell és az egymunkahelyes, párhuzamos gépes P modell kombinációjaként értelmezték. Megközelítésük igazi újdonságának az tekinthető, hogy megengedettnek tekintették bizonyos munkahelyek kihagyását a gyártási folyamat során. Egy gyors heurisztikus megoldási módszert javasoltak megvalósítható ütemtervek előállítására, valamint lokális keresési technikára alapozva három egyszerű algoritmusváltozatot fejlesztettek ki a legkésőbbi befejezési időpont és az átlagos késések csökkentésére. A keresés során az úgynevezett problématéren alapuló (problem-space-based) szomszédsági elvet követték melyet Storer és társai (1992) javasoltak [119]. A Leon és Ramamoorthy (1997) által eredetileg javasolt módszer véletlenszerűen választott szomszédos megoldások vizsgálatán alapult. Ezt a módszert Kurz és Askin (2001) továbbfejlesztették, elért eredményeiket a [83] munkájukban ismertették. Tovább bővítették FFL modelljüket (2003, 2004), munkasorrendtől függő gépátállítási időket vettek figyelembe, és azoknak megfelelően további megoldási módszereket javasoltak [84], [85].

Zdansky és Pozivil (2002) az FSMP feladat egy olyan modelljét dolgozták ki, amelyben a befejezési idő minimalizálására törekedtek, de figyelembe vették a gyártási

sorozatnagyságokat is. Egy olyan megoldási módszert fejlesztettek ki, amely a genetikus algoritmus elvének és a tabukeresés elvének egy sajátos kombinációjaként jelent meg [139].

Wang és társai (2003) egy hibrid neurálishálózat-alapú módszert fejlesztett ki az FFS feladatok megoldására [130]. Neurális háló állítja elő az egyes megoldási változatokat, majd ezek értékelését követően a megszerzett tapasztalatokat a háló tanítására használják fel. A megközelítésnek hatékonyságán és eredményességén túl további fontos jellemzője a rugalmasság és a viszonylag egyszerű implementálhatóság. Feladatspecifikus tudás beépítésével további – főként a módszer futási idejére vonatkozó – javítás érhető el.

Jungwattanaki és társai (2005) a rugalmas, egyutas ütemezési feladatok megoldására kétszintű heurisztikus megoldási módszert mutattak be [74]. Modelljükben különböző sebességű párhuzamos gépek szerepeltek, továbbá munkákhoz rendelt befejezési határidőket is figyelembe vettek. A munkákhoz tartozó műveletek a modellben nem szakíthatók meg, átlapolások nincsenek megengedve. Az ütemezés célja kettős: egyrészt a munkák súlyozott befejezési idejének, másrészt a késő munkák számának minimalizálása. Az első fázisban felépítő jellegű heurisztikus módszereket alkalmaznak, amelyeket az alap FS feladat jól ismert heurisztikus algoritmusaira építenek (Palmer, CDS, Gupta, Dannenbring, és NEH heurisztika). A felsorolt algoritmusokat az első munkahelyre alkalmazzák, majd heurisztikus prioritási szabályokat használnak a többi gépre (pl: érkezési sorrend, First In First Out, FIFO). A második fázisban iteratív módon javítják a megoldásokat genetikus algoritmus és szimulált hűtés elveket felhasználva. A szerzők modelljük továbbfejlesztésével munkák sorrendjétől függő átállási időket is figyelembe vettek [75], elért eredményeiket a futási eredményekkel együtt mutatták be [76].

### 2.1.3. FFS modellek és megoldási módszereik alkalmazhatósága

Az ismert modellváltozatokról és a megoldási módszerekről különböző szempontok szerint rendszerezett, részletes összefoglalót készített Linn és Zhang (1999) [91], Wang (2005) [132], továbbá Kis és Pesch (2005) [79], valamint Quadt és Kuhn (2007) [108].

Az FFS feladatok megoldási módszereinek szempontjából Quadt és Kuhn alapvetően egzakt (optimal) és heurisztikus (heuristic) osztályokat különböztettek meg. A heurisztikus módszereket további két csoportba sorolták aszerint, hogy a részekre bontás (decomposition) vagy az egységben kezelés (holistic) elve érvényesül a megoldómódszerekben. A szétbontásra épülő módszereket további munka- (job), munkahely- (stage) és részproblémaorientált (batching, loading, sequencing) csoportokra bontották annak megfelelően, hogy a módszer mire helyezi a hangsúlyt.

Megállapítható, hogy a kutatások nagy része teljesen egyenértékű párhuzamos gépekre vonatkozik, ilyenek például [60], [21], [90] [133]. A gyakorlatban azonban gyakran különböző képességű és hatékonyságú gépek dolgoznak párhuzamosan ugyanabban a szerepkörben eltérő költségekkel.

A témakörhöz tartozó ismert megoldások többsége optimalizálási kritériumként egyetlen célfüggvényt használ. Ez az esetek többségében (a raktárra történő gyártás alapesetét feltételezve) a megrendelt munkacsoport legkésőbbi befejezési idejének minimalizálását jelenti. Kevesebb megoldási módszer született (a határidőre történő gyártásnak megfelelően) a késésekkel kapcsolatos célfüggvények valamelyikének minimalizálására [92], [137]. A speciális feladatváltozatokra kifejlesztett módszerek alkalmazhatósága más célfüggvényekre erősen korlátozott, részletesebb modellekre pedig még komoly módosításokkal sem igazán alkalmazhatók.

Az egycélú ütemezéshez viszonyítva a több célfüggvény értékének kompromisszumos optimumát biztosító (multi-objective, multi-criteria) megoldások előállítása terén kevesebb kutatási eredmény született. Jellemzően olyan módszerek léteznek, amelyek csak egy elsődleges és egy másodlagos, vagy egy elsődleges és több alacsonyabb prioritású cél elérésére koncentrálnak [75]. Ugyanakkor, az ilyen megoldási módszerek – az ütemezési hatékonyság és a számítási sebesség növelése érdekében alkalmazott heurisztikák miatt – megváltozott célok kezelésére nem vagy csak nehezen alkalmazhatók. Léteznek különböző általános elvek és módszerek többcélú kombinatorikus optimálási feladatok megoldására (pl.: [114], [94], [53]), azonban dinamikus rendszerekben, gyakran változó fontosságú célok esetén a bonyolult paraméterezés miatt ezek alkalmazása nehézkes.

A kutatások jelentős része az aktuálisan vizsgált modelltulajdonság kiemelésére helyezi a hangsúlyt, az egyszerűbb kezelhetőség érdekében háttérbe kerülnek az egyéb jellemzők. Az FFS modellekben rendszerint folyamatosan elérhető gépek szerepelnek (pl.: [75], [139], [87], [20]). Kivételnek számító esetekben jelennek csak meg a gépek rendelkezésre állására vonatkozó időbeli korlátozások, és ilyenkor is csak egyéb tényezők egyszerűsítésével, elhagyásával kerülnek bemutatásra. Allaoui és Artiba (2006) például a gépek korlátozott rendelkezésre állását úgy vizsgálták HFS környezetben, hogy modelljükben két munkahelyet definiáltak. Az első munkahelyen csak egyetlen gép, míg a másodikon több párhuzamos gép szerepelt. Szétválasztás és korlátozás alapú megoldási módszert használtak a legkésőbbi befejezési időpont minimalizálására.

A munkák sorrendjétől függő vagy attól független gépbeállítási/átállítási időket figyelembe vevő FFS modelleket és módszereket Allahverdi és társai (1999) [22], (2007) [23] rendszerezték. Azok a megoldási módszerek, amelyek a sorozatnagyságra (a rendelések bontására és/vagy egyesítésére) vonatkozó kérdésekkel is foglalkoznak, rendszerint csak a teljes rendeléscsoport átfutási idejét, vagy a befejezési időpont csökkentését célozzák meg. Jó példák ezekre a Lee és társai (1997) [87], valamint Zdansky és Pozivil (2002) [139] által bemutatott módszerek. A sorozatnagyságok kezelését is tartalmazó ütemezési feladatokról és azok megoldási módszereiről részletes elemzést készített Zhu és Wilhelm (2006) [140].

Az egyik fontos kutatási terület, mely az utóbbi időben hangsúlyt kapott, az előidejű (prediktív) ütemezés és a valós idejű folyamatmenedzsment (Process Management) összehangolása. Ennek az integrációnak több aspektusa is van, például a „bizonytalanságkezelés”, a „kockázatmenedzsment”, a „prioritásmenedzsment”, a „viselkedés alapú termelésirányítás” stb. [122], [98]. A problémakör a gyártásirányítás fontos beavatkozási tevékenységeként kezeli a termelési folyamatok részbeni vagy

teljes újraütemezését. A témával kapcsolatos eredményekről, a különböző újraütemezési stratégiákról a [126], [105], és [127] publikációk is beszámolnak.

A kutatók egyetértenek abban, hogy az újraütemezést az eredeti ütemezési feladat jelentős módosításaként kell tekinteni, ahol új korlátozások és új célok jelennek meg. A probléma megoldásterét általában ezek a feltételek szűkítik, de járulékos nehézséget okoznak a speciális gyártásirányítási követelmények (pl. anyagellátási, minőségbiztosítási, logisztikai stabilitási igények) kielégítésének lehetőségei. Fontos új követelményként fogalmazható meg, hogy az alkalmazott modellnek lehetővé kell tenni a gyártásirányító személy (üzemvezető, művezető, üzemmérnök) olyan interaktív ütemezési javaslatainak és döntéseinek beillesztését, amely az eredeti és a járulékos korlátozások és célok kielégítését egyaránt biztosítja. Az újraütemezés speciális igényeivel kapcsolatos összefoglalások találhatók pl.: a [111], [27] irodalomban.

Magyarországon a diszkrét gyártási folyamatok modellezése és a gyártási folyamatok ütemezése több évtizede kutatás és fejlesztés tárgya. A műszaki egyetemek (BME, ME) gépgyártástechnológiai és alkalmazott informatikai tanszékei mellett eredményes kutatás-fejlesztés folyik az MTA SZTAKI-ban, és korábban a GTI-ben is. Az ütemezéselmélet témakörében eredményeket publikáltak további egyetemek kutatócsoportjainak, alkalmazott matematika, operációkutatás, informatika tanszékeinek kutatói is. Példaként említhető meg néhány jellemző eredmény az elmúlt évekből. MTA SZTAKI: [44], [43], [78], [77], [82], [97], [125]; BME: [117], [115], [116]; ME: [49], [68], [122]; ELTE: [129], [124]; SZTE: [19], [71], [72], [71]; PTE: [45], [46]; VE: [63], [95].

Összefoglalva megállapítható, hogy a tématerület jelentős elméleti háttere és a publikációk nagy száma ellenére a jelenleg ismert többgépes termelésütemezési modellek, és azok megoldási módszerei még nem veszik egyszerre figyelembe az igény szerinti tömeggyártás jellegzetességeit: több művelet együttes végrehajtására képes gépeket, technológiai útvonalalternatívákat, gépek változó rendelkezésre állási időintervallumait, gépenként eltérő termelési intenzitásokat és selejtarányokat, sorrendfüggő átállási időket, valamint az esetenként gyorsan változó termelési célokat, prioritásokat. Szükség van tehát ezeknek a modelleknek a kiterjesztésére, továbbfejlesztésére és hatékony megoldási módszerek kifejlesztésére, amelyek aztán új, hatékony funkcionális komponensei lehetnek a számítógépes MES alkalmazásoknak.

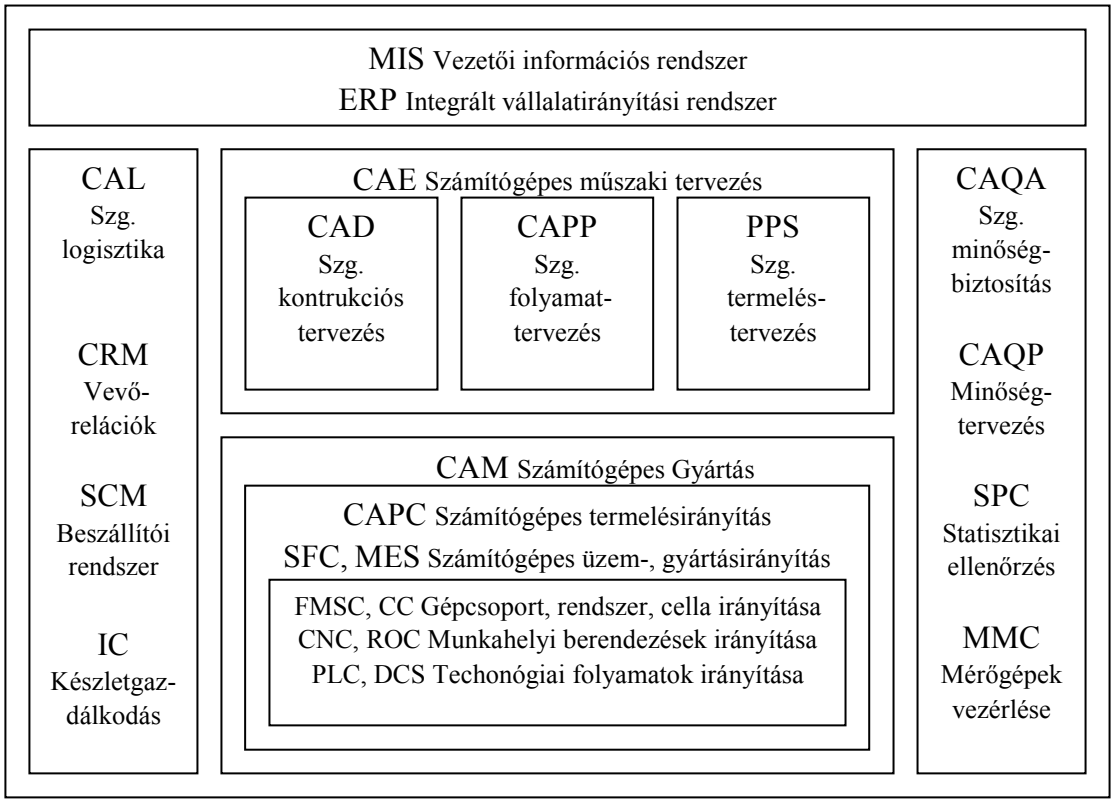
## 2.2. Diszkrét termelési folyamatok irányítása

Ismeretes, hogy a korszerű vállalatirányítási tevékenység kiterjed a termelésfejlesztés, a választékbővítés, a vállalati költséggazdálkodás, a piackutatás és feltárás területeire és más, a vállalat működésével összefüggő egyéb területekre is, amelyek a megrendelések beérkezésétől a késztermék kiszállításáig tartó, szűkebb értelemben vett termelési folyamatokon kívül esnek. Egy vállalat egészének működését – vagyis a vállalaton belül végbemenő tevékenységi folyamatokat – a tervezés, az előkészítés, a végrehajtás és az ellenőrzés szakaszokra lehet bonthani [121].

Az ipari technológiák egy részében a termékegyedek és a folyamatelemek térben és időben egymástól jól elkülöníthetők (pl.: alkatrészgyártás és szerelés esetén a műveletek, operációk). Az ilyen jellengű technológiai folyamatokra alapozott termelést szokás diszkrét termelési folyamatnak nevezni.

Általában a diszkrét termelési folyamatok irányítása több, hierarchikusan egymásra épülő szinten megy végbe. A magasabb szint hosszabb időszakot fog át és csak a fontosabb szempontokkal foglalkozik. Az alacsonyabb szinten a felsőbb szintről kapott irányítási döntések alapján hozzák meg az egyre részletesebb, de egyre rövidebb időperiódusra vonatkozó irányítási döntéseket. Ha az alacsonyabb szint megoldhatatlan feladatot kap (pl. azért, mert a körülmények közben megváltoztak), akkor visszajelez a közvetlenül felette lévő szintre, ahol módosítást kell végrehajtani.

A termelési rendszerek folyamatainak számítógépes tervezése és irányítása a gyakorlati esetek nagy többségében hierarchikus struktúrák segítségével valósul meg. A termeléstervezés és termelésirányítás területén is általában elkülönül a hosszú távú stratégiai tervezés és menedzsment, a közép távú aggregált tervezés és a rövid távú – esetenként valós idejű – termelésütemezés és -végrehajtás feladatköre. Ez utóbbit a nemzetközi irodalom SFC (Shop Floor Control) néven tárgyalja. A magyar műszaki terminológia a legyakrabban az „üzemirányítás” és a „gyártásirányítás” kifejezéseket használja (1. ábra).




**1. ábra: Számítógépes termelésinformatikai rendszerek és funkciók**

### 2.2.1. A gyártásirányítás feladatai és céljai

A gyártás az ipari termelés anyagainak, alkatrészeinek, szerelvényeinek és késztermékeinek előállítására irányuló műszaki-gazdasági tevékenység [48].

A diszkrét gyártási folyamatok alapeleme a művelet. A műveletek meghatározott gépeken, berendezéseken, munkahelyeken meghatározott sorrendben, különböző szerszámok, készülékek használatával, meghatározott technológiai adatokkal végezhetők el. A műveletek között rendszerint bonyolult kapcsolatrendszer áll fenn (pl. megelőzési reláció stb.). A műveletek végrehajtását a munkahelyek (pl. kézi munkahely vagy automatizált technológiai berendezés stb.) valósítják meg. A munkahelyek rendszerint gyártócsoportokba, gyártócellákba, gyártórendszerekbe, gyártóműhelyekbe szervezve funkcionálnak. Az ilyen gyártórendszerek irányításában önállóan is megfogalmazható részfeladatok megvalósítását szoros integrációban kell megoldani.

A gyártásirányítás legfontosabb feladatai: (1) a közép távú termelési tervek lebontása részletes, rövid távú feladatokra, a feladatok ütemezése, (2) a feladatok végrehajtásához szükséges anyagi, humán és információs feltételek biztosítása, (3) a feladatok eszközökhöz rendelése, kiosztása és elindítása, (4) a folyamatok valós idejű irányítása és megfigyelése, (5) a végrehajtás minőségének biztosítása, (6) az eredmények értékelése, szükség esetén korrekciója, (7) a bizonytalanságok és a kivételes események kezelése.

A gyártóműhelyek irányítása erősen interaktív jellegű. A döntéseket a gyártórendszer vagy a műhely vezetői hozzák. A döntések támogatására számítógépes rendszereket alkalmaznak, amelyek nagy mennyiségű információt dolgozhatnak fel. A kapcsolódó információk alapvetően három forrásból származnak: (1) a műszaki és üzleti tervezőrendszerektől a vállalati helyi hálózaton keresztül; (2) a műhely- vagy gyártórendszer-menedzsmenttől interaktív felületen keresztül; (3) a műhely megmunkáló rendszereit képviselő gyártócelláktól, mérő- és anyagkezelő celláktól, önálló munkahelyektől, üzemi terminálóktól, üzemi hálózaton keresztül.

A számítógépes gyártásirányítási feladatok támogatására – a vállalatirányítási feladatok támogatására szolgáló ERP (Enterprise Resources Planning) rendszerekhez hasonló – több funkcionális komponensből álló MES (Manufacturing Execution System) rendszerek alakultak ki. A MES célja a gyártási folyamatok végrehajtásának optimális irányítása, az aktuális üzleti és műszaki célok lehető legjobb megvalósítása.

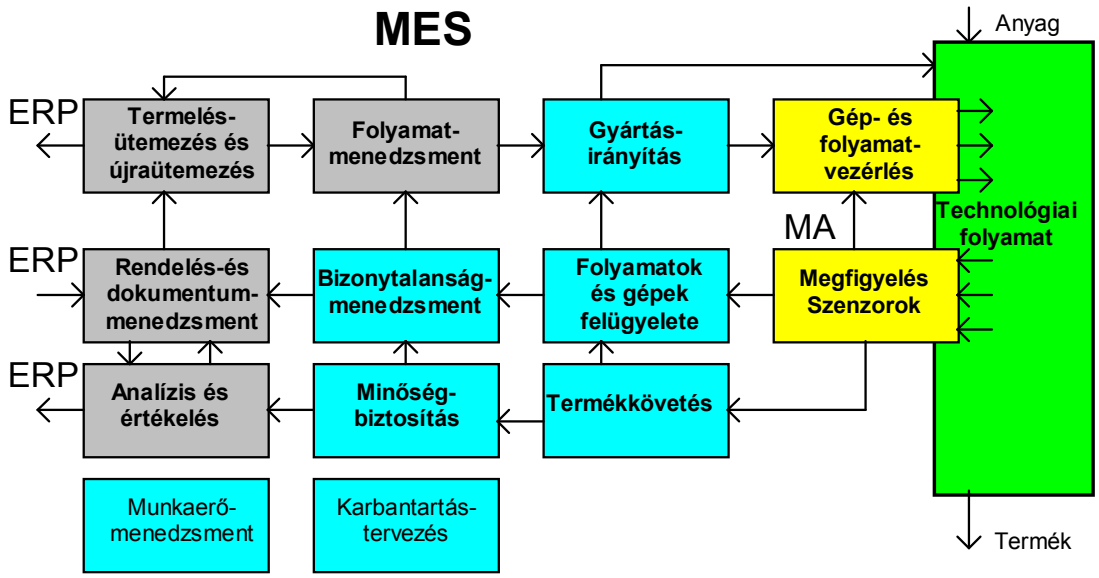
### 2.2.2. MES – Számítógépes gyártásirányító rendszerek

A MES hardver és szoftver komponensek együttese, amely lehetővé teszi termelési tevékenységek (aktivitások) menedzselését és optimalizálását a belső rendelések kibocsátásától a késztermék előállításáig. Egy MES – ellenőrzött és percrekész adatok felhasználásával – gondoskodik a tervezett tevékenységek időbeli ütemezéséről, elindításáról, irányításáról, a végrehajtás felügyeletéről, visszajelzések és jelentések készítéséről és továbbításáról. A MES a termelési aktivitásokról feladatspecifikus információkat szolgáltat a vállalat döntési folyamatainak támogatásához.

A nemzetközi gyakorlatban az alkalmazott MES rendszerek száma folyamatosan növekszik. A MES-gyártók nemzetközi szövetsége a MESA a legfontosabb komponenseket a következő listával definiálja [96]:

1. Rendelésmenedzsment (Order management).
2. Rövid távú ütemezés (Detailed scheduling).
3. Gyártásirányítás (Dispatching production units).
4. Termelésidokumentum-menedzsment (Document control).
5. Termelési adatok gyűjtése (Data collection and acquisition).
6. Erőforrások allokációja és megfigyelése (Resource allocation and status tracking).
7. Munkaerőmenedzsment (Labor management).
8. Minőségbiztosítás (Quality management).
9. Folyamatmenedzsment (Process management).
10. Karbantartás-menedzsment (Maintenance management).
11. Termelésifolyamat-követés (Product tracking and genealogy).
12. Teljesítményanalízis (Performance analysis).

Mint az a 2. ábrán látható, a MES-funkciók egy többszörösen visszacsatolt bonyolult szabályozási rendszert alkotnak. A funkcióblokkok „testre szabása” ezért a kereskedelmi MES-rendszerek alkalmazásának egyik elengedhetetlen feltétele. A MES-funkcióblokkok fejlesztése ma is széleskörű elmélet- és gyakorlatorientált kutatás tárgya. Ez az értekezés ehhez a fejlesztő munkához kíván néhány új eredménnyel hozzájárulni. Az ábrán eltérő színnel jelöltem meg az értekezés eredményei által érintett MES funkciókat. Ezek közül a legfontosabb a feladatok részletes ütemezése és allokációja. A kiterjesztett modell alapján fejlesztett funkciók azonban egyidejűleg támogatják a termelési folyamatok teljesítményértékelését, a bizonytalanságkezelést, a feladatok újraütemezését is.




2. **ábra: MES funkciók**

# 3. KITERJESZTETT RUGALMAS FLOW SHOP ÜTEMEZÉSI MODELL

## 3.1. Műhelyszintű termelésprogramozási feladatok általános jellemzői az igény szerinti tömeggyártásban

Egy igény szerinti tömeggyártást folytató vállalat több (esetenként számos) különböző terméket állít elő. A gyártásirányítás szintjén jellemzően adott egy belső rendelésállomány, amelyet a külső rendelések és az előrejelzések figyelembevételével a vállalat termeléstervezési szintjén definiálnak. Minden egyes rendelés meghatározott típusú, adott darabszámú egyforma termék adott határidőre történő legyártását írja elő.

A tömeggyártás műhelyszintű irányításában előredefiniált logisztikai egységek szerepelnek. A logisztikai egység (pl. paletta, konténer) megadott darabszámú, adott típusú termék előállítására irányuló munkadarabok együttesét jelenti. A logisztikai egység munkadarabjai együtt kerülnek mozgatásra a gyártó-szerelő rendszerben, és azonos műveleteket kell rajtuk végrehajtani. Egy rendelés ilyen logisztikai egységek halmazának tekinthető, ahol a logisztikai egységek számát a rendelt mennyiség és a logisztikai egység termékfüggő mérete együttesen határozzák meg.

A belső rendelésekben szereplő termékek előállításához – a termék típusától függően – adott számú műveletet kell kötött sorrendben végrehajtani. A műveletek rendszerint további részoperációk sorozatából tevődhetnek össze, azonban a műveletek jellemzően nem szakíthatók meg, ezért ezek tekinthetők az ütemezés során a legkisebb allokációs egységeknek. Egy művelet adott időben történő elvégzéséhez megfelelő gép (munkahely), továbbá meghatározott típusú és mennyiségű (esetleg többféle) anyag és/vagy komponens együttes rendelkezésre állása szükséges. Ugyanakkor a termelés rugalmas jellegéből következik, hogy egy adott típusú termék alternatív anyagok, komponensek, gépek és útvonalak használatával is előállítható.

A gyártó-szerelő rendszerben rendszerint automatizált gépek, gépsorok, (esetleg kézi munkaszalagok) működnek, amelyek a terméktől függő gyártási sebességekkel, a munkák sorrendjétől függő átállási időkkel, rendelkezésre állási időintervallumokkal és egy adott termékcsoportra érvényes egymást követő műveletek sorozatával jellemezhetők. A gyártás során egy adott gép egyszerre csak egy feladaton dolgozhat, és egy adott feladaton egyszerre csak egy gép dolgozhat. A gépek között átmeneti tárolóhelyek vannak kialakítva, amelyek mérete – ütemezési szempontból – rendszerint nem korlátozott.

A termelési finomprogram elkészítésekor figyelembe kell venni, hogy az ütemezési időhorizonton a műhely bizonyos gépei már korábbi, még be nem fejezett, nem módosítható feladatokkal terheltek, így az utolsó végrehajtásra kiadott és érvényben lévő finomprogram következményei hatással vannak az új finomprogramra.

## 3.2. Az EFFS ütemezési feladatosztály formális leírása

Ismeretes, hogy a diszkrét termelési folyamatok műhelyszintű prediktív ütemezése mind elméleti, mind gyakorlati szempontból az erősen modellfüggő és komplex feladatok körébe tartozik. A napjaink ipari gyakorlatában egyre fontosabbá váló, rugalmas és igény szerinti tömeggyártás irányítása szükségessé teszi az ismert ütemezési modellek további jelentős kiterjesztését.

Az ütemezési feladatok formális leírásának eszközeként a szakirodalomban az $\alpha|\beta|\gamma$ formalizmus használata a legismertebb [34]. A szimbólumok jelentése a következő:

- $\alpha$: erőforráskörnyezetet leíró paraméterlista,
- $\beta$: korlátozásokat, végrehajtási jellemzőket leíró paraméterlista,
- $\gamma$: célfüggvényeket kitűző paraméterlista.

A listákban szereplő paraméterekre számos javaslat van az irodalomban.

Az igény szerinti tömeggyártás általános jellemzői alapján – a gyártási sorozatnagyságokra, a gépek allokálására és a munkák időbeli ütemezésére koncentrálva, – az anyagok, komponensek és humánerőforrások rendelkezésre állásának egyszerűsítésével – a szakirodalomban használt szimbolikus jelölésrendszer felhasználásával definiálható egy új ütemezési feladatosztály:

$$Fx, M_g, Q_{i,m}, Set_{i,j,m}, Cal_m \left| R_i, D_i, Exe_i, A_i \right| [f_1, f_2, ..., f_K], \qquad (3.1)$$

melyet *kiterjesztett rugalmas flow shop* (Extended Flexible Flow Shop, EFFS) ütemezési feladatosztálynak neveztem el.

Az egyes szimbólumok jelentése a következő:

$\alpha = [\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5]$

$\alpha_1$: A műveleti sorrend előírása. *F*: a műveletek (technológiai lépések) sorrendje kötött. *x*: a sorba kapcsolódó műveletek maximális száma.

$\alpha_2$: Az erőforrások jellege. $M_g$: többcélú, egyetlen vagy akár több művelet együttes végrehajtására is alkalmas gépcsoportok, valamint gépcsoportokon belül párhuzamos gépek is megengedettek.

$\alpha_3$: A párhuzamos (alternatív) gépek jellege. $Q_{i,m}$: egymástól független párhuzamos gépek, melyek mindegyike munkáktól függő, eltérő termelési sebességekkel (intenzitásértékekkel) működhet.

$\alpha_4$: A gépek átállítására vonatkozó előírás. $Set_{i,j,m}$: munkák sorrendjétől és géptől függő átállítási időadatok figyelembe vétele kötelező.

$\alpha_5$: Erőforrásokra vonatkozó speciális előírások. $Cal_m$: Gépekre előírt rendelkezésre állási időintervallumok. A gépek csak ezeken belül működhetnek.

$\beta = [\beta_1, \beta_2, \beta_3, \beta_4]$.

$\beta_1$: A munkák indíthatóságára vonatkozó előírás. $R_i$: munkánként előírt legkorábbi indítási időpont.

$\beta_2$: A munkák befejezésére vonatkozó előírás. $D_i$: munkánként előírt legkésőbbi befejezési időpont.

$\beta_3$: A munkák végrehajtására vonatkozó előírás. $Exe_i$: munkánként előírt végrehajtandó műveletek összefüggő sorozata.

$\beta_4$: A munkák gépekhez rendelésére vonatkozó korlátozás. $A_i$: munkánként előírt műveletvégzésre alkalmas gépek halmaza.

$\gamma = [\gamma_1, \gamma_2, ..., \gamma_K]$: A kijelölt célfüggvények, amelyeknek a minimalizálása a cél.

Az ütemtervek minőségét célfüggvények megfogalmazásával és azok kiértékelésével lehet számszerűsíteni. Valós környezetben nagyon sokféle termelési cél megfogalmazható. Ismeretes, hogy a termelési modellek széles körében az ismert termelési háromszög állapotváltozói: a lekötött készletszint, az erőforrás-kihasználás és a szállítókészség szükséges és elegendő mértékben meghatározzák a termelési folyamat minősítését. Ezek a makroparaméterek tovább bonthatók, adott követelményeket részletesebben kifejező mutatók fogalmazhatók meg. A célfüggvények között ezek egyaránt szerepelhetnek, azonban hatással vannak egymásra, kölcsönkapcsolatban állnak, így tetszőlegesen nem javíthatók. A termeléssel kapcsolatban megfogalmazott célok fontossága időben változhat.

A feladatosztály leírása jól mutatja az ütemezési feladatok sokszínűségét és erős modellfüggését. Az EFFS feladatosztály magában foglalja az alap P és FS feladatok, valamint a rugalmas FFS feladatok megszakítás nélküli, előzéses változatait.

## 3.3. Az ütemezési modell alapvető sajátosságai

Az ütemezési modellben minden logisztikai egység önálló, absztrakt munkát jelent. A feladat megoldása során ilyen munkák kerülnek ütemezésre. Nincs előre rögzített gyártási sorozatnagyság. A gyártási sorozatnagyságokat az adott gépen azonos beállítással készülő munkák darabszámai adják meg. A rendelési sorozatnagyság ennél nagyobb vagy kisebb is lehet. Az ütemezés során az egyes gépeken kialakuló gyártási sorozatnagyság egy munkaköteg, amely két egymást követő beállítási időintervallum közötti munkák sorozata. A köteg nagysága tehát dinamikusan változhat, megengedve a rendelési sorozat bontását és egyesítését.

A modellben az ütemezés előtt minden munkához hozzárendelhető egy korlátozás, legkorábbi kezdési időpont, amely a munka indíthatóságának időbeli korlátozását jelenti, figyelembe véve a gyártási-szerelési komponensek rendelkezésre állását.

Bevezetem a végrehajtási lépés fogalmát, amely alatt a technológiai lépések olyan sorozatát kell érteni, amely gépváltás nélkül elvégezhető. Egy multifunkcionális gépsorhoz tartozó végrehajtási lépés több technológiai lépést tartalmaz, míg egy hagyományos géphez tartozó csak egyet. Az ütemezés alapegységének nem a technológiai lépést, hanem a végrehajtási lépést tekintem.

A végrehajtási útvonal fogalmát a végrehajtási lépések olyan sorozataként definiálom, amelyben az előforduló technológiai lépések csak egyszeresen vannak lefedve, azaz a végrehajtási lépések közös része üres halmaz.

A végrehajtási lépések típusai alapján gépcsoportokat (géptípus osztályokat) definiálok. Egy gépcsoport azonos végrehajtási lépéssel jellemezhető, de más jellemzőkben eltérő képességű gépek adott elemszámú halmaza. Mivel a gépek nem minden terméktípust képesek kezelni, így a gépcsoportok terméktípusonként (vagy terméktípus-csoportonként) eltérőek. Egy végrehajtási útvonal végeredményben a megfelelő végrehajtási lépéseket realizáló gépcsoportok sorozatát jelenti.

Egy termelési finomprogramot funkcionális szempontból három részre bontok a hatékonyabb kezelés érdekében. Ezek a következők:

(1) Feladatok listája, amely munkánkénti bontásban rögzíti a választott végrehajtási útvonalat, valamint az érintett gépcsoportoknak megfelelően rendre kiválasztott gépet.
(2) Feladatok sorrendjének listája, amely gépenkénti bontásban definiálja a munkák végrehajtási sorrendjét.
(3) Tervezett időadatok listája, amely gépenkénti bontásban adja meg a munkákhoz tartozó tervezett kezdési időpontot, gépbeállítási időtartamot, műveleti időt és befejezési időpontot.

(Az (1) – (3) fogalmak részletes kifejtésére a 3.4.6. alfejezetben kerül sor.)

A bemutatott végrehajtás-szemléletű ütemezési modell jól illeszkedik az EFFS feladatosztály jellegzetességeihez, ugyanakkor egyszerűbb feladatok esetében is előnyösen használható.

## 3.4. Adatmodell, számítógépi reprezentáció

Az ütemezési modell építőelemeinek számítógépi realizálására kifejlesztettem egy speciális adatmodellt. Az adatmodell egyszerűsített vázának formális leírására a számítógépi programozásban általánosan alkalmazott tömbök és objektumok jelölésrendszerét használom.

Az adatmodellben indexelt tömböket használok az adatelérés egyszerűsítése és gyorsítása érdekében. A gyártási környezetben az ütemezés során használt entitások (pl.: megrendelések, munkák, gépek stb.) indexekkel vannak helyettesítve. Ezek nem-negatív egész számok. Az indexek hivatkoznak az egyes objektumok vektorokban elfoglalt pozíciójára. Ezáltal lehetővé válik az, hogy egy adott tömb adott elemére történő hivatkozásban a tömb indexe ugyanannak vagy egy másik tömbnek egy megfelelő értéke legyen. Az összetartozó értékeket a bázisvektorokban tárolt alapadatokra történő hivatkozások rendszere tartja össze. Ennek előnye főként az, hogy

egy tetszőleges elemből kiindulva – az indexelési szabályok betartásával – minden egyes vele kapcsolatban álló elem közvetlenül elérhető.

A tömbök elemeire történő hivatkozást szögletes zárójelekkel jelölöm, a következő formalizmus szerint:

TÖMB_NÉV[INDEX]
TÖMB_NÉV[SOR_INDEX][OSZLOP_INDEX]
TÖMB_NÉV[DIM_1_INDEX][DIM_2_INDEX][…][DIM_N_INDEX]

Olyan esetekben, amikor a tömb elemei mezőkből felépülő összetett adatszerkezetek (rekordok, általánosan objektumok), az egyes mezőkre a pont operátor és a mezőnév együttes használatával hivatkozom:

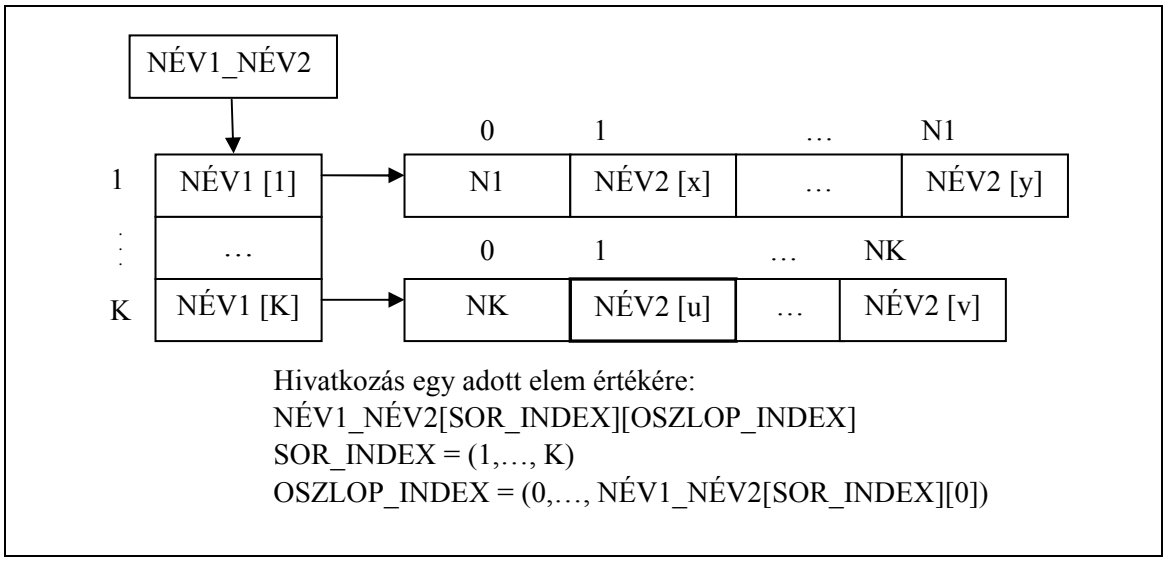
TÖMB_NÉV[INDEX].MEZŐ_NÉV
TÖMB_NÉV[SOR_INDEX][OSZLOP_INDEX].MEZŐ_NÉV
TÖMB_NÉV[DIM_1_INDEX][…][DIM_N_INDEX].MEZŐ_NÉV

Az objektumok mezői között további tömbök is szerepelhetnek.

TÖMB_NÉV[INDEX_1].MEZŐ_NÉV[INDEX_2] [INDEX_3]

(Az azonosítókban használt _ aláhúzásnak csak olvasást könnyítő szerepe van.)

Egy soronként eltérő elemszámú kétdimenziós tömb általános szerkezetét a 3. ábra szemlélteti. Ilyen kapcsolószerkezeteket használok két különböző (*NÉV1 és NÉV2*) tömb elemeinek összerendelésére, ahol a *NÉV1* az alaptömböt, *NÉV2* pedig a hozzákapcsolt tömböt jelenti.




**3. ábra: Kapcsolótömb általános szerkezete**

A bemutatott formalizmus használatával az ütemezési modell objektumai, azok attribútumai és a közöttük fennálló kapcsolatrendszer tömören, ugyanakkor áttekinthetően írható le.

### 3.4.1. Termékek

A gyártható terméktípusok $p$ $(p = 1,\ldots, N_P)$. A termékek alapattribútumai a következők:

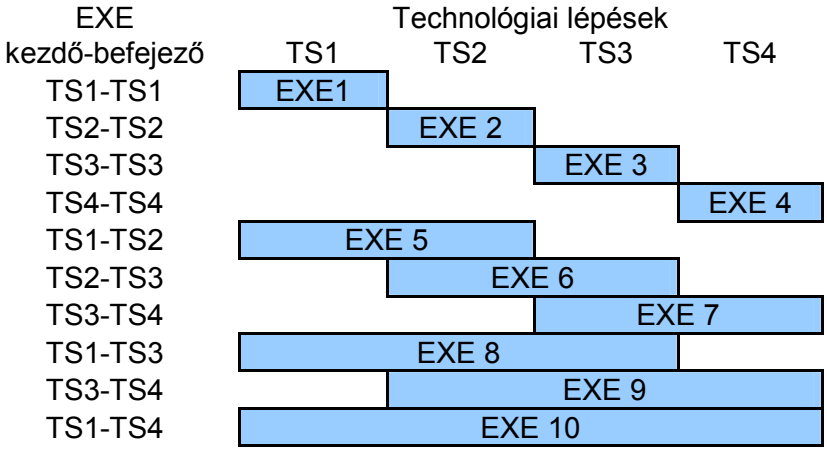
*P[p].BOM* a termék darabjegyzék-azonosítója. A lehetséges komponensbeépülési kombinációk adottak egy anyagjegyzékben, amelyben *ÉS* logikai operátorok továbbá *VAGY* logikai operátorok kapcsolják össze a komponenseket.

*P[p].EXE* a termék gyártási folyamatának azonosítója. A termékek legyártásához kötött sorrendű technológiai lépéseket $ts$ $(ts = 1,\ldots, N_{TS})$ kell végrehajtani. A különböző termékek előállításához a teljes technológiai lépéssorozatot vagy annak egy összefüggő részsorozatát kell végrehajtani (lásd pl.: 4. ábra $N_{TS} = 4$ ). Ilyen értelemben a gyártási folyamat típusainak darabszáma a következő összefüggéssel határozható meg:

$$N_{EXE} = \frac{N_{TS}(1 + N_{TS})}{2}. \qquad (3.2)$$

Bizonyítás:

Tetszőleges $N_{TS}$ számú technológiai lépés esetében a lehetséges gyártásifolyamat-típusok előállíthatók az 13.1 számú mellékletben leírt algoritmussal.




**4. ábra: Gyártásifolyamat-azonosítók négy technológiai lépés esetén ($N_{TS} = 4$)**

A gyártási folyamatok azonosítói ehhez hasonló módon definiálhatók olyan esetekben is, amikor megengedett bizonyos technológiai lépések kihagyása a termék előállítása során. Ilyen esetekben a „hiányos” sorozat felbontható kisebb összefüggő szakaszok sorozatára.

*P[p].S* a termék gyártásakor előírt gépbeállítás-típus azonosítója. A különböző termékek előállításához a gépeket megfelelő módon elő kell készíteni a munkák végrehajtása érdekében. Ez az azonosító az egyes beállítási operációkat és az előírt paraméterértékeket azonosítja.

*P[p].Q* legkisebb gyártható darabszám (logisztikai egység mérete). A gyártórendszerben logisztikai egységek mozoghatnak. Ezek mérete előre megadott, amely termékfüggő darabszámot jelent.

### 3.4.2. Rendelések és munkák

A gyártásra kiadott rendelésállomány belső rendeléseket *o* ($o = 1,\ldots, N_O$) tartalmaz. A rendelések alapattribútumai a következők:

*O[o].P* a gyártandó terméktípus azonosítója.

*O[o].NP* a megrendelt darabszám.

*O[o].CET* az előírt legkésőbbi befejezési időpont (határidő).

Minden egyes *o* rendelés munkákra bontható. Egy munka egy logisztikai egységnek felel meg.

A rendelésekhez további származtatott attribútumok rendelhetők:

*O[o].CST* legkorábbi kezdési időpont. A gyártandó terméktípus komponenseinek (*P[O[o].P].BOM*) együttes rendelkezésre állása alapján meghatározható az a legkorábbi időpont, amikor a rendeléshez tartozó munkák első technológiai lépésének végrehajtása elkezdhető.

*O[o].NJ* a rendeléshez tartozó munkák darabszáma, melyet a rendelésben szereplő igényelt darabszám és az adott termék legkisebb gyártható darabszámának hányadosa (szükség esetén felfelé kerekítve) ad meg:

$$\mathrm{O[o].NJ} = \left\lceil \frac{\mathrm{O[o].NP}}{\mathrm{P[\,O[o].P\,].Q}} \right\rceil . \qquad (3.3)$$

*O[o].FJ* a rendeléshez tartozó legkisebb sorszámú munka azonosítója. Az összes rendelés együttes munkáinak azonosítói decimális egészekből álló sorozatot alkotnak. Az első rendelés első munkájának azonosítója 1, utolsó munkájának azonosítója *O[1].NJ*. Egy tetszőleges *o* rendelés legkisebb sorszámú munkája: *O[o].FJ*, utolsó munkája: *O[o].FJ* + *O[o].NJ* – 1.

*O[o].RST* a rendelés teljesítésének kezdési időpontja, amely a rendeléshez tartozó munkák indítási időpontjai közül a legkorábbival egyezik meg.

*O[o].RET* a rendelésteljesítés befejezési időpontja, amely a rendeléshez tartozó munkák befejezési időpontjai közül a legkésőbbivel egyezik meg.

Minden logisztikai egységet önálló munkának tekintek. Összességében ezek az *i* ($i = 1,\ldots, N_J$) munkák kerülnek ütemezésre. A munkák alap attribútumai a következők:

*J[i].OID* a megrendelés azonosítója.

*J[i].RST* a munka kezdési időpontja, amely a munka első gyártási feladatának indítási időpontját jelenti.

*J[i].RET* a munka befejezési időpontja, amely a munka utolsó gyártási feladatának befejezési időpontját jelenti.

A munkákhoz kapcsolódó egyéb adatok hivatkozással elérhetők. Például:

*O[ J[i].OID ].CET* az *i* munkához tartozó határidő,

*P[ O[ J[i].OID ].P ].Q* az *i* munkához tartozó munkadarabok darabszáma.

### 3.4.3. Gépek és gépcsoportok

A műhelyben lévő gépek, gépsorok és kézi munkahelyek egy általános géptípus példányainak tekinthetők. A valódi gépsorok műveletvégző képességüktől függően egyben, vagy pedig több kisebb önálló részre bontva felelnek meg a modellben a gép objektumoknak. Minden gép *m* *(m=1,..., $N_M$)* a következő tulajdonságokkal jellemezhető:

*M[m].PR[p]* *(p= 1,..., $N_P$)* termékfüggő gyártási sebességek: időegység alatt megmunkálható *p* típusú termékek száma az *m* gépen.

*M[m].ST[p1][p2]* *(p1 = 1,..., $N_P$; p2 = 1,..., $N_P$)* terméksorrendtől függő átállási idők: az *m* gép átállításának ideje *p1* terméktípus gyártásáról *p2* terméktípus gyártására.

*M[m].NCAL* az *m* géphez tartozó rendelkezésre állási időintervallumok aktuális száma. Adott időhorizonton az *m* gép munkarendjét leíró kalendárium bejegyzéseinek száma.

*M[m].CAL[ci]* *(ci = 1,...,M[m].NCAL)* az *m* géphez tartozó rendelkezésre állási időintervallumok sorozata, ahol:

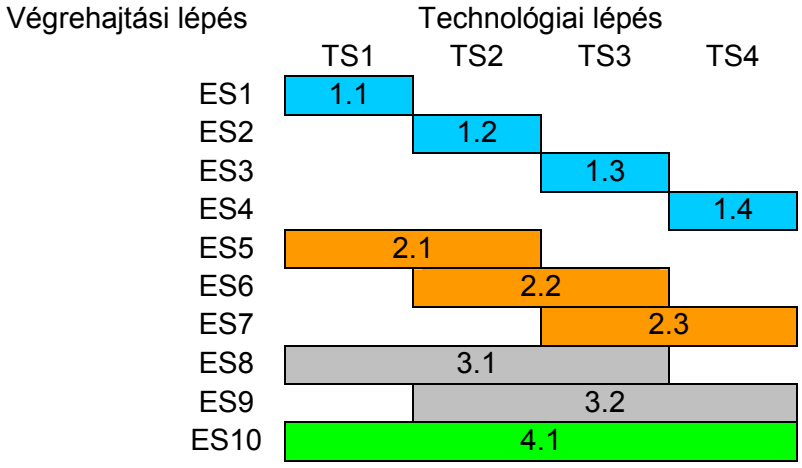
*M[m].CAL[ci].ST* a *ci*-edik intervallum kezdetét jelentő időpont,

*M[m].CAL[ci].ET* a *ci*-edik intervallum végét jelentő időpont.

A gépek csak a megadott időintervallumaikon belül dolgozhatnak.

*M[m].ES* az *m* gép műveletvégző képességét kifejező végrehajtásilépés-típus. Az összes technológiai lépést tartalmazó sorozat olyan összefüggő részsorozatát, amely gépváltás nélkül végrehajtható – mint önálló egységet – végrehajtási lépésnek *(es = 1,..., $N_{ES}$)* nevezem. Belátható, hogy a különböző végrehajtásilépés-típusok darabszáma és a technológiailépés-típusok darabszáma között – az *EXE* gyártásifolyamat-típusokhoz hasonlóan – az alábbi összefüggés áll fenn:

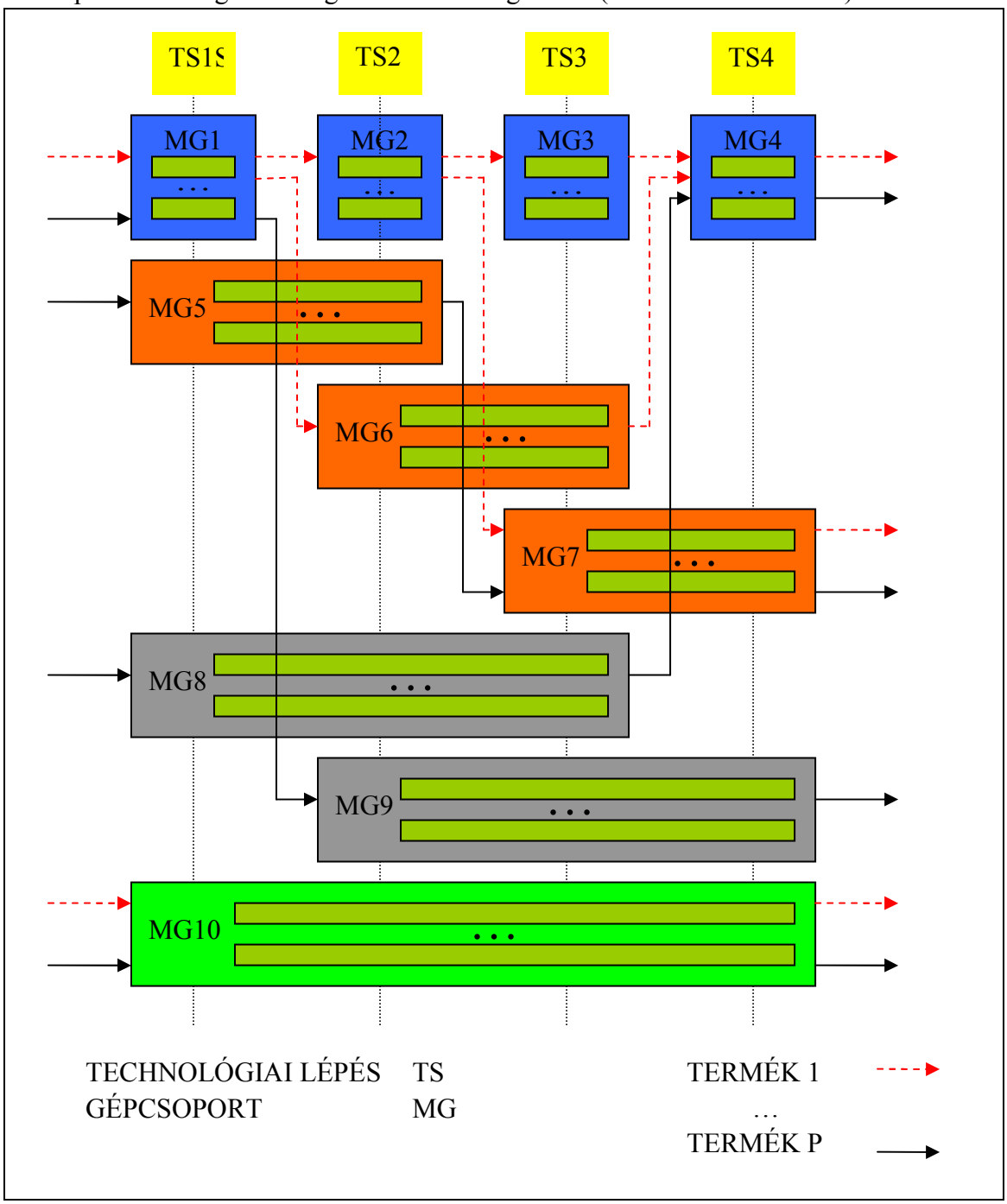
$$N_{ES} = \frac{N_{TS}(1+N_{TS})}{2}. \qquad (3.4)$$




**5. ábra: Végrehajtási lépések típusai négy technológiai lépés esetén ($N_{TS}$ = 4)**

A modellben szereplő gépek között vannak olyanok, amelyek csak egy, és vannak olyanok is, amelyek több egymást követő technológiai lépést képesek megszakítás

nélkül elvégezni (5. ábra $N_{TS} = 4$ ). A végrehajtásilépés-típusok egyértelműen azonosíthatók a kezdő és a befejező technológiai lépések együttesével, vagy pedig a kezdő technológiai lépés és a lépések számának együttesével. Előállításuk az *EXE* típusoknál megismert algoritmussal elvégezhető (13.1 számú melléklet).

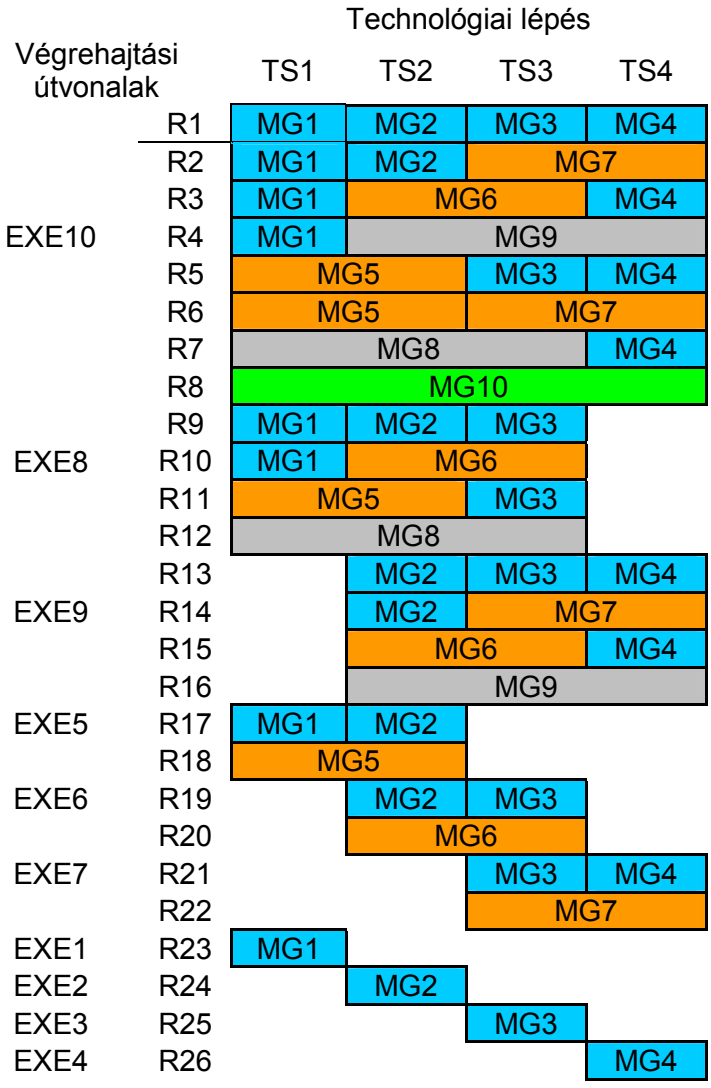



**6. ábra: EFFS modell gépcsoportjai négy technológiai lépés esetén ($N_{TS} = 4$)**

A végrehajtási lépések alapján a gépekből gépcsoportokat szervezek: *mg (mg = 1,..., $N_{MG}$)*, ahol $N_{MG} = N_{ES}$. Egy adott gépcsoporton belül azonos végrehajtásilépés-típussal, de más szempontból különböző képességekkel jellemezhető, párhuzamosan működő gépek lehetnek (6. ábra). A különböző gépcsoportba tartozó gépek száma eltérő lehet. Ezeket az összerendeléseket egy kapcsolótömbbel *MG_M* írom le (3. ábra: *NÉV1 = MG, NÉV2 = M*).

### 3.4.4. Végrehajtási útvonalak

A logisztikai egységek (munkák) végrehajtási útvonalakon mozognak. Az útvonalakat a végrehajtási lépések alapján definiálom. Egy útvonal a végrehajtási lépéseket realizáló gépcsoportok olyan sorozatát fogja egységbe, amelyben az igényelt technológiai lépések csak egyszeresen vannak lefedve, azaz a végrehajtási lépések közös része üres halmaz. A végrehajtási útvonalak azonosítására szolgáló sorszámok többféle szisztéma szerint is kioszthatók. Az azonosítóknak hivatkozási szerepük van, ezért azok kiosztása tetszőleges ugyan, de választás után rögzített. Egy lehetséges – a gyártási folyamat azonosítók által megkövetelt technológiai lépések számának csökkenő, azon belül a kezdő technológiai lépések növekvő sorrendje alapján megválasztott – elrendezés látható a 7. ábrán $N_{TS} = 4$ esetére.




**7. ábra: Végrehajtási útvonalak négy technológiai lépés esetén ($N_{TS} = 4$)**

Tetszőleges $N_{TS}$ számú technológiai lépés esetében a lehetséges végrehajtásiútvonal-típusok előállíthatók a 13.2 számú mellékletben leírt algoritmusok alkalmazásával. (Az *MG* gépcsoportok azonosítói – a definíciójukból következően – az *ES* végrehajtási lépések azonosítóival teljesen egyenértékűek).

A lehetséges végrehajtási útvonalak számát a következő összefüggés írja le:

$$N_R = \sum_{i=1}^{N_{TS}} (2^{(i-1)} \cdot (N_{TS} - (i-1))) . \qquad (3.5)$$

Bizonyítás:

A $2^{(i-1)}$ érték megadja egy *i* darab egymást követő technológiai lépésből álló gyártási folyamathoz tartozó lehetséges útvonalak darabszámát, $N_{TS} - (i-1)$ pedig az ilyen *i* hosszú különböző gyártási folyamat típusok darabszámát.

Minden *r (r = 1,...,$N_R$)* végrehajtási útvonalat a hozzá tartozó gépcsoportokkal együtt egy *R_MG* kapcsolótömbben definiálok (3. ábra: *NÉV1 = R, NÉV2 = MG*).

### 3.4.5. További speciális korlátozások

A gépek nem minden terméktípust képesek kezelni (az *m* gép *p* terméktípusra vonatkozó gyártási sebessége nulla is lehet: *M[m].PR[p] = 0*), így a gépcsoportok terméktípusonként eltérő összetételűek lehetnek. Ezek a terméktípustól függő csoportosítások hozzárendelhetők a termékekhez a következő módon:

*P[p].MG_M [mg][am]*, ahol:

*p =1,..., $N_P$;* a termékek,

*mg =1,..., $N_{MG}$;* a gépcsoportok,

*am = 1,..., P[p].MG_M[mg][0]*; az alkalmazható gépek indexei.

*P[p].MG_M[mg][am]* a *p* termék szempontjából az *mg* gépcsoporthoz tartozó *am*-edik gép.

Ebből az összerendelésből egyértelműen kiolvasható, hogy a különböző *p* terméktípusok gyártásához mely gépek használhatók. Ebből kiindulva és figyelembe véve a gépek képességeit és a termékek gyártási folyamatának végrehajtástípusait, a különböző *p* típusú termékekhez az azok gyártása során megengedett *r* végrehajtási útvonalak hozzárendelhetők és *P[p].R[ar]* formában leírhatók.

Indokolt azonban további hozzárendeléseket is megfogalmazni, amelyek lehetővé teszik a megrendelések szintjén az alkalmazható gépek halmazának speciális definiálását (adott számú technológiai lépés megvalósítására adott gépsor használata múltbeli okok miatt már kötelező), vagy a lehetőségek korlátozását (pl.: bizonyos gépek kizárása). Erre egyrészt az újraütemezési feladatok sajátos követelményei miatt van szükség. Másrészt lehetőséget ad arra, hogy akár egyedi esetekben emberi beavatkozással, akár valamilyen menedzselési stratégia szerint, azonos terméktípusra vonatkozó megrendelések teljesítéséhez részben eltérő gépállomány alkalmazása előírható legyen (pl.: különböző partneri viszony, minőségi elvárások, kockázatkezelés és egyéb okok miatt). Ennek érdekében a terméktípusokhoz hasonlóan a

megrendelésekhez is hozzárendelem *O[o].MG_M[mg][am]* formában egy kapcsolótömbbel az alkalmazható gépeket gépcsoportok szerint rendezve:
*o =1,..., $N_O$;* a megrendelések,
*mg =1,..., $N_{MG}$;* a gépcsoportok,
*am = 1,..., O[o].MG_M[mg][0]*; a használható párhuzamos gépek indexei.
*O[o].MG_M[mg][am]* az *o* megrendelés teljesítéséhez az *mg* gépcsoportból használható *am*-edik gép.

A szűkített gépállomány alapján, valamint egyéb okokból felmerülő teljes útvonalra vonatkozó megszorítások figyelembevételével a megvalósítható végrehajtási útvonalak meghatározhatók, és *O[o].R[ar]* formában hozzárendelhetők a megrendelésekhez.

Az ütemezési feladat megoldása során elkészítendő új ütemterv szempontjából a gépeknek már meglévő terhelését (amelyeket az utolsó érvényben lévő ütemterv határoz meg) egy *M[m].ENGAGED ($m = 1,..., N_M$)* attribútumban tárolom. Ebből kiolvasható az egyes gépek felszabadulásának időpontja. Ez a megoldás a rendszeres prediktív ütemezés során alkalmas az ütemtervek egymáshoz csatolására. A korábban beütemezett, még be nem fejezett munkák zárolásával kapcsolatos kérdéskört az értekezés későbbi, újraütemezéssel foglalkozó fejezetében részletesen vizsgálom.

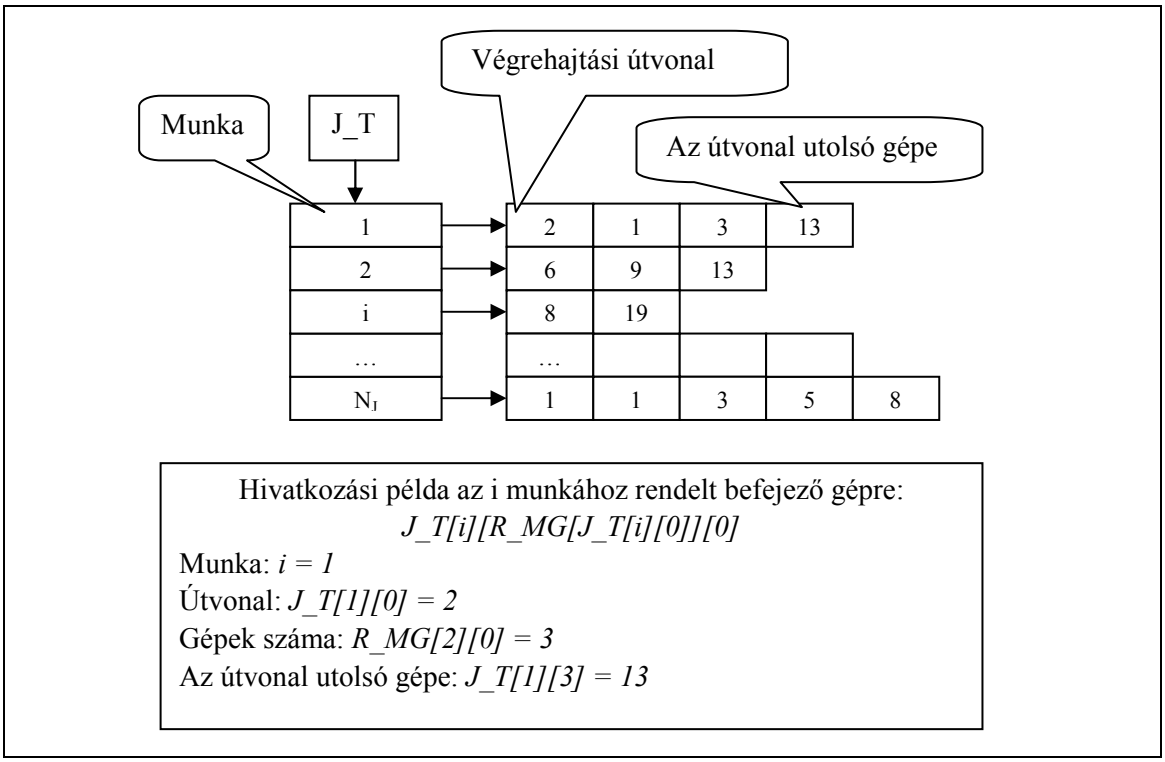
### 3.4.6. Ütemtervek és termelési finomprogramok

Az ütemezési feladatban szereplő munkák és gépek egymáshoz rendelését, valamint az így kialakuló gyártási feladatok végrehajtási sorrendjét együttesen definiáló objektumot ütemtervnek nevezem, jelölésére az *SCH* szimbólumot használom. Az előírt gyártási feladatok tervezett időadataival kibővített ütemtervet termelési finomprogramnak nevezem és *DSCH* szimbólummal jelölöm. (A *DSCH* magában foglalja az *SCH*-t.)

A termelési finomprogramot – annak érdekében, hogy minden részlete indexelhető legyen – három részből építem fel. Ezek a következők:

1. **Gyártási feladatok** (*J_T*)

Egy adott munka adott végrehajtási lépésének adott gépen történő megvalósítását nevezem gyártási feladatnak. A munkákhoz hozzárendelt végrehajtási útvonalakat a kiválasztott konkrét gépekkel együtt a *J_T[i][am]* kapcsolótömb rögzíti a következő módon:
*i* jelenti a munkát *($i = 1,..., N_J$)*,
*J_T[i][0]* jelenti az *i* munkához hozzárendelt útvonalat,
*R_MG[J_T[i][0]][0]* jelenti az útvonal bejárása során érintett gépek számát,
*J_T[i][am] (am=1,..., R_MG[J_T[i][0]][0])* jelenti az *i* munka *am*-edik végrehajtási lépéséhez rendelt gépet.
A *J_T* kapcsolótömb szerkezetének vázlata a 8. ábrán látható.




**8. ábra: A *J_T* kapcsolótömb szerkezete**

2. **Gyártási feladatok sorrendje** (*M_T*).

A gyártási feladatok gépenkénti sorrendjét az *M_T[m][ai]* kapcsolótömb mutatja:
*m (m= 1,…,$N_M$)* jelenti a gépet,
*M_T[m][0]* jelenti a munkák számát az *m* gépen,
*M_T[m][ai] (ai = 1,…, M_T[m][0])* jelenti az *m* gép munkasorozatának *ai*-edik munkáját (3. ábra: *NÉV1 = M, NÉV2 = J*).

3. **Gyártási feladatok időadatai** (*M_STET*).

Az *M_T* kapcsolótömb gyártási feladatainak időadatait az *M_STET* tömb tárolja a következő módon:
*M_STET[m][ai].ST* jelenti a kezdési időpontot,
*M_STET[m][ai].SetT* jelenti az gépátállás időtartamát,
*M_STET[m][ai].ProcT* jelenti a műveletvégzés időtartamát,
*M_STET[m][ai].ET* jelenti a befejezési időpontot.

A definiált gyártási feladatoknak két olyan adatát külön is indexelem, amelyek közvetett módon megkaphatók ugyan az 1. és 2. részek együtteséből, azonban közvetlen hivatkozással a termelési finomprogram kiértékelése nagymértékben felgyorsítható. Ezért egyrészt a termelési finomprogram 1. részének kibővítéseként eltárolom minden gyártási feladatnak a hozzá tartozó gép végrehajtási sorában elfoglalt *J_Tp[i][am]* pozícióját. Másrészt a 2. részben hasonló módon letárolom minden gyártási feladat esetében a hozzá tartozó munka végrehajtási lépésének *M_Tp[m][ai]* sorszámát.

### 3.4.7. Minősítést kifejező mutatószámok (performance index)

Egy megtervezett termelési finomprogram minősítése előre definiált, adott $N_K$ számú különböző célfüggvény kiértékelésével kapott függvényértékekkel – mint összehasonlításra alkalmas mutatószámokkal – megvalósítható és egyértelműen leírható a következő módon:

*k* a célfüggvény sorszáma *(k = 1,..., $N_K$)*,

*OBJ_VAL.F[k]* a *k*-adik célfüggvénynek a vizsgált termelési finomprogramra vonatkozó függvényértéke.

*F_Pri[k]* a *k*-adik célfüggvény fontosságát kifejező nemnegatív prioritásérték.

### 3.4.8. Gyártórendszer

A gyártórendszer adott időhorizontra vonatkozó tervezett vagy ténylegesen bekövetkezett állapotváltozási sorozatának (gyártási folyamatának) leírására az előzőekben részletesen bemutatott objektumokból felépített *SHOP* objektumot használom, a következő módon:

*SHOP.O* a rendelésállomány,

*SHOP.J* a munkaállomány,

*SHOP.M* az erőforrás-környezet,

*SHOP.DSCH* a végrehajtásra kiadott termelési finomprogram vagy a megvalósított termelési finomprogram,

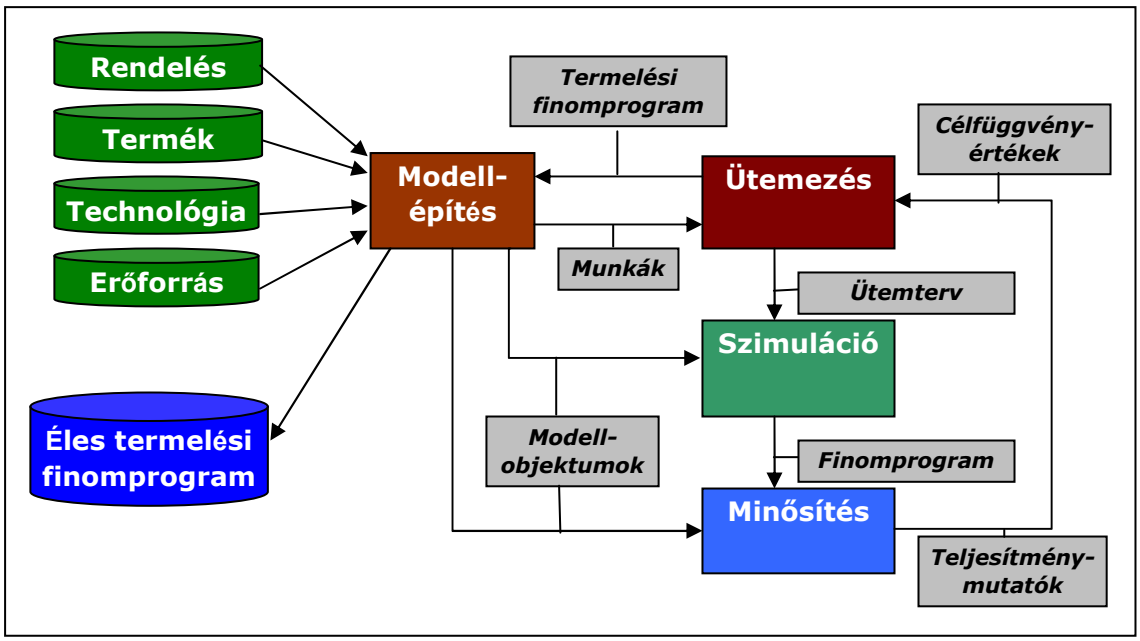
*SHOP.OBJ_VAL* a gyártási folyamat tervezett teljesítménymutatói vagy a gyártási folyamat tényleges teljesítménymutatói.

A bemutatott adatmodell fontos tulajdonsága, hogy input adatai egyszerűen feltölthetők a termelésinformatikai rendszerekben jellemzően tárolt alapadatok felhasználásával, továbbá a modell építőelemeinek attribútumai rugalmasan bővíthetők a kapcsolatrendszer változtatása nélkül.

# 4. TÖBBCÉLÚ INTEGRÁLT HEURISZTIKUS MEGOLDÁSI MÓDSZER

## 4.1. Megoldási koncepció

A termelési folyamat finomprogramozási feladatának megoldása során az ismert és pontosan meghatározott (technológiai, anyag- stb.) korlátozások figyelembe vételével, a gyártásra kiadott, ismert belső rendeléseken alapuló munkák elvégzéséhez gyártási erőforrások (gépek, eszközök stb.) allokálását és munkák indítási időpontjának sorozatát kell megtervezni úgy, hogy a definiált korlátozások teljesüljenek, és a vállalat magasabb szintjén megfogalmazott célok megvalósuljanak. Az EFFS feladatosztályba besorolható ütemezési feladatok megoldására egy többcélú integrált heurisztikus megoldási módszert fejlesztettem ki. A módszerre felépített ütemező szoftver működése során a feladat megoldását jelentő termelési finomprogram a 9. ábrán vázolt módon készül el.




**9. ábra: Az ütemező belső struktúrájának vázlata**

A termelésinformatikai rendszerek adatbázisában tárolt adatok felhasználásával a modellépítő komponens definiálja a rendszerben lévő objektumokat, és inicializálja azokat input adatokkal. Kiemelt fontosságú feladata a belső rendelések munkákra való felbontása és az érvényben lévő korlátozások definiálása. A korlátozások közül az előírt befejezési határidőket megsérthető (puha) korlátozásnak tekintve alkalmas mérőszámok megválasztásával ezek túllépésének minimalizálása ütemezési célként célfüggvények formájában jelenik meg. Minden más korlátozás kemény, azaz meg nem sérthető korlátozásként szerepel. A szükséges rendelkezésre állási, alkalmazhatósági és megvalósíthatósági vizsgálatok elvégzését követően a modellépítő komponens felépíti a teljes kapcsolatrendszert. Mindezek együttes célja az, hogy a feladatmegoldás során minden egyes döntési helyzetben a választható alternatív lehetőségek világosan és könnyen felismerhetők legyenek, ezáltal a döntés meghozatalának következményei biztosan megengedett és megvalósítható megoldást eredményezzenek.

A belső rendelések ütemezési alapegységekre bontásával önálló munkák jönnek létre. Ezek ütemezése egymástól függetlenül történik. Az ütemező modul minden egyes munkához hozzárendel egy megfelelő végrehajtási útvonalat, továbbá hozzárendel egy megfelelő gépet a kiválasztott útvonal minden egyes végrehajtási lépésének megfelelő gépcsoportból, és meghatározza minden érintett gépen a munkák végrehajtási sorban elfoglalt pozícióját. Ezáltal a munkákat feladatokra bontja fel. A gépeken a gyártási sorozatnagyságok és az azokat elválasztó átállítási műveletek dinamikusan, ütemezés közben alakulnak ki. Az ütemezési folyamat eredményeképpen egy megvalósítható ütemterv készül el.

Az ütemtervben szereplő feladatok időadatainak megtervezését egy megfelelően gyors szabályalapú szimulációs eljárás végzi, amely figyelembe veszi az egyes gépek rendelkezésre állási időintervallumait, az egyes gépeken az adott munkák sorrendje által meghatározott átállítási időket, a munka-gép összerendelések alapján számítható megmunkálási időket. A munka aktuális besorolása (korlátozások: időablak, sorrend, erőforrás, útvonal) alapján ismertté válik az egyes feladatok gépenkénti tervezett – és következményként a munkák, valamint megrendelések származtatott – indítási és befejezési időpontja.

Az időadatok felhasználásával az előírt célfüggvények kiértékelésére és a teljesítménymutatók kiszámítására kerül sor. A dinamikusan változó fontosságú, különböző dimenziójú és értékkészletű célfüggvények értékeinek együttes figyelembevételével az aktuálisan vizsgált megoldás és az addig legjobbnak ítélt megoldás egymáshoz viszonyított (relatív) minőségének számszerűsítésével kiértékelésre kerül a megoldás. Az ütemező iteratívan módosítja az aktuális ütemtervet, a lépéssorozat ismétlésével javul az eredmény.

A változó célokhoz való rugalmas alkalmazkodás úgy valósul meg, hogy a felhasználó az ütemezési folyamat megkezdése előtt – vagy akár a folyamat közben is – beállíthatja az előzetesen definiált célfüggvények aktuális fontosságának megfelelő prioritásértékeket.

## 4.2. Szimulációs modell

A gyártási folyamatot egy szabályalapú számítási eljárás szimulálja, melynek során az összerendelt munka-gép párosok időadatai kerülnek kiértékelésre. A szimuláció tervezése során feltételeztem, hogy a gépek között az átmeneti tárolók mérete gyakorlati szempontból nem korlátozott és a gépek közötti tranzakciós idők elhanyagolhatók. (A szimuláció alkalmas a különböző átmeneti tárolási stratégiák befogadására és tranzakciós idők definiálásával a gépek közötti anyagmozgatási idők figyelembe vételére is, ezek a működés lényegét nem befolyásolják.)

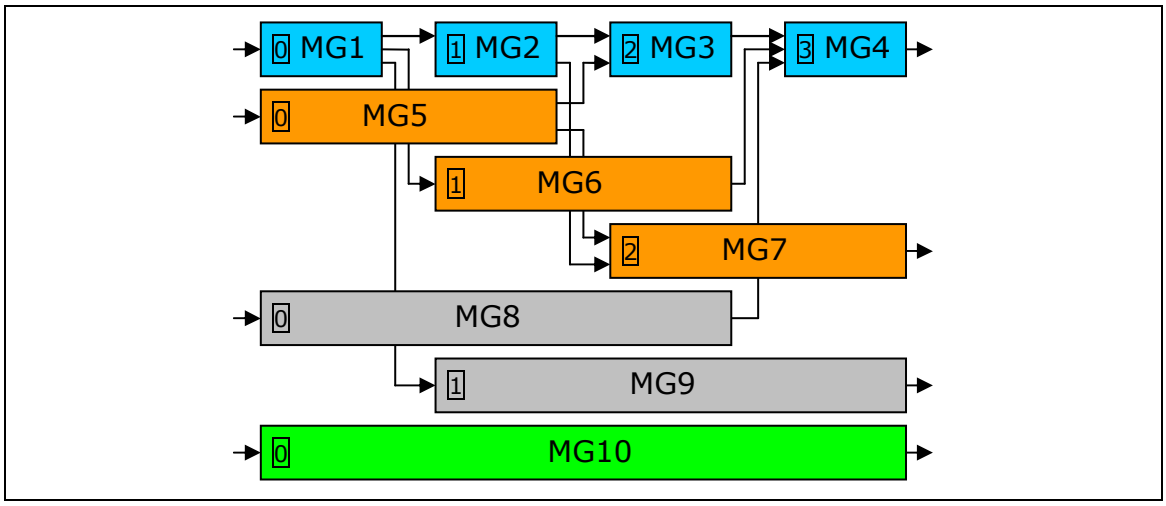
A szimuláció egy adott *SCH* ütemtervből indul ki, amelynek a két legfontosabb komponense a *J_T* feladatlistát definiáló kapcsolótömb és az *M_T* feladatsorrendet előíró kapcsolótömb. A gyártási feladatok nem szigorúan időrendi sorrendben kerülnek kiértékelésre, mert a gépek közötti gyakori átváltás lassítaná a szimulációt. A számítási folyamat minél gyorsabb végrehajtása érdekében a párhuzamos gépeken végzett műveletek szekvenciális műveletekké konvertálásával a gépeken előírt sorrendek szerint szakaszosan történik az időadatok számítása. Ehhez azonban szükséges a számítási lépések sorrendjének olyan megadási módja, amely figyelembe veszi a következőket:

- Egy adott *i* munkának egy adott *m* közbenső géphez tartozó időadatainak számításához szükséges – többek között – az *i* munka előző géphez tartozó befejezési időpontjának ismerete és az *m* gép előző munkájának befejezési időpontja is.
- A gyártási környezet tartalmaz útvonalcsatlakozásokat és -szétválásokat (csomópontok), ezért egy adott *m* gépre több azonos típusú és különböző típusú gépről is érkezhetnek munkák.
- Az azonos gépcsoportba tartozó gépek párhuzamosan működnek, így közöttük nem áll fenn előzési reláció.

A felsorolt jellemzőknek megfelelően célszerű a gépek helyett a gépcsoportok szintjén megfogalmazni a számítási műveletek sorrendjére vonatkozó korlátozásokat. A korlátozások leírhatók egy egyszerű (körútmentes) irányított gráffal, amelyben a csúcspontok jelentik a gépcsoportokat és az irányított élek mutatják a kötelező kiértékelési sorrendeket.

Minden egyes gépcsoporthoz hozzárendelhető a saját „befoka” (10. ábra). A befok $D_{IN}(mg)$ az adott *mg* gépcsoportba más gépcsoportból beérkező élek megengedett legnagyobb számát jelenti. Abban az esetben ha minden gépcsoportba tartozik legalább egy gép (teljes modell) a befok megegyezik a gépcsoport kezdő technológiai lépésének eggyel csökkentett értékével. A gépcsoportok keresett szekvenciája megkapható a befokok növekvő sorrendje alapján. A 10. ábrán vázolt esetben ez a sorrend a következő:

$$D_{IN}(4) > D_{IN}(3) \geq D_{IN}(7) > D_{IN}(2) \geq D_{IN}(6) \geq D_{IN}(9) > D_{IN}(1) \geq D_{IN}(5) \geq D_{IN}(8) \geq D_{IN}(10). \quad (4.1)$$




**10. ábra: Gépcsoportok kapcsolódásai négy technológiai lépés esetén ($N_{TS}$ = 4)**

Az azonos befokú gépcsoportok között a további, nem kritikus sorrendet meghatározhatják például a gépcsoportokra jellemző végrehajtási lépések úgy, hogy a több technológiai lépést integráló gépcsoport lesz a magasabb prioritású.

A gépcsoportokhoz rendelt prioritásokat egy *PRI_MG* kapcsolótömb rögzíti. A 10. ábrán vázolt esetben az összetartozó értékek a következők:

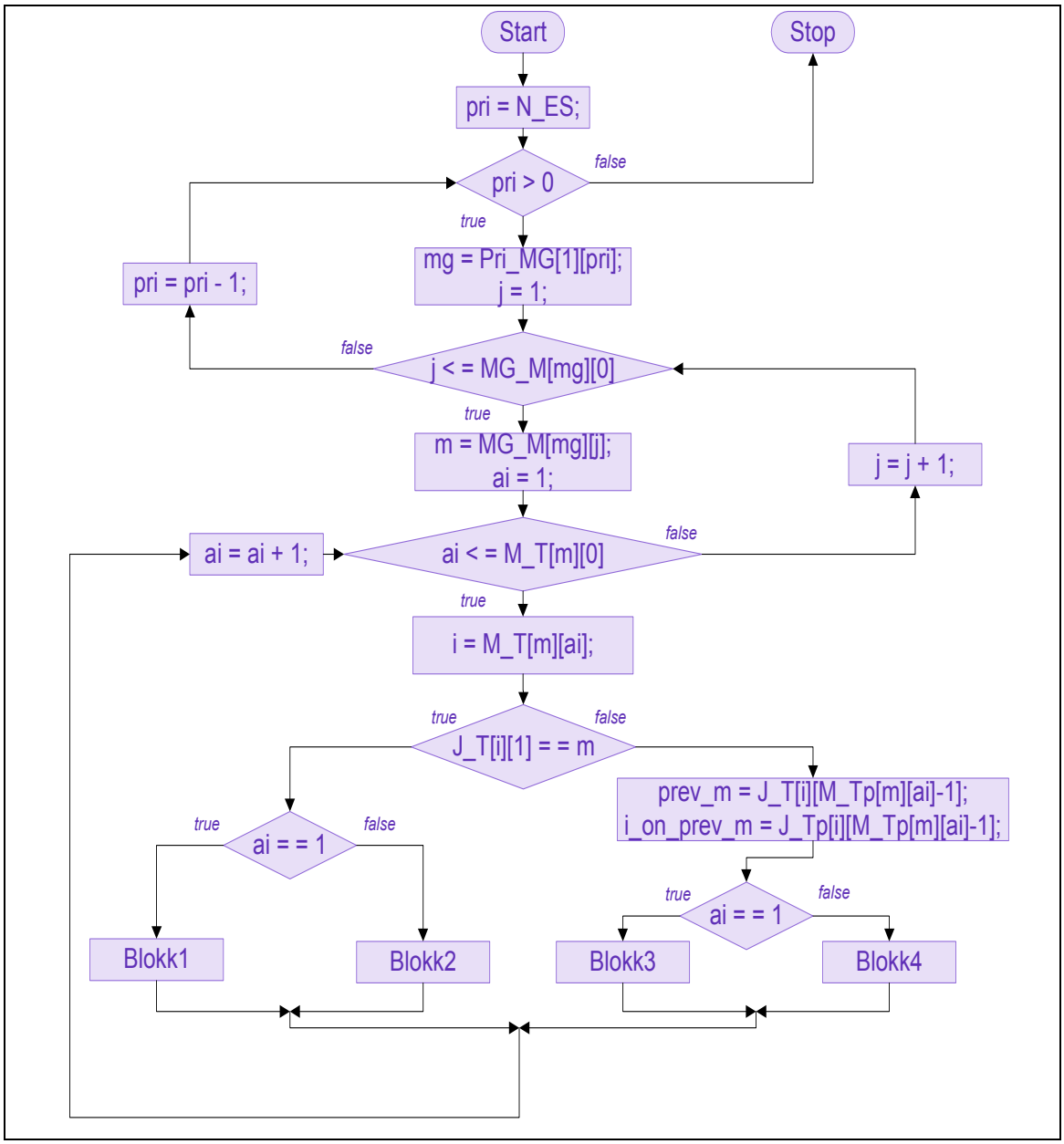
Prioritás: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Gépcsoport: {4, 3, 7, 2, 6, 9, 1, 5, 8, 10}

A gépcsoportokhoz rendelt prioritások csak és kizárólag az időadatok kiszámításának sorrendjét határozzák meg. Ezek biztosítják az egymásra épülő számítási részeredmények megfelelő sorrendű kiértékelését.

A szimuláció (kiértékelési szabályokra alapozott algoritmus) fontosabb lépései a 11. ábrán láthatók. A számítási folyamatban a prioritások nem növekvő sorrendjében kerülnek sorra a gépcsoportok. Az adott gépcsoporthoz tartozó összes gépen (párhuzamos gépek) az *M_T* tömb által meghatározott sorrendben kiszámításra kerülnek a feladatokhoz tartozó kezdési, előkészítési, műveleti és befejezési időadatok *(ST, SetT, ProcT, ET)*. Az eredmények bekerülnek az *M_STET* tömbbe.

A szimuláció során meghatározó szerepet játszanak a műveletkezdési időpontok. Az ütemtervben szereplő *m* gép végrehajtási sorának *ai*-edik pozíciójában szereplő *i* munka kezdési időpontját a következő értékek együttesen határozzák meg:

- az *i* munka legkorábbi kezdési időpontja *(O[J[i].OID].CST)*,
- az *i* munkának az előző gépen előállt befejezési időpontja *(M_STET[prev_m][i_on_prev_m].ET)*,
- az előző munkának az *m* gépen előállt befejezési időpontja *(M_STET[m][ai-1].ET)*,
- az *i* munkához tartozó átállási idő az *m* gépen *(M[m].ST[O[J[M_T[m][ai-1]].OID].P][O[J[i].OID].P])*,
- az *m* gép felszabadulásának időpontja az előző ütemterv terhelése alól *(M_ENGAGED[m])*, és
- az *m* gép munkarendje, azaz a rendelkezésre állási időintervallumok *(M[m].CAL[ci] (ci = 1,..., M[m].NCAL)*.




**11. ábra: A szimuláció folyamatábrája**

A különböző gyártástechnológiákhoz való alkalmazkodás érdekében a szimuláció alapvetően kétféle működési módot támogat:

1. Független átállások engedélyezettek: egy adott munka adott gépre érkezése előtt elkezdhető a gép beállítása/átállítása az adott munka végrehajtásához (pl. 12. ábra e2, e3).
2. Független átállások nem engedélyezettek: a gépek átállítása csak akkor kezdhető el, amikor az adott munka már a gépre megérkezett.

Mindkét működési mód esetében az *m* gép végrehajtási sorában az *i* munka *ai* sorszámának és az *m* gépnek az *i* munka végrehajtási útvonalához tartozó sorszámának (szerepének) figyelembe vételével négy esetet különböztetek meg (11. ábra Blokk1, Blokk2, Blokk3, Blokk4).

A független átállások engedélyezése (1. működési mód) esetén ezek a következők:

Blokk1: az *i* munka útvonalának első gépe az *m*,
az *m* gép első munkája az *i*.

```
o = J[i].OID;
pact = O[o].P;
M_STET[m][ai].SetT  = max_all_p( M[m].ST[p][pact] );
M_STET[m][ai].ST    = max( O[o].CST, M_ENGAGED[m] );
M_STET[m][ai].ProcT= P[pact].Q / M[m].PR[pact];
M_STET[m][ai].ET    = M_STET[m][ai].ST + M_STET[m][ai].SetT + M_STET[m][ai].ProcT;
Load_STET_to_CAL( M_STET[m][ai].ST, M_STET[m][ai].ET, 0, m, M);
```

Blokk2: az *i* munka útvonalának első gépe *m*,
az *m* gépnek egy közbenső munkája az *i*.

```
o = J[i].OID;
pact = O[o].P;
pprev = O[J[M_T[m][ai-1]].OID].P
M_STET[m][ai].SetT   = M[m].ST[pprev] [pact];
M_STET[m][ai].ST     = max( M_STET[m][ai-1].ET, O[o].CST );
M_STET[m][ai].ProcT = P[pact].Q / M[m].PR[pact]);
M_STET[m][ai].ET     = M_STET[m][ai].ST + M_STET[m][ai].SetT + M_STET[m][ai].ProcT;
Load_STET_to_CAL( M_STET[m][ai].ST, M_STET[m][ai].ET, M_ STET [m][ai-1].ET, m, M);
```

Blokk3: az *i* munka útvonalának egy közbenső gépe az *m*,
az *m* gépen az első munka az *i*.

```
o = J[i].OID;
pact = O[o].P;
M_STET[m][ai].SetT   = max_all_p( M[m].ST[p][pact] );
M_STET[m][ai].ST    = max(   M_STET[prev_m][i_on_prev_m].ET – M_STET[m][ai].SetT,
                             M_ENGAGED[m] );
M_STET[m][ai].ProcT = P[pact].Q / M[m].PR[pact];
M_STET[m][ai].ET     = M_STET[m][ai].ST + M_STET[m][ai].SetT + M_STET[m][ai].ProcT;
Load_STET_to_CAL( M_STET[m][ai].ST, M_STET[m][ai].ET, 0, m, M);
```

Blokk4: az *i* munka útvonalának egy közbenső gépe az *m*,
az *m* gépnek egy közbenső munkája az *i*.

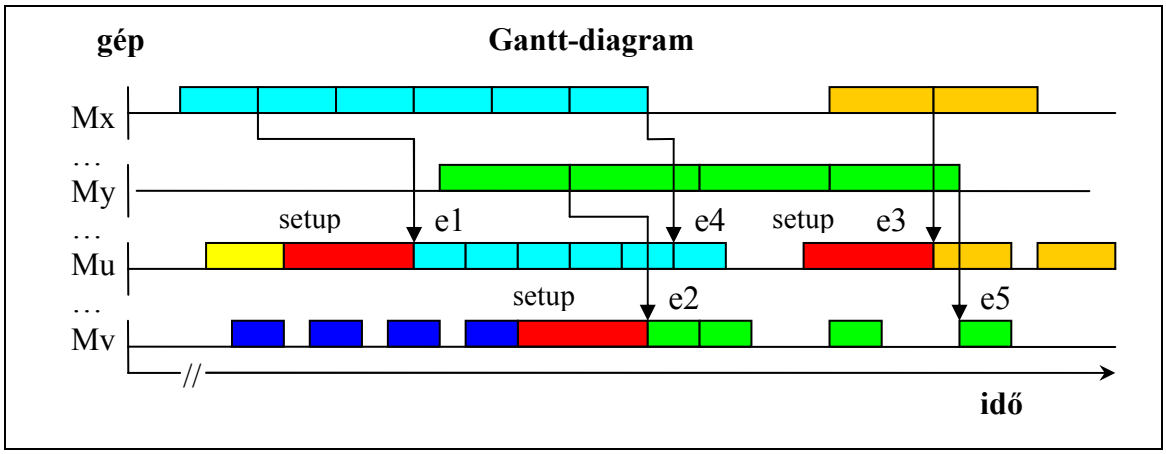
```
o = J[i].OID;
pact = O[o].P;
pprev = O[J[M_T[m][ai-1]].OID].P
M_STET[m][ai].SetT = M[m].ST[pprev] [pact];
if ( M_STET[prev_m][i_on_prev_m].ET <=  M_STET[m][ai -1].ET)
   M_STET[m][ai].ST = M_STET[m][ai -1].ET;
else
   M_STET[m][ai].ST = max( M_STET[prev_m][i_on_prev_m].ET –M_STET[m][ai].SetT,
                           M_STET[m][ai -1].ET);
M_STET[m][ai].ProcT    = P[pact].Q / M[m].PR[pact]);
M_STET[m][ai].ET       = M_STET[m][ai].ST + M_STET[m][ai].SetT + M_STET[m][ai].ProcT;
Load_STET_to_CAL( M_STET[m][ai].ST, M_STET[m][ai].ET, M_ STET [m][ai-1].ET, m, M);
```

A *Load_STET_to_CAL* függvény végzi az *i* munka kezdési és befejezési időpontjainak az *m* gép rendelkezésre állási időintervallumaihoz való illesztését, figyelembe véve a gép utolsó terhelésének befejezési időpontját. Ennek során, az *i* munka által képviselt fix méretű terhelést (időablakot) az *m* gép első (vagyis

legkorábban kezdődő) megfelelő méretű szabad időintervallumára balra illesztve helyezi el. Az algoritmusa váza a következő:

```
int _S::Load_STET_to_CAL (long *st_inout, long* et_inout, long last, int m, MACHINE* M)
{
  //---------------
  long st = *st_inout;
  long et = *et_inout;
  //---------------

  long size = et - st;
  long new_st = max(st, last);
  long new_et = new_st + size;
  int found = 0;
  int ci = 1;
  int cn = M[m].N_Cal;

          while (ci <= cn)
          {
           if ( new_st < M[m].CAL[ci].ET )
            {
              new_st = max (new_st, M[m].CAL[ci].ST);
              new_et = new_st + size;
              if ( new_et<= M[m].CAL[ci].ET)
                {
                found = 1;
                break;
                }
              else
               {
                ci++;
                if ( ci > cn ) {
                                found = 0;
                                break;
                               }
                new_st = M[m].CAL[ci].ST;
                new_et = new_st + size;
                continue;
               }
            }
           ci++;
          }

  //---------------
  *st_inout = new_st;
  *et_inout = new_et;
  //---------------
  return found;
}
```

A Blokk4 működését bemutató példák a 12. ábrán láthatók. A vázolt Gantt diagramon az *e1, e2, e3, e4, e5* szimbólumokkal jelölt események a munkák átvitelét jelentik a végrehajtási útvonal előző gépéről az aktuális gépre.




**12. ábra: Illusztratív példa a Blokk 4 működésére**

A kiemelt példákon jól megfigyelhető, hogy az

- *e1* esetben az adott gép előző munkájának befejezési ideje a meghatározó, ezután következhet a gép átállítása és a munka elindítása.
- *e2* esetben az előző munka befejezési időpontja kisebb, mint az aktuális munka befejezési időpontja az előző gépen, így ennek a különbségnek az értékével korábban kezdődhet a gép átállítása a munka tényleges megérkezése előtt.
- *e3* esetben az előző munka befejezési időpontja jóval kisebb, mint az aktuális munka befejezési időpontja az előző gépen, ezért a gép átállítása a teljes átállítási idővel korábbra tehető, így befejeződik a munka megérkezésének időpontjáig.
- *e4* esetben az előző munka befejezési időpontja a meghatározó, továbbá azonos terméktípusú munkák követik egymást, ezért nincs szükség átállításra,
- *e5* esetben sem szükséges átállítás és a munka előző gépről érkezése dominál.

A gyártási feladatok időadatainak ismeretében a munkák és a megrendelések időadatai származtathatók a következő módon:

Az *i* munka indítási időpontja:

*J[i].RST = M_STET[ J_T[i][ 1 ] ][ J_Tp[i][ 1 ] ].ST;*

a munkához tartozó első feladat indítási időpontja.

Az *i* munka befejezési időpontja:

*J[i].RET = M_STET[ J_T[i][fmg] ][ J_Tp[i][fmg] ].ET*; mivel

az *fmg* a befejező feladat gépcsoportja: *fmg = R_MG[J_T[i][0]][0]*,

ezen belül a konkrét hozzárendelt gép azonosítója: *J_T[i][fmg]* (8. ábra),

melyen az i munka pozíciója: *J_Tp[i][fmg]*.

Az *o* rendelés indítási időpontja:

$O[o].RST = \min_{ai} (J[i].RST);$ ahol $(ai = O[o].FJ, \ldots, O[o].FJ+O[o].NJ-1;)$

a rendeléshez tartozó legkorábban indított munka indítási időpontja.

Az *o* rendelés teljesítésének időpontja:

$O[o].RET = \max_{ai} (J[i].RET);$ ahol $(ai = O[o].FJ, \ldots, O[o].FJ+O[o].NJ-1;)$

a rendeléshez tartozó, utolsónak elkészült munka befejezési időpontja.

A szimuláció lefutásának eredményei az egyes gyártási feladatok tervezett időadatai, amelyek az *M_STET* tömbben találhatók, valamint a munkák és megrendelések ezekből származtatott indítási és befejezési időpontjai, amelyek rendre a *J[i].RST*, *J[i].RET, O[o].RST, O[o].RET* megfelelő objektumok attribútumaiból kiolvashatók.

A végrehajtás-szimuláció tehát a problématér transzformációját valósítja meg azáltal, hogy egy kiindulási „egyszerűsített" *SCH* ütemtervhez – amely a munkák és gépek megengedett összerendeléseit és a munkák végrehajtásának gépenkénti sorrendjét tartalmazza – a vázolt szabályok szerint egyértelműen hozzárendel egy megvalósítható, részletes *DSCH* termelési finomprogramot. A szimulációs modul a gyártási feladatok, munkák és megrendelések időadatait számítja ki, a döntéseket az ütemező modul hozza meg.

## 4.3. Megoldások értékelésének matematikai modellje

### 4.3.1. Optimalizálási célfüggvények

A megtervezett termelési finomprogramok minősítéséhez a különböző szempontokat reprezentáló célfüggvények kiértékelésével kapott eredmények adják a kiindulási alapot. Az ilyen összehasonlításra alkalmas mutatószámok típusa, dimenziója és értéke egyaránt tág határok között mozoghat.

A minősítendő termelési finomprogramban szereplő *i (i=1,...,$N_J$)* munkákhoz eleve adott – puha korlátozásként – az előírt befejezési határidő:

$$D_i = \text{O[J[i].OID].CET} . \tag{4.2}$$

A szimuláció következtében ismerté válik

a kezdési időpont: $R_i = \text{J[i].RST}$, (4.3)

és a befejezési időpont: $C_i = \text{J[i].RET}$. (4.4)

Ezeket felhasználva felírható az *i* munkára vonatkozó

határidőtől való eltérés: $L_i = C_i - D_i$, (4.5)

csúszás: $T_i = \max(0, L_i)$, (4.6)

korai befejezés: $E_i = \max(0, -L_i)$, (4.7)

átfutási idő: $Fl_i = C_i - R_i$. (4.8)

Az $R_i$, $C_i$, $L_i$, $T_i$, $E_i$, $Fl_i$, függvények bármelyikét általánosan $G_i$ függvénynek tekintve a leggyakrabban alkalmazott célfüggvények kifejezhetők a következő módon:

legnagyobb érték: $f = \max_i(G_i)$, (4.9)

összeg: $f = \sum_i G_i$, (4.10)

átlag: $f = \dfrac{\sum_i G_i}{N_J}$. (4.11)

A munkák fontosságának kifejezésére nemnegatív $v_i$ súlyokat lehet a munkákhoz hozzárendelni. A célfüggvényekben ezek figyelembe vehetők például a következő módon:

súlyozott maximum: $$f = \max_i (v_i \cdot G_i), \quad (4.12)$$

súlyozott összeg: $$f = \sum_i v_i \cdot G_i, \quad (4.13)$$

súlyozott átlag: $$f = \frac{\sum_i v_i \cdot G_i}{N_J}. \quad (4.14)$$

Számos további egyszerű vagy összetett célfüggvény is megfogalmazható.

Határidőhöz kapcsolódó fontos speciális szerepet betöltő célfüggvény

a késő munkák száma: $$f = |\{i \mid T_i > 0\}|, \quad (4.15)$$

a súlyozott termelési pontosság: $$f = \sum_i (u_i \cdot E_i + v_i \cdot T_i), \quad (4.16)$$

ahol az $u_i$ és a $v_i$ nemnegatív valós számok. Ezek a célfüggvények a rendelésre történő gyártásnál a „service level” minősítés legfontosabb összetevői.

A leggyakrabban használt nem határidőhöz kapcsolódó célfüggvény a a teljes munkacsoport együttes átfutási ideje (makespan):

$$f = \max_i (C_i) - \min_i (R_i). \quad (4.17)$$

Szintén határidőtől független fontos célfüggvény a gépátállítások száma:

$$f = |\{(m, ai) \mid m = 1,...,N_M; ai = 1,...,M\_T[m][0]; M\_STET[m][ai].Set > 0\}|. \quad (4.18)$$

A munkákhoz hasonlóan a megrendelésekre is értelmezhetők a vázolt célfüggvények. Ekkor az *i (i=1,…,$N_J$)* munkák szerepét az *o (o=1,…,$N_O$)* rendelések veszik át, valamint minden *o* rendelésre adott

az előírt befejezési határidő: $$D_o = \text{O[o].CET}, \quad (4.19)$$

a kezdési időpont: $$R_o = \text{O[o].RST}, \quad (4.20)$$

és a befejezési időpont: $$C_o = \text{O[o].RET}. \quad (4.21)$$

A vázolt célfüggvények az irodalomból ismert termelési háromszög állapotváltozói közül a szállítókészségre helyezik a hangsúlyt. További célfüggvények is megfogalmazhatók a lekötött készletszinthez és az erőforrás-kihasználáshoz kapcsolódóan. Ezeknek a paramétereknek a felhasználásával, az adott követelményeket (üzleti célokat) még pontosabban és részletesebben kifejező mutatók is megfogalmazhatók.

A bemutatott példákból jól látható, hogy az ütemezési feladatban alkalmazható optimalizálási célfüggvények igen sokfélék lehetnek. A részletes rövid távú ütemterv a gyártásirányítás – mint „szabályozás” – „alapjele”. A gyártásirányítás valós idejű célja ennek magvalósítása, követése. Egy sokoldalúan és hatékonyan használható ütemezővel szemben tehát fontos követelmény, hogy a termelési politika változó céljának megfelelően a célfüggvények rugalmasan változtathatók legyenek. Ellenkező esetben az „optimális” jelző értemét veszthetí.

### 4.3.2. Többcélú optimalizálási feladat megengedett megoldásainak összehasonlítása és értékelése

Olyan ütemezési feladatokban (és más optimalizálási feladatokban is), amelyekben az optimalizálás célja egyetlen célfüggvénnyel megadható, két különböző megoldás minőségének összehasonlítása egyszerűen elvégezhető. Ilyen esetekben elegendő a különböző megoldásokra vonatkozó célfüggvényértékeket kiszámítani és azok alapján egyértelműen számszerűsíthető a minőségbeli különbség.

Sokkal nehezebb a különböző megoldások minőségének összehasonlítása a többcélú optimalizálási feladatokban. Több optimalizálási cél esetén ugyanis különböző dimenziójú és értékkészletű célfüggvények szerepelhetnek, amelyek gyakran egymással valamilyen (legtöbbször pontosan nem is ismert) kölcsönkapcsolatban állhatnak. Az ilyen feladatoknak csak kivételes esetekben van olyan megoldása, amely az összes célfüggvény szempontjából egyszerre tekinthető optimálisnak. Kompromisszumos megoldás értelmezéséhez és megtalálásához szükséges annak definiálása, hogy milyen szempontok szerint és hogyan hasonlíthatók össze a lehetséges megoldások.

A többcélú optimalizálási feladatok egyik gyakran alkalmazott megoldási módszere az, hogy visszavezetik azokat valamilyen technika alkalmazásával egycélú optimalizálási feladatra. Legtöbbször erre a célra valamilyen összetett függvényt használnak – amely gyakran a célfüggvények súlyozott lineáris konbinációja –, és annak az optimumát próbálják meghatározni. Ennek a módszernek a nehézségét a súlyozótényezők értékének megadása jelenti, mivel ennek megválasztásától nagymértékben függ az optimális megoldás. Alkalmazását tovább nehezíti az, hogy nem különíthető el élesen a különböző célfüggvények összeadásához szükséges normalizálás és a fontosságbeli különbségek kifejezésére irányuló prioritások kiosztásának művelete.

Egy másfajta megközelítést megvalósító módszer a hierarchikus optimalizálás. Ennek során a különböző célfüggvények között valamilyen rangsort felállítva, az adott sorrendnek megfelelően egyenként kerül sor a célfüggvények minimalizálására, figyelembe véve a korábban elért eredmények köré felállított tartományokban engedélyezett mozgásokat. Legnagyobb hátránya ennek a módszernek az, hogy a célfüggvények fontosságának időbeli változását nagyon nehezen tudja követni.

Ehhez hasonló a célprogramozás módszerének alapgondolata is. Ebben a megközelítésben a különböző célfüggvényeket a lehetőségekhez mérten át kell transzformálni speciális korlátozásokká. Ezt követően olyan megoldás megtalálására kell koncentrálni, amely az ilyen módon kibővített korlátozásokat maradéktalanul kielégíti és a megmaradt egyetlen célfüggvény szempontjából a legjobb megoldásnak tekinthető. Dinamikus rendszerben, amikor az aktuális célfüggvénykészlet összetétele és az összetevők fontossága időről időre változik, az algoritmikus megoldás dominanciája miatt nehezen adaptálható a módszer.

Több kritérium együttes kezelésére alkalmas, hatékony eredményértékelő és -összehasonlító lehetőséget nyújtanak a Pareto-elven működő módszerek is. A megoldáskeresés koncepciója főként abban tér el az előzőekben vizsgált módszerektől,

hogy ebben a megközelítésben a különböző célfüggvények megmaradnak eredeti alakjukban, és párhuzamosan kerülnek kiértékelésre. Két megoldás minőségének összehasonlítása során a célfüggvények értékein túlmenően logikai kifejezések is szerepet játszanak. Nincs szükség a célfüggvények értékkészletének egységesítésére, mert a különböző megoldások esetében is rendre csak ugyanolyan típusú célfüggvények kerülnek összehasonlításra. A cél egy olyan $s_p$ megoldás megtalálása, melyről bármely irányba elmozdulva, a környezetében nincs olyan *s'* megoldás, amelyik legalább egy célfüggvény értékében jobb $s_p$ megoldásnál úgy, hogy az összes többi célfüggvény értéke legalább ugyanolyan jó mint az $s_p$ megoldás értékei. A módszer alkalmazhatóságát részben korlátozza az, hogy a célfüggvények eltérő fontossága a felhasználó által ilyenkor közvetlenül nem befolyásolható. Másrészt az összehasonlításra alkalmas megoldások generálása meglehetősen nehézkessé válhat nagyméretű, tagolt, nem homogén megoldástérben.

Léteznek tehát különböző módszerek és elvek, azonban a vizsgált ütemezési (optimalizálási) feladat sajátosságainak figyelembevételére, dinamikus rendszerben történő alkalmazásra a paraméterezés nehézségei miatt ezek kevésbé alkalmasak közvetlenül.

Egy új szemléletű, rugalmasan és könnyen adaptálható módszer kifejlesztésével az ismert megoldási módszerek hasznos tulajdonságai előnyösen ötvözhetők. A követelményeket a következőképpen fogalmaztam meg:

- Célszerű a feladatmegoldások jóságának összehasonlítása során egyszerű, egymástól független, időben változtatható, tetszőlegesen megválasztható nemnegatív értékű prioritásértéket hozzárendelni a célfüggvényekhez, kizárólag azok fontosságának kifejezésére.
- Előnyösebb lehet párosítva, rendre csak azonos típusú célfüggvények értékeit összehasonlítani, mint a különböző célfüggvényeket egységesíteni, és egyetlen összetett célfüggvénybe beépíteni.
- Rugalmasabb megoldási módszer hozható létre a változások relatív mértékének figyelembevételével, mint a célfüggvények értékeinek globális szemléletű értelmezésével.

A felsorolt megfontolásoknak megfelelően kifejlesztettem egy új módszert, amelyben két megoldás jóságának az összehasonlítása nem a megoldások külön-külön vett abszolút jóságának valamilyen módszerrel végrehajtott számszerűsítésén, majd ezek összehasonlításának eredményén alapul, hanem az egyik megoldásnak a másikhoz viszonyított (relatív) jósága kerül számszerűsítésre, és ennek alapján dönthető el az, hogy melyik tekinthető jobb megoldásnak.

Jelölje $S$ a megengedett megoldások halmazát.
Legyenek adottak az $f_1,...,f_K$ optimalizálási célok:

$$f_k : S \to \Re^+ \cup \{0\},\ \forall k \in \{1,2,...,K\}, \tag{4.22}$$

melyek mindegyike felírható minimalizálandó célfüggvények formájában.

A feladat a legjobb $s^* \in S$ „kompromisszumos" megoldás megtalálása.

$$\min_{s \in S}\left[f_1(s),...f_k(s),...f_K(s)\right],\ \forall k \in \{1,2,...,K\}. \tag{4.23}$$

Legyenek

$$w_k \in \Re,\ w_k \geq 0,\ \forall k \in \{1,2,...,K\}, \tag{4.24}$$

a célfüggvények aktuális fontosságát, „prioritását" kifejező tényezők, amelyeknek az aktuális értéke egymástól függetlenül, a felhasználó által tetszőlegesen beállítható.

Legyenek az $s_x, s_y, s_z \in S$ lehetséges (minden korlátozást kielégítő, megengedett) megoldások.

Jelölje *max* a következő operátort:

$$\max : \Re^2 \to \Re,\ \max(a,b) = \begin{cases} a, \text{ ha } a > b \\ b, \text{ egyébként} \end{cases}. \tag{4.25}$$

Legyen $D$ a következő függvény:

$$D : \Re^2 \to \Re,\ a,b \in \Re,\ D(a,b) = \begin{cases} 0, \text{ ha } \max(a,b) = 0 \\ \dfrac{b-a}{\max(a,b)} \cdot 100, \text{ egyébként} \end{cases}. \tag{4.26}$$

Legyen $F$ a következő függvény:

$$F : S^2 \to \Re,\ F(s_x, s_y) = \sum_{k=1}^{K} (w_k \cdot D(f_k(s_x), f_k(s_y))). \tag{4.27}$$

Ezek alapján az $F(s_x, s_y)$ előjeles érték jelenti az $s_y$ megoldásnak az $s_x$ megoldáshoz viszonyított (relatív) jóságát (minőségét).

Az $F$ kétváltozós függvény felhasználásával a következő alapvető relációs operátorok definiálhatók:

$s_x < s_y$ ($s_x$ jobb megoldás, mint $s_y$) akkor, ha $F(s_x, s_y) > 0$.

$s_x = s_y$ ($s_x$, $s_y$ azonosan jó megoldások) akkor, ha $F(s_x, s_y) = 0$.

$s_x > s_y$ ($s_x$ rosszabb megoldás, mint $s_y$) akkor, ha $F(s_x, s_y) < 0$.

Ezeket felhasználva a relációs operátorok köre tovább bővíthető:

$s_x \leq s_y$ ($s_x$ legalább olyan jó megoldás, mint $s_y$) akkor, ha $F(s_x, s_y) \geq 0$.

$s_x \geq s_y$ ($s_x$ legfeljebb olyan jó megoldás, mint $s_y$) akkor, ha $F(s_x, s_y) \leq 0$.

Az ilyen módon – a célfüggvények értékeire alapozva közvetlenül a lehetséges feladatmegoldások halmazán – értelmezett relációs operátorok felhasználásával lehetővé válik a különböző megoldások páronkénti összehasonlítása és értékelése. Ezáltal a valós számok körében alkalmazott relációs operátorokkal teljes mértékben megegyező módon, különböző stratégiák szerint kiválasztható a legjobb megoldás (ha csak egyetlen optimum létezik) vagy egy megoldás a legjobb megoldások halmazából (ha több optimum egyszerre létezik).

A célfüggvényekhez rendelt $w_k$ (4.24) prioritások (változók) növelik a módszer rugalmasságát azáltal, hogy interaktív bemenő paraméterekként funkcionálnak. Ez lehetővé teszi, hogy a felhasználó az egymástól független aktuális $w_k$ értékek megadásával egyszerűen definiálhassa az egyes célfüggvények figyelembevételének kívánatos mértékét a döntési szituációban. Ezeknek a prioritásoknak nincs függvény-normálási szerepe, kizárólag a célfüggvényben kifejezett döntéshozási szempont aktuális fontosságát fejezik ki. Gyakorlati szempontból a prioritásértékek nulla és egy célszerűen választott pozitív felső korlát közötti intervallumból kaphatnak értéket.

A *D* függvényben (operációban) (4.27) mindig páronként, rendre csak azonos típusú komponens-célfüggvények értékei kerülnek feldolgozásra, így nincs szükség a különböző típusú célfüggvények értékének „normálására". A célfüggvényértékek elvileg tetszőleges dimenziójúak lehetnek. Egyetlen szigorú korlátozás, hogy a célfüggvényeket olyan formában kell megfogalmazni a modellben, hogy értékkészletük a nemnegatív valós számok halmaza legyen (4.22). Ennek következménye, hogy a *D* operátor nevezőjében lévő *max* függvény (4.25) értéke csak akkor lehet nulla, ha mindkét (összehasonlítás alatt álló) célfüggvény-érték nulla. Minden más esetben a *max* operátor pozitív valós számot eredményez.

A számlálóban szereplő különbségképzés határozza meg a két célfüggvény-érték eltérésének előjeles értékét. Mivel ez kivonási művelet, az operáció nem kommutatív.

Az osztás és a 100 szorzótényező miatt válik a *D* operáció eredménye 0-100 közötti pozitív valós értékké. Végül, a célfüggvényérték-párok mindegyikére kiszámított relatív változások és a prioritásértékek szorzatösszege (skalár-szorzata, *F* függvény (4.27)) mutatja meg az eredő relatív változás végső minősítését.

A módszer jellemzője, hogy a nulla célfüggvényérték közelében is támogatja a nulla érték elérését és megtartását azáltal, hogy a százalékos változást felerősíti azokban az esetekben, amikor a vizsgált célfüggvény egyik megoldásra vonatkozó értéke nulla, a másik megoldásra vonatkozó értéke pedig nullánál nagyobb szám. Ennek főként olyan típusú célfüggvények esetében van különös jelentősége, amelyek ténylegesen felveszik a nulla értéket (ilyenek pl.: az ütemezéseknél nagy fontosságú késésekkel kapcsolatos célfüggvények).

A *D* operációban – a *max* operáció helyett – más alkalmasan megválasztott függvény is használható lenne (pl. $(a+b)/2$, $+\sqrt{a^2+b^2}$ stb.). A módszer lényege – a páronkénti relatív minősítés elve – megmaradna. A *max* függvény használatát egyszerűsége, a program futása közbeni gyors kiértékelhetősége és jó „kifejező ereje" indokolja.

A módszer még tovább általánosítható, ha a mindenkori optimalizálási feladatban szereplő célfüggvények jellegzetességeit figyelembe véve, komponensenként (vagy komponenscsoportonként) eltérő *D* operációvariánsok szerepelnek az *F* függvényben.

A bemutatott módszer olyan esetekben is hatékonyan alkalmazható, amikor az aktuális célfüggvénykészlet összetétele és az összetevők fontossága egy optimalizálási „politika" részei, és ez a politika időről időre változik.

## 4.4. Általános célú heurisztikus ütemezési módszer

A végrehajtás-szemléletű szimulációra alapozott megoldási koncepció következményeként a termelési finomprogram elkészítése az egyes munkák feladatokra bontására és a feladatok gépenkénti sorrendjének meghatározására redukálódik, miközben a lehetséges megvalósítható finomprogramok halmaza könnyebben kezelhetővé válik.

Ennek a redukált problémának a megoldására többféle heurisztikus módszer is számításba jöhet. Elemezve a heurisztikák teljesítményét, a következő koncepciót fejlesztettem ki.

A kezdeti megoldásokat előállító algoritmusok eredménye nem függ közvetlenül a célfüggvényektől és azok prioritásértékétől. A kiválasztott algoritmusokba épített heurisztikák és diszpécserszabályok határozzák meg az ütemterv minőségét. Általános cél az, hogy ne legyenek túlságosan szétdaraboltak a belső rendelések, ne legyen túl sok gépátállítás és gépvárakozás, ugyanakkor a munkák indítási sorrendje viszonylag összhangban legyen az előírt időablakokkal és a használható erőforrások képességeivel.

A továbbfejlesztő (javító) eljárás a gyakran változó célok együtteséhez való rugalmas alkalmazkodás képességének fokozása érdekében, a termelési finomprogramok minőségének összehasonlítására a 4.3.2. fejezetben bemutatott többcélú eredményértékelő módszert használja.

A kidolgozott eljárások a megoldási folyamat menetét figyelembe véve alapvetően két csoportba sorolhatók:

Az első csoportba tartoznak a felépítő jellegű heurisztikák. Ezek a módszerek az előző finomprogram még be nem fejezett részéből kiindulva, a kiválasztott soron következő ütemezendő munkához tartozó útvonalra, gépre és sorrendre vonatkozó döntéseket heurisztikus szabályok és szimulációs kiértékelés alapján hozzák meg.

A második csoportba tartoznak az iteratív javításon alapuló heurisztikák. Ezeknél a módszereknél egy kezdeti megvalósítható ütemtervből kiindulva megengedett módosítások és szimulációs kiértékelések iterálásával alakul ki az aktuális céloknak jobban megfelelő termelési finomprogram.

Ezekre a gyors és rugalmas, könnyen adaptálható megoldási módszerekre alapozva a kifejlesztett ütemező „gép” (engine) a hozzárendelési és sorrendi problémákat egy kétfázisú, kombinált heurisztikus módszerrel oldja meg.

A módszert a következők jellemzik:

- Az első fázisban gyors felépítő algoritmusok több „jó” kezdeti megoldást generálnak.
- A legjobb kezdeti megoldásból kiindulva a második fázisban többféle szomszédsági operátor kombinált alkalmazásával, többfunkciós tabulista-kezeléssel és folyamatos tanuláson alapuló, paraméteres szabályozással működő, heurisztikus megoldó algoritmus javítja a megoldást az optimális vagy közel optimális eredmény elérése érdekében.

### 4.4.1. Heurisztikus felépítő algoritmusok

A termelési finomprogram elkészítésének első fázisában az a cél, hogy rövid idő alatt egy „viszonylag jó” kezdeti ütemterv készüljön el. Az ütemezési feladatok változatossága (sok feladat, dinamikus sorozatnagyság, párhuzamosan működő gépek, alternatív technológiai útvonalak, több termeléspolitikai cél stb.) miatt szükséges, hogy az alkalmazott heurisztikák rugalmasan hozzáigazodjanak az aktuális feladathalmazhoz és azok végrehajtási környezetéhez.

A fejlesztés során azt tapasztaltam, hogy különösen a különböző termelési célok (célfüggvényrendszerek, -típusok és -prioritások) más-más heurisztikus algoritmussal kezelhetők hatékonyan. Mivel a termelési célokat kifejező menedzserindexek és prioritások (súlyok) sokfélék lehetnek, és ezek egymásra is hatással vannak, ezért a kezdeti fázisban „viszonylag jó” megoldás eléréséhez többféle heurisztikus algoritmusra lenne szükség. Ilyenkor azonban a specializált algoritmusok által előnyben részesített célfüggvények értékének javulása más célfüggvények értékének erőteljes romlásához vezethet.

Mindezek figyelembevételével a kezdeti megoldás előállítása során a legfontosabb termelési jellemzők egyensúlyban tartása lehet a cél. A vezérelvem az volt, hogy a lehető legrövidebb idő alatt egy olyan általános értelemben vett „viszonylag jó” ütemterv készüljön el, amelynek egyetlen célfüggvényértéke sem túl rossz. Ennek a legfontosabb előnye, hogy a második fázisban az iteratívan javító algoritmus bármely (az aktuális célfüggvényrendszernek megfelelő) irányba hasonló feltételekkel tudja majd elmozdítani a célfüggvényértékeket. A továbbiakban a kifejlesztett algoritmusok közül egy ilyen „terheléskiegyensúlyozó” elven működő heurisztikus algoritmust mutatok be részletesen.

A modellépítő az input adatok feldolgozása során előzetesen elkészít egy kapcsolótömböt, amelyben a gyártási megrendelések indexeit a megkövetelt gyártási-folyamat-típusok alapján csoportosítva tárolja el a következő algoritmus szerint:

```
Classify_orders_according_to_EXE(int ** E_O, ORDER* O, PRODUCT* P, int NO, int N_EXE)
{
int * E_O_number = new int[N_EXE];  //temp
 for (int e=0; e<=N_EXE; e++ )
    E_O_number[e]=0;
 for (int o=1; o<=NO; o++)
    E_O_number[P[O[o].P].EXE]++;
  //create memory model for order groups
  E_O = new int*[N_EXE + 1];
  for (int e=1; e<=N_EXE; e++)
  {   E_O[e] = new int[E_O_number[e]+1];
      E_O[e][0] = 0;
   }
  delete [] E_O_number;

  for (int o=1; o<=NO; o++)
  {  E_O[P[O[o].P].EXE][E_O[P[O[o].P].EXE][0]+1] = o;
     E_O[P[O[o].P].EXE][0]++;
  }
}
```

Az *E_O* kapcsolótömböt felhasználva a kezdeti ütemtervet felépítő (Basic Workload Balancing Algorithm, *BWBA*) terheléskiegyenlítő algoritmus fő lépései a következők:

- Az algoritmus kiveszi az *E_O* kapcsolótömbből az *o* megrendeléseket a gyártásifolyamat-típusok *EXE* azonosítói szerint növekvő sorrendben (3.4.1. fejezet 4. ábra).
- Az egyes megrendelések megbontás nélkül kerülnek ütemezésre. Az adott megrendeléshez tartozó munkák ugyanazokon a gépeken egymást követve haladnak végig.
- A megengedett *r* útvonalalternatívák közül az kerül kiválasztásra, amelyik mentén az érintett gépek a legmagasabb integráltsági fokkal rendelkeznek, vagyis több művelet együttes végrehajtására képesek (3.4.4. fejezet 7. ábra).
- Az algoritmus a kiválasztott útvonal által érintett *mg* gépcsoportokból egyenletes valószínűséggel véletlenszerűen választott konkrét *m* gépekhez rendeli hozzá a munkákat.
- A gépeken a munkák végrehajtási sorrendjét a beütemezés sorrendje határozza meg. Az aktuális ütemezett munka a terhelési lista végére kerül.

Az algoritmus a gépek egyenletes terhelésére törekszik. A magasabb integráltsági fokú gépeket főként a több technológiai lépést megkövetelő megrendelések számára osztja ki, és az egyszerűbb gépeket terheli a kevesebb műveletet igénylő megrendelések munkáival.

A *BWBA* algoritmus formális váza a következő:

```
int _S::generate_E_O_oriented_assign( ORDER * O, int** E_O, int N_EXE, int ** R_MG )
{
//E_O groups of orders acconding to the EXE type
//initializing
 for ( int m = 1; m<=NM; m++)
   M_T[m][0] = 0;
//loading
int o, route, nstep, step, mg, m, Last_Job, i, fi;
for (int e = 1; e <=N_EXE; e++ )
 for (int ao= 1; ao<=E_O[e][0]; ao++)
{
   o = E_O[e][ao];
   route = O[o].R[O[o].R[0]];
   nstep = R_MG[route][0];
   fi = O[o].FJ;
   J_T[fi][0] = route;
   for(step=1; step<=nstep; step++)
    {
      mg = R_MG[route][step];
      m = O[o].MG_M[mg][1+random(O[o].MG_M[mg][0])];
      J_T[fi][step] = m;
      M_T[m][0]++;
      M_T[m][M_T[m][0]] = fi;
      M_Tp[m][M_T[m][0]] = step;
      J_Tp[fi][step] = M_T[m][0];
    }

          //folytatás a következő oldalon
```

```
   Last_Job = O[o].FJ + O[o].NJ - 1;
   for ( i = fi+1; i <= Last_Job;  i++ )
   {
   J_T[i][0] = J_T[fi][0];
   for(step=1; step<=nstep; step++)
    {
      m = J_T[i][step] = J_T[fi][step];
      M_T[m][0]++;
      M_T[m][M_T[m][0]] = i;
      M_Tp[m][M_T[m][0]] = step;
      J_Tp[i][step] = M_T[m][0];
    }
  }
}
return 1;
}
```

Látható, hogy ennek a felépítésnek köszönhetően az algoritmus futási ideje (a végrehajtandó lépések száma) a megrendelések számával egyenesen arányos. A kezdeti ütemterv a vizsgált legnagyobb méretű problémák esetében is a másodperc töredéke alatt elkészül. Az előállított ütemtervek rendre sokkal jobb célfüggvényértékekkel rendelkeznek, mint a véletlenszerűen választott ütemtervek, ezáltal jó kiindulási alapot biztosítanak a javító algoritmusok számára (13.6. melléklet).

Az algoritmus továbbfejlesztésével különböző célfüggvényekre „kihegyezett” speciális változatok hozhatók létre:

- Heurisztikus prioritási szabályokkal a gépenkénti munkasorrendek variálhatók (Heuristic Easy-Priority & FIFO Combined Algorithm, *HEFCA*),
- A megrendelések beütemezésekor korlátozott iteráció-számmal a különböző alternatíváknak megfelelő ütemtervrészletek szimulációs kiértékeléssel összevethetők (Heuristic Inserting Algorithm, *HIA*).

Ezen algoritmusok (BWBA, HEFCA, HIA) felépítő jellege abban nyilvánul meg, hogy a beütemezett munkák gépekhez rendelését a további munkákra vonatkozó döntések meghozatalakor már nem változtatják meg.

Az így felépített induló ütemtervek lesznek a kereső-javító tervezési fázis kiinduló objektumai. Mivel az induló ütemtervek célfüggvényértékeit az ütemező számítja és tárolja, a javítás (vagy később az esetleges újraütemezés) során bekövetkezett minden változás a menedzserindexekben nyomon követhető.

## 4.4.2. Többcélú heurisztikus keresőalgoritmus

### 4.4.2.1. A keresőalgoritmus váza

A keresési folyamat során – egy heurisztikusan felépített kezdeti ütemtervből kiindulva – megengedett módosítások ismételt végrehajtásával alakul ki a végső ütemterv. A keresés a leállási feltétel teljesüléséig tart.

Egy közbenső lépés során az algoritmus bizonyos számú kiterjesztett ütemtervet készít prioritásértékekkel vezérelt szomszédsági operátorok alkalmazásával. Ha egy kiterjesztett ütemterv szerepel a tabulistán, akkor azt nem értékeli ki, figyelmen kívül hagyja. Ha egy kiterjesztett ütemterv nem szerepel a tabulistán, akkor felkerül a tabulistára, és a legkorábban felvett listaelem törlődik. A szimulációs kiértékelést követően, ha a kiterjesztett ütemterv jobb, mint az adott kiterjesztés legjobb ütemterve, akkor megjegyzésre kerül. A kiterjesztés legjobb ütemterve lesz a következő lépés kiterjesztésének kiindulási bázisa. Ha ez legalább olyan jó megoldás, mint a keresés során már megtalált legjobb megoldás, akkor ez kerül megjegyzésre.

A keresőalgoritmus formális váza a következő:

```
_S SCH_SEARCH(ORDER* O, JOB* J, MACHINE* M, int** MG_M,
        int STEPS, int LOOPS, int min_LOOP, int MAX_LENGHT,
        int* Nx_operator_priority, int Nx_number)
        //rendelések, munkák, gépek, gépcsoportok, lépések száma, adott lépéshez tartozó iterációk max. száma,
        //a kötelező iterációk száma, a tabulista elemeinek max. száma, operátorok prioritásai, oprtáorok száma
{
OBJ_VAL obj_val;                    //az aktuális célfüggvényértékeket tároló objektum
_S s0;                              //kezdeti megoldás
s0.build_init_solution();           //felépítő heurisztikus algoritmus
s0.calc_M_STET(MG_M, O, J, M);      //feladatok végrehajtásának szimulációja
s0.calc_job_order_time_and_obj_value(obj_val, O, J);  //időadatok és célfüggvények számítása
_S s_saved(s0);                     //a kiterjesztés kiinduló megoldása
_S s_ext(s0);                       //a kiterjesztés legjobb megoldása
OBJ_VAL obj_val_ext(obj_val);       //a kiterjesztés legjobb megoldásának célfüggvényértékei
_S s_best(s0);                      //a keresés során megtalált legjobb megoldás
OBJ_VAL obj_val_best(obj_val);      //a keresés során megtalát legjobb megoldás célfüggvényértékei
TABULIST TabuList(NULL);            //korábban kiértékelt, tiltott megoldások listája
int used_operator;                  //az aktuális kiterjesztésben a legjobb szomszédot előállító operátor
int Nx_selected_operator;           //a használatra kijelölt operátor
Nx_total = new long[Nx_number+1];   //a szomszédsági operátorok kiválasztásának számértékei
Nx_improve = new long[Nx_number+1]; //a szomszédsági operátorok által megnyert kiterjesztések száma
for (int n = 0; n <= Nx_number; n++)  //a szomszédsági operátorokhoz tartozó kezdőértékek megadása
      Nx_total[n] = Nx_improve[n] = 0;
for(int step = 1; step <= STEPS; step++)  //a keresés lépésenkénti végrehajtása
{
 if ( Scheduling_stop() == true ) break;  //leállás: időkorlát elérése/ nincs javulás/ felhasználói akció
 for (int ext = 1; ext <= LOOPS; ext++ )  // kiterjesztés: adott lépéshez tartozó szomszédság vizsgálata
 {
   s0 = s_saved;                    //az aktuális kiterjesztés kiindulási megdoldásának (bázis) betöltése
   Nx_selected_operator = select_Nx_operator( Nx_number, Nx_operator_priority ); //kiválsztás
   s0.perturb( Nx_selected_operator );  //a szomszédos megoldás elkészítése a kiválasztott operátorral
   if ( TabuList.does_TL_include( s0 ) ) continue;          //tiltott megoldás esetén új szomszédot készít
   else
     {
       TABU* newtabu = new TABU( s0 );                      //az aktuális megoldás kódolása
       TabuList.add_new_TABU_into_startpos(newtabu);        //felvétel a tabulista elejére
       if (TabuList.Get_N_of_TABU() > MAX_LENGHT)           //a tabulista méretétől függően
          TabuList.delete_TABU_from_lastpos();              //a legkorábban felvett tabu törlése
     }
s0.calc_M_STET(MG_M, O, J, M);                            //feladatok végrehajtásának szimulációja
s0.calc_job_end_time_and_obj_value(obj_value, O, J); //időadatok és célfüggvények számítása
//folytatás a következő oldalon
```

```
if ( ext == 1 )                                    //az aktuális kiterjesztés első megoldása
   {
    obj_value_ext = obj_value;                     //a megoldáshoz tartozó célfüggvény-értékek másolása
    s_ext = s0;                                    //az aktuális kiterjesztés legjobb megoldása
    used_operator = Nx_selected_operator;          //a legjobb megoldást előállító operátor megjegyzése
   }
   else
   {
      if ( obj_value_ext > obj_value )             //az aktuális megoldása jobb mint
        {                                          //a kiterjesztés során eddig megtalált legjobb megoldás
         obj_value_ext = obj_value;                //a megoldáshoz tartozó célfüggvényértékek másolása
         s_ext = s0;                               //az aktuális kiterjesztés legjobb megoldásának megjegyzése
         used_operator = Nx_selected_operator;     //a megoldást előállító operátor megjegyzése
         if ( ext > min_LOOP && obj_value_ext < obj_value_best ) break;
        } //adott számú iteráció után bejeződhet a kiterjesztés ha az eddigi legjobb megoldástól is jobbat talált
   }
}
s_saved = s_ext;                                   //a kiterjesztés legjobb megoldása lesz a következő kiterjesztés bázisa
Nx_total[ used_operator ]++;                       //a kiterjesztést megnyerő operátor győzelmeinek növelése
Nx_total[ 0 ]++;                                   //az összegzett győzelmek számának növelése
if ( obj_value_ext <= obj_value_best )             //az aktuális kiterjesztés legjobb megoldása legalább olyan jó
   {                                               //mint a keresés során megtalált eddigi legjobb megoldás
     obj_value_best = obj_value_ext;               //a megoldáshoz tartozó célfüggvényértékek másolása
     s_best = s_ext;                               //a legjobb megoldás megjegyzése
     Nx_improve[ used_operator ]++;                //a legjobb megoldást adó operátor jutalmazása
     Nx_improve[ 0 ]++;                            //a javítások összesített számának növelése
   }
Actualize_priority(Nx_operator_priority, Nx_improve, Nx_total); //az operátorok értékelése
}
delete [] Nx_total; delete [] Nx_improve; //lokális változók megszüntetése

return s_best;                                     //megtalált legjobb megoldás
}
```

#### 4.4.2.2. Tabulista

A tabulista „rövid távú memóriaként” működve biztosítja a keresési folyamat előrehaladását, támogatja a lokális optimumon való áthaladást, valamint megakadályozza a közelmúltban megvizsgált megoldásokra való visszatérést és a nem kívánt ismétlődő hurkok kialakulását.

Ennek az összetett funkciónak a megvalósítására – az irodalomban általánosan javasolt megoldásoktól eltérve – azt a módszert dolgoztam ki, hogy a tabulista időrendi sorrendben, kódolt alakban, a keresési folyamat során kiértékelt *megoldásokat* tárolja. A listán nyilvántartott megoldások (ütemtervek) legnagyobb darabszámát a *MAX_LENGHT* paraméter határozza meg. A tabulistán lévő megoldások tiltottak, az aktuális kiterjesztés során már nem tekinthetők szomszédnak.

A tabulista hossza nagymértékben befolyásolja a keresési folyamatot. Ennek értékétől függ az, hogy a keresési tér bejárása során az algoritmus mennyire „mély” lokális „gödörből” képes kijutni. Mivel a keresőalgoritmus minden egyes lépésben a legjobbnak ítélt szomszédos megoldást választja, egy rosszabb megoldásról indulva („magasabban” lévő pont) a legnagyobb csökkenés mentén lefelé halad a lokális

optimum irányába („alacsonyan fekvő pont”). A lokális optimum elérése után azonban a megoldástér felett értelmezett célfüggvények által meghatározott felület palástján lentről felfelé spirálszerű pályán mozogva próbál újra lefelé vezető utat találni. Ha a szomszédos megoldások száma nagy, akkor sajnos könnyedén visszajuthat a már ismert lokális optimumba. Kellően nagyra választott (méretezett) tabulistával azonban elérhető, hogy kijusson a „dombtetőre” és annak egy másik oldalán induljon újra lefelé egyre jobb megoldások megtalálásának reményében.

A kiterjesztés (szomszédos megoldások generálása és vizsgálata) során ellenőrzésre kerül az aktuális szomszédnak jelölt megoldás tabu volta, vagyis az, hogy szerepel-e már a tabulistán. Ha a tabulista mérete indokolatlanul nagy, akkor az ellenőrzések következtében nagyméretű feladatok esetében feleslegesen megnő a futási idő. Ez elkerülhető megfelelően megválasztott méretű tabulistával, valamint az összehasonlítások idejének csökkentésével. Ez utóbbi érdekében minden egyes kiértékelt ütemtervnek rendre csak a gépenkénti feladatlistája (*M_T*) kerül átmásolásra egy erre a célra létrehozott tabu objektumba és az kerül fel a tabulistára. Egy adott megoldás ellenőrzése során, a keresőgép a tabulistán időrendben visszafelé haladva (*a legutoljára felkerült előre* elv alkalmazásával) veszi elő az elemeket. Egy adott tabuobjektum és a jelölt ütemterv összehasonlítása során először a gépekhez rendelt (aktuális terheléseket reprezentáló) gyártási feladatok darabszámai kerülnek páronként összevetésre, amely még nagyméretű feladatok esetében is nagyon gyorsan elvégezhető. Ha ezekben nincs eltérés, csak akkor kerül sor a gépenkénti feladatlisták sorrendhelyes összehasonlítására az első eltérés észleléséig, vagy teljes egyezőség estén a feladatlista végének eléréséig. Ilyen módon, az aprólékos részletekre kiterjedő egyezőségvizsgálat helyett alkalmazott szisztematikus különbségkereséssel elfogadható sebesség érhető el.

A kidolgozott módszer további előnye, hogy megakadályozza a tiltott ütemtervek időadatainak szimulációs meghatározását, és a célfüggvények felesleges kiszámítását. Az így elért időmegtakarítás jóval nagyobb az ellenőrzésre fordított időtartamnál, így ez is jelentősen hozzájárul a feladatmegoldáshoz szükséges futási idő csökkentéséhez.

#### 4.4.2.3. Szomszédsági operátorok

A keresési folyamatban a szomszédsági operátorok állítják elő az ütemezési feladat megoldásait jelentő ütemterveket. A keresőmotor a soron következő ütemtervet az aktuális ütemtervben szereplő feladatok (*J_T* kapcsolótömbbel leírt) definíciójának és/vagy a feladatok (*M_T* kapcsolótömbbel meghatározott) végrehajtási sorrendjének megváltoztatásával hozza létre.

Az igen kis méretű ütemezési feladatoktól eltekintve a lehetséges módosítások körének teljes kifejtése (leszámlálása) a feladat jellegéből következő kombinatorikus robbanás miatt kivárható idő alatt nem valósítható meg. A lehető legjobb megoldás megtalálásához nélkülözhetetlen olyan módosítóoperátorok használata, amelyek:

- egyrészt kellően finom bejárást biztosítanak, ezáltal nem „lépnek” át jó megoldásokat,

- másrészt megfelelően nagy mértékű továbblépésekkel biztosítják, hogy a nagyobb távolságok elérése a kiértékelések következtében ne váljon túlságosan lassúvá, mert az a válaszidő értékét erőteljesen megnövelné.

A módosítóoperátorokkal szemben további részben összetartozó és részben ellentmondó követelmények is támaszthatók:

- A megengedett megoldások nem feltétlenül homogén és összefüggő keresési teréből az operátorok nem vezethetnek ki.
- Az operátoroknak biztosítaniuk kell a keresési tér bármely (pl. egy elszigetelt tartományban lévő) pontjának elérését is.
- A rugalmasság szempontjából fontos, hogy a módosítóoperátorok ne függjenek túlságosan egyetlen célfüggvénytől sem, ellenkező esetben ugyanis bizonyos célfüggvényekre nagyon hatékonyaknak bizonyulhatnak, míg más célfüggvények esetében elfogadhatatlanul gyengén teljesítenének.

A felállított követelményrendszer kielégítésére többféle módosítóoperátort definiáltam. Ezek mindegyike egy adott ütemterv következetesen végrehajtott, a kemény korlátozásokat maradéktalanul kielégítő megváltoztatására képes. Egy adott ütemtervből kiindulva a módosítóoperátorok önálló vagy kombinált alkalmazásával elérhető megoldáshalmazok unióját tekintem az adott ütemterv szomszédságának.

Az operátorok definíciójában bonyolult feltételek megfogalmazása nélkül biztosított az, hogy bármely megvalósítható ütemtervből készített új ütemterv szintén megvalósítható marad, hiszen az ütemezési modellből a felépített kapcsolatrendszernek köszönhetően a lehetőségek teljes tára, és annak határai is pontosan kiolvashatók. Ennek komoly előnye az, hogy nincs szükség a kiterjesztés során előállított ütemtervek konzisztenciájának további vizsgálatára, ezáltal a futási idő nem növekszik feleslegesen.

Alapvetően megrendelés-, munka-, feladat- és géporientált operátorokat fejlesztettem ki és vizsgáltam meg. Ezek független, valamint együttes értékelését követően csak a legjobbnak ítélt szomszédsági operátorokat építettem be a kereső-algoritmusba.

A keresőoperátorok bemutatása érdekében bevezetem az *_S* névvel azonosított osztályt, amely a *DSCH* termelési finomprogramot és az azon értelmezett metódusokat fogja egységbe.

Az *_S* osztály legfontosabb adattagjai a 3.4.6. fejezetben bemutatott kapcsolótömbök alapján a következők:

- *NO*: a termelési finomprogramban szereplő megrendelések száma,
- *NJ*: a munkák száma,
- *NM*: a gépek száma,
- *J_T*: a gyártási feladatok listája,
- *M_T*: a gyártási feladatok gépenkénti végrehajtási sorrendjének listája,
- *M_STET*: a gyártási feladatok időadatainak listája,
- *J_Tp*: a munkákhoz tartozó gyártási feladatok gépenkénti végrehajtási pozíciójának listája,
- *M_Tp*: a gépekhez rendelt gyártási feladatok munkákhoz tartozó végrehajtási lépéseinek listája.

Az *_S* osztálynak a keresési feladat megoldása szempontjából fontos metódusai a következők:
Alapvető műveletek:

- Adott munka és a hozzá tartozó gyártási feladatok kiemelése az ütemtervből:
  void _S::remove_job_from_sch(int i);
- Adott munka beillesztése az ütemtervbe, a munka elvégzéséhez szükséges gyártási feladatok és azok végrehajtási sorban betöltött helyének véletlenszerű megválasztásával:
  void _S::random_value_for_i_job(int i, ORDER* O, JOB* J);
- Adott lépésszámnál (*maxN*) nem hosszabb, véletlenszerűen választott hosszúságú permutációciklus definiálása:
  int _S::generate_cycle(int size_of_c, int* c, int maxN);
- Adott permutációciklus adott sorozaton történő végrehajtása:
  void do_pi(int *c, int m);
- Adott gépen a végrehajtási sorban szereplő feladatok sorrendjének véletlenszerű permutálása:
  int _S::random_do_pi_seq_change_on_m_machine(int m);
- Egy késő munka egyenletes valószínűséggel véletlenszerűen történő kiválasztása:
  int _S::random_select_latejob(ORDER* O, JOB* J, int n_of_latejob);

A felsorolt alapvető műveletekhez kapcsolódó algoritmusok bemutatása a 13.3. mellékletben megtalálható.

Módosítóoperátorok:

***P1 operátor***: Kiemeli a paraméterében megadott munkát az ütemtervből, majd újra beilleszti azt egyenletes valószínűséggel véletlenszerűen választott útvonal, gépek és pozíciók szerint.

```
int _S:: P1(int job, ORDER* O, JOB* J)
{
   remove_job_from_sch(job);
   random_value_for_i_job(job, O, J);
   return 1;
}
```

***P2 operátor***: A paraméterként megadott gépen megváltoztatja a munkák végrehajtási sorrendjét. A géphez rendelt munkák sorozatát egy kiindulási permutációnak tekinti, majd generál hozzá egy véletlen hosszúságú permutációciklust és azt alkalmazza a kiindulási permutációra. Az eredményül kapott új permutáció határozza meg a munkák sorrendjét a gépen.

```
int _S:: P2(int machine)
{
   if ( random_do_pi_seq_change_on_m_machine(machine) ==1 ) return 1;
   else return 0;
}
```

***P3 operátor***:A paraméterként megadott, módosításra kijelölt (*mod_job*) munkát kiemeli az ütemtervből, majd beszúrja a második (*fix_job*) paraméterben referenciának tekintendő munka mögé, minden gépen rendre a közvetlenül azt követő pozícióba. Ez úgy valósul meg, hogy a módosítandó munkához hozzárendeli a fix munka útvonalát és az ahhoz tartozó gépeket, majd minden egyes érintett gép végrehajtási sorába beszúrja a módosításra jelölt munkát a fix munkát közvetlenül követő munka elé, a többi munka jobbra léptetésével.

```
int _S:: P3(int fix_job, int mod_job)
{
  remove_job_from_sch(mod_job);
   int route = J_T[fix_job][0];
   J_T[mod_job][0] = route;
   int nstep = R_MG[ route ][0];
   int machine, new_pos, shift_pos;
   for(int step=1; step<=nstep; step++)
    {
      machine = J_T[mod_job][step] = J_T[fix_job][step];
      M_T[machine][0]++;
      new_pos =  J_Tp[fix_job][step];
      for (shift_pos=M_T[machine][0]-1;
           shift_pos>0 && shift_pos>=new_pos;
           shift_pos -- )
           {
            M_T[machine][shift_pos+1] = M_T[machine][shift_pos];
            M_Tp[machine][shift_pos+1] = M_Tp[machine][shift_pos];
            J_Tp[M_T[machine][shift_pos+1]][M_Tp[machine][shift_pos+1]] =
                 shift_pos+1;
           }
      M_T[machine][new_pos] = mod_job;
      M_Tp[machine][new_pos] = step;
      J_Tp[mod_job][step] = new_pos;
    }
return 1;
}
```

***P4 operátor***: A paraméterként megadott, módosításra kijelölt (*order*) megrendeléshez tartozó munkákat kiemeli az ütemtervből, és egyetlen egységként kezelve új helyre illeszti be azokat. Ezt úgy végzi el, hogy a megrendelés első munkáját az ütemtervből való kiemelés után újra beilleszti egyenletes valószínűséggel véletlenszerűen választott útvonal, gépek és pozíciók szerint (*P1* operátor). Ezt követően a megrendelés összes többi munkáját az első munka mögé (a *P3* operátor által meghatározott módon) helyezi el.

```
int _S::P4(int order, ORDER* O, JOB* J)
{
   int first_job = O[order].FJ;
   int next_job:
   remove_job_from_sch(first_job);
   random_value_for_i_job(first_job, O, J);
   if ( O[order].NJ>1 )
   for ( next_job = first_job+O[order].NJ-1;
         next_job > first_job;
         next_job--)
     P3(first_job, next_job);
 return 1;
}
```

Szomszédsági operátorok:

***N1 operátor***: Az ütemtervben szereplő rendelések közül egyenletes valószínűséggel véletlenszerűen kiválaszt egy megrendelést. Kiemeli az ütemtervből a teljes megrendelést (a hozzá tartozó munkák összességét), majd megbontás nélkül új helyre illeszti be.

```
void _S::N1(ORDER* O, JOB* J)
{
        int order = 1 + random(NO);
        P4(order, O, J);
}
```

***N2 operátor***: Az ütemtervben szereplő határidőt túllépő rendelések közül egyenletes valószínűséggel véletlenszerűen kiválaszt egy késő megrendelést. Kiemeli az ütemtervből a teljes megrendelést (a hozzá tartozó munkák összességét), majd megbontás nélkül új helyre illeszti be.

```
void _S::N2(ORDER* O, JOB* J, int n_of_latejob)
{
          int job = random_select_latejob(J, O, n_of_latejob);
          int order = J[job].OID;
          P4(order, O, J);
}
```

***N3 operátor***: Ez a két módosítást összevontan végrehajtó operátor az első lépésben az ütemtervben szereplő határidőt túllépő munkák közül egyenletes valószínűséggel véletlenszerűen kiválaszt egy késő munkát. Kiemeli az ütemtervből a munkát a hozzá tartozó gyártási feladatok összességével. Egyenletes valószínűséggel véletlenszerűen választott útvonal, valamint gépek hozzárendelésével újradefiniálja a munka elvégzéséhez szükséges gyártási feladatokat, és az adott gépeken szintén véletlenszerűen választott pozíciókba beilleszti. Második lépésként egy véletlenszerűen kiválasztott gépen megváltoztatja a gyártási feladatok végrehajtási sorrendjét.

```
void _S::N3(ORDER* O, JOB* J, int n_of_latejob)
{
        int job = random_select_latejob(J, O, n_of_latejob);
        P1(job, O, J);
        int machine = 1 + random(NM);
        P2(machine);
      }
```

***N4 operátor***: Az *N4* operátor működése megegyezik az *N3* operátor első lépésével, vagyis az ütemtervben szereplő határidőt túllépő munkák közül kiválaszt egy késő munkát és kiemeli az ütemtervből. Ezt követően újradefiniálja a munka elvégzéséhez szükséges gyártási feladatokat és a kiválasztott gépeken véletlenszerűen választott pozíciókba illeszti be azokat.

```
void _S::N4(ORDER* O, JOB* J, n_of_latejob)
{
         int job = random_select_latejob(J, O, n_of_latejob);
        P1(job, O, J);
}
```

***N5 operátor***: Az *N5* operátor működése hasonló az *N3* operátor működéséhez, a különbséget csak az jelenti, hogy itt az ütemtervben szereplő összes munka közül választ egy munkát és azt emeli ki az ütemtervből és ahhoz választ új útvonalat, gépeket és jelöl ki ezeknek megfelelően végrehajtási pozíciókat.

```
void _S::N5(ORDER* O, JOB* J)
{
      int job = 1 + random(NJ);
      P1(job, O, J);
}
```

#### 4.4.2.4. Szomszédsági operátorok általános célú alkalmazása

A bemutatott szomszédsági operátorok különböző feladatváltozatok esetében önmagukban is alkalmasak a keresési feladat megoldásterének feltérképezésére és megfelelő minőségű eredmény megtalálására elfogadható idő alatt. Azonban az EFFS ütemezési feladatosztály sokszínűsége – amely egyrészt az általánosan megfogalmazott építőelemek (erőforrások, munkák) egymáshoz való viszonyának, másrészt az időben eltérő fontosságú, változatos célfüggvények együttes alkalmazásának következménye – igényli, hogy a keresőalgoritmus széles problémakörben általános céllal is alkalmazható legyen.

A keresési folyamatban egy adott kiterjesztés során a következő szomszédos ütemterv elkészítésére eredményesen használható szomszédsági operátor kiválasztása az operátorok aktuális teljesítőképességére jellemző prioritásértékekkel egyenesen arányos valószínűséggel történik meg a következő algoritmus szerint:

```
int select_Nx_operator( int Nx_number, int * Nx_operator_priority )
{             //Nx_number: a szomszédsági operátorok száma
              //Nx_operator_priority:
              //a szomszédsági operátorok aktuális prioritásértékeit leíró vektor

              if (Nx_number ==1 ) return 1;

               int pri_sum = 0;
               for (int sj = 1; sj <= Nx_number; sj++)
                    pri_sum += Nx_operator_priority[sj];

               int random_i = 1 + random(pri_sum);
               int fix = 0;
               int op = 1;

                for (int sj = 1; sj <= Nx_number; sj++)
               {
                 fix += Nx_operator_priority[sj];
                 if ( fix >= random_i )
                    {
                     op = sj;
                     break;
                     }
               }

return op;    //a kiválasztott operátor sorszáma
}
```

Az operátorok prioritásai bizonyos lépésszám után frissítésre kerülnek. Minden egyes operátorhoz tartozó konkrét prioritásérték beállításának alapját egyrészt az adott lépésszámra vonatkoztatott megnyert kiterjesztések (vagyis az adott operátor által legjobb megoldást adó kiterjesztések) száma, másrészt a kiválasztások számának és a javuló módosítások számának aránya alapján felállított sorrend határozza meg.

A bemutatott szomszédsági operátorok együttes dinamikus alkalmazása és azok aktuális teljesítőképességének önálló számszerű értékelése egy rugalmas, prioritásértékekkel vezérelt szomszédságkiválasztási módszert eredményez, amely a keresési folyamat megfelelő irányításával nagymértékben hozzájárul az aktuális feladathoz való rugalmas alkalmazkodás biztosításához, a gyors és hatékony feladatmegoldáshoz.

Az elvégzett vizsgálatok eredményei – amelyeknek néhány jellemző részlete a 13.6. mellékletben található – azt mutatják, hogy a javasolt megoldási módszer nagyméretű feladatok esetében is hatékonyan alkalmazható, rugalmasan alkalmazkodik az aktuális célfüggvényekhez és rövid idő alatt szolgáltatja az eredményt.

# 5. ÚJRAÜTEMEZÉSI FELADATOK MEGOLDÁSA

## 5.1. Újraütemezési feladatok megfogalmazása

A termelés finomprogramozása a gyártásirányítás rövid távú tervezési és irányítási feladatának megvalósítását jelenti, melynek során adott a termelési rendszer aktuális állapota, a rendszerben fennálló érvényes korlátozások (beleértve a véges technológiai kapacitásokat, a rendelkezésre állásokat, a műveletek sorrendi előírásait és egyéb feltételeket is). Az előidejű (prediktív) ütemezési feladatban a gyártásra kiadott belső rendelések megvalósításához gyártási munkákat és műveleteket kell definiálni, majd ezek elvégzéséhez alkalmas gyártási erőforrásokat, valamint indítási és befejezési időpontokat kell tervezett módon úgy hozzárendelni, hogy a definiált korlátozások teljesüljenek, és a termelés teljesítményét mérő mutatók legalább kvázi optimálisak legyenek, azaz a menedzsment magasabb szintjén megfogalmazott célok megvalósuljanak.

Az előidejű ütemezés során, az ütemezés időpontjában rendelkezésre álló – részben a jelenlegi, részben a közlejövőben várható rendszerállapotra vonatkozó – ismeretek determinisztikus alkalmazásával megtervezett kvázi optimális termelési finomprogram önmagában nem garantálja az ütemezett időszakban a termelési folyamat optimális lefolyását. A fizikai és az üzleti folyamatok elkerülhetetlen sztochasztikája miatt szükség van a gyártás valós idejű folyamatos irányítására. Az irányítás során számos korrekciós beavatkozási döntést kell hozni. Számos szituáció szükségessé teszi a termelési folyamatok érvényes ütemezésének felfüggesztését, a folyamatok újraütemezését. Fontos követelmény lehet, hogy a termelés finomprogramozására szolgáló alkalmazás az irányítási feladat fontos összetevőjeként az újraütemezési feladatok megoldását is foglalja magába.

A termelés újraütemezése megfogalmazható a „viselkedés alapú” gyártásirányítás olyan beavatkozási folyamataként, amelynek során az elindított és végrehajtás alatt álló termelési finomprogram aktualizálásra, korrigálásra, újratervezésre kerül. Az újraütemezés indoka: reagálás a nem várt termelési és/vagy üzleti események fellépése nyomán kialakult, az eredetileg tervezett rendszerállapottól a megengedhetőnél nagyobb mértékben eltérő tényleges rendszerállapotra és annak várható következményeire.

Ebben a szemléletben a feladatmegoldást, a gyártásirányítási tevékenységet támogató *beavatkozási folyamat* elnevezés arra utal, hogy az előidejű ütemezési feladat

megoldására kidolgozott tervezési módszer kiterjesztésével a speciális követelményeket támasztó újraütemezési feladatok is hatékonyan támogathatók.

A korszerű gyártásirányítási rendszer többszörösen visszacsatolt, zárt hurkos szabályozás. A gyártási folyamat zavarainak elhárítása elsődlegesen a korszerű gépvezérlők (CNC, PLC, ROC) feladata. A második beavatkozási szint a művezetés, a gépbeállítások (setup) szintje. Ha a tervezett és a tényleges termelési mutatók (állapotjelzők) eltérése így sem korrigálható, és meghatározott limiteket meghalad, az újraütemezés elkerülhetetlen.

## 5.2. Az újraütemezési feladat speciális követelményei

Az újraütemezés általános célú, ugyanakkor hatékony megvalósítása érdekében szükséges az előidejű ütemezési feladat kiterjesztését elvégezni új korlátozások és új célok bevezetésével.

A probléma megoldásterét ezek az új feltételek szűkítik, ugyanakkor járulékos nehézséget okoznak a speciális gyártásirányítási követelmények (pl. gépátállítási, anyagellátási, minőségbiztosítási, logisztikai stabilitási igények) kielégítésére felállított új célfüggvények.

Ideálisnak tekinthető üzemszerű körülmények között a gyártórendszer állapota az ütemezett időhorizont bármely időpontjában pontosan megegyezik a tervezett állapottal. A gyakorlatban azonban számos váratlan esemény zavarhatja meg a termelés normális menetét, melyek következtében szükségessé válhat felülvizsgálni és indokolt esetben megváltoztatni az érvényes ütemtervet. Valós körülmények között gyakran előforduló váratlan események lehetnek a következők:

- A rendelésállomány megváltozása bizonyos megrendelések törlésének és/vagy új megrendelések beillesztésének hatására.
- Konkrét megrendelések prioritásának, határidejének, megrendelt termékek mennyiségének megváltozása.
- Az erőforrás-környezet megváltozása váratlan gépkiesés, gépek munkarendjének (kalendárium) megváltozása következtében.
- Bizonyos gépek átállítási időadatainak, selejtarányának, termelési sebességeinek és egyéb jellemző képességeinek tartós vagy átmeneti módosulása.
- Egyes gyártási feladatok végrehajtásával kapcsolatos váratlan események bekövetkezése (pl. szerszám, készülék, egyéb berendezés hiánya esetleg hibája miatt az előkészületi időadatoknak, vagy komponensek és anyagok késése következtében a műveleti időadatoknak a tervezettől való jelentős eltérése folytán stb.).
- Termelési célokat előíró célfüggvények vagy azok prioritásainak megváltozása.

Egy újraütemezési feladatban az előidejű ütemezési feladattal ellentétben eleve adott kezdeti feltételként (input adat) egy érvényben lévő (összehasonlítási alapnak tekintendő) termelési finomprogram, ami leírja a termelési folyamat elvárt menetét,

valamint ismertek az ehhez tartozó tervezett teljesítménymutatók. Adott továbbá az aktuális helyzetet tükröző származtatott termelési finomprogram, amelyben az előírttól eltérő, ténylegesen bekövetkezett állapotváltozások sorozatának megfelelő időadatok szerepelnek.

Fontos új követelményként fogalmazható meg, hogy az érvényben lévő ütemtervnek az újraütemezés időpontja előtt végrehajtott, vagy elkezdett része a feladatmegoldás során már nem változtatható meg (mert a múltban történt). Ugyanez a változtatásra vonatkozó tilalom érvényes az újraütemezés elkezdésének időpontja és a módosított ütemterv életbeléptetésének időpontja közötti időintervallumra is. Ezek az elkezdett és/vagy már befejezett gyártási feladatok, munkák és megrendelések hatással vannak a jövőre is. Például adott útvonalon már elindított munkák hátralévő gyártási feladatai a kezdeti erőforrás-alternatívák nem mindegyikére tehetők át, ilyen értelemben a beavatkozási lehetőségek időben előrehaladva csökkennek.

Az újraütemezés során a módosított termelési finomprogram minősítésének szempontjai között megjelenhetnek olyan új elemek is, amelyek kifejezik az ütemterv-változásokkal szemben támasztott speciális igényeket. Ilyenek lehetnek az egyedi (pl. bizonyos gépek átállítására, rendelések késésének engedélyezésére vonatkozó) illetve az összesített (pl. megváltozott allokációk vagy indítási sorrendek számának minimalizálására vonatkozó), valamint az egyéb termelési jellemzők változására vonatkozó elvárások.

## 5.3. Az ütemezési módszer kiterjesztése

Az 4.1. fejezetben bemutatott termelésprogramozási koncepciót kellően rugalmasnak találtam ahhoz, hogy az újraütemezés során fellépő további elvárásoknak is magas szinten megfeleljen.

### 5.3.1. A szimulációs modell általánosítása

A szimulációra alapozott problématér-transzformáció továbbra is alkalmazható azzal a módosítással, hogy az újraütemezés során a szabálybázisra épített algoritmus a már rögzített (a gyártási folyamat közben ismerté vált tényleges) időadatokat veszi figyelembe, ezáltal az egyes gyártási feladatok esetében már csak az ismeretlen időadatokat számítja ki.

### 5.3.2. A célfüggvények körének kibővítése

Az alkalmazott többcélú optimalizálási modellbe könnyedén beilleszthetők az új követelmények, megfelelően megválasztott célfüggvények formájában. Ezáltal a termelési folyamat elvárt minőségét definiáló célfüggvények és az újraütemezés speciális követelményeit megfogalmazó célfüggvények együttesen minősíthetik a módosított termelési finomprogramot. A különböző célfüggvények fontosságának

rugalmas kifejezésére a prioritásértékek hozzárendelésének technikája itt is használható.

Újraütemezés során előnyösen használható, mennyiségi és minőségi mutatókat logikai feltételekkel kombináló célfüggvényeket fogalmaztam meg és építettem be a modellbe. Ilyenek például a következők:

1. Az eredetitől eltérő útvonal bejárására ütemezett munkák száma.
2. Az eredetitől legalább egy eltérő párhuzamos gépre ütemezett munkák száma.
3. Az eredetileg tervezett befejezési státus (késő vagy nem késő) szempontjából megváltoztatott munkák száma.
4. Az eredetileg határidőre elkészülőnek tervezett, azonban az újraütemezés következtében késővé váló munkák száma.
5. Azoknak a megrendeléseknek a száma, amelyeknek van legalább egy az 1. pontban leírt feltételnek megfelelő munkája.
6. Azoknak a megrendeléseknek a száma, amelyeknek van legalább egy a 2. pontban leírt feltételnek megfelelő munkája.
7. Az eredetileg tervezett befejezési státus (késő vagy nem késő) szempontjából megváltoztatott megrendelések száma.
8. Az eredetileg határidőre elkészülőnek tervezett, azonban az újraütemezés következtében késővé váló megrendelések száma.
9. Az átállítások szempontjából megváltoztatott munkarendű gépek száma.
10. Új, eredetileg nem tervezett átállítással kibővített munkarendű gépek száma.

Az újraütemezés során tehát speciális célfüggvények állnak rendelkezésre, amelyek az eredetileg tervezett termelési finomprogram módosításának mértékét számszerűsítik, és ezek mint minimalizálandó célfüggvények jelennek meg a hagyományos célfüggvények mellett. A használható függvények készlete tovább bővíthető.

Az irodalomban az ilyen, bővített feltételrendszert kielégítő ütemezési feladatot *konzervatív* ütemezésnek nevezik. A konzervatív jelző arra utal, hogy a módosított ütemtervnek meg kell őriznie az eredeti ütemterv számos tulajdonságát.

A fentebb megfogalmazott speciális célfüggvények kiértékelésével kapott függvényértékeket – a korábban 3.4.7. fejezetben alkalmazott formalizmusnak megfelelően – beillesztettem a modell *OBJ_VAL* objektumába a következő módon:

*OBJ_VAL.F[k]* a *k*-adik célfüggvénynek a vizsgált termelési finomprogramra vonatkozó függvényértéke,

*k* a célfüggvény sorszáma *(k = 1,..., $N_K$)*,

*OBJ_VAL.DF[dk]* a *dk*-adik új célfüggvény – a vizsgált termelési finomprogram és az újraütemezés előtt érvényes finomprogram eltérésének mértékét kifejező – kiszámított értéke,

*dk* az új célfüggvény sorszáma *(dk = 1,..., $N_{DK}$)*,

Az új célfüggvényekhez tartozó aktuális fontosságot kifejező prioritásértékek a korábbi célfüggvényekhez hasonló módon jelennek meg a modellben:

*F_Pri[k]* a *k*-adik célfüggvény fontosságát kifejező nemnegatív prioritásérték.

*DF_Pri[dk]* a *dk*-adik új célfüggvény fontosságát kifejező nemnegatív prioritásérték.

### 5.3.3. Az újraütemezési feladat megoldása keresőalgoritmussal

Az új termelési finomprogram elkészítése az eredetileg érvényben lévő finomprogram végrehajtásából származó, az aktuális szituációt leíró finomprogramból kiindulva megfelelő módon végrehajtott elemi módosítások sorozatával elvégezhető. Ennek köszönhetően az előidejű ütemezésre használt többcélú keresési technika az újraütemezési feladatok megoldására is használható.

A keresőalgoritmus változtatás nélkül alkalmas lenne a még el nem indított munkák tervezett feladatainak módosítására, valamint az esetleg időközben megjelenő új munkák feladatokra bontására, azonban a definiált feladatok gépenkénti sorrendjének meghatározása már további korlátfeltételek kielégítését követeli meg. Újraütemezéskor ugyanis figyelembe kell venni, hogy a kiindulási ütemtervnek a múltban bekövetkezett része, valamint az új ütemterv elkészítéséhez és indításához szükséges időintervallumba eső része nem módosítható, ezáltal a gépeknek ezekbe az intervallumokba eső feladatait nem szabad a keresés során megváltoztatni.

További korlátozások jelentkeznek a már megkezdett munkák kezelésével kapcsolatban is. Ennek az oka az, hogy az eredetileg végrehajtásra kiadott ütemterv elindításától az új ütemterv életbe lépéséig terjedő váratlan eseményekkel terhelt időszakban végrehajtott vagy megkezdett gyártási feladatok aktuális állapota befolyásolja az érintett munkák és megrendelések el nem végzett feladataihoz hozzárendelhető gépek körét. Az újraütemezés során emiatt a keresési folyamatban használt módosítóoperátorok működésével szemben támasztott – az előidejű ütemezéskor a 4.4.2.3. fejezetben részletesen tárgyalt – hatékonysági és konzisztencia-elvárások betartása jelenti a legnagyobb kihívást.

#### 5.3.3.1. Zárolási technikák

Az újraütemezés sebességének növelése érdekében a tiltott módosítások elkerülését nem futásidőben kiértékelendő feltételek biztosítják, hanem – az előidejű ütemezés koncepciójában is alkalmazott elveknek megfelelően – az újraütemezés megkezdése előtt feltérképezésre kerülnek a megengedett alternatívák, és ezek rendszerezett formában beépülnek a modell kapcsolatrendszerébe. Ezáltal a döntési helyzetekben a választható alternatívák között csak az új feltételeket is maradéktalanul kielégítő lehetőségek jelennek meg. Az ütemezési modellben a speciális újraütemezési korlátozások ilyen irányú kibővítése hatékonyan megvalósítható zárolási technikák kombinált alkalmazásával. A szűkítő feltételek egyaránt vonatkozhatnak munkákra, megrendelésekre és gépekre is.

### Munkák zárolása

Az ütemezési modellben szereplő munka objektum attribútumainak kibővítésével indokolt olyan hozzárendeléseket is megfogalmazni, amelyek lehetővé teszik a munkák szintjén az alkalmazható gépek szűkített halmazának definiálását, és a nem módosítható feladatok leírását.

Ennek érdekében az $i$ $(i = 1,\ldots, N_J)$ munkákhoz hozzárendelem a következő zárolást leíró attribútumokat:

*J[i].LFS* a munka utolsó zárolt végrehajtási lépésének sorszáma. Nem zárolt munka esetén ennek értéke 0.

*J[i].FJ* a munka zárolásának módját definiáló objektum *(FREEZE_JOB)*:

*FJ.enabled_MG_M[mg][am]* a zárolt munka teljesítésére használható gépek halmaza gépcsoportok szerint rendezve:

$mg = 1,\ldots, N_{MG}$; a gépcsoportok,

*am = 1,..., J[i].FJ.enabled_.MG_M[mg][0]*; a használható párhuzamos gépek indexei.

*J[i].FJ.enabled_MG_M[mg][am]* az *i* munka teljesítéséhez az *mg* gépcsoportból használható *am*-edik párhuzamos gép.

*FJ.enabled_R[ar]* a szűkített gépállomány felhasználásával szervezhető megengedett végrehajtási útvonalak.

*ar = 1,..., J[i].FJ.enabled_R[0]*; a használható útvonalak indexei.

*J[i].FJ.enabled_R[ar]* az *i* munka teljesítéséhez használható *ar*-edik végrehajtási útvonal.

### Megrendelések zárolása

Mivel a zárolások a munkák szintjén egyértelműen definiáltak, ezáltal minden $o$ $(o = 1,\ldots, N_O)$ megrendelés zárolt státusa a hozzá tartozó munkák státusából származtatható:

*O[o].FO* ha az *o* megrendeléshez tartozó munkák között van zárolt, akkor értéke 1, ellenkező esetben értéke 0.

Ennek a mezőnek a jelzésértéke fontos szerepet játszik az újraütemezés során, segítségével számos ismétlődő feltételvizsgálat elkerülhető a módosítóoperátorok működése közben.

### Gépek zárolása

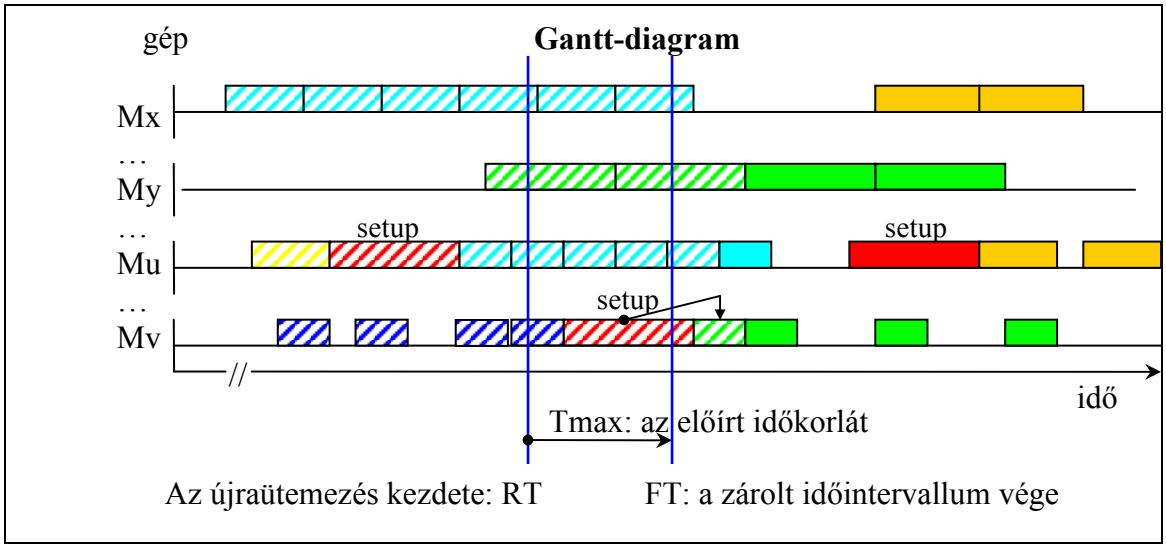
Az ütemezési feladat megoldása során elkészítendő új ütemterv szempontjából a gépeknek már teljesített feladatait, amelyeket az érvényben lévő ütemterv és a váratlan események együttesen határoztak meg, az újraütemezés során nem szabad módosítani. Így minden $m$ $(m = 1,\ldots, N_M)$ gépen az adott időszakra korábban beütemezett (befejezett vagy vérehajtás alatt álló) nem módosítható feladatok listája zárolás alá kerül, hozzárendelve a géphez a következő attribútumot:

*M[m].LFP* az *m* gép végrehajtási sorában az utolsó zárolt feladat sorszáma.

### A zárolást végző algoritmus

Az újraütemezés megkezdésének időpontja (*RT*) és a gyártásirányítási rendszer reakcióképességétől függően megválasztott, az újraütemezési feladat megoldására és az új ütemterv indításának előkészítésre fordítható időtartam (*Tmax*) együttesen – a 13.

ábrán vázolt módon – határozza meg a még érvényben lévő ütemterv zárolási időpontját (*FT*). Az újraütemezés elkezdésének időpontjában ezt az *FT* zárolási időpontot felhasználva pontosan meghatározhatók az érintett munkák, megrendelések és gépek. Ezeket az adatokat felhasználva megfogalmazhatók a jövőre vonatkozó korlátozások.




**13. ábra: Zárolási időpont értelmezése (FT)**

Egy adott munka vizsgálatát, majd a feltételek kiértékelésétől függően – az érintett megrendeléssel együtt – történő zárolását, valamint ezek következtében kialakuló korlátozott alternatívarendszer feltérképezését a következő algoritmus végzi:

```
void _S::freeze_job_order(int j, ORDER* O, JOB* J, MACHINE* M, long FT )
{
 int* enabled_R = J[j].FJ.enabled_R;
 int** enabled_MG_M = J[j].FJ.enabled_MG_M;
 enabled_R[0] = 0;
 J[j].LFS = 0;
 for (int mg = 1; mg<=N_MG; mg++)
    enabled_MG_M[mg][0] = 0;

 int order = J[j].OID;
 int route = J_T[j][0];
 int nstep = R_MG[route][0];
 int machine, position, step;
 int last_freeze_mg = 0;
 for (step = 1; step<= nstep; step++ )
 {
  machine = J_T[j][step];
  position = J_Tp[j][step];
  if ( M_STET[machine][position].ST <= FT )
       {
        O[order].FO = 1;
        last_freeze_mg = M[machine].ES;
        enabled_MG_M[ last_freeze_mg ][ 1 ] = machine;
        enabled_MG_M[ last_freeze_mg ][ 0 ] = 1;
        J[j].LFS = step;
       }
 }        //folytatás a következő oldalon
```

```
 if ( last_freeze_mg == 0 )
 {
  for ( int r = O[order].R[0]; r >= 0; r--)   //minden útvonal engedélyezett
  enabled_R[r] = O[order].R[r];                //nincs zárolás

  for ( int mg = 1, amg; mg <= N_MG; mg++ )
   for ( amg = O[order].MG_M[mg][0]; amg>=0; amg--)
      enabled_MG_M[mg][amg] = O[order].MG_M[mg][amg];
 }
 else
 {
  for ( int mg = 1, amg; mg <= N_MG; mg++ ) //van korlátozás
   if ( EndStep_of_MG( last_freeze_mg ) < StartStep_of_MG( mg ) )
      for ( amg = O[order].MG_M[mg][0]; amg>=0; amg--)
         enabled_MG_M[mg][amg] = O[order].MG_M[mg][amg];

   //megengedett útvonalalternatívák feltérképezése
    int nr = 0;
    int candidate_route;

    for (int pr=1, pr_ok=1, mg_ok=0; pr<=O[order].R[0]; pr++)
    {
       pr_ok = 1;
       candidate_route = O[order].R[pr];

       for (int mg = 1; mg <= R_MG[ candidate_route ][0]; mg++)
       {
          if ( enabled_MG_M[ R_MG[candidate_route][mg] ][0] > 0)
                mg_ok = 1;
          else
               {
                mg_ok = 0;
                pr_ok = 0;
                break;
                }
      }

       if (pr_ok == 1)
         {
          nr++;
          enabled_R[0]=nr;
          enabled_R[nr]= candidate_route;
         }
    }
  }
}
```

Az újraütemezés megkezdése előtt a munkákra, megrendelésekre és gépekre vonatkozó zárolásokat a következő algoritmus végzi el:

```
void _S::freeze_jobs_orders_and_machines( ORDER* O, JOB* J, MACHINE* M, long FT )
{
int examine_order;
int examine_job;
int examine_machine;
int position;
for (examine_order = 1; examine_order <= NO; examine_order++)
   O[examine_order].FO = 0;

for (examine_job = 1; examine_job <= NJ; examine_job++)
  {
  J[examine_job].FJ = new FREEZE_JOB(O[J[examine_job].OID].MG_M);
  freeze_job_order(examine_job, O, J, M, FT);
  }

for ( examine_machine = 1; examine_machine <= NM; examine_machine++)
  {
   M[examine_machine].LFP = 0;
   for (position = 1; position <= M_T[examine_machine][0]; position++)
      if ( M_STET[examine_machine][position].ST <= FT )
        M[examine_machine].LFP = position;
      else
        break;
  }
}
```

### 5.3.3.2. Továbbfejlesztett szomszédsági operátorok

A zárolásokra alapozva az újraütemezés speciális korlátozásai betarthatók. Ebből az következik, hogy az előidejű ütemezésre használt szomszédsági operátorok megfelelő továbbfejlesztésével a többcélú heurisztikus keresőalgoritmus képes az újraütemezési feladat megoldására.

A továbbfejlesztett algoritmusokat a már ismertetett *_S* osztályhoz rendelt új metódusok formájában valósítottam meg:

Zárolásokat nem sértő elemi műveletek:

- Paraméterben adott, részlegesen zárolt munka beillesztése részben zárolt környezetbe:
  void _S::random_value_for_i_FREEZE_job(int i, ORDER* O, JOB* J, MACHINE* M, _S* s_init)
- Paraméterben adott zárolásmentes munka beillesztése részben zárolt környezetbe:
  void _S::random_value_for_i_job_in_FREEZE_env(int i, ORDER* O, JOB* J, MACHINE* M)
- Paraméterben adott, részben zárolt géphez tartozó feladatsorrend megváltoztatása véletlen hosszú permutációciklus alkalmazásával:
  int _S::random_do_pi_seq_change_on_m_machine_FREEZE(int m, MACHINE* M)

Ezeknek az újraütemezéshez használt elemi műveleteknek az algoritmusait a 13.4. melléklet tartalmazza.

Módosítóoperátorok:

***P1_FREEZE operátor***: Kiemeli a paraméterében megadott munkát az ütemtervből, majd újra beilleszti egyenletes valószínűséggel véletlenszerűen választott útvonal, gépek és pozíciók szerint az érvényben lévő zárolások megsértése nélkül.

```
int _S:: P1_FREEZE(int job, ORDER* O, JOB* J, MACHINE* M, _S * s_init)
{
    if ( J[job].LFS > 0 )
     {
       remove_job_from_sch(job);
       random_value_for_i_FREEZE_job(job, O, J, M, s_init);
     }
   else
     {
      remove_job_from_sch(job);
      random_value_for_i_job_in_FREEZE_env(job, O, J, M);
     }
    return 1;
}
```

***P2_FREEZE operátor***: A paraméterként megadott gépen megváltoztatja a munkák végrehajtási sorrendjét, hasonlóan az előidejű ütemezés során használt *P2* operátorhoz, azzal a különbséggel, hogy a géphez tartozó zárolt munkákat nem módosítja.

```
int _S:: P2_FREEZE(int machine, MACHINE* M)
{
      if ( random_do_pi_seq_change_on_m_machine_FREEZE(machine, M)==1 )
            return 1;
      else
            return 0;
}
```

***P3_FREEZE operátor***: Abban az esetben, ha a paraméterekben megadott mindkét munka zárolásmentes, a módosításra kijelölt (*mod_job*) munkát kiemeli az ütemtervből, majd beszúrja a második (*fix_job*) paraméterben referenciának tekintendő munka mögé, minden gépen rendre a közvetlenül azt követő pozícióba az előidejű ütemezésnél (4.4.2.3. fejezetben) ismertetett *P3* operátor alkalmazásával.

```
int _S::P3_FREEZE(int fix_job, int mod_job, ORDER* O, JOB* J)
{
    if ( J[fix_job].LFS == 0 && J[mod_job].LFS == 0)
     {
      P3(fix_job, mod_job);
      return 1;
     }
 return 0;
}
```

***P4_FREEZE operátor***: A paraméterként megadott, módosításra kijelölt (*order*) megrendeléshez tartozó első munkát kiemeli az ütemtervből, majd újra beilleszti egyenletes valószínűséggel véletlenszerűen választott útvonal, gépek és pozíciók szerint (*P1_FREEZE* operátor) a zárolások megsértése nélkül. Ezt követően a megrendelés összes többi munkáját – mind a munkák, mind a gépek zárolását figyelembe véve – beszúrja az első munka mögé az első megengedett pozicióba.

```
int _S::P4_FREEZE(int order, ORDER* O, JOB* J, MACHINE* M, _S * s_init)
{
   int first_job = O[order].FJ;
   int next_job;
-- P1_FREEZE( first_job, O, J, M, s_init)
   if ( O[order].NJ > 1 )
   for ( next_job = first_job + O[order].NJ - 1;
         next_job > first_job;
         next_job--)
    if ( J[next_job].LFS > 0 )
     {
       remove_job_from_sch(next_job);
       random_value_for_i_FREEZE_job(next_job, O, J, M, s_init);
     }
    else
     {
       if ( P3_FREEZE(first_job, next_job, O, J) == 0 )
           {
            remove_job_from_sch(next_job);
            random_value_for_i_job_in_FREEZE_env(next_job, O, J, M);
           }
     }

 return 1;
}
```

Szomszédsági operátorok:

***N1_FREEZE operátor***: Az ütemtervben szereplő rendelések közül egyenletes valószínűséggel véletlenszerűen kiválaszt egy megrendelést. Kiemeli az ütemtervből a teljes megrendelést (a hozzá tartozó munkák összességét), majd a zárolásoktól függően a lehető legkisebb mértékű megbontással új helyre illeszti be.

```
void _S::N1_FREEZE(ORDER* O, JOB* J, MACHINE* M, _S* s_init)
{
        int order = 1 + random(NO);
        P4_FREEZE(order, O, J, M, s_init);
}
```

***N2_FREEZE operátor***: Az ütemtervben szereplő határidőt túllépő rendelések közül egyenletes valószínűséggel véletlenszerűen kiválaszt egy késő megrendelést. Az előző operátorral azonos módon kiemeli az ütemtervből a teljes megrendelést, majd a zárolásoktól függően a lehető legkisebb mértékű megbontással új helyre illeszti be.

```
void _S::N2_FREEZE(ORDER* O, JOB* J, MACHINE* M, _S* s_init, int n_of_latejob)
{
        int job = random_select_latejob(O, J, n_of_latejob);
        int order = J[job].OID;
        P4_FREEZE(order, O, J, M, s_init);
}
```

***N3_FREEZE operátor***: Két módosítást összevontan végrehajtó operátor, amely az érvényben lévő zárolások megsértése nélkül egyrészt adott munkát újraütemez, majd pedig adott gép feladataitnak sorrendjét rendezi át.

```
void _S::N3_FREEZE(ORDER* O, JOB* J, MACHINE* M, _S* s_init, int n_of_latejob)
{
      int job = random_select_latejob(O, J, n_of_latejob);
      P1_FREEZE(job, O, J, M, s_init);
      Int machine = 1 + random(NM);
      P2_FREEZE(machine, M);
}
```

***N4_FREEZE operátor***: Az operátor működése megegyezik az előző *(N3_FREEZE)* operátor első lépésével.

```
void _S::N4_FREEZE(ORDER* O, JOB* J, MACHINE* M, _S* s_init, int n_of_latejob)
{
      int job = random_select_latejob(O, J, n_of_latejob);
      P1_FREEZE(job, O, J, M, s_init);
    }
```

***N5_FREEZE operátor***: Az operátor működése hasonló az (*N3_FREEZE)* operátor működéséhez, a különbséget csak az jelenti, hogy itt az ütemtervben szereplő összes munka közül választ ki egy munkát módosításra.

```
void _S::N5_FREEZE(ORDER* O, JOB* J, MACHINE* M, _S* s_init)
{
    int job = 1 + random(NJ);
    P1_FREEZE(job, O, J, M, s_init);
}
```

### 5.3.3.3. Az újraütemező algoritmus váza

Az újraütemezési feladat megoldása érdekében továbbfejlesztett többcélú heurisztikus keresőalgoritmus egyszerűsített váza – az előző fejezetekben bemutatott módszereket felhasználva – a következőképpen írható le:

```
_S RESCH_SEARCH(SHOP& Shop_Before_Resch, SHOP& Shop_Released, int** MG_M,
        long freeze_time, int STEPS, int LOOPS, int min_LOOP, int MAX_LENGHT,
        int* Nx_operator_priority, int Nx_number)
        //aktuális állapot, tervezett állapot, gépcsoportok, zárolás időpontja, lépések száma, iterációk max. száma,
        //a kötelező iterációk száma, a tabulista elemeinek max. száma, operátorok prioritása, operátorok száma
{
_S s_init(Shop_Before_Resch.DSCH);  //a gyártóműhely aktuális állapotát leíró termelési finomprogram
ORDER O(Shop_Before_Resch.O);       //az aktuális  rendelések és azok attribútumai
JOB J(Shop_Before_Resch.J);         //az aktuális  munkák és azok attribútumai
MACHINE M(Shop_Before_Resch.M);     //az erőforrások aktuális jellemzői
OBJ_VAL obj_val(Shop_Before_Resch.OBJ_VAL);  //az aktuális állapotra vonatkozó célfüggvényértékek
s_init.freeze_jobs_orders_and_machines( O, J, M, freeze_time); //munkák, rendelések, gépek zárolása
_S s0(s_init);    //kezdeti megoldást tároló objektum
s0.calc_M_STET_ FREEZE(MG_M, O, J, M, s_init, freeze_time);          //végrehajtás-szimuláció
s0.calc_job_order_time_and_obj_value(obj_val, O, J);   //időadatok és célfüggvényekértékek számítása
s0.calc_diff_obj_val(obj_val, O, J, s_init, Shop_Released.O, Shop_Released.J); //stabilitásmutatók
        //változásorientált újraütemezési célfüggvényértékek számítása
_S s_saved(s0);                     //a kiterjesztés kiinduló megoldása
_S s_ext(s0);                       //a kiterjesztés legjobb megoldása
OBJ_VAL obj_val_ext(obj_val);       //a kiterjesztés legjobb megoldásának célfüggvényértékei
_S s_best(s0);                      //a keresés során megtalált legjobb megoldás
OBJ_VAL obj_val_best(obj_val);      //a keresés során megtalát legjobb megoldás célfüggvényértékei
TABULIST TabuList(NULL);            //korábban kiértékelt, tiltott megoldások listája
int used_operator;                  //az aktuális kiterjesztésben a legjobb szomszédot előállító operátor
int Nx_selected_operator;           //a használatra kijelölt operátor
Nx_total = new long[Nx_number+1];   //a szomszédsági operátorok kiválasztásának számértékei
Nx_improve = new long[Nx_number+1];   //a szomszédsági operátorok által megnyert kiterjesztések száma
for (int n = 0; n <= Nx_number; n++)    //a szomszédsági operátorokhoz tartozó kezdőértékek megadása
      Nx_total[n] = Nx_improve[n] = 0;

for(int step = 1; step <= STEPS; step++)   //a keresés lépésenkénti végrehajtása
{
 if ( Scheduling_stop() == true ) break;   //leállás: időkorlát elérése/ nincs javulás/ felhasználói akció
 for (int ext = 1; ext <= LOOPS; ext++ )  //kiterjesztés: adott lépéshez tartozó szomszédság vizsgálata
 {
   s0 = s_saved;                    //az aktuális kiterjesztés kiindulási megdoldásának (bázis) betöltése
   Nx_selected_operator = select_Nx_operator( Nx_number, Nx_operator_priority ); //kiválsztás
   s0.perturb_FREEZE( Nx_selected_operator );        //a szomszédos megoldás elkészítése
                                                     //a kiválasztott operátor alklamazásával
 if ( TabuList.does_TL_include( s0 ) )  continue;    //tiltott megoldás esetén új szomszédot készít
 else
    {
    TABU* newtabu = new TABU( s0 );                  //az aktuális megoldás kódolása
    TabuList.add_new_TABU_into_startpos(newtabu);    //felvétel a tabulista elejére
    if (TabuList.Get_N_of_TABU() > MAX_LENGHT)       //a tabulista méretétől függően
       TabuList.delete_TABU_from_lastpos();          //a legkorábban felvett tabu törlése
    }
```

//folytatás a következő oldalon

```
      //feladatok végrehajtásának szimulációja, az dőadatok, a célfüggvényekértékek és a stabilitásmutatók számítása
s0.calc_M_STET_ FREEZE(MG_M, O, J, M, s_init, freeze_time);
s0.calc_job_end_time_and_obj_value(obj_value, O, J);
s0.calc_diff_obj_val(obj_val, O, J, s_init, Shop_Released.O, Shop_Released.J);

if ( ext == 1 )                                  //az aktuális kiterjesztés első megoldása
   {
    obj_value_ext = obj_value;                   //a megoldáshoz tartozó célfüggvény-értékek másolása
    s_ext = s0;                                  //az aktuális kiterjesztés legjobb megoldása
    used_operator = Nx_selected_operator;        //a legjobb megoldást előállító operátor megjegyzése
   }
   else
   {
      if ( obj_value_ext > obj_value )           //az aktuális megoldása jobb mint
        {                                        //a kiterjesztés során eddig megtalált legjobb megoldás
         obj_value_ext = obj_value;              //a megoldáshoz tartozó célfüggvény-értékek másolása
         s_ext = s0;                             //az aktuális kiterjesztés legjobb megoldásának megjegyzése
         used_operator = Nx_selected_operator; //a megoldást előállító operátor megjegyzése
         if ( ext > min_LOOP && obj_value_ext < obj_value_best ) break;
        } //adott számú iteráció után bejeződhet a kiterjesztés ha az eddigi legjobb megoldástól is jobbat talált
   }
}

s_saved = s_ext;                 //a kiterjesztés legjobb megoldása lesz a következő kiterjesztés bázisa
Nx_total[ used_operator ]++;     //a kiterjesztést megnyerő operátor győzelmeinek növelése
Nx_total[ 0 ]++;                 //az összegzett győzelmek számának növelése
if ( obj_value_ext <= obj_value_best )   //az aktuális kiterjesztés legjobb megoldása legalább olyan jó
   {                                     //mint a keresés során megtalált eddigi legjobb megoldás
     obj_value_best = obj_value_ext;     //a megoldáshoz tartozó célfüggvény-értékek másolása
     s_best = s_ext;                     //a legjobb megoldás megjegyzése
     Nx_improve[ used_operator ]++;      //a legjobb megoldást adó operátor jutalmazása
     Nx_improve[ 0 ]++;                  //a javítások összesített számának növelése
   }
Actualize_priority(Nx_operator_priority, Nx_improve, Nx_total); //az operátorok értékelése
}

delete [] Nx_total; delete [] Nx_improve;        //lokális változók megszüntetése

return s_best;                                   //megtalált legjobb megoldás
}
```

A bemutattott koncepciót felhasználva a kiterjesztett EFFS modellel hatékonyan kezelhetők mind az előidejű termelésprogramozási, mind az újraütemezési feladatok. Az általánosított probléma megoldására kifejlesztett új szemléletű integrált módszer végrehajtás-szimulációra alapozott problématér-transzformációval, heurisztikus (előkészítő, rendszerező) algoritmusok valamint többcélú keresési technika kombinált alkalmazásával egyidejűleg támogatja a rendelések bontására és/vagy egyesítésére, a gyártási sorozatnagyságok dinamikus meghatározására, a technológiai alternatívák kezelésére, a gépi erőforrások allokálására, a gyártási feladatok definiálására és azok végrehajtásának időbeli ütemezésére vonatkozó döntéseket.

# 6. TERMELÉSPROGRAMOZÓ SZOFTVER

## 6.1. Az integrált termelésprogramozó szoftver koncepciója

A rugalmas tömeggyártásban fellépő műhelyszintű termelésirányítási feladatok esetében a hagyományos MES alkalmazások az igények egy részét nem tudják maradék nélkül kielégíteni [68], [69], [70], [88].

A gyártásirányítás hatékonysága nagymértékben növelhető valós idejű visszacsatolt szabályozás megvalósításával. Ennek érrdekében szükség van a tervezett (ütemezett) és a tényleges termelésirendszer-állapot ismételt összevetésére, értékelésére és indokolt esetben korrekciós beavatkozások kezdeményezésére.

A kívánt állapotot a mindenkor érvényes termelési finomprogram (a hozzá tartozó specifikációkkal és egyéb információkkal együtt: *SHOP*) képviseli, amely a termelés teljesítményét mérő mutatók előidejű optimalizálásával készült. Az éles finomprogram valós termelési teljesítményét mérő mutatók a MES termeléskövető komponenséből visszacsatolhatók és összevethetők a tervezettekkel. A termelési folyamat további jövőbeli alakulására vonatkozó becsült menedzsermutatók kellően gyors termelésifolyamat-szimulátorral számíthatók. Az értékelés egyik lehetséges következményeként a hatékony beavatkozás érdekében újraütemezésre kerülhet sor, amelyet a termelésprogramozó szoftver automata megoldómotorja és/vagy annak interaktív kézi beavatkozást biztosító szolgáltatásrendszere támogathat.

A műhelyszintű viselkedésalapú termelésifolyamat-szabályozás megvalósításához tehát szükséges, hogy a termelésprogramozó szoftver megfeleljen a vele szemben támasztott funkcionális elvárásoknak. Ezek közül a legfontosabbak a következők:

- A termelési finomprogram végrehajtás-szimulációjának és értékelésének megvalósítása.
- Gyártási rendszerállapotok összehasonlítása.
- Termelési szituációk összehasonlító elemzése.
- Ütemezési és újraütemezési feladatok hatákony megoldása.
- Lehetséges és megengedett felhasználói beavatkozások támogatása.
- MES rendszerbe történő beágyazás biztosítása.

A bemutatott koncepciónak megfelelően, a megfogalmazott szempontok figyelembevételével objektumoritentált paradigmát követve megterveztem és megvalósítottam a javasolt termelésprogramozó szoftver egy olyan prototípusát, amely rugalmasan konfigurálható és bővíthető modulokból épül fel, ennek köszönhetően a

szolgáltatások rendszere az aktuális (gyári körülményeket tükröző) igényeknek megfelelően testreszabható, kiterjeszthető pl. a termelési folyamat szimulációja, az optimalizálási célfüggvénykészlet, az összevethető termelési adatok rendszere stb. tekintetében.

## 6.2. A megvalósított termelésprogramozó szoftver

A kidolgozott EFFS ütemezési modell, valamint a kifejlesztett megoldási módszerek és algoritmusok felhasználásával megterveztem és C++ Builder szoftverfejlesztő környezetben objektumorientált módon megvalósítottam a többfunkciós integrált ütemező és újraütemező szoftver prototípusát.

### 6.2.1. Az alkalmazott szoftvertechnológia

#### 6.2.1.1. Objektumorientált szoftverfejlesztés

A vizsgált termelésprogramozási probléma megfogalmazását, modellezését és a megoldási módszerek kifejlesztését objektumorientált módszertan felhasználásával végeztem el. Ennek köszönhetően a kifejlesztett rendszer modellje együttműködő objektumokból épül fel. A tervezés és az implementáció során olyan programegységeket alakítottam ki, amelyek alkalmasak az objektumok megvalósítására.

A koncepció feltevése, hogy az előidejű termelési finomprogramozás (ütemezés) és a viselkedés alapú korrekciós újraütemezés modelljének objektumai nagyrészt azonosak.

Az objektum a rendszermodell egyedileg azonosítható szereplője, amely a belső szerkezetével, állapotával és viselkedésével jellemezhető. Az objektum egységbe zárja az aktuális állapotát tároló adatokat (adattag vagy attribútum) és azok szerkezetét, valamint a rajtuk végrehajtható – az objektum viselkedését meghatározó – műveleteket (tagfüggvények vagy metódusok). Az objektumnak a renszerben meghatározott szerepköre van.

Az objektumok egyrészt közvetlen üzeneteken keresztül kommunikálnak egymással. Minden objektum az üzenetek egy meghatározott készletét képes elfogadni és értelmezni. Az elküldött üzenet (név, paraméterek) által hordozott információ vezérlő jellegénél fogva a fogadó objektumnak különböző reakcióit váltja ki.

Az objektumok kommunikációjának másik eszköze az esemény, amely azonosítható, pillanatszerű történést jelent. Hasonlóan az üzenetekhez a név statikus, a paraméterek dinamikus információt hordoznak. A paraméterek leírják az azonosított esemény bekövetkezésének körülményeit. Az eseményeket objektumok időbeli viselkedése hozza létre, majd a környezet gondoskodik arról, hogy más objektumok értesüljenek az esemény megtörténtéről és hozzájussanak az esemény paramétereihez.

Egy objektum a hozzá érkező üzenet hatására vagy egy esemény bekövetkezésének hatására valamilyen cselekvést hajt végre. Az objektum viselkedésének pontos leírása, implementációja a metódusainak kódjában található. Az objektum azonos tartalmú üzenetekre eltérően képes reagálni aktuális állapotától függően. Az objektum viselkedését tehát attributumai által hordozott állapotinformációi determinálják és ezek az objektum élete során változhatnak.

A megegyező viselkedésű és struktúrájú objektumok egy közös osztályba tartoznak, ezáltal egy adott objektum az őt definiáló osztálynak egy adott példánya.

Az objektumorientált szemlélet láthatóan hatékonyan alkalmazható a különböző MES funkciók, így az előidejű (prediktív) tervezés és a korrekciós (proaktív) újra-ütemezés támogatására.

#### 6.2.1.2. Programozási nyelv és fejlesztői környezet

Az imlementáció elkészítésére a C++ programozási nyelvet használtam, amely nyelvi szinten kiválóan támogatja az objektumorientált paradigma alkalmazását a programozói lehetőségek jelentős mértékű korlátozása nélkül. A nyelv specifikációjának köszönhetően rugalmas, tömör, magasszintű, hatékony forráskódkód, valamint a rendelkezésre álló fordítóprogramok segítségével futásidőben gyors program hozható létre.

A termelésprogramozó szoftver elkészítéséhez a Borland C++ fejlesztői környezet 5.0 verzióját használtam. Ez az integrált fejlesztői környezet rendelkezik kétutas erőforrásszerkesztővel. Ennek segítségével a grafikus felhasználói interfész egyrészt grafikus tervezőfelület használatával, másrészt közvetlen forrásszerkesztő szolgáltatások segítségével kombinált módon az előre gyártott komponensek felhasználásával, valamint meglévő elemek továbbfejlesztésével egyszerűen megvalósítható.

A Borland C++ hatékony eszközökkel támogatja az objektumorientált szoftverfejlesztést. Számos hasznos funkció menürendszer, eszköztárak segítségével elérhető, a forráskód strukturális vázának elkészítését beépített varázslók segítségével is támogatja. A hibakeresésre szintén hatékony módszereket biztosít.

### 6.2.2. A termelésprogramozó szoftver strukturális felépítése

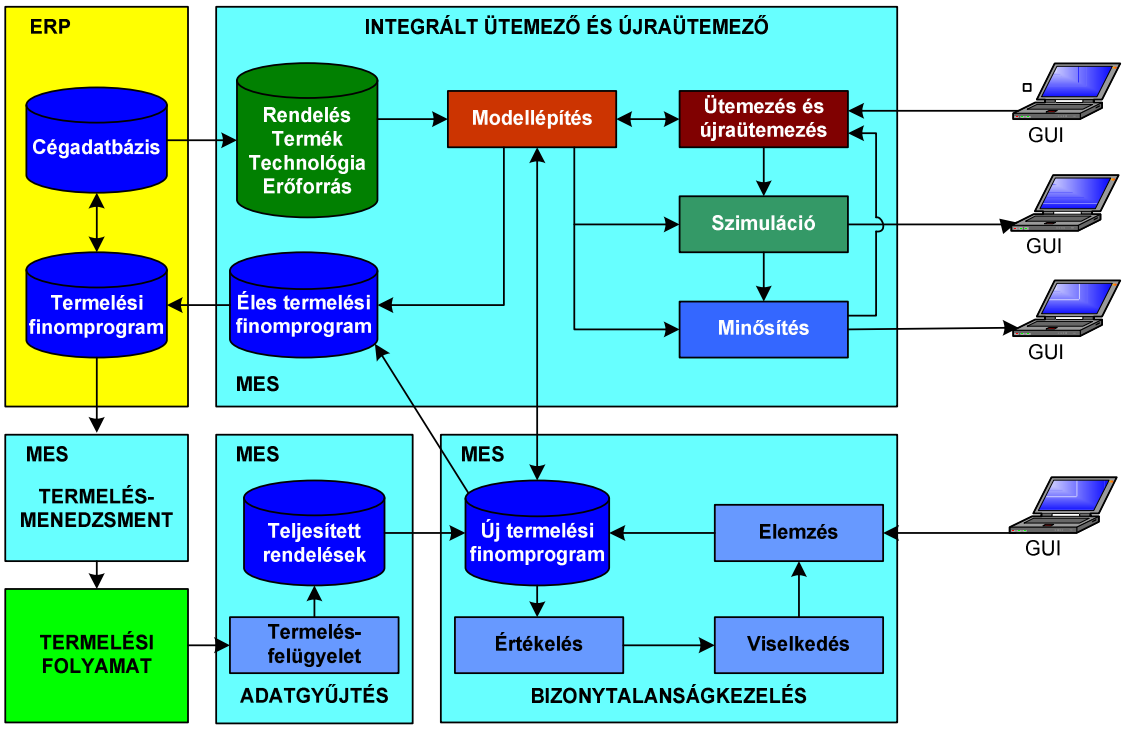
#### 6.2.2.1. Funkcionális modell

A kifejlesztett termelésprogramozó mintarendszer jelenlegi verziója önálló alkalmazásként virtuális környezetben működik. A szoftver a működéshez szükséges bemeneti (input) adatokat adatbázisból vagy lokális fájlokból tudja kiolvasni. A fejlesztéshez szükség volt egy paraméterezhető problémagenerátor létrohozására, valamint váratlan események hatását leíró modul megvalósítására is.

A keretrendszernek a felhasználó által elérhető funkciói a következők:

- input adatok megadása/megtekintése,
- célfüggvények és prioritások kezelése,
- ütemezési paraméterek kezelése,
- előidejű termelésütemezés,
- korrekciós újraütemezés,
- tervezett és tényleges állapotok összehasonlítása,
- termelési finomprogramok szerkesztése/módosítása (kézi ütemezés),
- eredmények, diagramok, jelentések létrehozása és megtekintése,
- termelési finomprogramok mentése/végrehajtás indítása,
- tesztfeladatok készítése,
- váratlan események generálása (valós környezetben az utolsó két funkcióra nincs szükség).

Az alkalmazás funkcionális modelljét és a termelésinformatikai rendszerben betöltött szerepét a 14. ábra vázolja.




**14. ábra: Integrált termelésprogramozó szoftver funkcionális kapcsolatai**

A szoftver a termelés előidejű tervezésére, ütemezésére valamint a termelési finomprogram időközbeni korrekciós újratervezésére (a bizonytalanságokból és zavarokból keletkező hibák elhárítására, csökkentésére) közös és többcélú funkcionális komponenseket használ. Ezek a következők:

- többfunkciós paraméterkezelő,
- modellépítő komponens,
- ütemező és újraütemező megoldómotor (interaktív grafikus felülettel a felhasználói beavatkozás támogatására),
- termelésifolyamat-szimulátor,
- terljesítményértékelő komponens.

Az integrált ütemező és újraütemező szoftver funkcionális komponenseinek feladata és a rendszerben betöltött szerepe az 4.14. fejezetben a megoldási módszer kifejtése során részletesen bemutatásra került.

#### 6.2.2.2. Architekturális modell

A megvalósított szoftver együttműködő dinamikus és statikus objektumok rendszeréként működik. A termelési finomprogramot előállító objektumok dinamikusak, futási időben keletkeznek a bemenő adatokban szereplő kódolt információ feldolgozása során, valamint az ütemezési folyamat közben. A tervezés során kiemelkedően fontos szempont volt, hogy az ütemező algoritmusnak a futási idő szempontjából kritikus szakaszában (ciklusban végrehajtott módosítás, szimuláció, kiértékelés) a lehető legkevesebb új objektum keletkezzen. Ilyen esetekben ugyanis a nélkülözhetetlen memórifoglalások következtében a végrehatás meglehetősen lelassulhat.

A legfontosabb objektumok attribútumait és azok kapcsolatrandszerét a 3.4 fejezetben részletesen bemutattam. Az architekturális tervezés során a különböző objektumtípusokat adattagjaik és tagfüggvényeik együttesével definiálva önálló osztályként valósítottam meg. A legfontosabb osztályok a következők:

- Termék – PRODUCT.
- Rendelés – ORDER.
- Munka – JOB.
- Gép – MACHINE.
- Termelési finomprogram – _S.
- Minősítő elem (célfüggvényértékek együttese) – OBJ_VAL.
- Gyártórendszer – SHOP.

A szoftverben alkalmazott számos objektumra programozástechnikai szempontból van szükség, ezek támogatják a kommunikációt és a számítógépi megjelenítést. Ilyen segédobjektumok például az alkalmazásobjektum (applikáció), mely a szoftver keretét adja, a felhasználói interfész objektumai (pl. ablakok, vezérlőelemek), az adatbázis-kezelő objektumok, a fájlkezelő objektumok és egyéb segédobjektumok.

A termelésprogramozó szoftverben felhasznált osztályok részletes bemutatására terjedelmi okokból nem térek ki.

#### 6.2.2.3. Interfészek

A termelésprogramozó szoftver alapvetően három különböző interfészt használ. Ezek a következők:

- Adatbázis-kezelő interfész,
- fájlkezelő interfész,
- grafikus felhasználói interfész.

A termelésprogramozó szoftvernek valós környezetbe történő integrálását legegyszerűbben adatbázis-kapcsolatokon keresztül lehet megvalósítani. A jelenlegi szoftververzió relációs adatbázisokat kezelő objektumokon keresztül képes alapvető tranzakciók (pl. táblakezelés, SQL lekérdezés) megvalósítására. Erre a célra az BDE (Borland Database Engine) objektumait használtam fel.

Az értekezésemhez mellékelt szoftververzionak az input tesztadatokat szabványos szöveges (text) fájlformátumban kell megadni. Az eredményként elkészített termelési finomprogramot a hozzá tartozó információkkal együtt, a különböző jelentéseket valamint a naplózási és egyéb beállítási adatokat szöveges fájlba menti el. A fájlkezelés nyelvi szinten az fstream.h header fájl osztályainak felhasználásával készített objektumokon keresztül valósul meg.

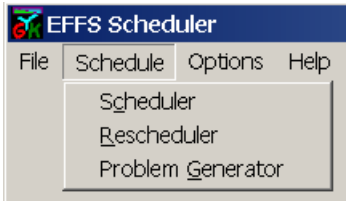
A szoftvereszköz fejlesztésekor a VCL (Visual Component Library) osztálykönyvtárat használtam. Az abban definiált osztályokból származnak öröklődéssel többek között a Windows operációs rendszeren történő futtatáshoz szükséges alapverő objektumok (pl. ablakok, kijelzőelemek, vezérlőelemek, egyéb grafikus objektumok). A felhasználói interfész részletes bemutatására a következő fejezetben kerül sor.

### 6.2.3. A szoftver szolgáltatásai

#### 6.2.3.1. Felhasználói felület

A kezelői felület kialakításakor arra törekedtem, hogy a termelésprogramozó szoftver egyszerűen kezelhető legyen, ugyanakkor a felhasználó számára a lehető legnagyobb mértékű beavatkozást tegye lehetővé, továbbá hatékony operációkészlettel támogassa a beavatkozások megvalósítását és a döntések meghozatalát.

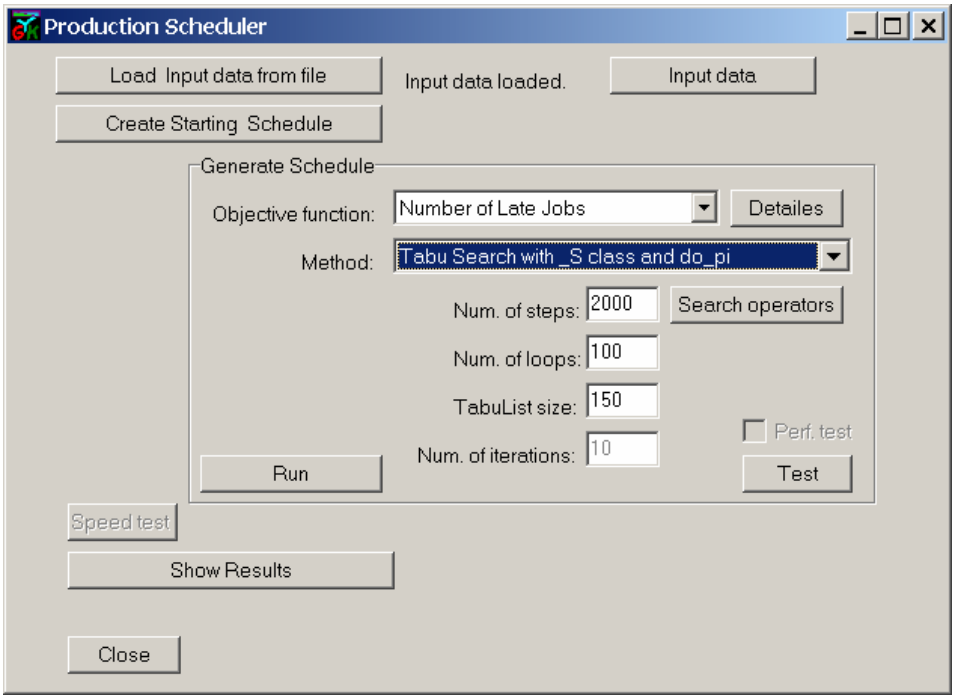
Az alkalmazás nyitóablakának főmenüjéből indítható el az előidejű ütemező, az újraütemező és a tesztprobléma-generátor (15. ábra).



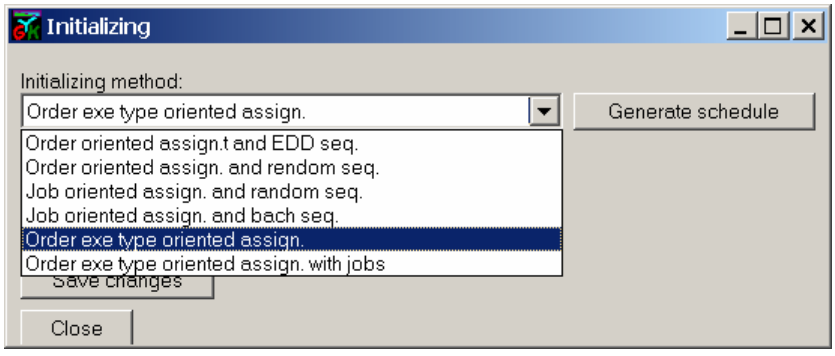

**15. ábra: A nyitóablak legfontosabb menüpontjai**

Az előidejű ütemező ablaka a 16. ábrán látható. Innen érhetők el a következő funkciók:

1. az input adatok betöltése (erőforráskörnyezet, megrendelések),
2. az input adatok megtekintése,
3. aktuális prioritásértékek hozzárendelése a célfüggvényekhez,
4. kezdeti megoldást adó heurisztikus módszer indítása,
5. a használni kívánt megoldómódszer kiválasztása,
6. kezdeti prioritásértékek hozzárendelése a keresőoperátorokhoz,
7. a keresési paraméterek beállítása (a megengedett lépészám, a kiterjesztések és a a tabulista elemszámának maximumai),
8. a megoldási módszerek tesztelésének, értékelésének és vizsgálatának indítása,
9. eredmények megtekintése.



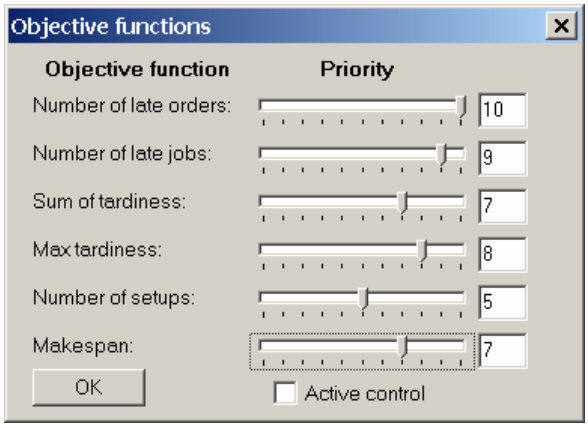

**16. ábra: Az ütemező ablaka**




**17. ábra: Kezdeti megoldást készítő heurisztikus módszerek**

A *Create Starting Schedule* nyomógomb segítségével érhető el a kezdeti megoldás előállítását vezérlő dialógusablak (17.ábra). Az *Initializing* lista mutatja a rendelkezésre álló heurisztikus algoritmusokat. A *Generate Schedule* nyomógombbal indítható el a kiválasztott módszer.

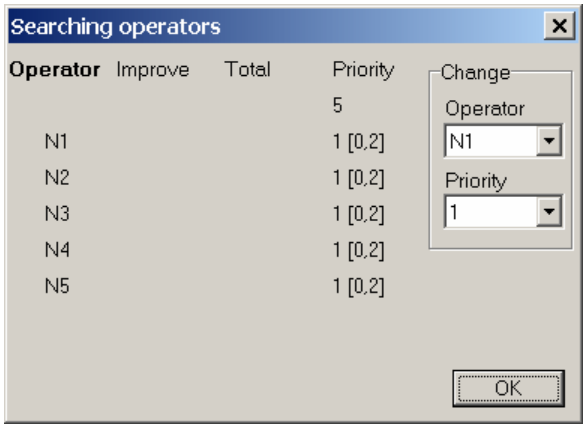
A ütemező főablakának (*Production Scheduler) Detailes* nyomógombja nyitja meg a célfüggvények prioritásainak beállítására szolgáló *Objective functions* ablakot (18. ábra).




**18. ábra: Ütemezési célok aktuális fontosságának beállítása**

Az ütemező főablakának *Method* listájában szerepelnek a kezdeti ütemterv iteratív javítására használható keresőalgoritmusok, algoritmusváltozatok és egyéb módszerek. A kiválasztott módszer aktuális paraméterei beállíthatók a *Generate Schedule* csoportosítópanelen elhelyezett vezérlőelemekkel.

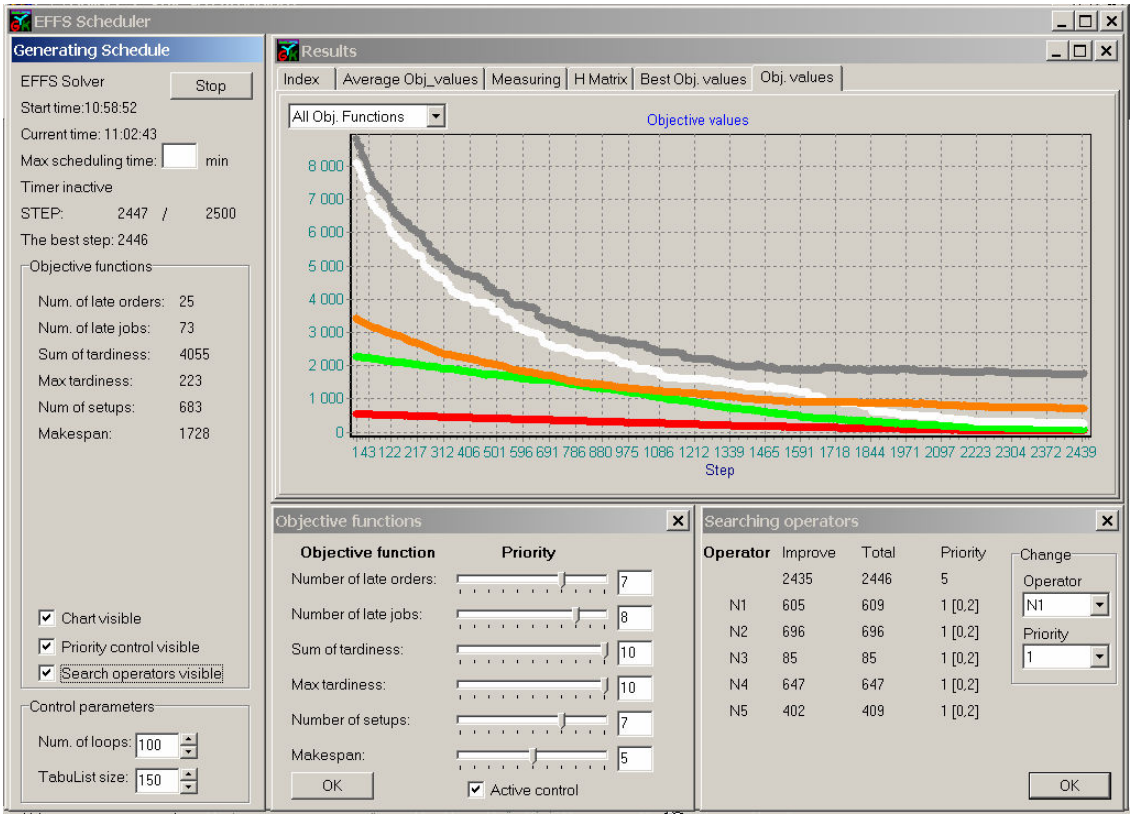
A *Search operators* nyomógomb megnyomásával érhető el a keresőoperátorok prioritásainak beállítására szolgáló *Searching operators* ablak (19. ábra).




**19. ábra: A szomszédsági operátorok prioritásértékei**

A főablak *Run* nyomógombja indítja el a megoldási folyamatot. Ennek a felhasználó által történő folyamatos nyomonkövetését és aktív vezérlését a 20. ábrán látható eszközök támogatják. A *Generating Schedule* ablakban láthatók a keresési folyamat állapotát jellemző idő- és lépésszámértékek, a keresési folyamat során

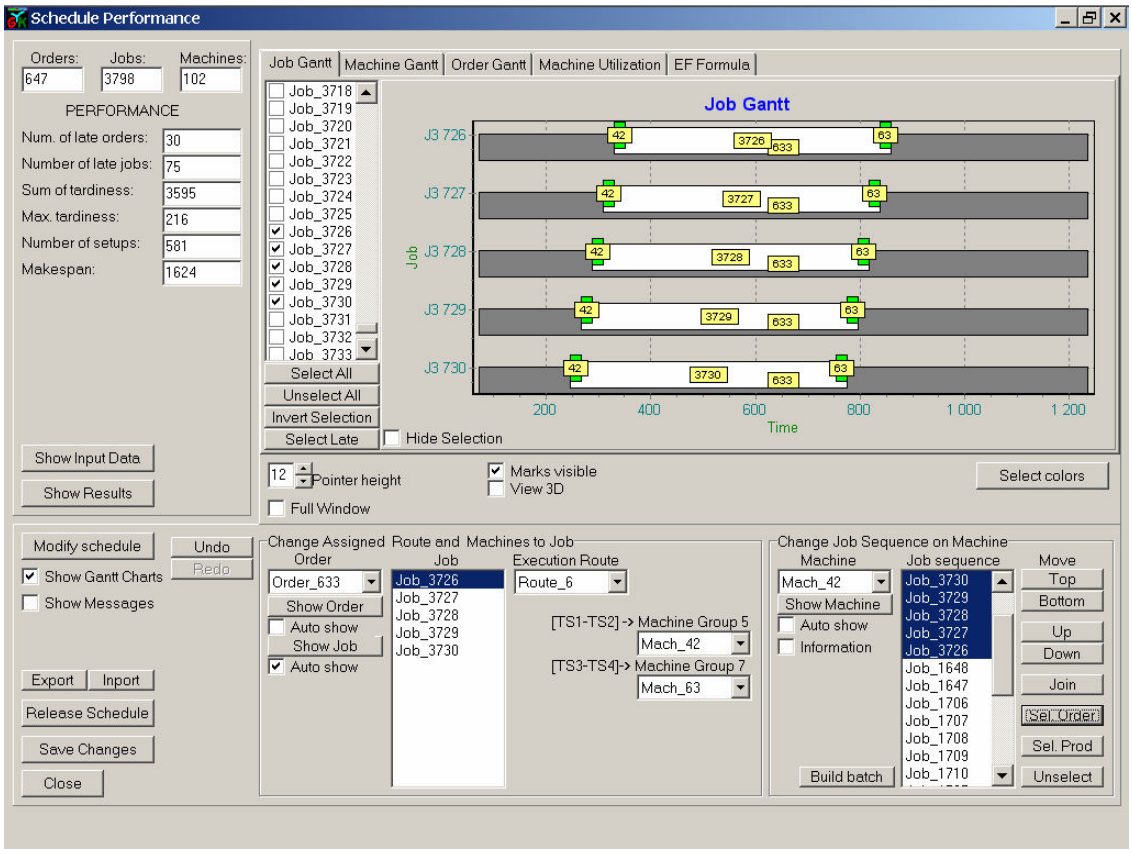
megtalált legjobb megoldás célfüggvényértékei és a keresési paraméterek. A *Results* ablakban (nagyítható, testreszabható diagram formájában) a célfüggvényértékek grafikonjai láthatók a megtett lépések számának függvényében. Az *Objective functions* és a *Searching operators* ablakok vezérlőelemeinek segítségével a felhasználó a keresési folyamat közben változtathatja a prioritásértékeket.



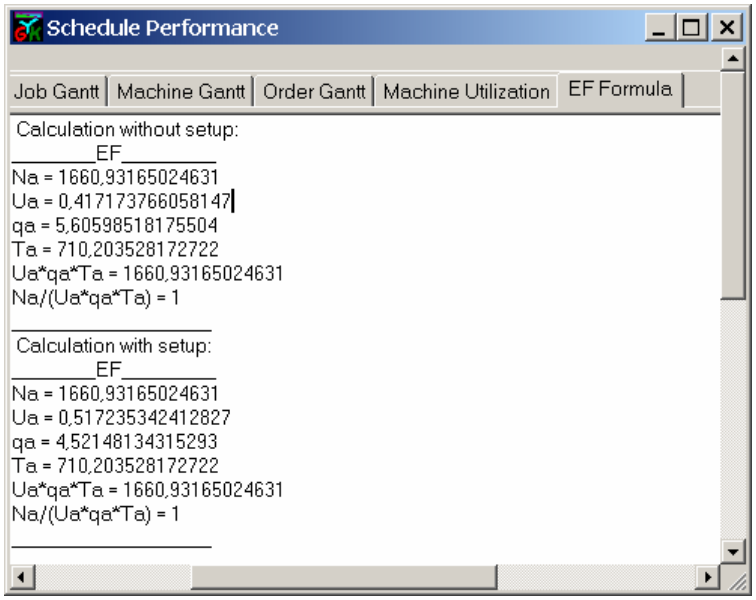

**20. ábra: Az ütemező működésének vezérlésére szolgáló eszközök**

A (heurisztikus algoritmussal, vagy keresőalgoritmussal) elkészített termelési finomprogram részletei, valamint annak várható teljesítményét jelző célfüggvényértékek, mutatószámok, diagramok és táblázatok a *Schedule Performance* ablak (21. ábra) eszközeinek segítségével vizsgálhatók és szerkeszthetők.

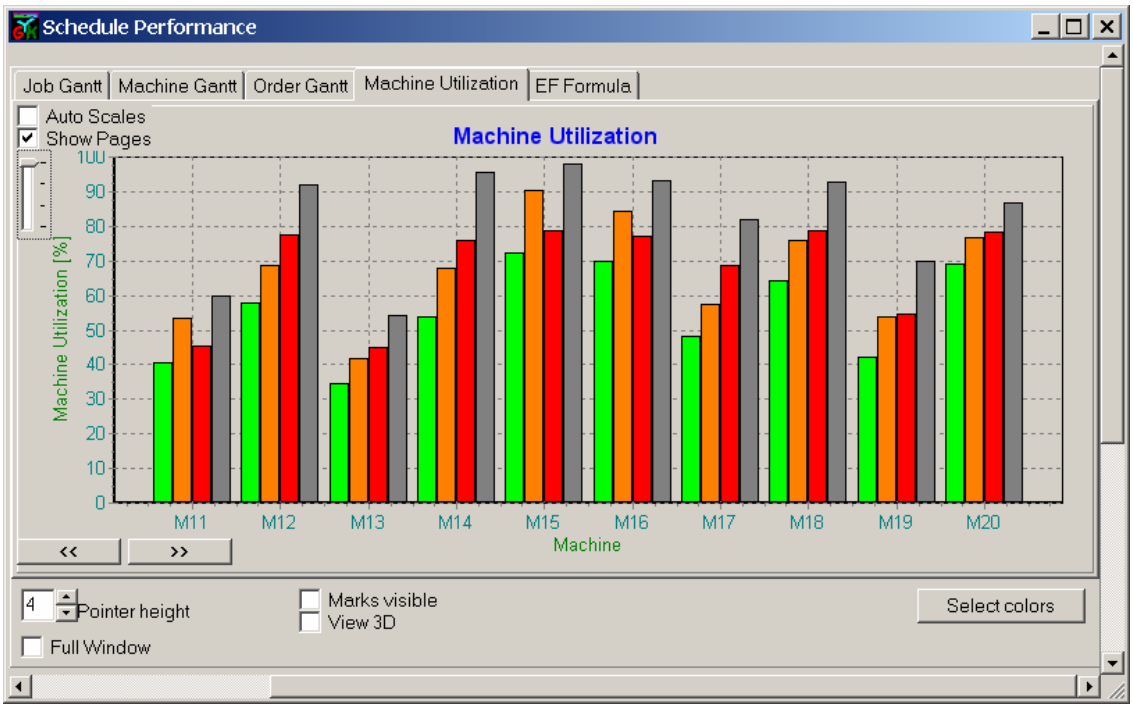
A *Performance* panelen láthatók a célfüggvények értékei. Az ütemezési folyamatot jellemző információk a *Show Results* nyomógomb megnyomására jelennek meg. A termelési finomprogram részleteinek tanulmányozását munka-, gép- és megrendelésorientált Gantt-diagramok valamint gépkihasználtsági diagramok és egyéb számított értékek segítik (22. és 23. ábra). A diagramok tetszőlegesen nagyíthatók, kicsinyíthetők, tartalmuk mozgatható és léptethető. A megjelenő adatok szelektálhatók, elrejthetők. A tájékozódást a feliratok és a térhatású megjelenítőeszközök, valamint az előredefiniált és testreszabható színsémák tovább segítik.




**21. ábra: Értékelést és szerkesztést támogató szolgáltatások**

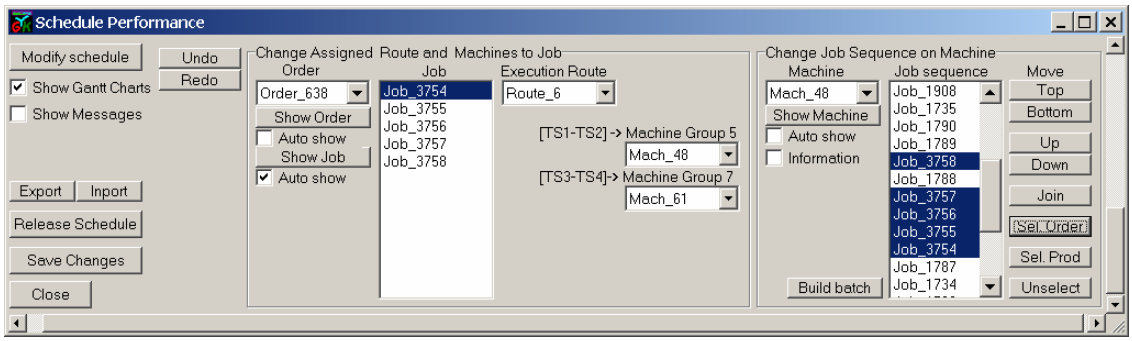



**22. ábra: Tervezett menedzsermutatók**




**23. ábra: Gépkihasználtsági mutatók diagramja**

A felhasználó számára a szoftver előredefiniált szerkesztési műveleteket, interaktív operációkészletet biztosít (24. ábra). Ezeket az eszközöket a *Scheduling performance* ablak *Modify schedule* nyomógombjával lehet aktiválni. A vezérlőeszközök alkalmasak a termelési finomprogram konzisztens módosítására (hozzárendelési, sorrendi módosítások) és a megjelenítés vezérlésére is (módosítás nélkül a kiválasztott elemekre fókuszált megjelenítéssel). A vezérlőeszközök kölcsönkapcsolatban állnak egymással és a kijelzésre használt objektumokkal, bármelyik állapotának megváltoztatása a többi vezérlő aktualizálását vonja maga után. Az operációkészlet csak megengedett (kemény korlátozásokat nem sértő) műveletek végrehajtását teszi lehetővé. Az elvégzett műveletek hatása az ablak eredménykijelző objektumain (a háttérben futó szimulációs és kiértékelő taszkoknak köszönhetően) azonnal látható.

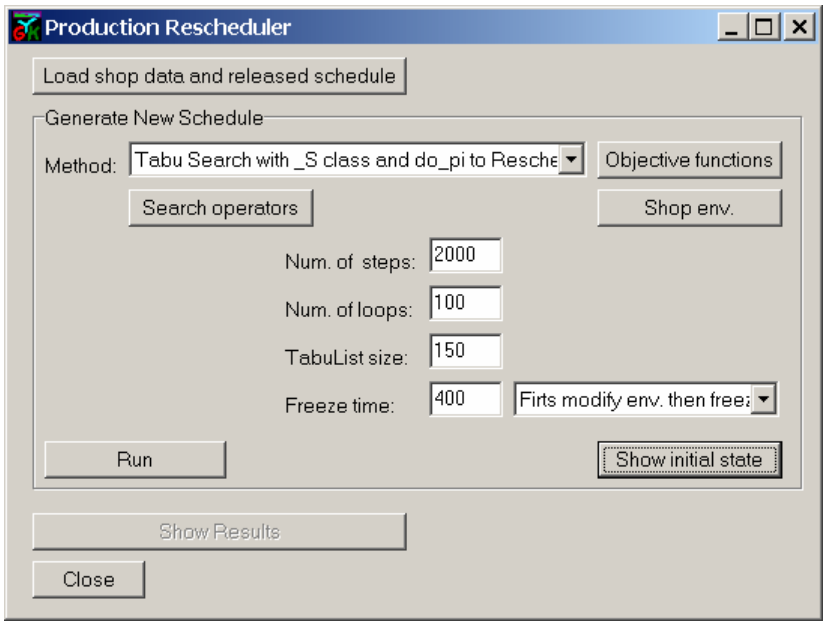



**24. ábra: A termelési finomprogram kézi szerkesztését támogató operációkészlet**

Néhány alapvető művelet a következő:

- A kiválasztott megrendelés (*Order*) munkáinak végrehajtására vonatkozó döntések megtekintése, módosítása.
  - A kiválasztott munka végrehajtását előíró adatok átvétele az érintett megrendelés egy másik munkájától.
- A kiválasztott munka (*Job*) gyártási feladatokra történő bontása, a végrehajtására vonatkozó adatok megtekintése, módosítása.
  - A végrehatási útvonal (*Execution Route*) közvetlen beállítása/megváltoztatása a megengedett útvonalak listájából egy konkrét útvonal kiválasztásával (gyártási feladatokra bontás).
  - Az adott munka kiválasztott gyártási feladatának konkrét géphez rendelése/áthelyezése a rendelkezésre álló megengedett párhuzamos gépek közül egy konkrét gép kijelölésével.
- A kiválasztott gépen (*Machine*) a gyártási feladatok sorrendjének szerkesztése.
  - Egyetlen vagy több munka egyidejű kiválasztása módosításra egérműveletekkel és/vagy lekérdezéssel (*Sel.Prod, Sel.Order, Unselect*).
  - Kiválasztott munka (munkák) előre/hátra léptetése a végrehajtási sorban (*Up/Down*).
  - Kiválsztott munka (munkák) mozgatása a lista elejére/végére (*Top/Bottom*).
  - Kiválasztott munkák összekapcsolása (*Join*).
  - A végrehajtási sorban lévő munkáknak a gépbeállítás-típusoktól függő rendezése (*Build batch*).
- Az aktuális tervezési, szerkesztési állapotnak megfelelő termelési finomprogram mentése/visszatöltése (*Export/Import*).
- A végrehajtott módosítások visszavonása/ismételt végrehajtása (*Undo/Redo*).

Az elkészített termelési finomprogram a *Release schedule* gomb megnyomásával élesíthető, elküldhető végrehajtásra (az értekezéshez mellékelt verzióban ez a művelet fájlba írást eredményez).




**25. ábra: Az újraütemező ablaka**

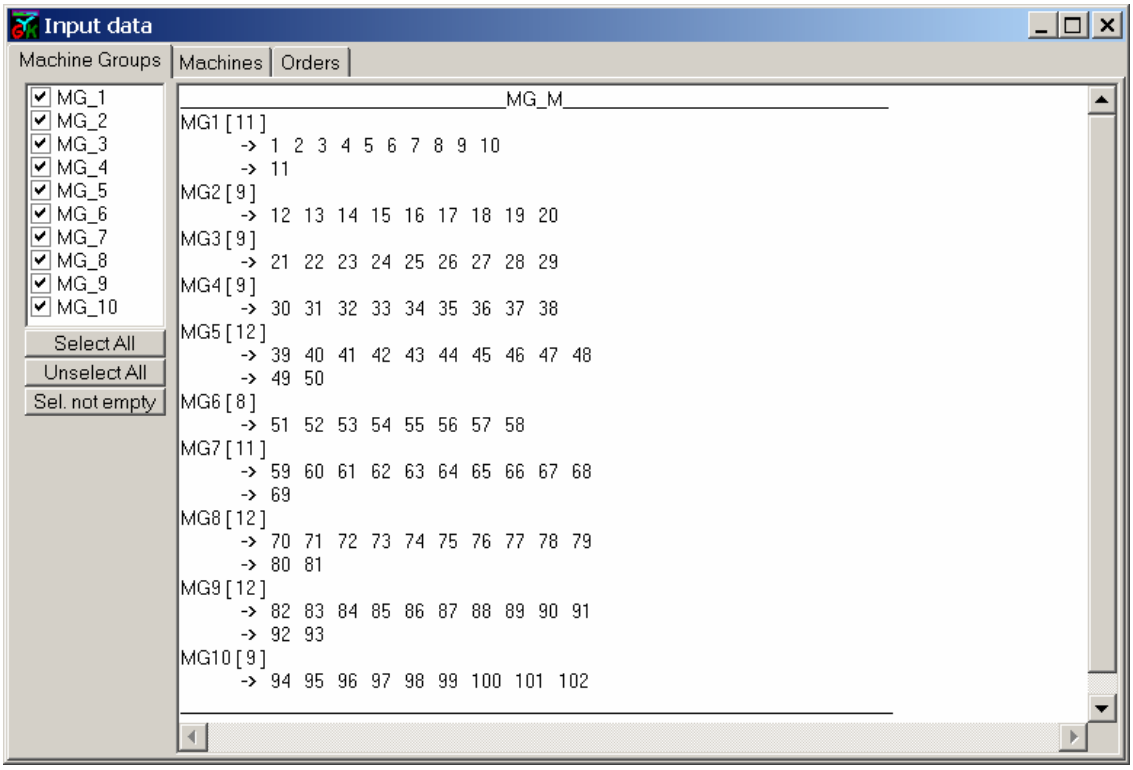
Az újraütemezési funkció a szoftver nyitóablakának *Rescheduler* menüpontján keresztül érhető el, ennek felhasználói felülete a 25. ábrán látható. Ezen keresztül érhetők el a következő szolgáltatások:

1. az input adatok betöltése (a gyártórendszer aktuális állapota, a tervezett termelési finomprogram a hozzátartozó adatokkal),
2. a gyártórendszer állapotának megtekintése, változások, váratlan események és zavarok kezelése,
3. aktuális prioritásértékek hozzárendelése a célfüggvényekhez és a speciális újraütemezési célfüggvényekhez,
4. a használni kívánt megoldómódszer kiválasztása,
5. kezdeti prioritásértékek hozzárendelése a keresőoperátorokhoz,
6. a keresési paraméterek beállítása (max. lépészám, kiterjesztés, tabulista elemszáma)
7. zárolási időpont beállítása,
8. zárolási stratégia kiválasztása,
9. a kezdeti megoldás (zárolt aktuális állapot) megtekintése.

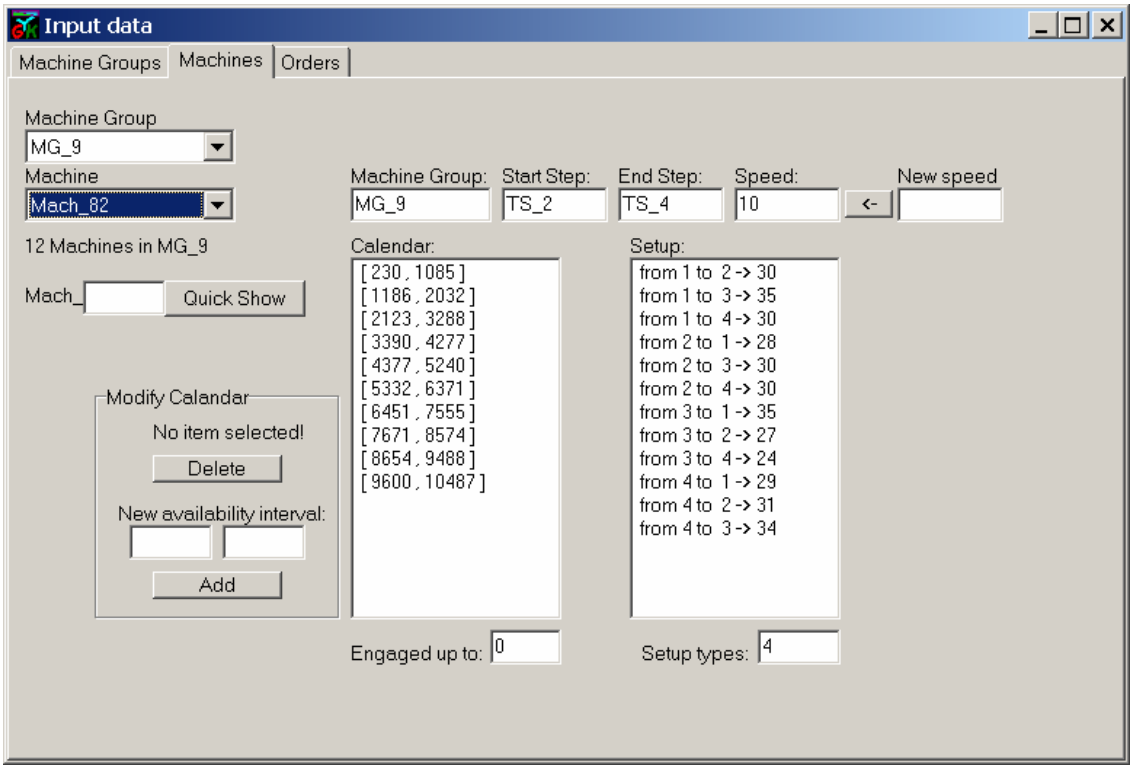
A termelési folyamat állapota tényleges üzemi körülmények között a MES rendszerből, a rendelésekre vonatkozó információk az ERP rendszerből adatbázis-kapcsolaton keresztül visszacsatolhatók a termelésprogramozó szoftverbe. A fejlesztés során ezek hiányában a szoftvert virtuális környezetben teszteltem, a változások előidézésére felhasználói felületet dolgoztam ki.

Az input adatok betöltését követően a felhasználó megtekintheti a gyártórendszer gépeit funkcionális gépcsoportokba szervezett elrendezésben (26. ábra), megváltoztathatja a gépek egyes tulajdonságait (27. ábra), valamint módosíthatja a megrendelések bizonyos attribútumait (28. ábra).

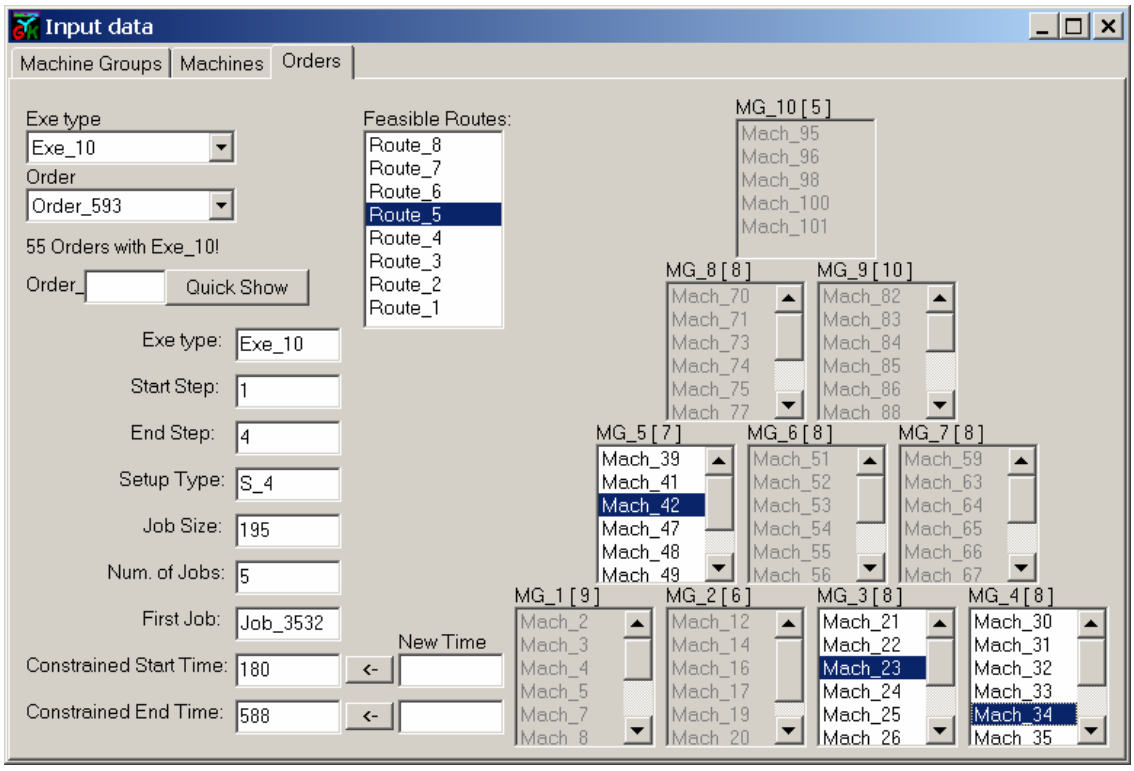
Ezek a megvalósított funkciók a szimulációs szolgáltatásokkal együtt lehetővé teszik, hogy ténylegesen még be nem következett, de már előjelekből kikövetkeztethető események bekövetkezésének várható hatása elemezhetővé váljon.




**26. ábra: Gépek és gépcsoportok megtekintése**

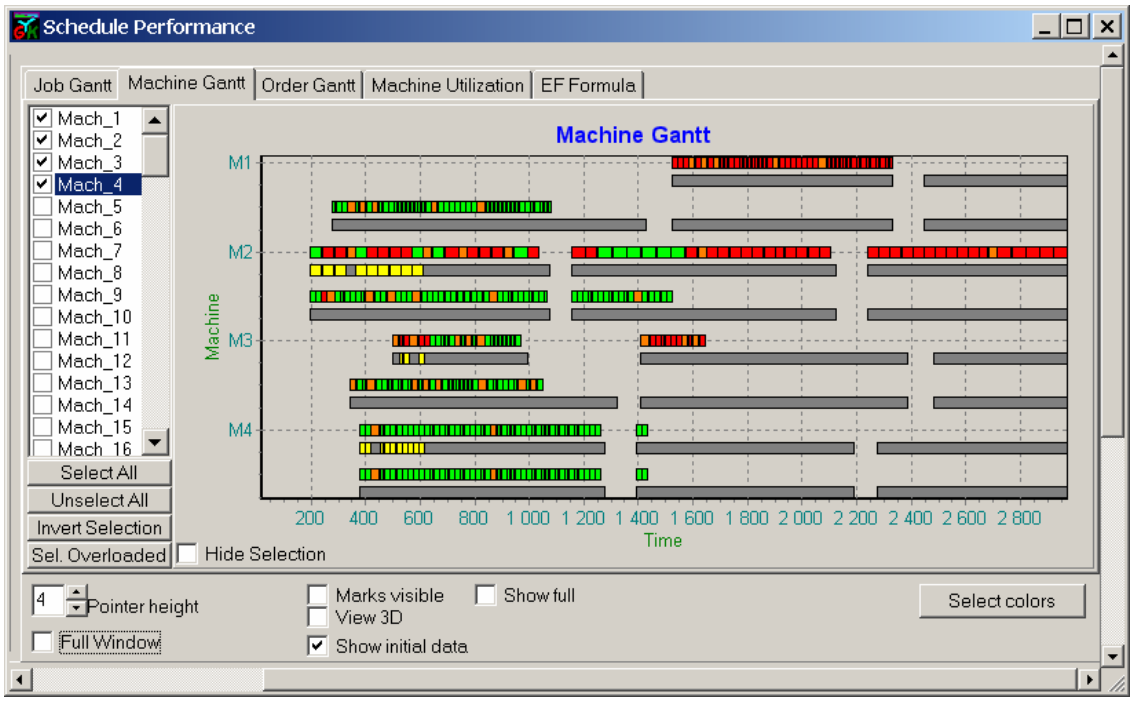



**27. ábra: Gépek attribútumainak megtekintése/megváltoztatása**




**28. ábra: Megrendelések attribútumainak megtekintése/megváltoztatása**

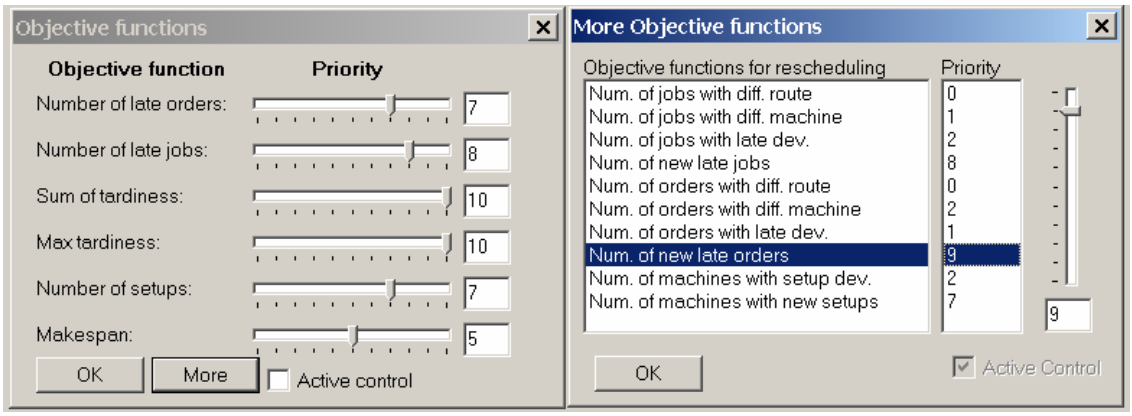
A felhasználó a gyártási folyamat aktuális állapotát és annak következményeit (a korábban bemutatott *Schedule Performance* ablak eszközeinek felhasználásával) összevetheti az előzetesen megtervezettel, melynek eredményeként korrekciós újraütemezést kezdeményezhet.



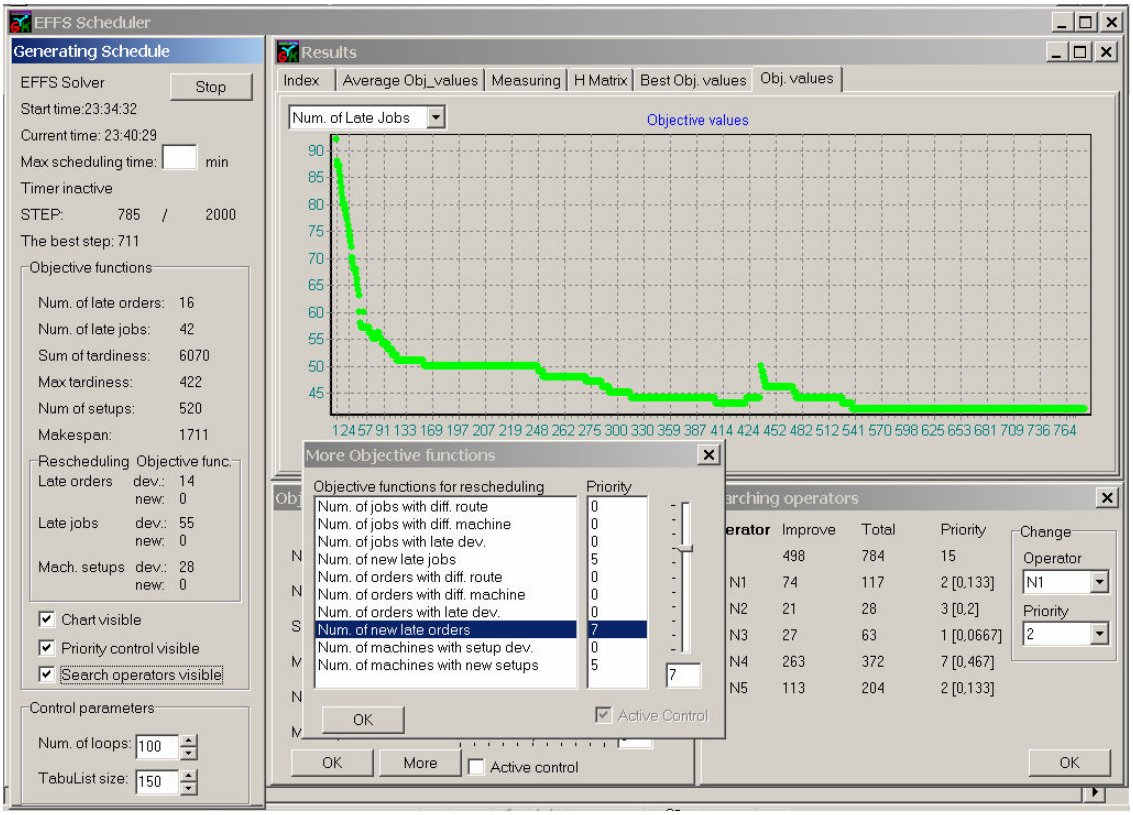

**29. ábra: Újraütemezési szituáció átmeneti gépkiesések következtében**

A 29. ábra például egy olyan szituációt mutat, amely közel egyidőben fellépő különböző váratlan események (M1 gép átmeneti meghibásodása, M2 sebességének lecsökkenése, M3 gép munkarendjének változása) miatt következett be. A Gantt-diagramban minden gép esetében látható a tényleges állapot és annak várható következménye (felső sor), valamint a tervezett tevékenységsorozat (alsó sor).

Az újraütemezés megkezdése előtt a felhasználó az aktuális céloknak megfelelően beállíthatja a célfüggvények prioritásait (30.ábra), az ütemezési paramétereket és a keresőoperátorok prioritásait.




**30. ábra: Az ütemezési és újraütemezési célfüggvények prioritásértékei**




**31. ábra: Az újraütemező működésének vezérlésére szolgáló eszközök**

A megoldási folyamat vezérlésére és az eredmények megtekintésére az előidejű ütemezésnél leírt eszközök állnak a felhasználó rendelkezésére. Ekkor azonban ezek

hatásköre kiterjed a speciális újraütemezési célok (változásorientált célfüggvények) és a korlátozások (zárolás, speciális keresőoperátorok) kezelésére (31. ábra).

#### 6.2.3.2. Üzemmódok

A kifejlesztett szoftver alapvetően háromféle működési módot támogat:

1. Automata üzemmód.
2. Kézi üzemmód.
3. Kombinált (félautomata hibrid) üzemmód.

A szoftver automata üzemmódban az input alapadatok ismeretében felhasználói beavatkozás nélkül – a megadott paraméterek figyelembevételével – képes elkészíteni az új termelési finomprogramot. Előidejű ütemezési feladat megoldása során heurisztikus szabályok alapján készít egy kiindulási ütemtervet, majd keresőalgoritmussal javítja a megoldást a leállási feltétel teljesüléséig (4. fejezet). Újraütemezési feladat megoldása során az aktuális állapotnak megfelelően készít egy kiindulási ütemtervet, a megadott paraméterek szerint elvégzi a szükséges zárolásokat, majd az aktuális céloknak megfelelő új termelési finomprogramot keresőalgoritmussal állítja elő (5. fejezet).

Az alkalmazás rendelkezik olyan felhasználói felülettel, amelynek segítségével az ütemezést végző felhasználó készítheti el az ütemtervet. Az ütemezés során minden munkához a felhasználó rendelheti hozzá annak végrehajtási útvonalát, majd kiválaszthatja a megfelelő gépet az útvonal minden egyes végrehajtási lépésének megfelelő gépcsoportból. Meghatározhatja továbbá a gépeken az egyes munkák sorrendjét. A döntéshozatal vonatkozhat egyetlen munkára, vagy a munkák egy csoportjára, vagy akár teljes megrendelésekre is. A rendszer megjeleníti az egyes döntési helyzetekben a lehetséges alternatívákat, és ezekből a felhasználó választja ki, melyiket kívánja használni. A rendszer valós időben számítja ki és megjeleníti meg az ütemtervben szereplő megrendelések, munkák, gépek adatait, valamint a teljes ütemtervet jellemző célfüggvények, egyéb menedzsermutatók értékeit.

A hibrid ütemezés a kézi és az automatikus ütemezési mód kombinációját jelenti. A felhasználó dönti el, hogy a kiindulási ütemterv hogyan jön létre: kézi ütemezéssel készíti el, heurisztikus szabályok kiválasztása után az automata ütemező hozza létre, vagy e kettő kombinációjából származzon. Ezt követően a felhasználó beállítja a paramétereket és elindítja az automatikus ütemezést. Az erre a célra kifejlesztett felületen keresztül nyomon követi a megoldási folyamatot, közben bármikor aktívan beavatkozhat az ütemezés menetébe a prioritások és paraméterek futás közben történő megváltoztatásával. Megszakíthatja a keresési folyamatot, módosításokat végezhet az ütemtervben, megváltoztathatja az ütemezési paramétereket, továbbfuttathatja a keresést.

A megvalósított szoftver tehát automata, kézi és kombinált üzemmódban, a termelési célfüggvények és mutatók számításával, részletes és áttekinthető grafikus diagramok, táblázatos jelentések készítésével támogatja a gyártásirányító döntéshozást. Fontos jellemzője az alkalmazásnak, hogy megrendelés-, munka-, feladat- és gépszintű konzisztens beavatkozást biztosító interaktív operációkészlettel támogatja a felhasználót az ütemezési és újraütemezési feladatok megoldásában.

# 7. ÚJ TUDOMÁNYOS EREDMÉNYEK

**1. tézis: Új, kiterjesztett rugalmas Flow Shop ütemezési modell és annak számítógépi reprezentációja.**

Ismeretes, hogy a diszkrét termelési folyamatok műhelyszintű prediktív ütemezése mind elméleti, mind gyakorlati szempontból az erősen modellfüggő és komplex feladatok körébe tartozik. A klasszikus „*Flow Shop*" ütemezési feladat formális modellje egy hármassal jellemezhető: $(M, J, f)$ ahol $M$ az erőforrások, $J$ a munkák és $f$ az irányítási célok leírása. Ebben a modellben az optimális ütemterv előállítása már a klasszikus egyutas, előzésmentes esetre is NP-teljes feladat. A napjaink ipari gyakorlatában egyre fontosabbá váló, rugalmas és igény szerinti tömeggyártás irányítása szükségessé teszi az ismert ütemezési modellek további jelentős kiterjesztését.

***Kidolgoztam egy új, kiterjesztett ütemezési feladatosztályt (Extended Flexible Flow Shop, EFFS) és annak számítógépi reprezentációját a rövid távú, műhelyszintű termelésprogramozási feladatok modellezésére, figyelembe véve az igény szerinti tömeggyártás legfontosabb általános jellemzőit és követelményeit.***

Az új, kiterjesztett modellt a következők jellemzik:

1.1 Egyszerre jelennek meg benne: (1) több művelet együttes végrehajtására képes összetett gépek (gépsorok), (2) párhuzamos gépekből funkciók szerint szervezett gépcsoportok, (3) terméktípustól függő végrehajtásiútvonal-alternatívák, (4) géptől és terméktől függő termelési intenzitások, (5) géptől függő változó rendelkezésre állási időintervallumok, (6) munkák sorrendjétől függő gépátállítási idők, (7) munkák indítására és befejezésére vonatkozó időkorlátok.

1.2 Lehetővé teszi a munkák dinamikus kezelését, a rendelések összevonását és/vagy szétbontását. A modellnek ez a tulajdonsága a tömeggyártás „sorozatnagyság" problémájának egy lehetséges értelmezését nyújtja.

1.3 Egyidejűleg több és változó fontosságú optimalizálási célt valamint interaktív tervezői beavatkozásokat is képes kezelni a termelésirányítás rugalmasan változó igényeinek kielégítése érdekében.

Definiáltam az ütemezési modell építőelemeit, részletesen megadtam a hozzájuk tartozó attribútumokat és leírtam azok kapcsolatrendszerét. Kifejlesztettem egy számítógépi implementálásra közvetlenül alkalmas adatmodellt, amelyben indexelt

hivatkozásrendszert alkalmaztam az adatkezelés egyszerűsítése és gyorsítása érdekében. A modell alkalmazhatóságát a számítógépes implementáció teszt-feladatainak elemzése igazolta.

A modellt az értekezés 3. fejezete ismerteti. A modellt a [10], [8], [7], [13], [11], és az [1], [3] publikációkban is részletesen bemutattam.

**2. tézis: Többfázisú heurisztikus módszer az EFFS ütemezési feladatok megoldására.**

Az ismert FFS ütemezési feladatokhoz viszonyítva az 1. tézisben definiált EFFS termelésprogramozási feladatokban a végrehajtási útvonalak és a párhuzamos gépek alternatívái, valamint a dinamikus sorozatnagyság-képzés lehetőségei jelentősen megnövelik a megvalósítható ütemtervek keresési terét. A méretnövekedésnek a feladat megoldása szempontjából nehezítő hatását fokozza a megengedett megoldások terének szerkezetváltozása is. Ez a változás – többek között – a többműveletes végrehajtási lépések, a gépenként változó termelési intenzitások, és az átállítási időtartamok munka-függőségének következménye. További speciális követelményt jelent a gépek terhelhetőségénél egyes korábban ütemezett terhelések figyelembe vétele. Ezek következményeként az EFFS feladatok megoldására az ismert ütemezési módszerek közvetlenül nem alkalmazhatók, szükség van a gyakorlatban előforduló méretű és nehézségű feladatokat megoldó, elfogadható futási idejű új módszerek kifejlesztésére.

***Az EFFS modellel kezelhető termelésprogramozási feladatok megoldására kifejlesztettem egy új szemléletű integrált módszert, amely végrehajtás-szimulációra alapozott problématér-transzformációval, heurisztikus algoritmusok valamint keresési technikák kombinált alkalmazásával egyidejűleg támogatja a rendelések bontására és/vagy egyesítésére, a gyártási sorozatnagyságok dinamikus meghatározására, a technológiai alternatívák kezelésére, a gépi erőforrások allokálására, a gyártási feladatok definiálására és azok végrehajtásának időbeli ütemezésére vonatkozó döntéseket.***

A módszer fontosabb jellemzői:

2.1 A belső rendelések ütemezési alapegységekre bontásával önálló munkák jönnek létre. Ezek ütemezése egymástól függetlenül, végrehajtási lépésekre alapozott feladatok (taszkok) formájában történik (a csomagkapcsolt hálózati adattovábbítás elvének egy speciális alkalmazásával). Ezáltal a gépeken a gyártási sorozatnagyságok és az azokat elválasztó átállítási műveletek dinamikusan, ütemezés közben alakulnak ki.

2.2 A technológiai alternatívák teljes körű rendszerezését és megvalósíthatósági (érvényességi) vizsgálatát követően azok beépülnek az EFFS modellben szereplő építőelemek attribútumai által definiált kapcsolatrendszerbe. Minden egyes döntési helyzetben a választás már csak megengedett megoldást eredményezhet, ezért nincs szükség további megvalósíthatósági vizsgálatra.

2.3 Az EFFS feladatosztályhoz tartozó (gépek aktuális terhelésére és rendelkezésre állására, legkorábbi műveletkezdésre, műveleti sorrendre, rendelések határidejére

vonatkozó) korlátfeltételek és a speciális követelmények (eltérő termelési intenzitások, változó átállítási időtartamok, különböző méretű logisztikai egységek) figyelembevételét egy szabályalapú számítási eljárás végzi, amely egy jól definiált ütemterv adott gyártási környezetben történő végrehajtásának gyors szimulációjára is alkalmas.

2.4 A végrehajtás-szimuláció az ütemtervben szereplő gyártási feladatokhoz, munkákhoz és megrendelésekhez tartozó kezdési, előkészítési, műveleti és befejezési időadatok kiszámításával a problématér transzformációját valósítja meg. Egy egyszerűsített ütemtervhez – amely a munkák és gépek megengedett összerendeléseit és a munkák végrehajtásának gépenkénti sorrendjét tartalmazza – egyértelműen hozzárendel egy megvalósítható, minősített, részletes termelési finomprogramot.

2.5 A hozzárendelési és sorrendi problémákat egy kétfázisú, kombinált heurisztikus módszer oldja meg. Az első fázisban gyors felépítő algoritmusok (BWBA, HEFCA, HIA, EHIA) több „jó" kezdeti megoldást generálnak. A legjobb kezdeti megoldásból kiindulva a második fázisban többféle szomszédsági operátor (N1, N2, N3, N4, N5) kombinált alkalmazásával, többfunkciós tabulista-kezeléssel és folyamatos tanuláson alapuló, paraméteres szabályozással működő, tabukeresési megoldóalgoritmus javítja a megoldást.

A definiált EFFS ütemezési feladatok megoldására tesztadatok és/vagy összehasonlításra alkalmas eredmények a szakirodalomból közvetlenül nem nyerhetők. Az új, integrált ütemezési módszer vizsgálatára ezért egy számítógépi keretrendszert hoztam létre. A rendszer egymástól jól elkülönített funkcionális modulokból áll, így az egyes modulok, a megvalósított funkciók, valamint azok különböző algoritmusai önmagukban is, csoportosan is tesztelhetők, összehasonlíthatók. A módszer helyességét és hatékonyságát – erre a célra fejlesztett problémagenerátorral előállított – különböző méretű és nehézségű tesztfeladatokon elvégzett futási eredmények támasztják alá.

A megoldási módszer részleteit, a kifejlesztett algoritmusokat az értekezés 4. fejezete részletesen ismerteti, valamint a [10], [7], [13], [12], [11], és az [1], [2], [3] publikációkban nyomon követhetők a módszer kifejlesztésének fontosabb fokozatai.

**3. tézis: Új módszer többcélú kombinatorikus optimalizálási feladat megengedett megoldásainak egymáshoz viszonyított minősítésére.**

Ismeretes, hogy a többcélú, kombinatorikus optimalizálási feladatoknak csak kivételes esetekben van olyan megoldása, amely az összes célfüggvény szempontjából egyszerre tekinthető optimálisnak. Kompromisszumos megoldás értelmezéséhez és megtalálásához szükséges annak definiálása, hogy milyen szempontok szerint és hogyan hasonlíthatók össze a lehetséges megoldások. Léteznek különböző elvek és módszerek erre a célra, azonban dinamikus rendszerekben ezek alkalmazása – főként a paraméterezés bonyolultsága miatt – nehézkes, így indokolt új, rugalmasan, könnyen adaptálható módszerek kifejlesztése.

***Matematikai modellt és megoldási módszert dolgoztam ki a többcélú kombinatorikus optimalizálási feladatokban előírt – dinamikusan változó fontosságú, különböző dimenziójú és értékkészletű – célfüggvények együttes figyelembevételére és a megengedett megoldások egymáshoz viszonyított (relatív) minőségének számszerűsítésére.***

A módszer többféle feladat-megoldási stratégiában is jól felhasználható az aktuális céloknak legjobban megfelelő megoldás megtalálására. A módszert – termelési finomprogramok minőségének összehasonlítására – objektumorientált elvek alapján, relációs operátorok újraértelmezésével implementáltam. Kimutattam, hogy a módszer széles problémakörben felhasználható ismert keresési algoritmusok (metaheurisztikák) támogatására, kombinatorikus többcélú optimálási feladatok megoldására.

A módszert és annak tulajdonságait az értekezés 4.3.2. fejezete részletesen ismerteti, valamint az [1], [3], [12] és a [4] publikációkban is bemutattam.

**4. tézis: Integrált termelésprogramozási modell a rugalmas tömeggyártásban fellépő változások és zavarok kezelésének támogatására.**

A rugalmas tömeggyártásban fellépő műhelyszintű termelésirányítási feladatok esetében a hagyományos MES alkalmazások az igények egy részét nem tudják maradék nélkül kielégíteni. Hatékony, zárt (visszacsatolt) irányítás (szabályozás) megvalósítása érdekében szükség van a tervezett és a tényleges termelésirendszer-állapot ismételt összevetésére és szükség esetén korrekciós beavatkozások kezdeményezésére. A kívánt állapotot a mindenkori érvényes termelési finomprogram képviseli, amely a termelés teljesítményét mérő mutatók előidejű optimalizálásával készült. Ehhez szükséges, hogy az ütemezőben rendelkezésre álljon egy gyors termelésifolyamat-szimulátor. Az éles finomprogram valós termelési teljesítményét mérő mutatók a MES rendszerből visszacsatolhatók és összevethetők a tervezettekkel. Az értékelés egyik lehetséges következményeként a hatékony beavatkozás érdekében újraütemezésre kerülhet sor, amelyet a ütemező megoldó heurisztikája és/vagy egy interaktív kézi ütemezést biztosító szolgáltatás támogathat.

***A MES szintű gyártásirányítás és bizonytalanságkezelés ütemezési és újraütemezési feladatainak támogatására kifejlesztettem egy új, integrált modellt és kidolgoztam annak egy számítógépi (szoftver) reprezentációját. A megoldás alkalmas az igény szerinti tömeggyártás műhelyszintű termelésirányítási feladatainak integrált, kiterjesztett funkciójú kezelésére.***

A modellt a következők jellemzik:

4.1 Az előidejű ütemezésre használt megoldási módszer továbbfejlesztett, kibővített formában újraütemezési feladatok megoldására is alkalmas. Ez kezdeti megoldásnak az eredetileg érvényben lévő ütemtervet vagy annak egy interaktív szerkesztésű következményét tekinti. A minősítés kritériumai között ilyenkor megjelennek olyan új elemek is, amelyek kifejezik az ütemtervváltozásokkal szemben támasztott speciális igényeket. Ilyenek lehetnek egyedi (pl. egyes kiemelt gépek átállítására, rendelések késésének engedélyezésére vonatkozó)

illetve összesített (pl.: megváltozott allokációk vagy indítási sorrendek számának minimalizálására vonatkozó) elvárások is.

4.2 A termelés rövid távú programozására valamint a termelési bizonytalanságokból és zavarokból keletkező hibák elhárítására (csökkentésére) funkcionális komponensekből felépített termelésprogramozó szoftvert hoztam létre. Az alkalmazás funkcionális modelljét, és a termelésinformatikai rendszerben betöltött szerepét a 14. ábra vázolja.

4.3 A szoftver új, többfunkciós koncepciót valósít meg a következő fő komponensekkel: (1) többfunkciós paraméterkezelő, (2) modellépítő komponens, (3) heurisztikán és keresési metaheurisztikán alapuló feladatmegoldó ütemező és újraütemező (interaktív grafikus felülettel a felhasználói beavatkozás támogatására), (4) gyors termelésifolyamat-szimulátor a taszkok időadatainak meghatározására és (5) termelési teljesítménymutatókat számító és minősítő komponens.

4.4 A modellt megvalósító szoftver a gyártásirányító vezetői döntéshozást automata, kézi és kombinált üzemmódban, termelési célfüggvények és mutatók számításával, részletes és áttekinthető grafikus diagramok, táblázatos jelentések készítésével támogatja.

4.5 Fontos jellemzője az alkalmazásnak, hogy megrendelés-, munka-, feladat- és gépszintű konzisztens beavatkozást biztosító interaktív operációkészlettel támogatja a felhasználót az ütemezési és újraütemezési feladatok megoldásában, ezáltal lehetővé teszi a természetes (felhasználói) intelligencia és a megoldási módszerekben realizált mesterséges (számítógépi) intelligencia kombinált alkalmazását.

A kidolgozott modell és a megoldóalgoritmusok alapján megterveztem és C++ Builder szoftverfejlesztő környezetben objektumorientált módon megvalósítottam a többfunkciós, integrált termelésprogramozó szoftver prototípusát, amely alkalmas: (1) Többműveletes, EFFS típusú termelésirányítási feladatok többkritériumos optimumközeli ütemezésére. (2) A tervezett és a megvalósult termelési célok összevetésére. (3) Viselkedés alapú beavatkozás támogatójaként a feladatok újrafogalmazására. (4) A feladatok újraütemezésére. A szoftver alkalmazhatóságát próbafeladatokon végrehajtott tesztek eredményei támasztják alá.

A modell jellemzőit, a szoftver felépítésének részleteit, az alkalmazott információs technológiát, valamint az interaktív felhasználói felület tulajdonságait az értekezés 5. és 6. fejezete részletesen ismerteti. A legfontosabb jellemzők az [1], [2], [3], [7], [12], [11] és a [4] publikációkban is bemutatásra kerültek.

# 8. AZ EREDMÉNYEK HASZNOSÍTHATÓSÁGA

A Miskolci Egyetem Alkalmazott Informatikai Tanszékén Erdélyi Ferenc vezetésével végzett kutatómunkám egyik feladata volt hozzájárulni az MTA SZTAKI vezette *„VITAL" Valós idejű, kooperatív vállalatok* (NKTH 2/010/2004) kutatási projekt *Integrált kapacitástervezés, erőforrás-menedzsment és ütemezés* klasztere keretében folyó kutatási és fejlesztési munka támogatásához. Az értekezésben összefoglalt eredmények (ütemezési modell, megoldási koncepció, az alkalmazott módszerek és a megvalósított funkciók) szóbeli prezentáció és írásos kutatási jelentések formájában kerültek belső felhasználásra.

Az értekezésben összefoglalt eredmények a rugalmas tömeggyártás más területein is hasznosíthatók. Mintafeladatokon produkált futási eredmények azt mutatják, hogy a kifejlesztett EFFS ütemezési modell és annak megoldómódszerei alkalmasak a definiált feladatosztályba sorolható sokféle ütemezési feladat megoldására. A megvalósított szoftverprototípus adatbáziskezelő rendszerekhez (DBMS) kapcsolódhat, ezáltal termelésinformatikai rendszerekbe ágyazható. Gyakorlati alkalmazásához szükség lehet a konkrét termelési adatbázis-struktúrához illesztő (adatkonverziós) modul módosítására, a felmerülő igényeknek megfelelően további célfüggvények, termelési mutatók definiálására, valamint a konkrét gyártó-szerelő rendszer speciális jellemzőinek és igényeinek megfelelően a szimulátor módosítására, esetleg továbbfejlesztésére.

Az ütemező szoftver könnyen kezelhető grafikus felülete, beépített probléma-generátora, valamint eredménykijelző és -értékelő szolgáltatásai lehetővé teszik a felsőfokú oktatásban való felhasználását is (pl.: ME Alkalmazott Informatikai Tanszék által oktatott „Számítógépes gyártásirányítás", „Számítógépes termelésirányítás", „Termelési rendszerek és folyamatok" és „Számítógéppel integrált gyártás" c. tárgyak laborgyakorlatainak keretein belül).

A többcélú optimalizálási feladat megoldásainak egymáshoz viszonyított (relatív) minőségének számszerűsítésére kifejlesztett ( a harmadik tézisben összefoglalt) matematikai modell és megoldási módszer – termelésütemezési problémáktól független tulajdonsága következtében – széles feladatkörben felhasználható ismert keresési algoritmusok és metaheurisztikák támogatására, kombinatorikus többcélú optimalizálási feladatok megoldására.

# 9. TOVÁBBI KUTATÁSI FELADATOK

A továbbfejlesztés egyik iránya lehet az EFFS modell bővítése. Ez vonatkozhat egyrészt a modell specializálására, finomítására további részletek kibontásával, modellezésével és beépítésével. Ilyenek lehetnek például a gyártási folyamat során felhasznált anyagokkal, beépülő komponensekkel és azok alternatíváival mint nem megújuló erőforrásokkal kapcsolatos kérdések részletes vizsgálata, korlátozott méretű átmeneti tárolók, anyagmozgatási idők figyelembevétele és hatásuk vizsgálata. Általános irányú funkcionális bővítés is kitűzhető cél lehet, melynek során a modellel kezelhető ütemezési feladatok körét tovább lehet szélesíteni. Ilyenek lehetnek például a művelet-végrehajtási (pl. megszakíthatóságra vonatkozó) jellemzők változtatásából származó feladatok, a műveleti sorrendekre vonatkozó előírások általánosításából eredő („job shop”, „open shop” vagy „general shop” típusú) feladatok kezelésére irányuló kutatások.

A továbbfejlesztés egy másik lehetséges irányát jelentheti a tisztán determinisztikus előidejű ütemezési (proactive) és újraütemezési (reactive) feladok vizsgált körének bővítése sztochasztikus ütemezési feladatok irányába, adott eloszlások szerint véletlenszerűen generált elemek (pl. időközben beérkező megrendelések, ütemezés közben megváltozó műveleti és rendelkezésre állási idők, egyéb jellemzők) bevonásával.

További kutatási feladatot jelenthetnek a szimulációt megvalósító, a megoldást előállító és a kiértékelést végző algoritmusok sebességének növelésére (összességében az adott minőségű eredmények előállításához szükséges futási idő csökkentésére, vagy adott futási időkorlát mellett elérhető még jobb minőségű eredmények előállítására) irányuló törekvések.

Hosszú távú fejlesztési célként említhető meg az ütemező szoftver és további (meglévő vagy fejlesztés alatt álló) alkalmazások felhasználásával az ME AIT termelésinformatikai laboratóriumában virtuális körülmények között működő teljes funkcionalitású gyártásirányító rendszer kialakítása kutatási és oktatási céllal.

# 10. AZ ÉRTEKEZÉS TÉMAKÖRÉBEN KÉSZÍTETT SAJÁT PUBLIKÁCIÓK

Angol nyelven megjelent tudományos közlemények:


[1] **KULCSÁR, GY.**, ERDÉLYI, F., 2007, *A New Approach to Solve Multi-Objective Scheduling and Rescheduling Tasks,* International Journal of Computational Intelligence Research, Vol. 3, No. 4, (megjelenés alatt).

[2] TÓTH, T., ERDÉLYI, F., **KULCSÁR, GY.**, 2008, *Decision Supporting of Production Planning and Control by means of Key Production Performance Measuring Indicators,* Seventh International Symposium on Tools and Methods of Competitive Engineering, TMCE 2008, April 21–25, 2008, Kusadasi, Turkey, (közlésre elfogadva).

[3] **KULCSÁR, GY.**, ERDÉLYI, F., HORNYÁK, O., 2007, *Multi-Objective Optimization and Heuristic Approaches for Solving Scheduling Problems,* IFAC Workshop on Manufacturing Modeling, Management and Control, MIM 2007, Budapest, November 14-16, 2007, pp. 127-132.

[4] **KULCSÁR, GY.**, ERDÉLYI, F., 2007, *Multi-Objective Searching Methods for Solving Scheduling and Rescheduling Problems,* microCAD 2007 International Scientific Conference, Miskolc, Hungary, pp. 133-138.

[5] HORNYÁK, O., ERDÉLYI, F., **KULCSÁR, GY.**, 2006, *Behaviour-Based Control for Uncertainty Management in Manufacturing Execution Systems*, Proceedings of the 8th International Conference on The Modern Information Technology in the Innovation Processes of the Industrial Enterprises, MITIP 2006. September 11-12, 2006, Budapest, Hungary, pp. 73-79.

[6] HORNYÁK, O., ERDÉLYI, F., **KULCSÁR, GY.**, 2006, *Detailed Scheduling and Uncertainty Management in Customized Mass Production*, 12th International Conference on Machine Design and Production, Kusadasi, Turkey, pp. 423-439.

[7] **KULCSÁR, GY.**, ERDÉLYI, F., 2006, *Modeling and Solving of the Extended Flexible Flow Shop Scheduling Problem,* Production Systems and Information Engineering, A Publication of the University of Miskolc, Vol. 3, Miskolc, Hungary, pp. 121-139.

[8] **KULCSÁR, GY.**, ERDÉLYI, F., 2006, *Extended Flexible Flow Shop Scheduling Problem with Limited Machine Availability,* microCAD 2006 International Scientific Conference, Miskolc, Hungary, pp. 181-186.

[9] **KULCSÁR, GY**., ERDÉLYI, F., 2005, *Production Goals and Heuristics Methods in Extended Flexible Flow Shop Scheduling Problems,* Forum of PhD Students, Miskolc, Hungary, pp. 104-109.

[10] **KULCSÁR, GY**., HORNYÁK, O., ERDÉLYI, F., 2005, *Shop Floor Decision Supporting and MES Functions in Customized Mass Production,* Machine Engineering, Vol. 5. pp. 138 - 152.

Magyar nyelven megjelent tudományos közlemények:

[11] **KULCSÁR, GY**., 2006, *Az igény szerinti tömeggyártás termelés-programozásának heurisztikus megoldási módszere,* Gyártás-2006 Magyar Tudományos Konferencia, pp. 130-140.

[12] **KULCSÁR, GY**., 2006, *Az igény szerinti tömeggyártás termelés-programozásának egy hibrid megoldási módszere,* Doktoranduszok Fóruma, Miskolc, pp. 104-111.

[13] **KULCSÁR, GY**., 2006, *Az igény szerinti tömeggyártás termelés-programozásának modellezése és heurisztikus megoldása,* GÉP, A Gépipari Tudományos Egyesület Műszaki Folyóirata, LVII. évfolyam, 2006/10 szám, pp. 3-13.

[14] **KULCSÁR, GY**., ERDÉLYI, F., 2006, *Kiterjesztett rugalmas Flow Shop ütemezési feladat modellezése és megoldása,* Fiatal Műszakiak Tudományos Ülésszaka, Kolozsvár, Románia, pp. 223- 226.

[15] HORNYÁK, O., ERDÉLYI, F., **KULCSÁR, GY**., 2005, *Valós idejű gyártásirányítás (MES) funkciók fejlődése, modellek és módszerek*, Informatika a felsőoktatásban 2005, CD, Debrecen.

[16] **KULCSÁR, GY**., ERDÉLYI, F., 2004, *Technológiai alternatívák hatása rövid távú termelés ütemezési feladatok megoldására,* Doktoranduszok Fóruma, Miskolc, pp. 173-178.

# 11. HIVATKOZOTT IRODALOM


[17] ACERO-DOMINGUEZ, M., J., PATERNINA-ARBOLEDA, C., D., 2004, *Scheduling Jobs on a K-Stage Flexible Flow Shop Using a TOC-Based (Bottleneck) Procedure,* Proceedings of the 2004 Systems and Information Engineering Design Symposium, pp. 295-298.

[18] AGARWAL, A., COLAK, S., ERYARSOY, E., 2006, *Improvement Heuristic for the Flow-Shop Scheduling Problem: An Adaptive-Learning Approach,* European Journal of Operational Research, Vol. 169, No. 3, pp. 801-815.

[19] AHR, D., BÉKÉSI, J., GALAMBOS, G., OSWALD, M., REINELT, G., 2004, *An Exact Algorithm for Scheduling Identical Coupled Task Problems,* Mathematical Methods of Operations Research, Vol. 59, No. 2, pp. 193-203.

[20] ALCARAZ, J., MAROTO, C., RUIZ, R,, 2006, *Two New Robust Genetic Algorithms for the Flowshop Scheduling Problem,* Omega, Vol. 34, pp. 461-476.

[21] ALISANTOSO, D., KHOO, L., P., JIANG, P., Y., 2003, *An Immune Algorithm Approach to the Scheduling of a Flexible PCB Flow Shop,* The International Journal of Advanced Manufacturing Technology, Vol. 22, No. 11-12, pp. 819-827.

[22] ALLAHVERDI, A., GUPTA, J., N., D., ALDOWAISAN, T., 1999, *A Review of Scheduling Research Involving Setup Considerations,* Omega, Vol. 27, No. 2, pp. 219-239.

[23] ALLAHVERDI, A., NG, C., T., CHENG, T., C., E., KOVALYOV, M., Y., 2007, *A Survey of Scheduling Problems with Setup Times or Costs,* European Journal of Operational Research, In Press, Corrected Proof, Available online 13 November 2006.

[24] ALLAOUI, H., ARTIBA, A., 2006, *Scheduling Two-Stage Hybrid Flow Shop with Availability Constraints,* Computers and Operations Research, Vol. 33, pp. 1399-1419.

[25] ARTHANARI, T., S., RAMAMURTHY, K., G., 1971, *An Extension of Two Machines Sequencing Problem,* Journal of Operational Research, Vol. 41, pp. 641-648.

[26] ASKIN, G., A., STANDRIDGE, C., R., 1993, *Modeling and Analysis of Manufacturing Systems,* John Wiley and Sons Inc., New York.

[27] AYTUG, H., LAWLEY, M., A., MCKAY, K., MOHAN, S., UZSOY, R., 2005, *Executing Production Schedules in the Face of Uncertainties: A Review and*

*some Future Directions,* European Journal of Operational Research, Vol. 161, pp. 86-110.

[28] BLAZEWICZ, J., BREIT, J., FORMANOWICZ, P., KUBIAK, W., SCHMIDT, G., 2001, *Heuristic Algorithms for Two-Machine Flowshop with Limited Machine Availability,* Omega, Vol. 29, pp. 599-608.

[29] BLAZEWICZ, J., PESCH, E., STERNA, M., WERNER, F., 2004, *Open Shop Scheduling Problems with Late Work Criteria,* Discrete Applied Mathematics, Vol. 134, No. 1-3, pp. 1-24.

[30] BLAZEWICZ, J., PESCH, E., STERNA, M., WERNER, F., 2005, *A Comparison of Solution Procedures for Two-Machine Flow Shop Scheduling with Late Work Criterion,* Computers and Industrial Engineering, Volume 49, Issue 4, pp. 611-624.

[31] BOTTA-GENOULAZ, V., 2000, *Hybrid Flow Shop Scheduling with Precedence Constraints and Time Lags to Minimize Maximum Lateness,* International Journal of Production Economics, Vol. 64, No. 1-3, pp. 101-111.

[32] BRAH, S., A., HUNSUCKER, J., L., 1991, *Branch-and-Bound Algorithm for the Flow Shop with Multiple Processors,* European Journal of Operations Research, Vol. 51, pp. 88-99.

[33] BRAH, S., A., LOO, L., L., 1999, *Heuristics for Scheduling in a Flow Shop with Multiple Processors,* European Journal of Operational Research, Vol. 113, No. 1, pp. 113-122.

[34] BRUCKER, P., 1998, *Scheduling Algorithms,* Secong, Revised and Enlarged Edition, Springer-Verlag, Berlin.

[35] BRUCKER, P., KNUST, S., 2006, *Complexity Results for Scheduling Problems,* http://www.mathematik.uni-osnabrueck.de/research/OR/class/

[36] CAMPBELL, H., G., DUDEK, R., A., SMITH, M., L, 1970, *A Heuristic Algorithm for the n-Job m-Machine Sequencing Problem,* Management Science, Vol. 16, No. 10, pp. 630-637.

[37] CHANG, S., C., LIAO, D., Y., 1994, *Scheduling Flexible Flow Shops with No Setup Effects,* IEEE Transactions on Robotics and Automation, Vol. 10, No. 2, pp. 112-122.

[38] CHEN, C., L., VENKATESWARA, S.,V., ALJABER, N., 1995, *An Application of Genetic Algorithms for Flow Shop Problems,* European Journal of Operational Research, Vol. 80, pp. 389-396.

[39] CHENG, T., C., E., GUPTA, J., N., D., WANG, G., 2000, *A Review of Flowshop Scheduling Research with Setup Times,* Production and Operations Management, Vol. 8, No. 3, pp. 262-282.

[40] CHENG, T., C., E., WANG, G., 2000, *An Improved Heuristic for Two-Machine Flowshop with an Availability Constraint,* Operation Research Letters, Vol. 26, pp. 223-229.

[41] CHROBAK, M., CSIRIK, J., IMREH, CS., NOGA, J., SGALL, J., WOEGINGER, G., 2001, *The Buffer Minimization Problem for Multiprocessor Scheduling with Conflicts,* Proceedings of ICALP 2001, LNCS 2076, pp. 862-874.

[42] CRESCENZI, P., KANN, V., 2005, *A Compendium of NP Optimization Problems,* http://www.nada.kth.se/~viggo/problemlist/

[43] CSÁJI, B., CS., MONOSTORI, L., 2005, *Stochastic Reactive Production Scheduling by Neurodynamic Learning,* 16th IFAC World Congress, July 4-8, 2005, Prague, Czech Republic, (CD version is available).

[44] CSÁJI, B., CS., MONOSTORI, L., 2005, *Stochastic Reactive Production Scheduling by Multi-Agent Based Asynchronous Approximate Dynamic Programming,* Lecture Notes in Artificial Intelligence, Springer, Vol. 3690, Proceedings of the 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005), September 15-17, 2005, Budapest, Hungary, pp 388-397.

[45] CSÉBFALVI, G., 2000, *A New Exact Resource Leveling Procedure for the Multiple Resource Constrained Project Scheduling Problem,* Proc. North American Productivity Workshop, Jun 15-21, 2000, Schenectady, New York, USA.

[46] CSÉBFALVI, G., KONSTANTINIDIS, P., 1998, *Egy implicit leszámláláson alapuló új erőforrás kiegyenlítő eljárás,* Szigma, Vol. 29, pp. 43-52.

[47] DAYA, B., M., AL-FAWZAN, M., 1998, *A Tabu Search Approach for the Flow Shop Scheduling Problem,* European Journal of Operational Research, Vol. 109, pp. 88-95.

[48] ERDÉLYI, F., 2004, *Technológiai folyamatok automatizálásának alapjai,* TÓTH, T., *Termelési rendszerek és folyamatok* 11. fejezete, p. 305-384, Miskolci Egyetemi Kiadó.

[49] ERDÉLYI, F., TÓTH, T., 2006, *Decision Supporting of Production Planning and Control (PPC) by Means of Rate-Type State Variables,* Sixth International Symposium on Tools and Methods of Competitive Engineering, TMCE 2006, April 18-22, Ljubljana, (under review).

[50] FRAMINAN, J., M., GUPTA, J., N., D., LEISTEN, R., 2002, *A Review and Classification of Heuristics for Permutation Flowshop Scheduling with Makespan Objective,* Technical Report OI/PPC-2001/02, Version 1.2 – 20/07/2002 (http://taylor.us.es/cicyt/reports/TROI-2001-02.pdf).

[51] FRAMINAN, J., M., LEISTEN, R., RAJENDRAN, C.,, 2003, *Different Initial Sequences for the Heuristic of Nawaz, Enscore and Ham to Minimize Makespan, Idletime or Flowtime in the Static Permutation Flowshop Sequencing Problem,* International Journal of Production Research, Vol. 41, No. 1, pp. 121-148.

[52] GAREY, M., R., JOHNSON, D., S., SETHI, R., 1976, *The Complexity of Flowshop and Jobshop Scheduling,* Mathematics of Operations Research, Vol. 1, No. 2, pp. 117-129.

[53] GEIGER, M., J., 2006, *Foundations of the Pareto Iterated Local Search Metaheuristic,* Proceedings of the 18th International Conference on Multiple Criteria Decision Making, June 19-23, 2006, Chania, Greece. http://www.dpem.tuc.gr/fel/mcdm2006/Papers/Geiger.pdf

[54] GRABOWSKI, J., WODECKI, M., 2002, *A Very Fast Tabu Search Algorithm for the Permutation Flow Shop Problem with Makespan Criterion,* Research Report PRE 64/2002, Institute of Engineering Cybernetics, Technical University of Wroclaw, Wroclaw, Poland.

[55] GUPTA, J., N., D., 1971, *A Functional Heuristic Algorithm for the Flow Shop Scheduling Problem,* Operational Research Quarterly, Vol. 22, pp. 39-47.

[56] GUPTA, J., N., D., 1979, *An Improved Lexicographic Search Algorithm for the Flow Shop Scheduling Problem,* Computers and Operational Research, Vol. 6, No. 2, pp. 117-120.

[57] GUPTA, J., N., D., 1988, *Two-stage, hybrid flow shop scheduling problem,* The Journal of the Operational Research Society, Vol. 39, No. 4, pp. 359-364.

[58] GUPTA, J., N., D., DARROW, W., P., 1986, *The Two-Machine Sequence Dependent Flow Shop Scheduling Problem,* European Journal of Operational Research, Vol. 24, No. 3, pp. 439-446.

[59] GUPTA, J., N., D., HARIRI, A., M., A., POTTS, C., N., 1997, *Scheduling a Two-Stage Hybrid Flowshop with Parallel Machines at the First Stage,* Annals of Operations Research, Vol. 69, pp. 171-191.

[60] GUPTA, J., N., D., KRÜGER, K., LAUFF, V., WERNER, F., SOTSKOV, Y., N., 2002, *Heuristics for hybrid flow shops with controllable processing times and assignable due dates,* Computers & Operations Research, Vol. 29, No. 10, pp. 1417-1439.

[61] GUPTA, J., N., D., TUNC, E., A., 1991, *Schedules for a Two-Stage Hybrid Flowshop with Parallel Machines at the Second Stage,* International Journal of Production Research, Vol. 29, No. 7, pp. 1489-1502.

[62] GUPTA, J., N., D., TUNC, E., A., 1994, *Scheduling a Two-Stage Hybrid Flowshop with Separable Setup and Removal Times,* European Journal of Operational Research, Vol. 77, pp. 415-428.

[63] HOLCZINGER, T., ROMERO, J., PUIGJANER, L., FRIEDLER, F., 2002, *Scheduling of Multipurpose Batch Processes with Multiple Batches of the Products,* Hungarian Journal of Industrial Chemistry, Vol. 30, pp. 305-312.

[64] HONG, T., P., WANG, T., T., 1999, *Heuristic Palmer-Based Fuzzy Flexible Flow-Shop Scheduling Algorithm,* IEEE International Fuzzy Systems Conference Proceedings, FUZZ-IEEE'99, New York: IEEE, Vol. 3, pp. 1493-1497.

[65] HONG, T., P., WANG, T., T., 2000, *Fuzzy Flexible Flow Shops at Two Machine Centers for Continuous Fuzzy Domains,* Information Sciences, Vol. 129, No. 1-4, pp. 227-237.

[66] HONG, T., P., WANG, T., T., WANG, S., L., 2001, *A Palmer-Based Continuous Fuzzy Flexible Flow-Shop Scheduling Algorithm,* Soft Computing, Vol. 5, No. 6, pp. 426-433.

[67] HOOGEVEEN, J., A., LENSTRA, J., K., VELTMAN, B., 1996, *Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard,* European Journal of Operational Research, Vol. 89, No. 1-2, pp. 172-175.

[68] HORNYÁK, O., ERDÉLYI, F., 2004, *Manufacturing Execution System for Advanced Shop Floor Control,* Proceedings of The Eleventh International Conference on Machine Design and Production, October 13-15, Antalya, Turkey. pp. 48-52.

[69] HORNYÁK, O., ERDÉLYI, F., 2005, *Development of MES Functionalities for Supporting Management Decisions,* MicroCAD'2005 International Computer Science Conference, 10-11 March 2005, Miskolc, Hungary, pp. 134-142.

[70] HORNYÁK, O., ERDÉLYI, F., 2006, *Uncertainty Management in Manufacturing Execution Systems,* MicroCAD'2006 International Computer Science Conference, 16-17 March 2006, Miskolc, Hungary, pp. 295-303.

[71] IMREH, CS., 2001, *Scheduling on a Two Layer Multiprocessor Architecture,* Acta Cybernetica, Vol. 15, pp. 163-172.

[72] IMREH, CS., 2003, *Scheduling Problems on Two Sets of Identical Machines,* Computing, Vol. 70, pp. 277-294.

[73] JOHNSON, S., M., 1954, *Optimal Two- and Three-Stage Production Schedules with Setup Times Included,* Research Logistics Quarterly, Vol. 1, pp. 61-68.

[74] JUNGWATTANAKI, J., REODECHA, M., CHAOVALITWONGSE, P., WERNER, F., 2005, *An Evaluation of Sequencing Heuristics for Flexible Flowshop Scheduling Problems with Unrelated Parallel Machines and Dual Criteria,* Preprint 2005-28, Mathematics Preprint Servers, Otto-von-Guericke-University, Magdeburg, Germany.

[75] JUNGWATTANAKI, J., REODECHA, M., CHAOVALITWONGSE, P., WERNER, F., 2006, *Algorithms for Flexible Flow Shop Scheduling Problems with Unrelated Parallel Machines, Setup Times, and Dual Criteria,* Preprint 2006-26, Mathematics Preprint Servers, Otto-von-Guericke-University, Magdeburg, Germany.

[76] JUNGWATTANAKI, J., REODECHA, M., CHAOVALITWONGSE, P., WERNER, F., 2006, *Sequencing Algorithms for Flexible Flow Shop Problems with Unrelated Parallel Machines, Setup Times and Dual Criteria,* Preprint 2006-49, Mathematics Preprint Servers, Otto-von-Guericke-University, Magdeburg, Germany.

[77] KÁDÁR, B., MONOSTORI, L., CSÁJI, B., 2005, *Adaptive Approaches to Increasing the Performance of Production Control Systems,* CIRP Journal of Manufacturing Systems, Vol. 34, No. 1, pp. 349-352.

[78] KÁDÁR, B., PFEIFFER, A., MONOSTORI, L., 2005, *Stability-oriented Evaluation of Rescheduling Strategies by using Simulation,* Computers in Industry, Elsevier, (paper accepted in 2006).

[79] KIS T., PESCH, E., 2005, *A Review of Exact Solution Methods for the Non-Preemptive Multiprocessor Flowshop Problem,* European Journal of Operational Research, Vol. 164, No. 3, pp. 573-695.

[80] KIS, T., ERDŐS, G., 2005, *Specification: Automatic Scheduling System with Basic Scheduling Engine for GE Lighting,* (VITAL project NKTH 2/010/2004: internal report).

[81] KOCHAR, S., MORRIS, R., J., T., 1987, *Heuristic Methods for Flexible Flow Line Scheduling,* Journal of Manufacturing Systems, Vol. 6, No. 4, pp. 299-314.

[82] KOVÁCS, A., EGRI, P., KIS, T., VÁNCZA, J., 2005, *Proterv-II: An integrated Production Planning and Scheduling System,* In: Principles and Practice of Constraint Programming, Proc. of CP2005, (ed.: van Beek, P.,), Springer LNCS.

[83] KURZ, M., E., ASKIN, R., G., 2001, *An Adaptable Problem-Space-Based Search Method for Flexible Flow Line Scheduling,* IIE Transactions, Vol. 33, No. 8, pp. 691-693.

[84] KURZ, M., E., ASKIN, R., G., 2003, *Comparing Scheduling Rules for Flexible Flow Lines,* International Journal of Production Economics, Vol. 85, No. 3, pp. 371-378.

[85] KURZ, M., E., ASKIN, R., G., 2004, *Scheduling Flexible Flow Lines with Sequence-Dependent Setup Times,* European Journal of Operational Research, Vol. 159, No. 1, pp. 66-82.

[86] LEE, C., Y., 1997, *Minimizing the Makespan in the Two-Machine Flowshop Scheduling Problem with an Availability Constraint,* Operation Research Letters, Vol. 20, pp. 129-139.

[87] LEE, I., SIKORA, R., SHAW, M., J., 1997, *A Genetic Algorithm-Based Approach to Flexible Flow-Line Scheduling with Variable Lot Sizes,* IEEE Transactions on Systems, Man and Cybernetics, Vol. 27, No. 1, pp. 36-54.

[88] LENGYEL, A., ERDÉLYI, F., 2006, *Behaviour Based Combined Approaches to Uncertainty Management in Manufacturing Systems,* Proceedings of the 6th International Workshop on Emergent Synthesis, IWES'06, 18-19 August, 2006, Kashiwa, Japan, pp. 77-82.

[89] LEON, V.,J., RAMAMOORTHY, B., 1997, *An Adaptable Problem-Space-Based Search Method for Flexible Flow Line Scheduling,* IIE Transactions, Vol. 29, No. 2, pp. 115-125.

[90] LIN, H., T., LIAO, C., J., 2003, *A Case Study in a Two-Stage Hybrid Flow Shop with Setup Time and Dedicated Machines,* International Journal of Production Economics, Vol. 86, No. 2, pp. 133-143.

[91] LINN, R., ZHANG, W., 1999, *Hybrid Flow Shop Scheduling: A Survey,* Computers and Industrial Engineering, Vol. 37, No. 1-2, pp. 57-61.

[92] LIU, C., Y., CHANG, S., C., 2000, *Scheduling Flexible Flow Shops with Sequence-Dependent Setup Effects,* IEEE Transactions on Robotics and Automation, Vol. 16, No. 4, pp. 408-419.

[93] LOMINCKI, Z., A., 1965, *A Branch and Bound Algorithm for the Exact Solution of the Three-Machine Scheduling Problem,* Operational Research Quarterly, Vol. 16, pp. 89-100.

[94] LOUKIL, T., TEGHEM, J., TUYTTENS, D., 2005, *Solving Multi-Objective Production Scheduling Problems Using Metaheuristics,* European Journal of Operational Research, Vol. 161, pp. 42-61.

[95] MAJOZI, T., FRIEDLER, F., 2006, *Maximization of Throughput in a Multipurpose Batch Plant Under Fixed Time Horizon: S-Graph Approach,* Ind. Eng. Chem. Res., Vol. 45, pp. 6713-6720.

[96] MESA INTERNATIONAL, 1997, *MES Explained: A High Level Vision,* White Paper, Number 6, (http://www.mesa.org/whitepapers/).

[97] MONOSTORI, L., CSÁJI, B., CS., 2006, *Stochastic Dynamic Production Control by Neurodynamic Programming,* Annals of the CIRP, Vol. 55, pp. 473-478.

[98] MONOSTORI, L., VÁNCZA, J., KIS, T., KÁDÁR, B., VIHAROS, ZS., J., 2006, *Real-Time, Cooperative Enterprises: Requirements and Solution Approaches,* Preprints of the 12th IFAC Symposium on Information Control Problems in Manufacturing, INCOM-2006, May 16-19, 2006, Saint Etienne, France pp. 591-596.

[99] NAWAZ, M., ENSCORE, JR., E., HAM, I., 1983, *A Heuristic Algorithm for the m-Machine, n-Job Flow-Shop Sequencing Problem ,* OMEGA, Vol. 11, No. 1, pp. 91-95.

[100] NOWICKI, E., SMUTNICKI, C., 1996, *A Fast Tabu Search Algorithm for the Flow Shop Problem,* European Journal of Operational Research, Vol. 91, pp. 160-175.

[101] NOWICKI, E., SMUTNICKI, C., 1998, *The Flow Shop with Parallel Machines: a Tabu Search Approach,* European Journal of Operational Research, Vol. 106, No. 2-3, pp. 226-253.

[102] OGBU, F., A., SMITH, D., K., 1990, *The Application of the Simulated Annealing Algorithm to the Solution of the n/m/$C_{max}$ Flow Shop Problem,*. Computers and Operations Research, Vol. 17, No. 3, pp. 243-253.

[103] OSMAN, I., H., POTTS, C., N., 1989, *Simulated Annealing for Permutation Flow-Shop Scheduling,* OMEGA International Journal of Management Science, Vol. 17, No. 6, pp. 551-557.

[104] PALMER, D., S., 1965, *Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time – a Quick Method of Obtaining a Near Optimum,* Operational Research Quarterly, Vol. 16, No. 1, pp. 101-107.

[105] PETROVIC, D., DUENAS, A., 2006, *A Fuzzy Logic Based Production Scheduling/Rescheduling in the Presence of Uncertain Discruptions,* Fuzzy Sets and Systems, Vol. 157, pp. 2273-2285.

[106] PONNAMBALAM, S., G., ARAVINDAN, P., CHANDRASEKARAN, S., 2001, *Constructive and Improvement Flow Shop Scheduling Heuristics: an Extensive Evaluation,* Production Planning & Control, Vol. 12, No. 4, pp. 335-344.

[107] *Production Scheduling Portal*, http://www.productionscheduling.com/production_scheduling_books.html

[108] QUADT, D., KUHN, H., 2007, *A Taxonomy of Flexible Flow Line Scheduling Procedures,* European Journal of Operational Research, Vol. 178, pp. 686-698.

[109] RAJENDRAN, C., CHAUDHURI, D., 1992, *Scheduling in n-Job, m-Stage Flow Shop with Parallel Processors to Minimize Makespan,* International Journal of Production Economics, Vol. 27, pp. 137-143.

[110] RAJENDRAN, C., CHAUDHURI, D., 1993, *Heuristic Algorithm for Scheduling in a Flow Shop to Minimize Total Flowtime,* International Journal of Production Economics, Vol. 29, pp. 65-73.

[111] RANGSARITRATSAMEE, R., FERRELL, W., G., KURZ, M., B., 2004, *Dynamic Rescheduling that Simultaneously Considers Efficiency and Stability,* Computers and Industrial Engineering, Vol. 46, No. 1, pp. 1-15.

[112] REEVES, C., R., YAMADA, T., 1998, *Genetic Algorithms, Path Relinking, and Flow Shop Problem,* Evolutionary Computation, Vol. 6, pp. 45-60.

[113] SCHMIDT, G., 2000, *Scheduling with Limited Machine Availability,* European Journal of Operational Research, Vol. 121, pp. 1-15.

[114] SMITH, K., L., EVERSON, R., M., FIELDSEND, J., E., 2004, *Dominance Measures for Multi-Objective Simulated Annealing,* Proceedings of the 2004

Congress on Evolutionary Computation, CEC 2004, June 19-23, 2004, Portland OR, USA, pp. 23-30.

[115] SOMLÓ, J., 2006, *Új utak a gépipari rendszerek gyártásütemezésében,* Gyártás-2006 Magyar Tudományos Konferencia, pp. 155-161.

[116] SOMLÓ, J., SAWKIN, A., V., 2006, *Periodic and Transient Switched Serves Schedules for FMS,* Robotics and Computer Integrated Manufacturing, Vol. 22, pp.93-112.

[117] SOMLÓ, J., SAWKIN, A., V., ANUFRIEV, A., KONCZ, T., 2004, *Pragmatic Aspects of the Solution of FMS Scheduling Problems using Hybrid Dynamical Approach,* Robotics and Computer Integrated Manufacturing, Vol. 20, No. 1, pp. 35-49.

[118] STEFÁN, P., 2003, *Combined Use of Reinforcement Learning and Simulated Annealing: Algorithms and Applications,* Ph.D. Thesis, József Hatvany Ph.D. Program in Information Science, Miskolc, Hungary.

[119] STORER, R., H., WU, S., D., VACCARI, R., 1992, *A New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling,* Management Science, Vol. 38, No. 10, pp. 1495-1509.

[120] TAILLARD, E., 1990, *Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem,* European Journal of Operational Research, Vol. 47, No. 1, pp. 65-74.

[121] TÓTH, T., 2004, *Termelési rendszerek és folyamatok,* Miskolci Egyetemi Kiadó, Miskolc.

[122] TÓTH, T., ERDÉLYI, F., 2006, *New Consideration of Production Performance Management for Discrete Manufacturing,* Proceedings of the 8th International Conference on The Modern Information Technology in the Innovation Processes of the Industrial Enterprises, September 11-12, 2006, Budapest, Hungary, pp. 435-444.

[123] ULLMAN, J., D., 1975, *NP-Complete Scheduling Problem,* Journal of Computing System Science, Vol. 10, pp. 384-393.

[124] VAIK, ZS., 2005, *On Scheduling Problems with Parallel Multi-Purpose Machines,* Technical Report TR-2005-02, MTA-ELTE EGRES, Egerváry Research Group on Combinatorial Optimization.

[125] VÁNCZA, J., MÁRKUS, A., 2000, *An Agent Model for Incentive-based Production Scheduling,* Computers in Industry, Vol. 43, No. 2, pp. 173-187.

[126] VIEIRA, G., HERMANN, J., LIN, E., 2000, *Predicting the Performance of Rescheduling Strategies for Paralell Machines Systems,* Journal of Manufacturing Systems, Vol. 19, No. 4, pp. 256-266.

[127] VIEIRA, G., HERMANN, J., LIN, E., 2003, *Rescheduling Manufacturing Systems: A Framework of Strategies, Policies and Methods,* Journal of Scheduling, Vol. 6, No. 1, pp. 35-58.

[128] VÍZVÁRI B., 2004, *Ütemezéselmélet,* IVÁNYI, A., (szerk.), *Informatikai algoritmusok* 9. fejezete, p. 364-415, ELTE Eötvös Kiadó.

[129] VÍZVÁRI, B., DÓSA, GY., 2004, *Az LPT(k)' algoritmus egyforma párhuzamos gépek ütemezésére,* Alkalmazott Matematikai Lapok, Vol. 21, pp. 269-289.

[130] WANG, H., JACOB, V., ROLLAND, E., 2003, *Design of Efficient Hybrid Neural Networks for Flexible Flow Shop Scheduling,* Expert Systems, Vol. 20, No. 4, pp. 208-231.

[131] WANG, L., LI, D., 2002, *A Scheduling Algorithm for Flexible Flow Shop Problem,* Proceedings of the 4th World Congress on Intelligent Control and Automation, New York: IEEE, Vol. 4, pp. 3106-3108.

[132] WANG, W., 2005, *Flexible Flow Shop Scheduling: Optimum, Heuristics, and Artificial Intelligence Solutions,* Expert Systems, Vol. 22, No. 2, pp. 78-85.

[133] WANG, W., HUNSUCKER, J., L., 2003, *An Evaluation of the CDS Heuristic in Flow Shops with Multiple Processors,* Journal of the Chinese Institute of Industrial Engineers, Vol. 20, No. 3, pp. 295-304.

[134] WITTROCK, R., J., 1985, *Scheduling Algorithm for Flexible Flow Lines,* IBM Journal of Research and Development Vol. 29, pp. 401-412.

[135] WITTROCK, R., J., 1988, *An Adaptable Scheduling Algorithm for Flexible Flow Lines,* Operations Research, Vol. 36, No. 3, pp. 445-453.

[136] YAMADA, T., 2003 *Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problems,* Ph.D. Thesis, Kyoto University, Japan.

[137] YANG, Y., KREIPL, S., PINEDO, M., 2000, *Heuristics for Minimizing Total Weighted Tardiness in Flexible Flow Shops,* Journal of Scheduling, Vol. 3, No. 2, pp. 89-108.

[138] YING, K., C., LIAO, C., J., 2004, *An Ant-Colony System for Permutation Flow Shop Sequencing,* Computers and Operations Research, Vol. 31, No. 5, pp. 791-801.

[139] ZDANSKY, M., POZIVIL, J., 2002, *Combination Genetic/Tabu Search Algorithm for Hybrid Flowshops Optimization,* Proceedings of ALGORITMY 2002 Conference on Scientific Computing, pp. 230-236.

[140] ZHU, X., WILHELM, W., E., 2006, *Scheduling and Lot Sizing with Sequence-Dependent Setup: A Literature review,* IIE Transactions, Vol. 38, pp. 987-1007.

# 12. KÖSZÖNETNYILVÁNÍTÁS

Értekezésemben a Miskolci Egyetem Gépészmérnöki és Informatikai Karán, a Hatvany József Informatikai Tudományok Doktori Iskola keretein belül végzett kutatási munkám során elért eredményeket foglaltam össze.

Ezúton fejezem ki köszönetemet témavezetőmnek Dr. Erdélyi Ferencnek, aki figyelmemet a tudomány felé irányította, rendkívül széles körű és kiemelkedő szakmai ismereteivel, gyakorlati tapasztalataival munkámat mindvégig támogatta, a dolgozat megírásához nélkülözhetetlen segítséget nyújtott.

Köszönetet mondok Prof. Dr. Tóth Tibor tanszékvezető úrnak és egyben a doktori iskola vezetőjének, aki a kutatómunkához szükséges feltételeket megteremtette, és hasznos tanácsokkal látott el.

Köszönetet mondok Dr. Dudás Lászlónak, aki a munkámat folyamatosan nyomon követte, és főként mesterséges intelligenciához, szoftvertechnológiához kapcsolódó hasznos tanácsaival, észrevételeivel segítette fejlesztéseimet.

Megköszönöm Körei Attilának azokat a hasznos észrevételeket, amelyekkel hozzájárult a dolgozatban alkalmazott matematikai formalizmusok finomításához.

Köszönöm a Miskolci Egyetem Alkalmazott Informatikai Tanszék teljes kollektívájának szakmai és erkölcsi támogatását.

Köszönetet mondok mindazoknak, akik közvetlen vagy közvetett módon elősegítették, támogatták munkámat, ezáltal hozzájárultak ahhoz, hogy a dolgozat elkészüljön.

Köszönet illeti meg korábbi tanáraimat, oktatóimat, akik megalapozták szakmai képzésemet.

Hálámat és köszönetemet fejezem ki családtagjaimnak biztatásukért és támogatásukért.

# 13. MELLÉKLETEK

## 13.1. Melléklet: Gyártásifolyamat-típusok és végrehajtási-lépés-típusok meghatározásának algoritmusa

A következő algoritmus adott *N_TS* számú technológiai lépésnek megfelelő EFFS környezetben definiálja a végrehajtásilépés-típusokat. A módszer alkalmas a technológiai lépések összefüggő részsorozatát jelentő gyártásifolyamat-típusok meghatározására is. Ekkor *ES* szerepét *EXE* veszi át.

```
void ES_TS_generation(int N_TS, int ** ES_TS)
{                                                    //N_TS: a technológiai lépések száma
        int N_ES = (N_TS*(1+N_TS))/2;                //N_ES: a végrehajtási lépések száma

           ES_TS = new int * [N_ES+1];                //ES_TS: kapcsolótömb
           for ( int es = 1; es<=N_ES; es++ )         //es: a végrehajtási lépések azonosítója
            {
            ES_TS[es] = new int [N_TS+1];            //a: a hozzárendelés futóindexe
            for (int a = 0; a<= N_TS; a++)
               ES_TS[es][a] = 0;
            }

           for ( int i = 1, es = 1, st, d; i <= N_TS; i++ )  //i: a végrehajtási lépések hossza
           {
            for ( st = 1; st <= N_TS-(i-1); st++ )   //st: a kezdő technológiai lépés
            {
             for ( d = 0; d < i; d++ )               //d: a technológiai lépések futóindexe
              {
                 ES_TS[es][0]++;
                 ES_TS[es][ES_TS[es][0]] = st + d;
              }
             es++;
            }
           }
}
```

## 13.2. Melléklet: Végrehajtásiútvonal-típusok meghatározása

Algoritmus adott *N_TS* számú technológiai lépésnek megfelelő EFFS környezetben értelmezhető végrehajtási útvonalaknak végrehajtási lépésekből történő felépítésére:

```
void R_ES_generation(int N_TS, int ** R_ES )
{
                    //N_TS: a technológiai lépések darabszáma
                    //R_ES: útvonal és végrehajtási lépések kapcsolótömbje

                    //a generálás változói
  int N_R;          //N_R: a végrehajtási útvonalak darabszáma
  int i;            //i: az útvonal által lefedett technológiai lépések darabszáma
  int nr;           //nr: i hosszúságú különböző útvonaltípusok száma
  int r;            //r: az útvonal futóindexe
  int st;           //st: első technológiai lépés
  int a;            //a: a hozzárendelés futóindexe
                    //a transzformálás változói
  int R;            //R: az útvonal azonosítója
  int step:         //step: az útvonalhoz tartozó végrehajtási lépések futóindexe
  int sh_st;        //sh_st: növekmény a kezdő lépéstől számítva
  int act_st;       //act_st: aktuális kezdő technológiai lépés

                    // a kapcsolótömb deklarációja
  N_R = 0;
  for (int i = 1; i <= N_TS; i++)
     N_R += pow(2, (i-1)) * (N_TS - (i-1));

  int ** R_ES = new int * [N_R+1];
  for ( r = 1; r <= N_R; r++ )
   {
    R_ES[r] = new int [N_TS+1];
    for ( a = 0; a<= N_TS; a++)
       R_ES[r][a] = 0;
    }
                    //R_ES felépítése a részsorozatok méretére alapozva
                    //rekurzívan az első lépést meghatározva
  r = 0;
  for (i = N_TS; i >= 1; i--)
    {
     nr = pow(2, (i-1));
     for ( st = 1; st+(i-1) <= N_TS; st++)
     {
      first_ES_length_recursive(R_ES, i, r, 1);
      r += nr;
     }
   }
                    //folytatás a következő oldalon
```

```
                    //R_ES transzformálása a végrehajtási lépések azonosításával
R = 0;
sh_st = 0;
act_st = 1;
 for (int i = N_TS; i >= 1; i-- )
  {
    for ( st = 1; st <= N_TS-(i-1); st++ )
    {
      nr = pow(2, (i-1));
      for ( r = 1; r <= nr; r++)
       {
         R++;
         act_st = st;sh_st = 0;
         for( step = 1; step <= R_ES[R][0]; step ++)
          {
           if (step == 1)
            {
            sh_st += R_ES[R][step];
            R_ES[R][step] = get_ES_ID( N_TS, act_st, R_ES[R][step]);
           }
           else
           {
            int temp = R_ES[R][step];
            R_ES[R][step] = get_ES_ID( N_TS, act_st + sh_st, R_ES[R][step]);
            sh_st += temp;
           }
          }
        }
     }
    }
```

Rekurzív módszer adott *N_TS* vagy annál kevesebb számú technológiai lépésből álló sorozatoknak (végrehajtási útvonalaknak) összefüggő technológiai lépésekből álló részsorozatokkal (végrehajtási lépésekkel) történő lefedésére:

```
int first_ES_length_recursive(int ** R_ES, int N_TS, int fix_R, int fix_pos)

{               //R_ES: az útvonal és a végrehajtási lépések kapcsolótömbje
                //N_TS: a technológiai lépések száma
                //fix_R: a kifejtendő útvonal azonosítója
                //fix_pos: a kifejtendő útvonal első még le nem fedett végrehajtási lépése

   int r = 0;  //r: az útvonal aktuális hívási pontjától számított sorszáma
   int fl;     //fl: az útvonal első végrehajtási lépésének a hossza
   int na;     //na: az fl hosszú végrehajtási lépéssel kezdődő útvonalak száma
   int a;      //a: az útvonal futóindexe
   int st;     //st: az első technológiai lépés
   int fst;    //fst: az első még le nem fedett technológiai lépés

                    //folytatás a következő oldalon
```

```
   for ( fl = 1; fl <= N_TS; fl++ )
   {
    if ( fl == N_TS ) na = 1;                    //egyetlen lépés lefedi (ilyen csak egyféle van)
    else na = pow (2, (N_TS-fl)-1);              //i azaz  N_TS - fl hosszú rész marad szabadon
                                                 //ebből 2^(i-1) féle van

    for ( a = 1; a <= na; a++ )
    {
     r++;
     if ( a==1 )
       first_ES_length_recursive(R_ES, N_TS-fl, fix_R+r-1, fix_pos+1);

     st = 1;
     fst = st + fl;
     R_ES[fix_R+r][fix_pos] = fl;
     R_ES[fix_R+r][0]++ ;
    }
   }
  return 1;
}
```

Algoritmus adott *TS_start* technológiai lépéstől kezdődő, adott *ES_length* számú technológiai lépést lefedő *ES* végrehajtási lépés azonosítására *N_TS* számú technológiai lépést tartalmazó környezetben:

```
int get_ES_ID(int N_TS, int TS_start, int ES_length)
{
                        //N_TS: a technológiai lépések száma
                        //TS_start: a végrehajtási lépés első technológiai lépése
                        //ES_length: a végrehajtási lépés hossza

   int es;              //es: a végrehajtási lépés azonosítója
   int i;               //i: a végrehajtási lépések hossza
   int st;              //st: az első technológiai lépés
   int d;               //d: a technológiai lépések futóindexe

  es = 1;
  for ( i = 1; i <= N_TS; i++ )
   {
    for ( st = 1; st <= N_TS-(i-1); st++ )
     {
      for ( d = 0; d < i; d++ )
       {
          if ( st == TS_start && (d+1) == ES_length )
             return es;
       }
       es++;
     }
   }
   return -1;
}
```

## 13.3. Melléklet: Az ütemezés során használt szomszédsági operátorok alapvető műveletei

Adott munka és a hozzá tartozó gyártási feladatok kiemelése az ütemtervből:

```
void _S::remove_job_from_sch(int i)
{
        int pos, target, machine;
        int nstep = R_MG[J_T[i][0]][0];
        J_T[i][0] = 0;
        for(int step=1; step<=nstep; step++)
        {
          machine = J_T[i][step];
          J_T[i][step] = 0;
          target = J_Tp[i][step];
          J_Tp[i][step] = 0;

           for (pos = target; pos<M_T[machine][0]; pos++)
              {
              M_T[machine][pos] = M_T[machine][pos+1];
              M_Tp[machine][pos] = M_Tp[machine][pos+1];
              J_Tp[M_T[machine][pos]][M_Tp[machine][pos]] = pos;
              }

          M_T[machine][0]--;
        }
}
```

Adott munka beillesztése az ütemtervbe, a munka elvégzéséhez szükséges gyártási feladatok és azok végrehajtási sorban betöltött helyének véletlenszerű megválasztásával:

```
void _S::random_value_for_i_job(int i, ORDER* O, JOB* J)
{
   int shift_pos, new_pos, nr, r, route, nstep, step, mg, npm, m, machine;
   nr = O[J[i].OID].R[0];
   r = 1 + random(nr);
   route = O[J[i].OID].R[r];
   J_T[i][0] = route;
   nstep = R_MG[route][0];
   for(step=1; step<=nstep; step++)
    {
      mg = R_MG[route][step];
      npm = O[J[i].OID].MG_M[mg][0];
      m = 1 + random(npm);
      machine = J_T[i][step] = O[J[i].OID].MG_M[mg][m];
      M_T[machine][0]++;
      new_pos = 1 + random(M_T[machine][0]);


                    //folytatás a következő oldalon
```

```
    for (shift_pos = M_T[machine][0] - 1; shift_pos > 0 && shift_pos >= new_pos; shift_pos -- )
          {
           M_T[machine][shift_pos+1] = M_T[machine][shift_pos];
           M_Tp[machine][shift_pos+1] = M_Tp[machine][shift_pos];
           J_Tp[M_T[machine][shift_pos+1]][M_Tp[machine][shift_pos+1]] = shift_pos+1;
          }

     M_T[machine][new_pos] = i;
     M_Tp[machine][new_pos] = step;
     J_Tp[i][step] = new_pos;
    }
}
```

Adott lépésszámnál (maxN) nem hosszabb, véletlenszerűen választott hosszúságú permutációciklus definiálása:

```
int _S::generate_cycle(int size_of_c, int* c, int maxN)
 {
  int * temp = new int [maxN+1];
  for (int i=1; i<=maxN; i++)
  {
   temp[i] = i;
  }
 if (size_of_c == 0) size_of_c = 1+random(maxN);
 c[0] = size_of_c;
 for (int i=1, index, step, max = maxN; i<=size_of_c; i++)
    {
     step = 1+random(max);
     index = get_pos(step,temp,1,maxN);
     if (index == -1) { return -1; }
     c[i] = temp[index];
     temp[index] = -1;
     max--;
    }
  delete [ ] temp;
 return 1;
 }
```

Adott permutációciklus adott sorozaton történő végrehajtása:

```
void _S::do_pi(int *c, int m)
 {
   int tmp = M_T[m][c[ c[0] ]];
   int tmp_p = M_Tp[m][c[ c[0] ]];
   for (int i=c[0], tmp; i>1; i--)
    {
     M_T[m][c[ i ]] = M_T[m][ c[i-1] ];
     M_Tp[m][ c[ i ]] = M_Tp[m][ c[i-1] ];
     J_Tp[ M_T[m][c[ i ]] ][ M_Tp[m][ c[ i ]] ] = c[i];
    }
    M_T[m][c[1]] = tmp;
    M_Tp[m][c[1]] = tmp_p;
    J_Tp[tmp][tmp_p] = c[1];
}
```

Adott gépen a végrehajtási sorban szereplő feladatok sorrendjének véletlenszerű permutálása:

```
int _S::random_do_pi_seq_change_on_m_machine(int m)
{
 if (M_T[m][0] > 1)
    {
     int size_of_c;
     int *c;

    size_of_c = 1+random(M_T[m][0]);
    c = new int[size_of_c+1];
    if ( generate_cycle(size_of_c, c, M_T[m][0]) )
    do_pi(c, m);
    delete [ ] c;
    return 1;
   }
 return 0;
}
```

Egy késő munka egyenletes valószínűséggel végrehajtott, véletlenszerű kiválasztása a listából:

```
int _S::random_select_latejob(ORDER* O, JOB* J, int n_of_latejob)
{
  int si = 1 + random(n_of_LateJob);
  int li = 0;
  for (int s=1; s<=NJ; s++)
  if ( J[s].RET > O[J[s].OID].CET )
     {
        li++;
        if (si==li) return s;
      }
return 0;
}
```

## 13.4. Melléklet: Az újraütemezés során használt szomszédsági operátorok alapvető műveletei

Zárolás következtében csökkentett gépállománnyal végezhető munka beillesztése az ütemtervbe, a munka befejezéséhez szükséges gyártási feladatok és azok végrehajtási sorban betöltött helyének véletlenszerű megválasztásával:

```
void _S::random_value_for_i_FREEZE_job(int i, ORDER* O, JOB* J, MACHINE* M, _S* s_init)
{
   int shift_pos, new_pos, nr, r, route, nstep, step, mg, npm, m, machine;
   nr = J[i].FJ.enabled_R[0];
   r = 1 + random(nr);
   route = J[i].FJ.enabled_R[r];
   J_T[i][0] = route;
   nstep = R_MG[route][0];
   for(step=1; step<=nstep; step++)
    {
      mg = R_MG[route][step];
      npm = J[i].FJ.enabled_MG_M[mg][0];
      m = 1 + random(npm);
      machine = J_T[i][step] = J[i].FJ.enabled_MG_M[mg][m];
      M_T[machine][0]++;
      if (J[i].LFS >= step)
         new_pos = s_init->J_Tp[i][step];
      else
         {
          new_pos = 1 + random(M_T[machine][0]);
          if ( new_pos <= M[machine].LFP ) new_pos = M[machine].LFP + 1;
         }

      for (shift_pos = M_T[machine][0] - 1; shift_pos > 0 && shift_pos >= new_pos; shift_pos -- )
          {
           M_T[machine][shift_pos+1] = M_T[machine][shift_pos];
           M_Tp[machine][shift_pos+1] = M_Tp[machine][shift_pos];
           J_Tp[M_T[machine][shift_pos+1]][M_Tp[machine][shift_pos+1]] = shift_pos+1;
          }

      M_T[machine][new_pos] = i;
      M_Tp[machine][new_pos] = step;
      J_Tp[i][step] = new_pos;
    }
}
```

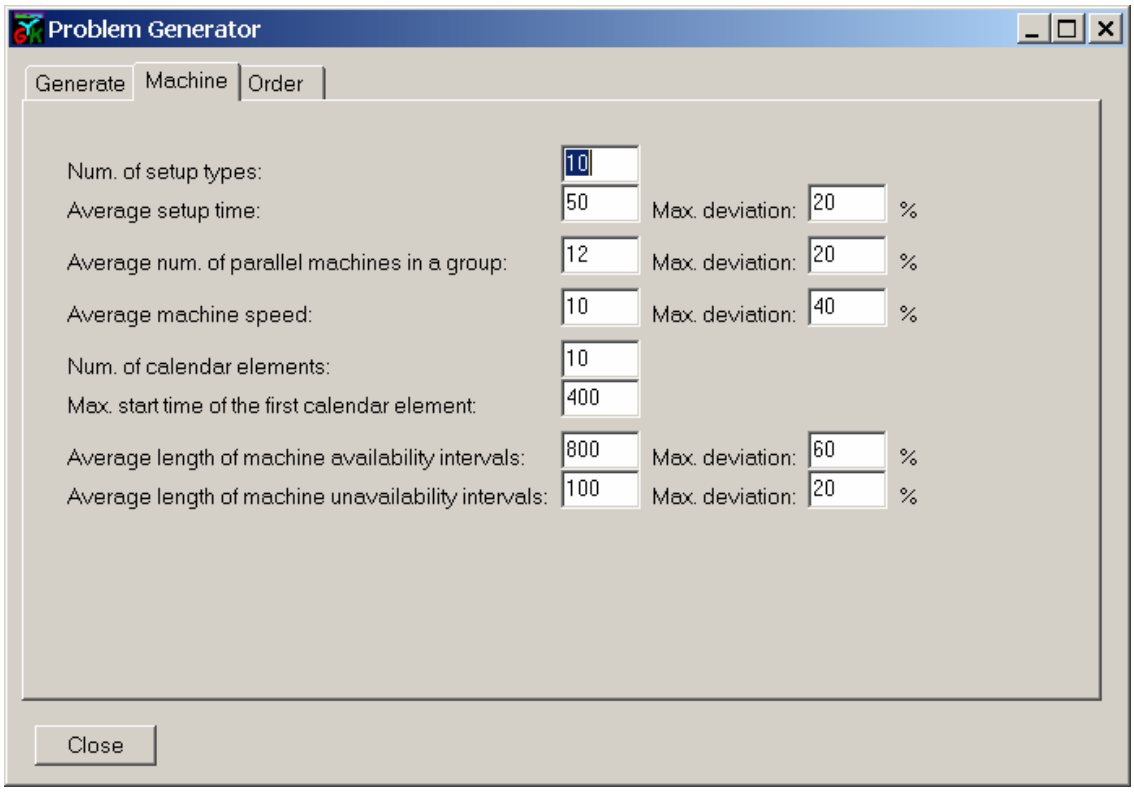
Adott, zárolással nem terhelt munka beillesztése az ütemtervbe a gépek zárolt időszakának megsértése nélkül a munka elvégzéséhez szükséges gyártási feladatok és azok végrehajtási sorban betöltött helyének véletlenszerű megválasztásával:

```
void _S::random_value_for_i_job_in_FREEZE_env(int i, ORDER* O, JOB* J, MACHINE* M)
{
  int shift_pos, new_pos, nr, r, route, nstep, step, mg, npm, m, machine;
   nr = O[J[i].OID].R[0];
   r = 1 + random(nr);
   route = O[J[i].OID].R[r];
   J_T[i][0] = route;
   nstep = R_MG[route][0];
   for(step=1; step<=nstep; step++)
    {
      mg = R_MG[route][step];
      npm = O[J[i].OID].MG_M[mg][0];
      m = 1 + random(npm);
      machine = J_T[i][step] = O[J[i].OID].MG_M[mg][m];
      M_T[machine][0]++;
      new_pos = 1 + random(M_T[machine][0]);

      if ( new_pos <= M[machine].LFP )
          new_pos = M[machine].LFP + 1;

      for (shift_pos = M_T[machine][0] - 1; shift_pos > 0 && shift_pos >= new_pos; shift_pos -- )
           {
            M_T[machine][shift_pos+1] = M_T[machine][shift_pos];
            M_Tp[machine][shift_pos+1] = M_Tp[machine][shift_pos];
            J_Tp[M_T[machine][shift_pos+1]][M_Tp[machine][shift_pos+1]] = shift_pos+1;
           }

      M_T[machine][new_pos] = i;
      M_Tp[machine][new_pos] = step;
      J_Tp[i][step] = new_pos;
    }
}
```

Adott gépen a végrehajtási sorban szereplő feladatok sorrendjének véletlenszerű permutálása a sorozat adott hosszúságú kezdő (zárolt) részének módosítása nélkül:

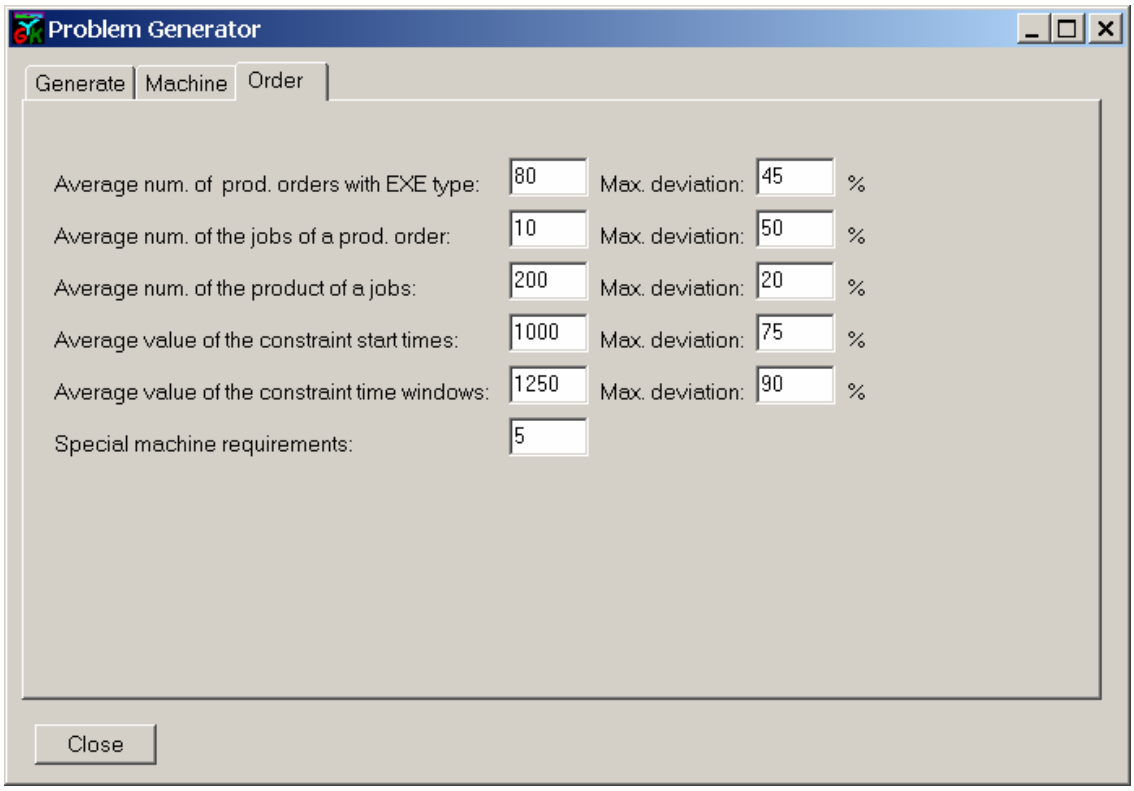
```
int _S::random_do_pi_seq_change_on_m_machine_FREEZE(int m, MACHINE* M)
{
 if (M_T[m][0] - M[m].LFP > 1)
    {
     int size_of_c;
     int *c;
    size_of_c = 1+random(M_T[m][0] - M[m].LFP);
    c = new int[size_of_c+1];
    if ( generate_cycle(size_of_c, c, M_T[m][0] - M[m].LFP) )
    for (int k=1; k<=size_of_c; k++ )
        c[k] += M[m].LFP;

    do_pi(c, m);
   delete [ ] c;
    return 1;
   }
 return 0;
}
```

## 13.5. Melléklet: A tesztfeladatok előállítására használt problémagenerátor paraméterei




**32. ábra: A gépek paraméterei**

*Num. of setup types:* gépbeállítás-típusok száma.
*Average setup time:* gépbeállítás átlagos időtartama.
*Average num. of parallel machines in a group:* a különböző gépcsoportokban rendelkezésre álló párhuzamos gépek átlagos száma.
*Average machine speed:* a gépek termékfüggő gyártási sebességeinek átlagétéke (időegység alatt megmunkálható termékek száma).
*Num. of calendar elements:* a gépek rendelkezésre állási időintervallumainak száma.
*Max. start time of the first calendar element:* az első rendelkezésre állási intervallum kezdő időpontjának legnagyobb megengedett értéke.
*Average length of machine availability intervals:* a gépek rendelkezésre állási időintervallumainak átlagos hossza.
*Average length of machine unavailability intervals:* a rendelkezésre állási időintervallumok közötti időszakaszok átlagos hossza.
*Max. deviation:* az átlagos értéktől való legnagyobb eltérés százalékban kifejezve.
A beállított paraméterek alapján a számértékek generálása kváziegyenletes valószínűséggel történik.




**33. ábra: A megrendelések, termékek és munkák paraméterei**

*Average num. of prod. orders with EXE type:* a különböző gyártásifolyamat-típusokra vonatkozó belső megrendelések átlagos száma.

*Average num. of the jobs of a prod. order:* a belső rendelésekhez tartozó munkák átlagos száma.

*Average num. of the product of a jobs:* a munkákhoz (logisztikai egységekhez) tartozó munkadarabok átlagos száma.

*Average value of the constraint start times:* a munkákhoz tartozó legkorábbi kezdési időpont átlagos értéke.

*Average value of the constraint time windows:* a rendelések teljesítésére rendelkezésre álló időtartam átlagos hossza.

*Special machine requirements:* a megrendelések teljesítéséhez szükséges gépek speciális képességeinek előírására szolgáló számérték (1: nincs korlátozás, 10: legnagyobb mértékű korlátozás).

*Max. deviation:* az átlagos értéktől való legnagyobb eltérés százalékban kifejezve.

A beállított paraméterek alapján a számértékek generálása kváziegyenletes valószínűséggel történik.

## 13.6. Melléklet: Futási eredmények

A bemutatott heurisztikus megoldási módszer pontosságának vizsgálatához kifejlesztettem egy leszámlálásra épülő ütemező algoritmust, amely kisméretű feladatok esetében megadja az optimális megoldások számát és az elsőnek megtalált optimális megoldáshoz tartozó célfüggvényértékeket. Ezzel a módszerrel hasonlítottam össze a termelésprogramozó szoftver megoldóalgoritmusait. A különböző összetételű és nehézségi fokú tesztfeladatok vizsgálata során azt tapasztaltam, hogy a megvizsgált leszámlálható méretű feladatok megoldása során a többcélú heurisztikus kereső-algoritmus minden esetben megtalálta az optimális megoldást (vagy az optimális megoldások közül egyet, ha egyszerre több is létezett).

A megoldási módszer hatékonyságának vizsgálatához nagyméretű ütemezési feladatokat készítettem a 13.5. mellélletben bemutatott paraméterezéssel. Az elvégzett vizsgálatok eredményei azt mutatják, hogy a javasolt megoldási módszer nagyméretű feladatok esetében is hatékonyan alkalmazható, rugalmasan alkalmazkodik az aktuális célfüggvényekhez és rövid időn belül szolgáltatja az eredményeket.

A következőkben leszámlálható méretű feladatokon végzett összehasonlító vizsgálatok futási eredményeit, valamint valós gyártási adadokhoz hasonló nagyméretű, nehéz feladatok megoldása során kapott eredményeket mutatok be. (Tesztkörnyezet: Intel Pentium 4 2,8 GHz CPU, 512MB DDR333 RAM, Microsoft Windows XP OS.)

Az alkalmazott formalizmus a következő:

*Input file*: ütemezési feladatot leíró fájl.
*Prob_Sizes*: a feladat jellemző méretei.
*Machine*: gépek száma.
*Orders*: megrendelések száma.
*Jobs*: munkák száma.
*Setup_Types*: gépbeállítás-típusok száma.
*Search_Pars*: keresési paraméterek.
*maxSTEP*: legnagyobb megtehető lépésszám (leállási feltétel).
*maxLOOP*: adott lépés kiterjesztéseinek maximális száma.
*minLOOP*: adott lépés kiterjesztésének minimális száma (kötelezően megvizsgálandó szomszédos megoldások száma).
*maxSTEP_impless*: a javulás nélkül megtehető lépések legnagyobb száma (leállási feltétel).
*maxTABU*: a tabulistán tárolt megoldások megengedett legnagyobb száma.
*Nx_op_pri*: a szomszédsági operátorok (N1, N2, N3, N4, N5) prioritásértékei.
*ObjF*: a célfüggvények.
*LOrder*: a késő megrendelések száma.
*LJob*: a késő munkák száma.
*sumT*: a munkák késéseinek az összege.
*maxT*: a legnagyobb késés.
*Set*: a gépátállítások száma.

*maxC*: a rendeléscsoport teljesítésének befejezési időpontja.
*ObjPri*: a célfüggvények prioritásértékei.
*rand*: véletlenszerűen felépített megoldás.
*heu*: a *BWBA* heurisztikus algoritmussal felépített megoldás.
*search*: a többcélú keresőalgoritmussal javított megoldás.
*F(r,x)*: az adott sorban lévő megoldásnak a *rand* megoldáshoz viszonyított minősége.
*F(h,x)*: az adott sorban lévő megoldásnak a *heu* megoldáshoz viszonyított minősége.
A 4.3.2 fejezetben definiált (4.27) képletre alapozva: „-" jobb megoldást, „+" rosszabb megoldást és „0" azonosan jó megoldást jelöl.
*RunT*: futási idő (másodperc pontossággal mérve, a felépítő algoritmusok futási ideje nincs feltüntetve, mert minden esetben kisebb mint 1s).
*BStep*: a keresés során a legjobb megoldást adó lépés sorszáma.
*Avp(10it)*: az iterációk eredményének átlagértékei.
*Enum*: a leszámláló algoritmussal kapott egzakt eredmény.
*Number of evaluated schedule*: a leszámlálás során kiértékelt ütemtervek száma.
*best solution (first)*: az első optimális megoldás sorszáma.
*number of optimal solutions*: az optimális ütemtervek száma.
*Ts_x*: kisméretű ütemezési feladaton végzett teszt eredményei (*x*: a feladat sorszáma).
*Tl_x*: nagyméretű ütemezési feladaton végzett teszt eredményei (*x*: a feladat sorszáma).
A feladatok pontos specifikációját (gépek, megrendelések, munkák és korlátozások) terjedelmi okokból itt nem részletezem, azok az értekezés CD mellékletén megtalálhatók és a termelésprogramozó szoftver segítségével könnyen megtekinthetők (lásd: 26- 28. ábrák).

## Kisméretű ütemezési feladatok megoldásának eredményei:

***Ts_1:***

Performance test. Input file :C:\kulcsar\EFFS\s_DATA\si1\si1.txt

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Prob_Sizes: | | Machine:11 | | Orders:16 | | Jobs:18 | | Setup_Types:3 | | |
| Search_Pars: | | maxSTEP:200 | | maxLOOP:100 | | minLOOP:10 | | maxImpless:20 | | maxTABU:150 |
| Nx_op_pri: | | N1: 1 | N2: 1 | N3: 1 | N4: 1 | N5: 1 | | | | |
| ObjF: | LOrder | LJob | sumT | maxT | Set | maxC | F(r,x) | F(h,x) | RunT | BStep |
| ObjPri: | 7 | 8 | 10 | 10 | 7 | 5 | | | | |
| 1: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 5 | 627 | 175 | 15 | 523 | | | | |
| heur | 2 | 3 | 398 | 175 | 13 | 505 | - | | | |
| search | 0 | 0 | 0 | 0 | 13 | 430 | - | - | 0s | [2] |
| 2: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 5 | 627 | 175 | 15 | 523 | | | | |
| heur | 2 | 3 | 398 | 175 | 13 | 505 | - | | | |
| search | 0 | 0 | 0 | 0 | 13 | 430 | - | - | 0s | [2] |
| 3: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 5 | 627 | 175 | 15 | 523 | | | | |
| heur | 2 | 3 | 398 | 175 | 13 | 505 | - | | | |
| search | 0 | 0 | 0 | 0 | 13 | 430 | - | - | 0s | [2] |
| 4: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 5 | 627 | 175 | 15 | 523 | | | | |
| heur | 2 | 3 | 398 | 175 | 13 | 505 | - | | | |
| search | 0 | 0 | 0 | 0 | 13 | 430 | - | - | 0s | [2] |
| 5: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 5 | 627 | 175 | 15 | 523 | | | | |
| heur | 2 | 3 | 398 | 175 | 13 | 505 | - | | | |
| search | 0 | 0 | 0 | 0 | 13 | 430 | - | - | 0s | [2] |
| 6: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 5 | 627 | 175 | 15 | 523 | | | | |
| heur | 2 | 3 | 398 | 175 | 13 | 505 | - | | | |
| search | 0 | 0 | 0 | 0 | 13 | 430 | - | - | 0s | [2] |
| 7: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 5 | 627 | 175 | 15 | 523 | | | | |
| heur | 2 | 3 | 398 | 175 | 13 | 505 | - | | | |
| search | 0 | 0 | 0 | 0 | 13 | 430 | - | - | 0s | [2] |
| 8: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 5 | 627 | 175 | 15 | 523 | | | | |
| heur | 2 | 3 | 398 | 175 | 13 | 505 | - | | | |
| search | 0 | 0 | 0 | 0 | 13 | 430 | - | - | 0s | [2] |
| 9: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 5 | 627 | 175 | 15 | 523 | | | | |
| heur | 2 | 3 | 398 | 175 | 13 | 505 | - | | | |
| search | 0 | 0 | 0 | 0 | 13 | 430 | - | - | 0s | [2] |
| 10: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 5 | 627 | 175 | 15 | 523 | | | | |
| heur | 2 | 3 | 398 | 175 | 13 | 505 | - | | | |
| search | 0 | 0 | 0 | 0 | 13 | 430 | - | - | 0s | [2] |
| Avp(10it): | | ====== | ====== | ====== | ====== | ====== | ====== | | | |
| rand: | 4 | 5 | 627 | 175 | 15 | 523 | | | | |
| heur: | 2 | 3 | 398 | 175 | 13 | 505 | - | | | |
| search: | 0 | 0 | 0 | 0 | 13 | 430 | - | - | 0s | |
| Optimum: | | ====== | ====== | ====== | ====== | ====== | ====== | | | |
| Enum: | 0 | 0 | 0 | 0 | 13 | 430 | | | 1 (min) and 32 (sec) | |

Number of evaluated schedule: 10501632
best solution (first): 9585589
number of optimal solutions: 144
End of file.

***Ts_2:***

Performance test. Input file :C:\kulcsar\EFFS\s_DATA\si2\si2.txt

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Prob_Sizes: | | Machine:11 | | Orders:14 | | Jobs:17 | | Setup_Types:3 | | |
| Search_Pars: | | maxSTEP:200 | | maxLOOP:100 | | minLOOP:10 | | maxImpless:20 | | maxTABU:150 |
| Nx_op_pri: | | N1: 1 | N2: 1 | N3: 1 | N4: 1 | N5: 1 | | | | |
| ObjF: | LOrder | LJob | sumT | maxT | Set | maxC | F(r,x) | F(h,x) | RunT | BStep |
| ObjPri: | 7 | 8 | 10 | 10 | 7 | 5 | | | | |
| 1: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 6 | 556 | 184 | 14 | 523 | | | | |
| heur | 2 | 4 | 338 | 166 | 12 | 430 | - | | | |
| search | 2 | 4 | 182 | 88 | 12 | 430 | - | - | 0s | [2] |
| 2: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 6 | 556 | 184 | 14 | 523 | | | | |
| heur | 2 | 4 | 338 | 166 | 12 | 430 | - | | | |
| search | 2 | 4 | 182 | 88 | 12 | 430 | - | - | 0s | [2] |
| 3: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 6 | 556 | 184 | 14 | 523 | | | | |
| heur | 2 | 4 | 338 | 166 | 12 | 430 | - | | | |
| search | 2 | 4 | 182 | 88 | 12 | 430 | - | - | 0s | [2] |
| 4: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 6 | 556 | 184 | 14 | 523 | | | | |
| heur | 2 | 4 | 338 | 166 | 12 | 430 | - | | | |
| search | 2 | 4 | 182 | 88 | 12 | 430 | - | - | 0s | [2] |
| 5: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 4 | 6 | 556 | 184 | 14 | 523 | | | | |
| heur | 2 | 4 | 338 | 166 | 12 | 430 | - | | | |
| search | 2 | 4 | 182 | 88 | 12 | 430 | - | - | 1s | [2] |
| 6: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 3 | 5 | 421 | 166 | 13 | 448 | | | | |
| heur | 2 | 4 | 338 | 166 | 13 | 430 | - | | | |
| search | 2 | 4 | 182 | 88 | 12 | 430 | - | - | 0s | [10] |
| 7: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 3 | 5 | 421 | 166 | 13 | 448 | | | | |
| heur | 2 | 4 | 338 | 166 | 13 | 430 | - | | | |
| search | 2 | 4 | 182 | 88 | 12 | 430 | - | - | 0s | [10] |
| 8: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 3 | 5 | 421 | 166 | 13 | 448 | | | | |
| heur | 2 | 4 | 338 | 166 | 13 | 430 | - | | | |
| search | 2 | 4 | 182 | 88 | 12 | 430 | - | - | 0s | [10] |
| 9: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 3 | 5 | 421 | 166 | 13 | 448 | | | | |
| heur | 2 | 4 | 338 | 166 | 13 | 430 | - | | | |
| search | 2 | 4 | 182 | 88 | 12 | 430 | - | - | 0s | [10] |
| 10: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 3 | 5 | 421 | 166 | 13 | 448 | | | | |
| heur | 2 | 4 | 338 | 166 | 13 | 430 | - | | | |
| search | 2 | 4 | 182 | 88 | 12 | 430 | - | - | 0s | [10] |
| Avp(10it): | | ====== | ====== | ====== | ====== | ====== | ====== | | | |
| rand: | 3.5 | 5.5 | 488.5 | 175 | 13.5 | 485.5 | | | | |
| heur: | 2 | 4 | 338 | 166 | 12.5 | 430 | - | | | |
| search: | 2 | 4 | 182 | 88 | 12 | 430 | - | - | 0.1s | |
| Optimum: | | ====== | ====== | ====== | ====== | ====== | ====== | | | |
| Enum: | 2 | 4 | 182 | 88 | 12 | 430 | | | 3 (min) and 4 (sec) | |

Number of evaluated schedule: 22334976
best solution (first): 20438377
number of optimal solutions: 48
End of file.

***Ts_3:***

Performance test. Input file :C:\kulcsar\EFFS\s_DATA\si3\si3.txt

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Prob_Sizes: | | Machine:6 | | Orders:12 | | Jobs:13 | | Setup_Types:4 | | |
| Search_Pars: | | maxSTEP:200 | | maxLOOP:100 | | minLOOP:10 | | maxImpless:20 | | maxTABU:150 |
| Nx_op_pri: | | N1: 1 | N2: 1 | N3: 1 | N4: 1 | N5: 1 | | | | |
| ObjF: | LOrder | LJob | sumT | maxT | Set | maxC | F(r,x) | F(h,x) | RunT | BStep |
| ObjPri: | 7 | 8 | 10 | 10 | 7 | 5 | | | | |
| 1: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 7 | 7 | 8404 | 3254 | 17 | 4972 | | | | |
| heur | 6 | 6 | 7761 | 3254 | 17 | 4972 | - | | | |
| search | 1 | 1 | 561 | 561 | 14 | 2486 | - | - | 0s | [9] |
| 2: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 7 | 7 | 8404 | 3254 | 17 | 4972 | | | | |
| heur | 6 | 6 | 7761 | 3254 | 17 | 4972 | - | | | |
| search | 1 | 1 | 561 | 561 | 14 | 2486 | - | - | 0s | [9] |
| 3: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 7 | 7 | 8404 | 3254 | 17 | 4972 | | | | |
| heur | 6 | 6 | 7761 | 3254 | 17 | 4972 | - | | | |
| search | 1 | 1 | 561 | 561 | 14 | 2486 | - | - | 0s | [9] |
| 4: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 7 | 7 | 8404 | 3254 | 17 | 4972 | | | | |
| heur | 6 | 6 | 7761 | 3254 | 17 | 4972 | - | | | |
| search | 1 | 1 | 561 | 561 | 14 | 2486 | - | - | 0s | [9] |
| 5: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 7 | 7 | 8404 | 3254 | 17 | 4972 | | | | |
| heur | 6 | 6 | 7761 | 3254 | 17 | 4972 | - | | | |
| search | 1 | 1 | 561 | 561 | 14 | 2486 | - | - | 0s | [9] |
| 6: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 7 | 7 | 8404 | 3254 | 17 | 4972 | | | | |
| heur | 6 | 6 | 7761 | 3254 | 17 | 4972 | - | | | |
| search | 1 | 1 | 561 | 561 | 14 | 2486 | - | - | 0s | [9] |
| 7: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 7 | 7 | 8404 | 3254 | 17 | 4972 | | | | |
| heur | 6 | 6 | 7761 | 3254 | 17 | 4972 | - | | | |
| search | 1 | 1 | 561 | 561 | 14 | 2486 | - | - | 0s | [9] |
| 8: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 7 | 7 | 8404 | 3254 | 17 | 4972 | | | | |
| heur | 6 | 6 | 7761 | 3254 | 17 | 4972 | - | | | |
| search | 1 | 1 | 561 | 561 | 14 | 2486 | - | - | 1s | [9] |
| 9: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 7 | 7 | 8404 | 3254 | 17 | 4972 | | | | |
| heur | 6 | 6 | 7761 | 3254 | 17 | 4972 | - | | | |
| search | 1 | 1 | 561 | 561 | 14 | 2486 | - | - | 0s | [5] |
| 10: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 7 | 7 | 8404 | 3254 | 17 | 4972 | | | | |
| heur | 6 | 6 | 7761 | 3254 | 17 | 4972 | - | | | |
| search | 1 | 1 | 561 | 561 | 14 | 2486 | - | - | 0s | [5] |
| Avp(10it): | | ====== | ====== | ====== | ====== | ====== | ====== | | | |
| rand: | 7 | 7 | 8404 | 3254 | 17 | 4972 | | | | |
| heur: | 6 | 6 | 7761 | 3254 | 17 | 4972 | - | | | |
| search: | 1 | 1 | 561 | 561 | 14 | 2486 | - | - | 0.1s | |
| Optimum: | | ====== | ====== | ====== | ====== | ====== | ====== | | | |
| Enum: | 1 | 1 | 561 | 561 | 14 | 2486 | | | 4 (sec) | |

Number of evaluated schedule: 663552
best solution (first): 657660
number of optimal solutions: 4
End of file.

A további kisméretű feladatok esetében csak az átlagolt értékek szerepelnek.

***Ts_4:***

```
Performance test.   Input file :C:\kulcsar\EFFS\s_DATA\si4\si4.txt
Prob_Sizes:     Machine:6       Orders:12       Jobs:14         Setup_Types:4
Search_Pars:    maxSTEP:200     maxLOOP:100     minLOOP:10      maxImpless:20   maxTABU:150
Nx_op_pri:      N1: 1   N2: 1   N3: 1   N4: 1   N5: 1
ObjF:   LOrder  LJob    sumT    maxT    Set     maxC    F(r,x)  F(h,x)  RunT    BStep
ObjPri: 7       8       10      10      7       5
Avp(10it):      ======  ======  ======  ======  ======  ======
rand:   7       8       11693   3289    17      5007
heur:   6       7       11050   3289    17      5007    -
search: 1       1       561     561     14      2486    -       -       0.1s
    Optimum:    ======  ======  ======  ======  ======  ======
Enum:   1       1       561     561     14      2486                    2 (min) and 3 (sec)
Number of evaluated schedule:   16588800
best solution (first):          16438362
number of optimal solutions:    24
End of file.
```

***Ts_5:***

```
Performance test.   Input file :C:\kulcsar\EFFS\s_DATA\si5\si5.txt
Prob_Sizes:     Machine:6       Orders:6        Jobs:9          Setup_Types:6
Search_Pars:    maxSTEP:200     maxLOOP:100     minLOOP:10      maxImpless:20   maxTABU:150
Nx_op_pri:      N1: 1   N2: 1   N3: 1   N4: 1   N5: 1
ObjF:   LOrder  LJob    sumT    maxT    Set     maxC    F(r,x)  F(h,x)  RunT    BStep
ObjPri: 7       8       10      10      7       5
Avp(10it):      ======  ======  ======  ======  ======  ======
rand:   5       7       1843    595     12      1155
heur:   5       7       2211    812     10      1268    +
search: 0       0       0       0       9       621     -       -       0s
    Optimum:    ======  ======  ======  ======  ======  ======
Enum:   0       0       0       0       9       621                     9 (min) and 11 (sec)
Number of evaluated schedule:   94348800
best solution (first):          33145201
number of optimal solutions:    160
End of file.
```

***Ts_6:***

```
Performance test.   Input file :C:\kulcsar\EFFS\s_DATA\si6\si6.txt
Prob_Sizes:     Machine:2       Orders:5        Jobs:5          Setup_Types:4
Search_Pars:    maxSTEP:200     maxLOOP:100     minLOOP:10      maxImpless:20   maxTABU:150
Nx_op_pri:      N1: 1   N2: 1   N3: 1   N4: 1   N5: 1
ObjF:   LOrder  LJob    sumT    maxT    Set     maxC    F(r,x)  F(h,x)  RunT    BStep
ObjPri: 7       8       10      10      7       5
Avp(10it):      ======  ======  ======  ======  ======  ======
rand:   3       3       1829    934     8       1478
heur:   3       3       1829    934     8       1478    0
search: 2       2       419     334     8       1484    -       -       0.1s
    Optimum:    ======  ======  ======  ======  ======  ======
Enum:   2       2       419     334     8       1484                    0 (sec)
Number of evaluated schedule:   14400
best solution (first):          11254
number of optimal solutions:    2
End of file.
```

***Ts_7:***

```
Performance test.   Input file :C:\kulcsar\EFFS\s_DATA\si7\si7.txt
Prob_Sizes:     Machine:2       Orders:7        Jobs:7          Setup_Types:4
Search_Pars:    maxSTEP:200     maxLOOP:100     minLOOP:10      maxImpless:20   maxTABU:150
Nx_op_pri:      N1: 1   N2: 1   N3: 1   N4: 1   N5: 1
ObjF:   LOrder  LJob    sumT    maxT    Set     maxC    F(r,x)  F(h,x)  RunT    BStep
ObjPri: 7       8       10      10      7       5
Avp(10it):      ======  ======  ======  ======  ======  ======
rand:   4       4       3292    1424    12      1968
heur:   4       4       3292    1424    12      1968    0
search: 4       4       402     233     6       1790    -       -       0s
    Optimum:    ======  ======  ======  ======  ======  ======
Enum:   4       4       402     233     6       1790                    2 (min) and 8 (sec)
Number of evaluated schedule:   25401600
best solution (first):          13192298
number of optimal solutions:    6
End of file.
```

***Ts_8:***

```
Performance test.   Input file :C:\kulcsar\EFFS\s_DATA\si8\si8.txt
Prob_Sizes:     Machine:3       Orders:9        Jobs:9          Setup_Types:4
Search_Pars:    maxSTEP:200     maxLOOP:100     minLOOP:10      maxImpless:20   maxTABU:150
Nx_op_pri:      N1: 1   N2: 1   N3: 1   N4: 1   N5: 1
ObjF:   LOrder  LJob    sumT    maxT    Set     maxC    F(r,x)  F(h,x)  RunT    BStep
ObjPri: 7       8       10      10      7       5
Avp(10it):      ======  ======  ======  ======  ======  ======
rand:   6       6       2372    586     9       1130
heur:   4       4       1860    670     10      1214    -
search: 3       3       399     160     8       1129    -       -       0s
    Optimum:    ======  ======  ======  ======  ======  ======
Enum:   3       3       399     160     8       1129                    1 (min) and 20 (sec)
Number of evaluated schedule:   14515200
best solution (first):          5672390
number of optimal solutions:    4
End of file.
```

***Ts_9:***

```
Performance test.   Input file :C:\kulcsar\EFFS\s_DATA\si9\si9.txt
Prob_Sizes:     Machine:1       Orders:10       Jobs:10         Setup_Types:4
Search_Pars:    maxSTEP:200     maxLOOP:100     minLOOP:10      maxImpless:40   maxTABU:150
Nx_op_pri:      N1: 1   N2: 1   N3: 1   N4: 1   N5: 1
ObjF:   LOrder  LJob    sumT    maxT    Set     maxC    F(r,x)  F(h,x)  RunT    BStep
ObjPri: 7       8       10      10      7       5
Avp(10it):      ======  ======  ======  ======  ======  ======
rand:   4       4       3411    1385    10      2949
heur:   4       4       3411    1385    10      2949    0
search: 1       1       130     130     8       2754    -       -       0s
    Optimum:    ======  ======  ======  ======  ======  ======
Enum:   1       1       130     130     8       2754                    17 (sec)
Number of evaluated schedule:   3628800
best solution (first):          1534957
number of optimal solutions:    1
End of file.
```

## Nagyméretű ütemezési feladatok megoldásának eredményei:

***Tl_1:***

Performance test. Input file :C:\kulcsar\EFFS\l_DATA\li1\li1.txt

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Prob_Sizes: | | Machine:119 | | Orders:393 | | Jobs:2173 | | Setup_Types:4 | | |
| Search_Pars: | | maxSTEP:2000 | | maxLOOP:100 | | minLOOP:10 | | maxImpless:40 | | maxTABU:150 |
| Nx_op_pri: | | N1: 1 | N2: 1 | N3: 1 | N4: 1 | N5: 1 | | | | |
| ObjF: | LOrder | LJob | sumT | maxT | Set | maxC | F(r,x) | F(h,x) | RunT | BStep |
| ObjPri: | 7 | 8 | 10 | 10 | 7 | 5 | | | | |
| 1: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 277 | 997 | 1344918 | 4143 | 1740 | 4535 | | | | |
| heur | 104 | 408 | 118741 | 1222 | 964 | 1710 | - | | | |
| search | 2 | 3 | 33 | 14 | 243 | 998 | - | - | 3min 32s | [1656] |
| 2: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 274 | 943 | 1209326 | 4067 | 1745 | 4517 | | | | |
| heur | 106 | 417 | 117679 | 1300 | 971 | 1724 | - | | | |
| search | 2 | 3 | 46 | 22 | 234 | 1068 | - | - | 2min 54s | [1525] |
| 3: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 282 | 989 | 1276966 | 4106 | 1817 | 4503 | | | | |
| heur | 111 | 431 | 117251 | 1201 | 970 | 1640 | - | | | |
| search | 2 | 3 | 46 | 22 | 250 | 1035 | - | - | 2min 36s | [1402] |
| 4: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 278 | 1006 | 1620700 | 4893 | 1796 | 5345 | | | | |
| heur | 113 | 420 | 115699 | 1304 | 965 | 1746 | - | | | |
| search | 2 | 3 | 46 | 22 | 265 | 1032 | - | - | 1min 46s | [1146] |
| 5: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 278 | 999 | 1398662 | 4073 | 1787 | 4562 | | | | |
| heur | 107 | 436 | 118624 | 1228 | 949 | 1697 | - | | | |
| search | 2 | 3 | 33 | 14 | 228 | 1031 | - | - | 3min 17s | [1593] |
| 6: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 277 | 1009 | 1462062 | 4556 | 1799 | 4910 | | | | |
| heur | 105 | 419 | 122077 | 1396 | 968 | 1884 | - | | | |
| search | 2 | 3 | 46 | 22 | 228 | 995 | - | - | 3min 41s | [1701] |
| 7: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 284 | 1034 | 1420264 | 4247 | 1814 | 4770 | | | | |
| heur | 111 | 436 | 117843 | 1240 | 954 | 1669 | - | | | |
| search | 2 | 3 | 46 | 22 | 237 | 1055 | - | - | 3min 59s | [1872] |
| 8: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 272 | 971 | 1260134 | 4022 | 1768 | 4474 | | | | |
| heur | 111 | 440 | 119894 | 1359 | 960 | 1783 | - | | | |
| search | 2 | 3 | 46 | 22 | 244 | 1058 | - | - | 2min 48s | [1489] |
| 9: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 275 | 975 | 1355598 | 4216 | 1784 | 4668 | | | | |
| heur | 111 | 446 | 112442 | 1260 | 963 | 1675 | - | | | |
| search | 3 | 5 | 92 | 31 | 304 | 1014 | - | - | 1min 49s | [1189] |
| 10: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 278 | 998 | 1298440 | 4088 | 1746 | 4568 | | | | |
| heur | 113 | 454 | 125931 | 1243 | 976 | 1722 | - | | | |
| search | 2 | 3 | 46 | 22 | 254 | 1061 | - | - | 2min 55s | [1491] |
| Avp(10it): | | ====== | ====== | ====== | ====== | ====== | ====== | | | |
| rand: | 277.5 | 992.1 | 1.3e+06 | 4241.1 | 1779.6 | 4685.2 | | | | |
| heur: | 109.2 | 430.7 | 118618 | 1275.3 | 964 | 1725 | - | | | |
| search: | 2.1 | 3.2 | 48 | 21.3 | 248.7 | 1034.7 | - | - | 2min 55s | |

| ObjF: | LOrder | LJob | sumT | maxT | Set | maxC |
|---|---|---|---|---|---|---|
| ObjPri: | 7 | 8 | 10 | 10 | 7 | 5 |
| | --- | --- | --- | --- | --- | --- |
| s_1: | 2 | 3 | 33 | 14 | 243 | 998 |
| s_2: | 2 | 3 | 46 | 22 | 234 | 1068 |
| s_3: | 2 | 3 | 46 | 22 | 250 | 1035 |
| s_4: | 2 | 3 | 46 | 22 | 265 | 1032 |
| s_5: | 2 | 3 | 33 | 14 | 228 | 1031 |
| s_6: | 2 | 3 | 46 | 22 | 228 | 995 |
| s_7: | 2 | 3 | 46 | 22 | 237 | 1055 |
| s_8: | 2 | 3 | 46 | 22 | 244 | 1058 |
| s_9: | 3 | 5 | 92 | 31 | 304 | 1014 |
| s_10: | 2 | 3 | 46 | 22 | 254 | 1061 |
| best solution: | s_5 | | | | | |

Rel. quality of s_row: - better, + worse, 0 equal.

| | s_1 | s_2 | s_3 | s_4 | s_5 | s_6 | s_7 | s_8 | s_9 | s_10 |
|---|---|---|---|---|---|---|---|---|---|---|
| s_1 | 0 | - | - | - | + | - | - | - | - | - |
| s_2 | + | 0 | - | - | + | + | - | - | - | - |
| s_3 | + | + | 0 | - | + | + | + | + | - | - |
| s_4 | + | + | + | 0 | + | + | + | + | - | + |
| s_5 | - | - | - | - | 0 | - | - | - | - | - |
| s_6 | + | - | - | - | + | 0 | - | - | - | - |
| s_7 | + | + | - | - | + | + | 0 | - | - | - |
| s_8 | + | + | - | - | + | + | + | 0 | - | - |
| s_9 | + | + | + | + | + | + | + | + | 0 | + |
| s_10 | + | + | + | - | + | + | + | + | - | 0 |

End of file.

A *Tl_1* feladat generálására használt paraméterek a 32. és 33. ábrának megfelelő sorrendben:

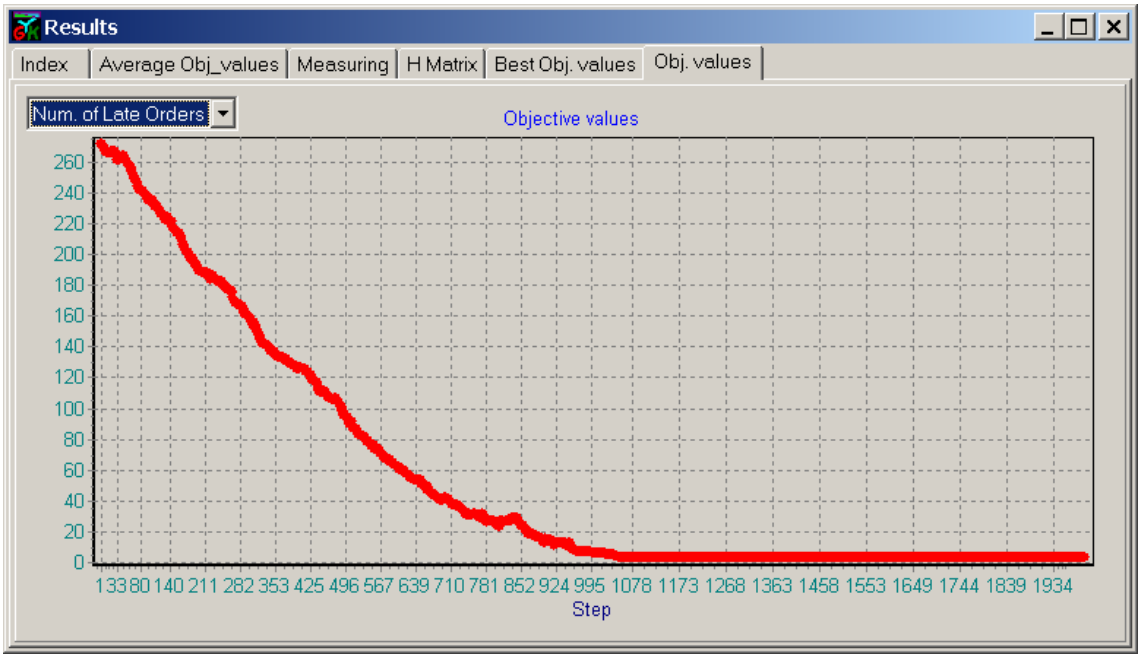
*Machine*

4
30 20
12 20
10 20
10
400
1000 20
100 20

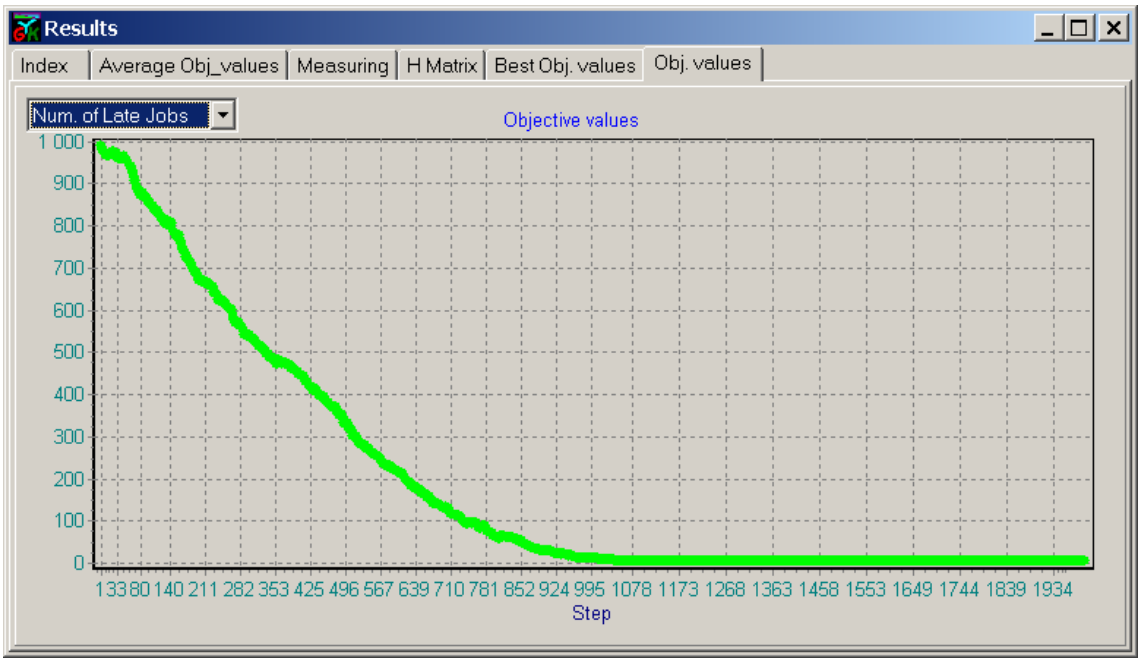
*Order*

40 20
6 20
200 20
300 20
1000 80
8

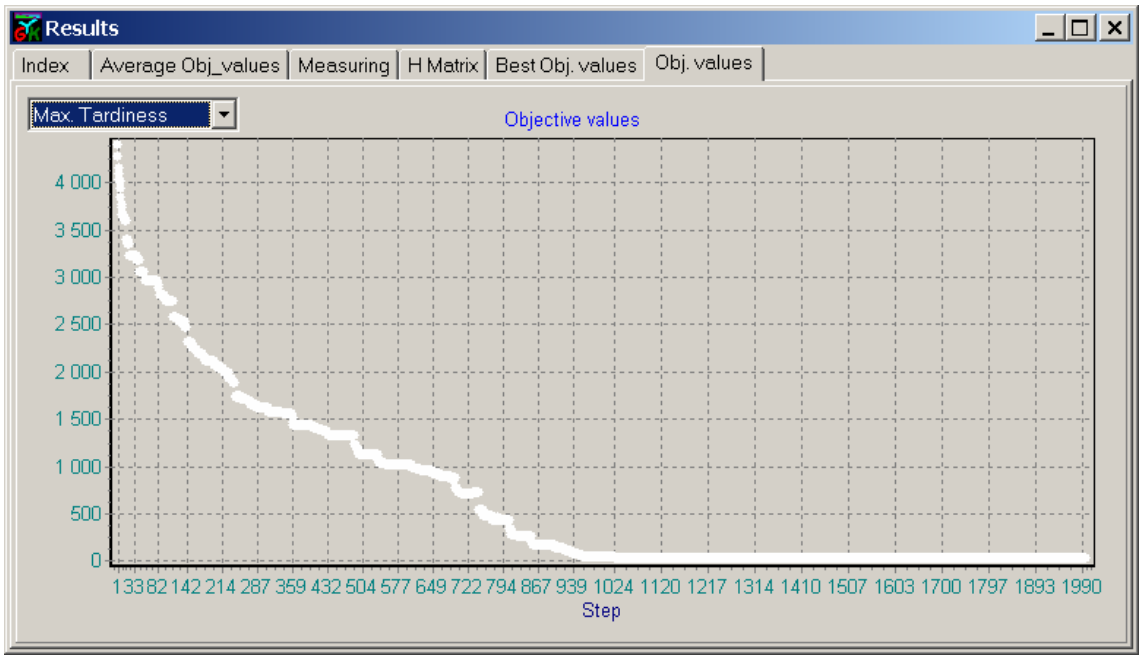
A keresési folyamat során a célfüggvények alakulása a megtett lépések függvényében, véletlenszerűen felépített kezdeti ütemtervből kiindulva:



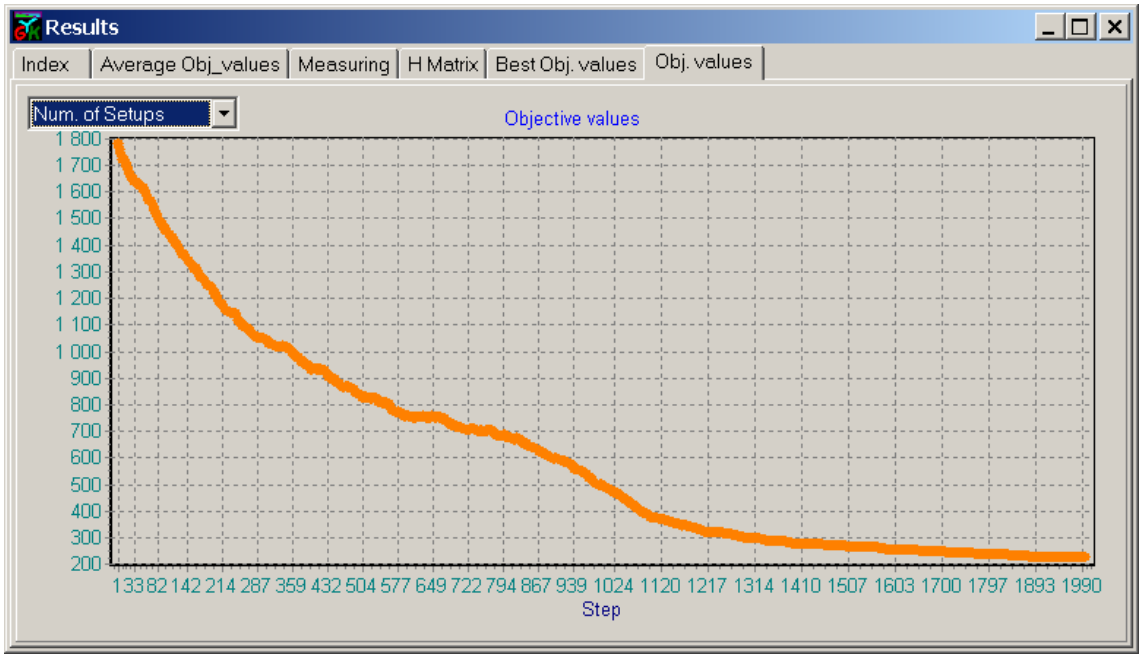

A késő megrendelések számának változása.



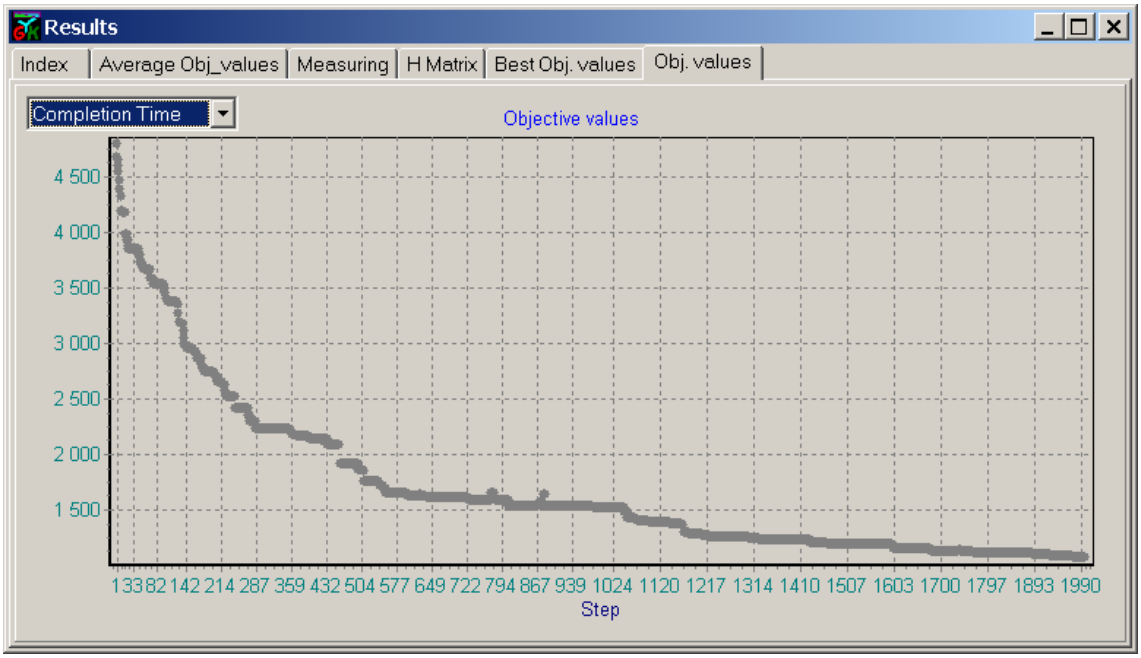

A késő munkák számának változása.




A legnagyobb késés változása.

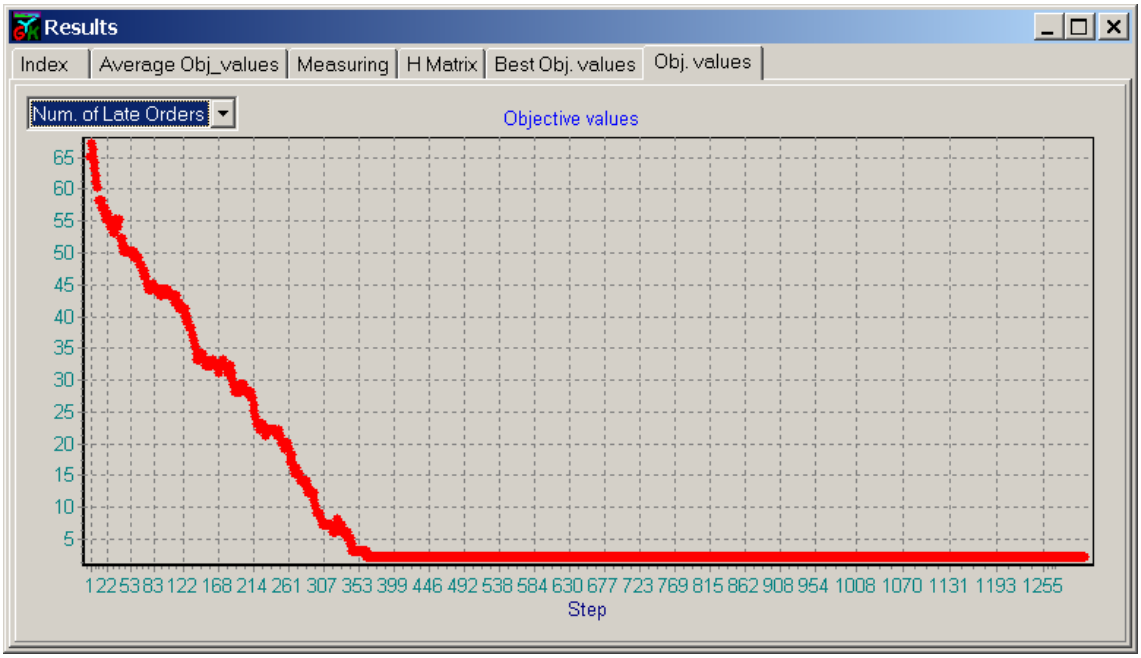



A gépátállítások számának változása.

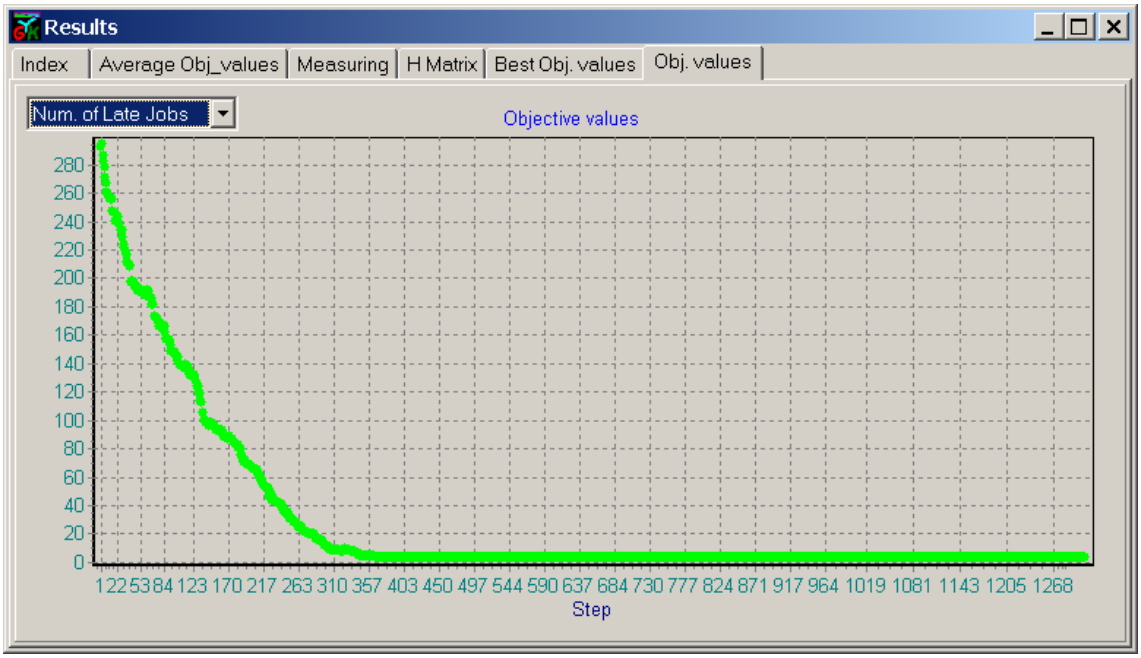



A legkésőbbi befejezési időpont változása.

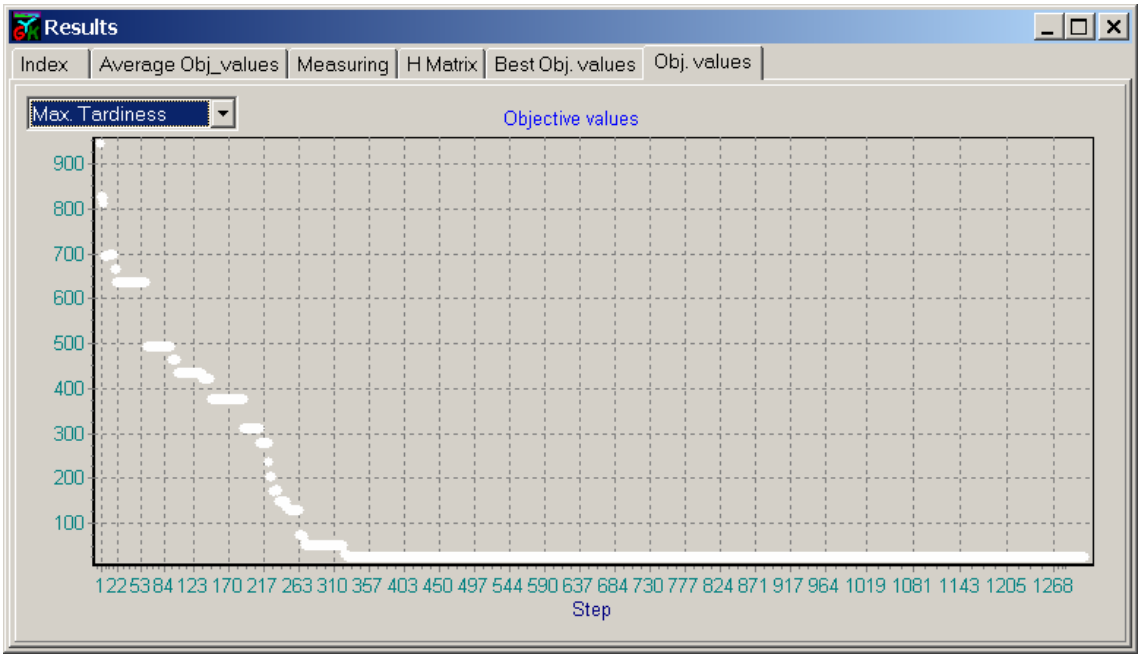
A keresési folyamat során a célfüggvények alakulása a megtett lépések függvényében, *BWBA* heurisztikusan felépített kezdeti ütemtervből kiindulva:



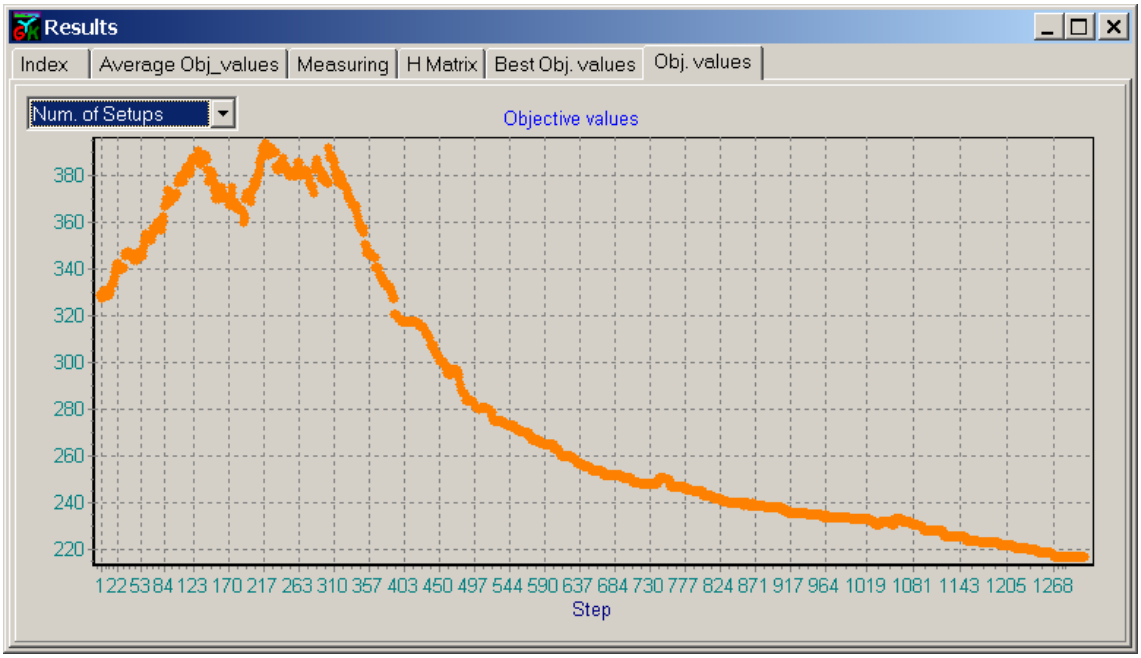

A késő megredelések számának változása.



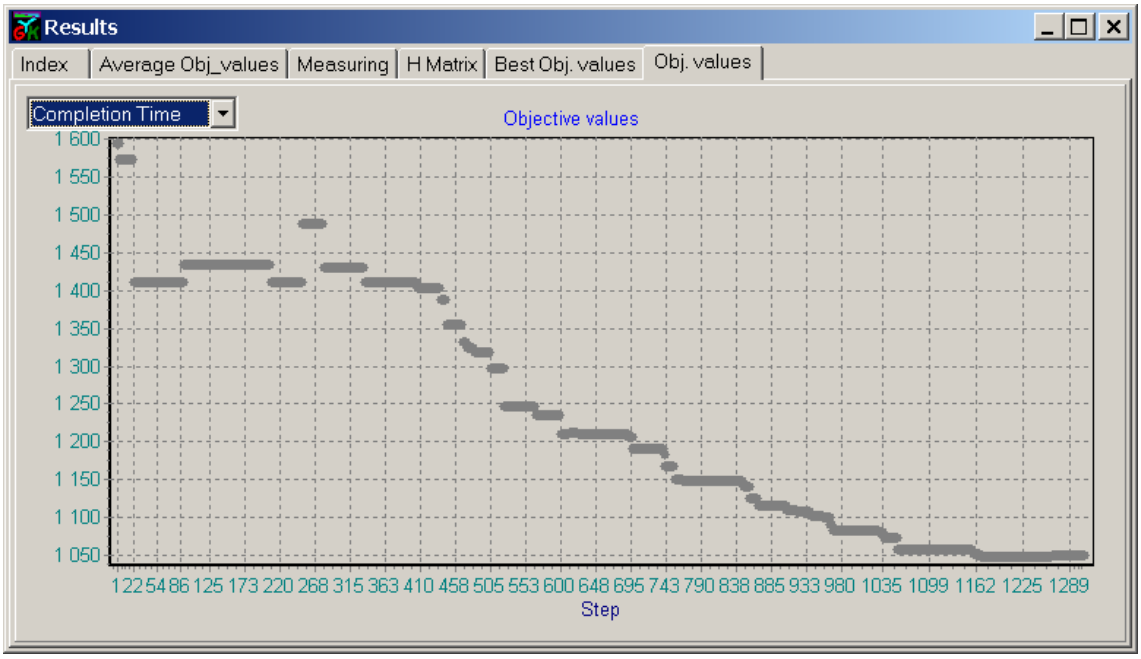

A késő munkák számának változása.




A legnagyobb késés változása.




A gépátállítások számának változása.




A legkésőbbi befejezési időpont változása.

***Tl_2:***

Performance test. Input file :C:\kulcsar\EFFS\l_DATA\li2\li2.txt

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Prob_Sizes: | | Machine:56 | | Orders:424 | | Jobs:2360 | | Setup_Types:4 | | |
| Search_Pars: | | maxSTEP:2500 | | maxLOOP:100 | | minLOOP:10 | | maxImpless:40 | | maxTABU:150 |
| Nx_op_pri: | | N1: 1 | N2: 1 | N3: 1 | N4: 1 | N5: 1 | | | | |
| ObjF: | LOrder | LJob | sumT | maxT | Set | maxC | F(r,x) | F(h,x) | RunT | BStep |
| ObjPri: | 7 | 8 | 10 | 10 | 7 | 5 | | | | |
| 1: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 343 | 1364 | 3605777 | 8024 | 1573 | 8519 | | | | |
| heur | 202 | 929 | 436308 | 1636 | 681 | 2413 | - | | | |
| search | 16 | 60 | 5829 | 302 | 288 | 1686 | - | - | 5min 12s | [2499] |
| 2: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 339 | 1372 | 3885252 | 8925 | 1536 | 9438 | | | | |
| heur | 207 | 934 | 458321 | 1731 | 692 | 2498 | - | | | |
| search | 16 | 62 | 6499 | 286 | 331 | 1762 | - | - | 3min 15s | [1829] |
| 3: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 344 | 1403 | 3737266 | 8243 | 1579 | 8772 | | | | |
| heur | 205 | 926 | 466921 | 2090 | 689 | 2537 | - | | | |
| search | 16 | 62 | 6735 | 286 | 312 | 1678 | - | - | 3min 8s | [1854] |
| 4: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 338 | 1387 | 3707010 | 8688 | 1527 | 9303 | | | | |
| heur | 201 | 908 | 446960 | 1685 | 676 | 2437 | - | | | |
| search | 15 | 59 | 5515 | 302 | 308 | 1738 | - | - | 3min 9s | [1783] |
| 5: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 338 | 1373 | 3532821 | 8316 | 1499 | 8985 | | | | |
| heur | 200 | 924 | 463197 | 1767 | 689 | 2610 | - | | | |
| search | 13 | 53 | 5726 | 317 | 292 | 1743 | - | - | 3min 46s | [1969] |
| 6: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 335 | 1391 | 4006473 | 8368 | 1589 | 8746 | | | | |
| heur | 203 | 927 | 463073 | 2127 | 687 | 2578 | - | | | |
| search | 16 | 64 | 5893 | 286 | 311 | 1688 | - | - | 4min 11s | [2088] |
| 7: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 340 | 1389 | 3616455 | 8152 | 1533 | 8623 | | | | |
| heur | 213 | 953 | 460564 | 1705 | 712 | 2337 | - | | | |
| search | 12 | 51 | 5127 | 302 | 319 | 1696 | - | - | 3min 37s | [1925] |
| 8: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 341 | 1396 | 3966620 | 8315 | 1525 | 8795 | | | | |
| heur | 198 | 914 | 453509 | 1853 | 693 | 2462 | - | | | |
| search | 17 | 55 | 6381 | 318 | 325 | 1716 | - | - | 3min 15s | [1820] |
| 9: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 334 | 1383 | 3797006 | 8715 | 1534 | 9251 | | | | |
| heur | 203 | 922 | 450043 | 1819 | 685 | 2397 | - | | | |
| search | 17 | 61 | 5729 | 302 | 310 | 1740 | - | - | 3min 27s | [1891] |
| 10: | ------ | ------ | ------ | ------ | ------ | ------ | | | | |
| rand | 332 | 1368 | 3574951 | 7854 | 1536 | 8272 | | | | |
| heur | 201 | 918 | 455900 | 2114 | 702 | 2561 | - | | | |
| search | 18 | 76 | 7675 | 304 | 268 | 1650 | - | - | 5min 12s | [2494] |
| Avp(10it): | | ====== | ====== | ====== | ====== | ====== | ====== | | | |
| rand: | 338.4 | 1382.6 | 3.7e+06 | 8360 | 1543.1 | 8870.4 | | | | |
| heur: | 203.3 | 925.5 | 455480 | 1852.7 | 690.6 | 2483 | - | | | |
| search: | 15.6 | 60.3 | 6110.9 | 300.5 | 306.4 | 1709.7 | - | - | 3min 49s | |

| ObjF: | LOrder | LJob | sumT | maxT | Set | maxC |
|---|---|---|---|---|---|---|
| ObjPri: | 7 | 8 | 10 | 10 | 7 | 5 |
| | ------ | ------ | ------ | ------ | ------ | ------ |
| s_1: | 16 | 60 | 5829 | 302 | 288 | 1686 |
| s_2: | 16 | 62 | 6499 | 286 | 331 | 1762 |
| s_3: | 16 | 62 | 6735 | 286 | 312 | 1678 |
| s_4: | 15 | 59 | 5515 | 302 | 308 | 1738 |
| s_5: | 13 | 53 | 5726 | 317 | 292 | 1743 |
| s_6: | 16 | 64 | 5893 | 286 | 311 | 1688 |
| s_7: | 12 | 51 | 5127 | 302 | 319 | 1696 |
| s_8: | 17 | 55 | 6381 | 318 | 325 | 1716 |
| s_9: | 17 | 61 | 5729 | 302 | 310 | 1740 |
| s_10: | 18 | 76 | 7675 | 304 | 268 | 1650 |
| best solution: | s_7 | | | | | |

Rel. quality of s_row: - better, + worse, 0 equal.

| | s_1 | s_2 | s_3 | s_4 | s_5 | s_6 | s_7 | s_8 | s_9 | s_10 |
|---|---|---|---|---|---|---|---|---|---|---|
| s_1 | 0 | - | - | + | + | - | + | - | - | - |
| s_2 | + | 0 | + | + | + | + | + | - | + | - |
| s_3 | + | - | 0 | + | + | + | + | - | + | - |
| s_4 | - | - | - | 0 | + | - | + | - | - | - |
| s_5 | - | - | - | - | 0 | - | + | - | - | - |
| s_6 | + | - | - | + | + | 0 | + | - | - | - |
| s_7 | - | - | - | - | - | - | 0 | - | - | - |
| s_8 | + | + | + | + | + | + | + | 0 | + | - |
| s_9 | + | - | - | + | + | + | + | - | 0 | - |
| s_10 | + | + | + | + | + | + | + | + | + | 0 |

End of file.

A *Tl_2* feladat generálására használt paraméterek a 32. és 33. ábrának megfelelő sorrendben:

*Machine*

4
30 20
6 20
10 20
10
400
1000 20
100 20

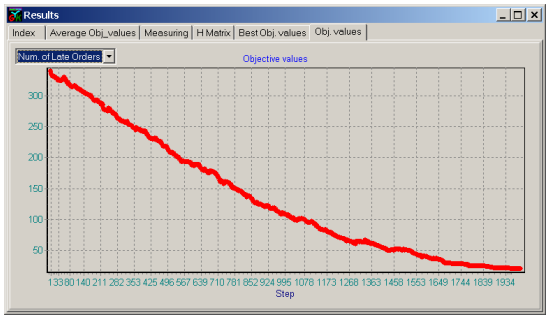
*Order*

40 20
6 20
200 20
300 20
1000 80
8

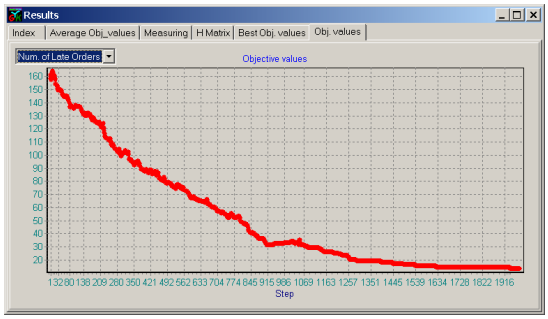
A következő ábrákon a célfüggvények alakulása látható a keresési folyamat során megtett lépések függvényében:

(a) véletlenszerűen felépített kezdeti ütemtervből kiindulva (bal oldali hasáb).

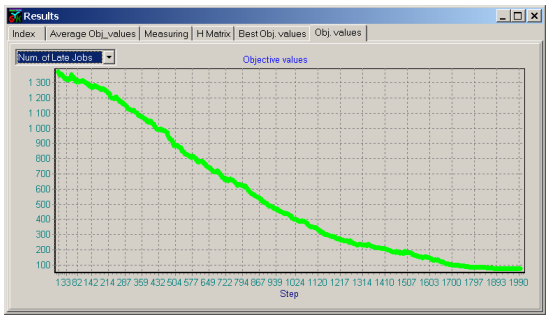
(b) a *BWBA* heurisztikus algoritmussal felépített kezdeti ütemtervből kiindulva (jobb oldali hasáb).



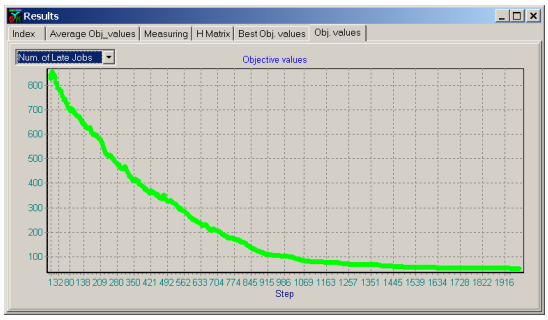

A késő megrendelések számának változása.



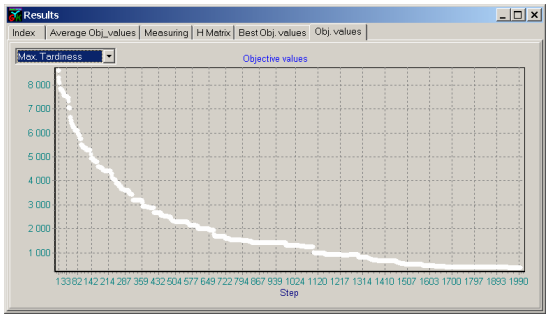

A késő megrendelések számának változása.



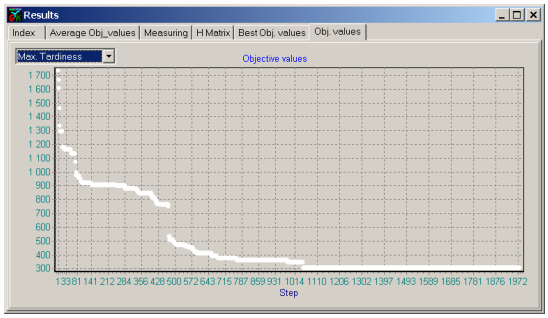

A késő munkák számának változása.



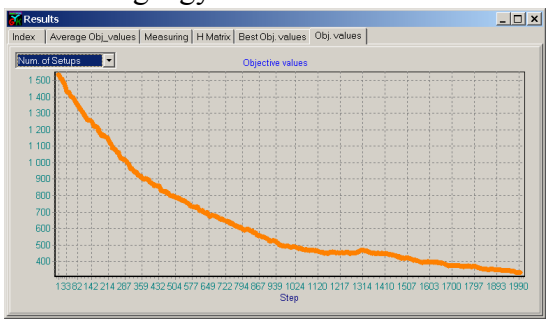

A késő munkák számának változása.



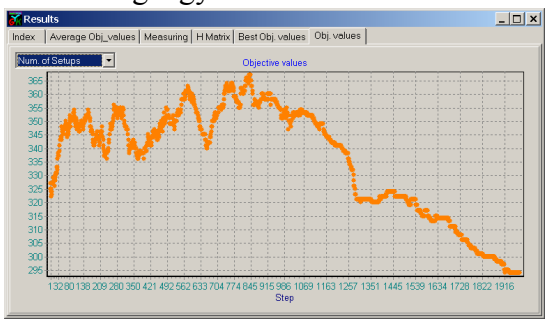

A legnagyobb késés változása.



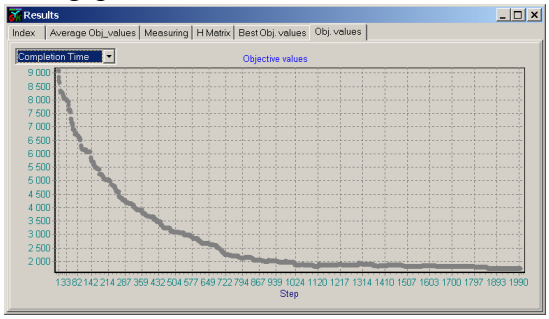

A legnagyobb késés változása.



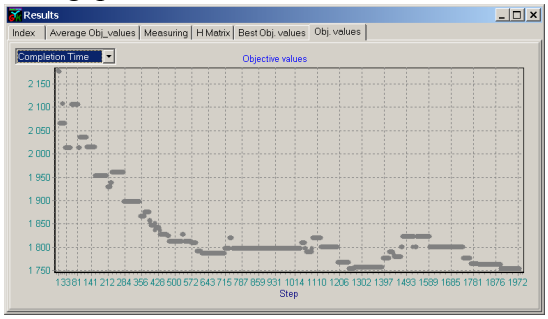

A gépátállítások számának változása.




A gépátállítások számának változása.




A legkésőbbi befejezési időpont változása.




A legkésőbbi befejezési időpont változása.

# SUMMARY

The theses summarize the main results of the author's research in the field of short time shop floor scheduling. A new extended flexible flow shop scheduling model and multi-phase heuristic solving methods are presented. The new approach focuses on creating near-optimal feasible schedule considering multiple objectives and constraints of customized mass production. The approach uses heuristic algorithms, multi-objective searching techniques, relative qualification of the feasible solutions and problem space transformation based on execution simulation. The proposed methods can be applied for solving large size problems effectively in a reasonable amount of time. Freezing techniques, performance indices and neighbouring operators are developed for using in the proposed multi-objective searching algorithm in order to satisfy additional constraints and requirements of rescheduling problems. A software-prototype is implemented based on the new integrated approach for supporting the scheduling and rescheduling tasks of the shop floor control and uncertainty management. Services of the software provide useful tools for supporting the user interaction in order to increase flexibility and effectiveness of the scheduling/rescheduling processes. This combined approach based on human and artificial intelligence can yield significant performance improvements.

The structure of the thesis is the following: Chapter 1 gives an introduction to the scope of the research and presents the main purpose of the work. Chapter 2 gives an overview of the flow shop and the flexible flow shop models and their solving methods published in the literature. The role and the main tasks of the manufacturing execution systems are also surveyed. Chapter 3 briefly summarizes the general characteristics of the shop floor fine scheduling problems in customized mass production. A formal specification of a new class of scheduling problems is presented. A special data model is proposed to realize the model and to accelerate the computation in question. In Chapter 4 a new integrated approach is proposed, which manages all the sub-problems (batching, assigning, sequencing and timing) simultaneously and solves them together. The details of the proposed methods are also demonstrated. Chapter 5 addresses the issues of the rescheduling tasks. A novel model and advanced methods are suggested for solving the rescheduling problems. Chapter 6 details the developed software representation of the new approach applied by the author. The software is also suitable for supporting the uncertainty management in manufacturing execution system environment. Final chapters of the thesis summarize the new scientific results, show the list of the publications and pose some directions in which the results can be further developed.

**Thesis 1 A new extended flexible flow shop scheduling model and its computer representation.**

***I defined a new class suitable for solving extended flexible flow shop (EFFS) problems. I elaborated a special model representation for computer application to be developed. The novelty of this model representation is as follows: it is able to take into consideration the most important general characteristics and requirements of customized mass production (CMP).***

**Thesis 2 Multi-phase heuristic method for solving EFFS scheduling problems.**

***A new integrated approach based method has been developed for solving fine scheduling problems, which can be managed by the EFFS model. The method supports the decision making of joining and/or dividing the production orders, calculation of the manufacturing lot sizes dynamically, management of the alternative technological routes, allocation of the machine resources, definition of the manufacturing tasks and scheduling its execution processes. The aforementioned method uses heuristic algorithms, searching techniques and problem space transformation based on execution simulation.***

**Thesis 3 A new method for relative qualification of the feasible solutions of a multi-objective combinatorial optimization problem.**

***A mathematical model and a solving method have been developed for managing the objective functions and evaluating the relative quality of the feasible solutions by comparing them to each other. The model has been applied to multi-objective combinatorial optimization problems, which include objective functions characterized by dynamically varying importance, different dimension and value range.***

**Thesis 4 An integrated fine scheduling model for supporting uncertainty management in customized mass production.**

***A new integrated model has been developed and its computer (software) representation has been implemented for supporting the scheduling and rescheduling tasks of the shop floor control and uncertainty management at the MES level. The solution is suitable for managing some problems of shop floor control in an integrated and extended functional form.***