DOKTORI ÉRTEKEZÉS

Láng Blanka

Pécs, 2010

PÉCSI TUDOMÁNYEGYETEM EGYETEM
KÖZGAZDASÁGTUDOMÁNYI KAR

GAZDÁLKODÁSTANI DOKTORI ISKOLA

Láng Blanka

A nettó jelenérték maximalizálása az erőforráskorlátos projekt ütemezés során - egy új harmónia kereső heurisztika

DOKTORI ÉRTEKEZÉS

Témavezetők: Dr. Csébfalvi Anikó, CSc, PhD
Dr. Csébfalvi György, CSc

Pécs, 2010

Tartalomjegyzék

Bevezetés

Disszertációnkban egy új harmóniakereső metaheurisztikát mutatunk be, amely a minimális időtartamú erőforráskorlátos ütemezések halmazán a projekt nettó jelenértékét maximalizálja.

Napjainkban a projekt ütemezés problémaköre egyre növekvő fontossággal bír. Különböző célokat valósítunk meg a projekt ütemezés során, a tevékenységek között fennálló elsőbbségi feltételek és esetlegesen fellépő erőforráskorlátok megléte mellett – például projekt időtartamának minimalizálása, erőforrás fogyasztás kisimítása – mely célok közül a projekt nettó jelenértékének maximalizálása (a projekt időtartamának minimalizálása mellett) talán a legéletszerűbb, pénzügyileg leginkább motivált cél különösképpen hosszú időtartamú projektek esetében.

A technikai fejlődésnek megfelelően a probléma méretek egyre nőnek, a valós életben pedig az erőforrások igen szűkösek, ez arra készteti a kutatókat, hogy egyre jobb és hatékonyabb egzakt és heurisztikus megoldásokat dolgozzanak ki.

A valós életben előforduló projektek általában nagy méretű, sok tevékenységből álló projektek. Az ilyen esetekben, amikor a projekt záró időpontja időben kitolódik – jellemzően például a gyógyszerfejlesztési projektek, az építőipari projektek, a jármű tervezés, vasútfejlesztés, haditechnikai védelmi projektek, elméleti alapkutatások – a nettó jelenérték maximalizálása és a projekt időtartamának minimalizálása egyaránt kiemelt gyakorlati jelentőséggel bír a projekt menedzserek számára a feladat ütemezés során.

Az első fejezetben a probléma terület legfontosabb fogalmait tekintjük át.

A második és harmadik fejezetben áttekintjük a(z) - elsősorban nemzetközi - szakirodalom korábbi eredményeit, a reguláris, illetve irreguláris modelleket, az erőforráskorlátos, illetve erőforráskorlát nélküli megoldásokat, az egzakt és a heurisztikus megoldásokat. Vizsgáljuk a korábbi időtartam minimalizáló és a nettó jelenérték maximalizáló eljárásokat is, mivel disszertáció-béli algoritmusunk két ilyen típusú célfüggvénnyel dolgozik. Kiemelten foglalkozunk a heurisztikus, és azon belül a metaheurisztikus megoldásokkal, mivel a probléma természetéből adódóan az ilyen megoldások bizonyulnak hatékonynak. Az irodalomban bemutatott eredményekkel alátámasztjuk azt az állításunkat, hogy a disszertációban tárgyalt problémára a jövőben a leghatékonyabb megoldásokat a metaheurisztikák között érdemes keresnünk. Bemutatjuk a területen jelenleg egyik leghatékonyabbnak bizonyuló heurisztika típust, a harmónia kereső metaheurisztikát.

Ezen előzmények után a negyedik fejezetben rátérünk új harmóniakereső algoritmusunk ismertetésére, amely a minimális időtartamú erőforráskorlátos ütemezések halmazán a projekt nettó jelenértékét maximalizálja. Az optimális ütemezés elméletileg két egészértékű bináris programozási feladat megoldását jelenti, ahol az első lépésben meghatározzuk a minimális időtartamú erőforráskorlátos ütemezések időtartamát, majd a második lépésben az optimális időtartamot feltételként kezelve megoldjuk a nettó jelenérték maximalizálási problémát minimális időtartamú erőforráskorlátos ütemezések halmazán. A probléma NP-nehéz[1] jellege miatt azonban az egzakt megoldás elfogadható idő alatt csak kisméretű projektek esetében képzelhető el.

A bemutatandó új metaheurisztika a Csébfalvi [2007] által a minimális időtartamú erőforráskorlátos ütemezések időtartamának meghatározására és a tevékenységek ennek

[1] NP-nehéz, nincs polinomiális idő alatt megoldható algoritmus, mely megoldást ad.

megfelelő ütemezésére kifejlesztett harmóniakereső metaheurisztika továbbfejlesztése. A továbbfejlesztés egyik legfontosabb pontja, hogy az új eljárás a rejtett erőforrás felhasználási konfliktusokat konfliktus javító elsőbbségi relációk beépítésével oldja fel.

Részletesen ismertetjük modellünket, az algoritmus főbb lépéseit, végigkövetjük egy konkrét projektre alkalmazva az ütemezés lépéseit, majd ismertetjük eljárásunk pszeudokódját.

Mivel legjobb tudomásunk szerint ilyen elsődleges - másodlagos szempont szerinti (bi-criteria) optimalizálásos megoldás korábban nem született a pénzmozgásos RCPSP problémára, eljárásunk eredményeit nem tudjuk összehasonlítani korábbi megoldások eredményeivel. Az ajánlott metaheurisztika hatékonyságának és életképességének szemléltetésére számítási eredményeket adunk a jól ismert és népszerű PSPLIB tesztkönyvtár J30 részhalmazán futtatva.

Az utolsó fejezetben megadjuk a lehetséges továbbfejlesztési irányokat.

A dolgozatban az alábbi jelöléseket alkalmaztuk:

1. táblázat Jelölések

A_t	A t időperiódusban aktív tevékenységek halmaza
D_i	Az i-edik tevékenység nem megszakítható időtartama
E	Az $i \rightarrow j$ elsőbbségi kapcsolatok halmaza
$i \rightarrow j$	Az i tevékenység a j tevékenység közvetlen előzménye
N	A valódi tevékenységek száma, $N = 1,2,...,N$
i	Az i-edik tevékenység indexe, $i \in \{1,2,...N\}$
$R_{i,r}$	Az i-edik tevékenység végrehajtásához szükséges erőforrás igény az r-edik erőforrásból, $r \in \{1,...R\}$
R	Az erőforrások száma
R_r	Az r-edik erőforrás felső korlátja.
$\overline{T}$	A tevékenységek időtartamainak összege
$X_{i,t}$	Bináris változó, $X_{i,t} \in \{0,1\}$, $i = \{1,2,...N\}$ $X_{i,t} = 1$ ha az i-edik tevékenység t időperiódusban kezdődik el, egyébként $X_{i,t} = 0$
C_i	Az i-edik tevékenység pénzmozgása.
$C_{i,t}$	A t időperiódusban kezdődő, i-edik tevékenység pénzmozgása a projekt kezdetére visszadiszkontálva, $i = \{1,2,...N\}$, $t \in T_i$
NPV	A projekt nettó jelenértéke
$U_{t,r}$	t időpontban, az r-edik erőforrásból felhasznált mennyiség
X_i	Az i-edik tevékenység kezdési időpontja

$\underline{X_i}$	Az i-edik tevékenység legkorábbi kezdési időpontja a nemkorlátos, csak elsőbbségi korlátokat kielégítő esetben
$\overline{X_i}$	Az i-edik tevékenység legkésőbbi kezdési időpontja a nemkorlátos, csak elsőbbségi korlátokat kielégítő esetben
T_i	Az i-edik tevékenység legkorábbi és legkésőbbi kezdési időpontja közé eső értékek.
T	A projekt időperiódusainak száma, $t \in \{1,2,...,T\}$
PS	A közvetlen megelőző-követő relációk halmaza.
PS^+	A PS halmaz és a konfliktus javító reláció halmaz úniójának nem redundáns részhalmaza.
$\underline{T}$	Az elsőbbségi korlátokat kielégítő projekt minimális időtartama
CU_{tr}^+	A t időperiódusban az újrainduló erőforrás egységek száma az r-edik erőforrásból
CU_{tr}^-	A t időperiódusban a leálló erőforrás egységek száma az r-edik erőforrásból

.

1. A projekt ütemezési probléma elemei

Egy projekt ütemezési probléma *tevékenység*ekből, köztük fennálló *elsőbbségi kapcsolat*okból, a tevékenységek végrehajtásához szükséges *erőforrás*okból és projekt teljesítmény-mértékekből (azaz *ütemezési cél*okból) áll (Kolisch-Padman [2001]). Az ütemezési problémák osztályozásához e négy komponens figyelembe vételét ajánlja Herroelen és ts. [1999]. Az alábbiakban e négy összetevő jellemzőit tekintjük át, kiegészítve a különböző projekt *ábrázolási módszerekkel*, *teszthalmaz*ok, és a *nettó jelenérték* fogalmának bemutatásával.

1.1. Tevékenységek és elsőbbségi feltételek, kritikus út

Egy projekt tevékenységekből (activities, tasks, operations, vagy jobs) áll, ezeket a tevékenységeket kell végrehajtanunk azért, hogy a projektet teljesítsük. Ezen tevékenységek egymással kapcsolatban állnak, a köztük fennálló elsőbbségi feltételek lehetnek vég-vég, kezdet-kezdet, kezdet-vég és vég-kezdet kapcsolatok. A legtöbbször az utóbbit használják nulla fáziskéséssel (time lag), azaz egy tevékenység pontosan akkor kezdődhet el, ha minden közvetlen előzménye befejeződött. Minden tevékenységet jellemez időtartama (duration), erőforrás igénye (resource requirement), és jellemezheti a hozzá kapcsolódó pénzmozgás (cash flow). A tevékenységhez társított pénzmozgás lehet kiadás (cash outflow), ekkor negatív a pénzmozgás, vagy bevétel (cash inflow), ekkor pozitív a pénzmozgás. Több módú esetben (multi-mode) a tevékenységeknek több végrehajtási módja lehet, azaz több erőforrást megengedve rövidülhet a tevékenység időtartama (a tevékenységek „effort-driven”-ek). Egy megvalósítási módú (single mode) esetben – leginkább ilyen esetekkel foglalkozunk disszertációnkban - minden tevékenységnek egy végrehajtási módja van, azaz erőforrás

igénye meghatározott, és a tevékenység időtartama előre adott. A tevékenységeket egyes modellekben megszakíthatónak tekintik, de disszertációnkban csak a nemmegszakítható esetekkel foglalkozunk, tehát ha egy tevékenység végrehajtását megkezdjük, akkor azt megszakítás nélkül végre kell hajtani.

Amennyiben adottak a projekt tevékenységei, az azok elvégzéséhez szükséges idő, és a tevékenységek közt fennálló elsőbbségi korlátok, a *kritikus út* módszer (Critical Path Method, CPM) olyan ütemezést ad, melynek időszükséglete minimális. Az eljárás során az egyes tevékenységek legkorábbi, illetve legkésőbbi kezdési időpontjait határozzuk meg. Először a nulladik, általvékenység kezdési idejét rögzítjük a nulla időpontban, majd rekurziót alkalmazva növekvő index szerint meghatározzuk a többi tevékenység kezdési idejét úgy, hogy figyelembe vesszük, hogy egy tevékenység legkorábban akkor kezdődhet el, ha az őt megelőző befejeződött, tehát $\underline{X}_j = \max(D_i + \overline{X}_i)$ minden j-re, ahol $i \to j$. Majd ugyanezt az eljárást elvégezzük visszafelé. Rögzítjük az $(n+1)$-ik tevékenység legkésőbbi kezdési időpontját a legkorábbi kezdési időpontjába: $\overline{X}_{N+1} = \underline{X}_{N+1}$. Ezután csökkenő index szerint meghatározzuk sorban a legkésőbbi kezdési időpontokat. Egy tevékenységet *kritikus*nak tekintünk, ha a legkorábbi kezdési ideje és a legkésőbbi kezdési ideje azonos, azaz késleltetése a projekt ugyanolyan mértékű késleltetését jelentené.

1.2. Erőforrások

A tevékenységek végrehajtásához erőforrásokra van szükség, melyekből gyakran szűkösek a készleteink.

Slowinsky és Weglarz [1978] háromféle erőforrás típust említ:

Megújuló (renewable) erőforrás például a munkaerő, gépek. Ebben az esetben az erőforrás mennyisége a felhasználás során nem csökken, az időegységenként rendelkezésre álló mennyiség független az erőforrás-felhasználástól, állandó a projekt során.

Nemmegújuló (non-renewable) erőforrás például a pénzügyi eszközök, anyagok. Ebben az esetben az erőforrás mennyisége a felhasználás során csökken, a felhasználás nem csak projekt szinten, hanem időegységenként is korlátozott. Így a nem-megújuló erőforrásokat *kétszeresen korlátozott (doubly constrained)* erőforrásként is emlegetik. Talbot [1982] megmutatta, hogy minden duplán korlátos erőforrás leírható egy megújuló és egy nem megújuló erőforrással.

A negyedik típust később vezették be Böttcher és ts. [1999]. *Részlegesen megújuló (partially renewable)* erőforrás mennyiségét a projekt egy részhalmazán korlátozzuk. Ilyen például annak a munkásnak a munkaideje, akinek heti óraszáma korlátos.

1.3. Ábrázolási módszerek

A projektek ábrázolása segít megérteni azok sajátosságait és létrehozni a számunkra leginkább megfelelő ütemezést. A projekteket hálókkal ábrázolhatjuk, melyek összefüggő, irányított, és körmentes gráfok.

Az egyik legelterjedtebb ábrázolási mód az AoN (Activity on Node) gráf, melyben a tevékenységeket a gráf csúcspontjaival, a köztük fennálló logikai kapcsolatokat (megelőző-követő) irányított élekkel jelöljük. Ezzel ellentétben az AoA (Activity on Arc) esetében a gráf csúcspontjai eseményeket ábrázolnak, a nyilak pedig tevékenységeket.

A két ábrázolási mód más-más pénzmozgás felfogást tükröz. Az AoA ábrázolási módszer esetén a pénzmozgások a pontokban következnek be, azaz amikor a

tevékenységek kezdődnek, illetve befejeződnek. Az AoN ábrázolási módszer esetén a pénzmozgásokat általában magukhoz a tevékenységekhez társítjuk.

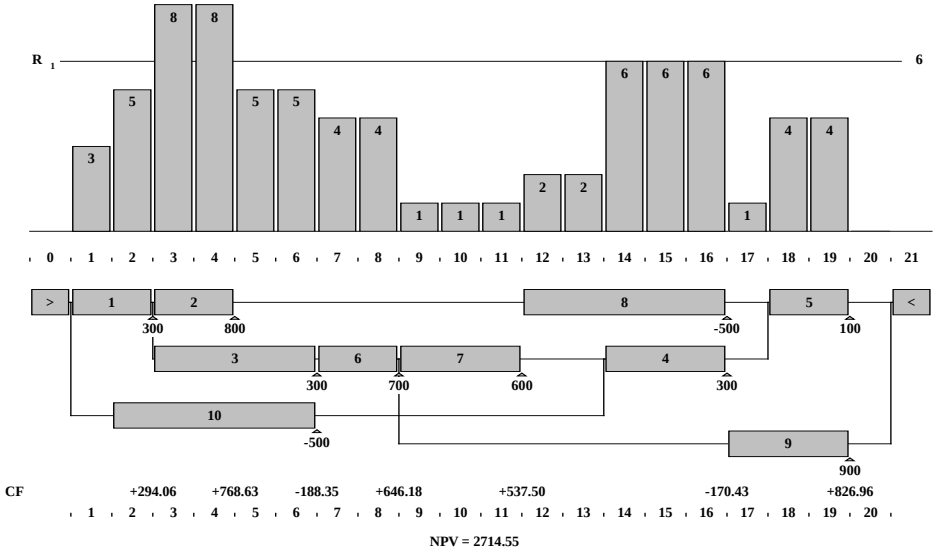
A fenti leginkább használt ábrázolási módok mellett egyéb használatos ábrázolási módszerek: erőforrás felhasználási sáv, oszlopdiagram, precedencia diagram, idősklázott CPM diagram, függőségi kapcsolatokkal kiegészített Gantt diagram - ez utóbbi az AoN gráfok egy speciális változata. Ebben az esetben az egyes csúcspontokat téglalapokkal helyettesítjük, az adott tevékenység időszükségletét a téglalap hossza képviseli. Vízszintes időtengelyek mentén helyezzük el a téglalapokat, ahol a téglalap bal széle a tevékenység kezdési idejére, a jobb széle a befejezési idejére illeszkedik. A tevékenységek között fennálló elsőbbségi feltételeket a téglalapok közötti összekötő vonalakkal, nyilakkal adjuk meg.

Természetesen minden ábrázolási módszernek vannak bizonyos esetekben előnyös, bizonyos esetekben előnytelen tulajdonságai, hiszen az ütemezési információk egy csoportját kihangsúlyozza, míg a többit elhanyagolja. Ennek ellenére szükséges őket alkalmazni, hogy a megfelelő ütemezés megtalálása érdekében a projekt szerkezetét minél jobban megértsük. A jelenlegi gyakorlatban a tevékenység-orientált szemlélet (AoN) túlsúlya figyelhető meg az esemény-orientált (AoA) szemlélettel szemben, mivel az előbbi alkalmazásával általában jobban áttekinthetőek a komplex kapcsolatok.

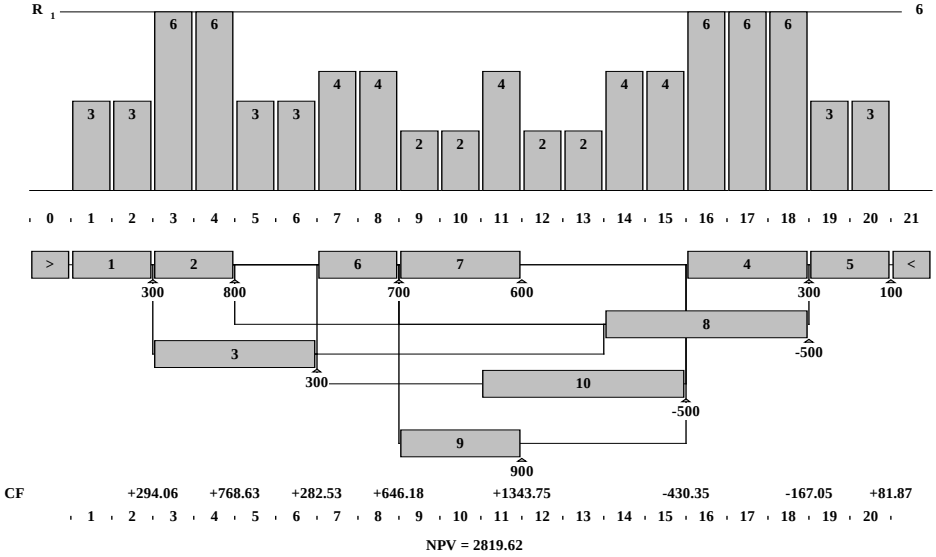
Az alábbi ábrákon ugyanazon projekt két különböző ütemezését láthatjuk. A projekt 10 tevékenységből áll, melyek egy erőforrást igényelnek. Az első ütemezés nem erőforráskorlátos, a projekt nettó jelenértéke *2 714,55*. A 10. tevékenységre konfliktusjavító feltétel bevezetésével elérjük, hogy a második ütemezés már erőforráskorlátos, és a nettó jelenértéke még nőtt is: *2 819,62*, köszönhetően annak, hogy a nagy negatív pénzmozgásos tevékenységeket igyekeztünk késleltetni, a pozitív

pénzmozgással rendelkezőket pedig minél korábbi időpontra ütemezni. A projekt időtartama közben nem változott.

1. ábra Projekt diszkontált pénzmozgásokkal



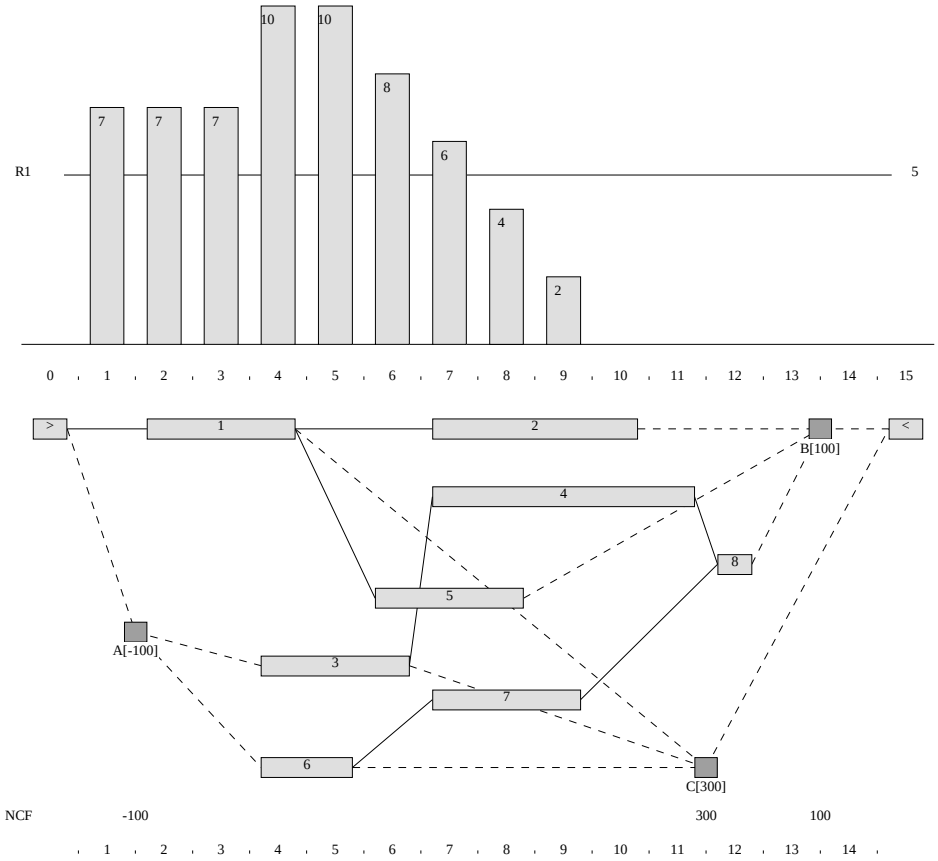
2. ábra Optimális ütemezés



A fenti megoldást bizonyos esetekben hasznos továbbgondolni. Érdemes figyelembe venni azt a legtöbbször elhanyagolt tényt, hogy az erőforrás felhasználás tevékenység orientált, míg a pénzmozgás eseményorientált. Negatív pénzmozgás általában egy tevékenységhalmaz megkezdése előtt következik be - ilyen például az anyagvásárlás -, pozitív pénzmozgás pedig tevékenység halmaz befejezése után, például részhatáridő teljesítésekor. Így a hagyományos ábrázolást kiegészíthetjük pénzmozgás eseményekkel, melyeket nulla időtartamú áltevékenységekkel ábrázolunk, melyekre az eredeti feltételekkel összhangban elsőbbségi feltételeket vezetünk be. A nulla időtartamú áltevékenységek a megelőző és következő tevékenység csoportok kezdő, illetve végpontjához kapcsolódnak elsőbbségi feltételekkel.

A 3. ábrán megfigyelhető a pénzmozgásokkal kibővített Gantt diagrammos ábrázolás. Egy kis méretű projekttel szemléltetünk, nyolc tevékenységet, egy erőforrást, két pozitív (*100, 300*) és egy negatív (*-100*) pénzmozgást feltételezünk. Az ábrázolt ütemezés nem erőforráskorlátos. A tevékenységeket téglalapokkal, a pénzmozgásokat körökkel ábrázoltuk, az elsőbbségi feltételeket egyenesekkel. A „normál” elsőbbségi feltételeket sima vonallal, a „nem triviális”, pénzmozgásokhoz kapcsolódó elsőbbségi feltételeket pontozott vonallal szemléltetjük.

3. ábra A pénzmozgások áltevékenységgel való szemléltetése



A 3. ábrán látható ábrázolási módszert alkalmazzuk a továbbfejlesztett algoritmusunk ismertetése során.

Ez az ábrázolás jó példa arra, hogy minden ábrázolásnak vannak előnyös és előnytelen tulajdonságai bizonyos esetekben. Amennyiben erőforráskorlátos tevékenységekből álló projekt ábrázolását tekintjük elsődleges célnak, legmegfelelőbb ábrázolási mód az 1. ábrán bemutatott módszer. Amennyiben azonban a tevékenységek pénzmozgásait is szeretnénk kihangsúlyozni az ábrázolás során, már a 3. ábrán bemutatott módszer az alkalmasabb, annak ellenére hogy így elveszítjük a tevékenység orientált ábrázolás bizonyos előnyeit.

1.4. Ellenőrzési technikák

A kutatóknak a hatékony megoldási eljárások keresése során az eljárások hatékonyságának összehasonlítása - a benchmarkolás – céljából szüksége volt hivatkozható adathalmazok kifejlesztésére. Davis [1969] készített először projekt ütemezési adathalmazokat tesztelés és összehasonlítás céljára.

Patterson [1984] 11 adathalmazt, Talbot [1978] további 10-et hozott létre. Patterson összegyűjtötte, rendszerezte az adathalmazokat, megadta a hozzájuk tartozó optimális megoldásokat, és napjainkban „Patterson probléma" néven hivatkozunk rájuk. A kutatók máig használják őket fejlesztéseik során. A Patterson problémák számos kitűnő tulajdonságuk mellett (széles körben használhatóak egzakt és heurisztikus megoldások során is, optimális projekt időtartam értékeket adnak, illetve két nagyon jól használható példát tartalmaznak, a P20-t, melyre igen nagyszámú, 2000 megoldás van, illetve a P90-t, melyre csupán 1 darab megoldás van) rendelkeznek egyes kutatók szerint egy hiányossággal: nem áll mögöttük egy pontosan definiált kísérleti terv. Ezért hozott létre Alvarez-Valdés és Tamarit [1994] új teszt halmazokat 5 előre meghatározott probléma paraméterrel.

Kolisch és ts. [1992] Demeulemeester és Herroelen [1992] „branch and bound" – a kifejezés többé-kevésbé elfogadott magyar fordítása „csoportosítás és korlátozás" - eljárását tanulmányozva megállapították, hogy az irodalomban eddig használt klasszikus teszthalmaz-beli problémák nem tesztelik kielégítően az egzakt megoldások hatékonyságát. Kolisch és ts. ezért egy új probléma paramétert ajánlottak, az *erőforrás erősséget* (resource strength), mely az igényelhető erőforrás alsó és felső korlátja konvex kombinációjának mértéke. Ezzel az új paraméterrel hoztak létre új teszt halmazokat. Külön könyvtárba rendezték az egymódú és a többmódú algoritmusok

teszthelésére szolgáló feladatokat. Mivel disszertációnkban ismertetett eljárás egymódú, ennek megfelelően az egymódú könyvtárral dolgoztunk. Ebben a könyvtárban 30, 60, 120 tevékenységből álló halmazokat hoztak létre, rendre J30, J60, J120 néven. Rendre 480, 480, 600 projektet tartalmaznak. Minden halmaz tartalmaz könnyen megoldható, nehéz és nagyon nehéz eseteket is. Minden feladatra az alkalmazott kísérleti tervnek megfelelően 10 változatot hoztak létre a különböző probléma karakterek értékét változtatva. A J30 halmazra ismertek az optimális megoldások, egzakt algoritmusokkal számolták ki. A J60 és J120 halmazokra nem ismert optimális megoldás, csak arra tudunk támaszkodni, amit már korábban beküldtek rájuk megoldásnak, illetve a relaxált megoldásokra. Ezeket úgy nyerjük, hogy a bináris feltételt – 0-1 kezdési időpont – relaxáljuk, folytonos változónak tekintjük, tehát a tevékenységeket megszakíthatónak tekintjük. Így kapunk alsó korlátokat. Az erőforráskorlátos projekt ütemezési problémák generálásához PROGEN programjukat használták.

Az egy megvalósítási móddal jellemezhető eseteken túlmenően publikáltak több megvalósítási móddal jellemezhető esetekre is teszteket. Kiegészítették a problémákat minimális és maximális fáziskésésekkel, és különböző erőforrás-költség célokkal is. Az összes teszt halmazt összegyűjtötték, és közkézre bocsátottak a http://129.187.106.231/psplib/ Internet címen, ahol bárki tesztelheti új megoldását, és be is küldheti, ha úgy találja, hatékonyabb, mint a meglévőek. A teszthalmazok alkalmazása jelenleg minden tudományos összehasonlítás elismert módszere. Mi is a PSPLIB tesztkönyvtár részhalmazaira adunk futási eredményeket hibrid algoritmusunkra vonatkozólag.

A projekt NPV-vel kapcsolatos kutatások motiválták Russelt, hogy oly módon módosítsa a Patterson problémákat, hogy a tevékenységekhez pénzmozgásokat társított.

Azonban túl nagy pénzmozgásokat rendelt a projektek befejeződési idejéhez, és ezért nem lehetett szignifikáns eltérést megfigyelni azok között az eljárások teljesítménye között, melyeket a projekt időtartamának minimalizálásra fejlesztettek ki elsősorban, és az NPV maximalizálásra is használták, és azok között az eljárások teljesítménye között, melyeket kifejezetten az NPV maximalizálásra fejlesztettek ki, figyelembe véve az NPV természetét.

Ennek a problémának a megoldására Padman és ts. [1997] kifejlesztettek egy un. PDS adathalmazt, melyben hat paraméterrel jellemezték a projekt környezetet: projekt mérete, projekt háló struktúrája, a pénzmozgás gyakorisága, az erőforráskorlátosság szintje, tőkeköltség, nyereség (profit margin).

1.5. A nettó jelenérték fogalma

Minden pénzügyi számítás végső soron a *jelenérték* (present value) fogalmára vezethető vissza. A jelenérték fogalma azt a tényt szemlélteti, hogy a pénzt ma befektetve a jövőben nagyobb értéket kapunk vissza. A jelenérték azt mondja meg, hogy mennyi a jövőben kapott pénz mostani értéke.

A pénz *nettó jelenértéke* (net present value) a pozitív és negatív előjelű pénzmozgások jelenértékeinek összege. A közismert NPV szabály szerint egy befektetést akkor érdemes megvalósítani, ha annak nettó jelenértéke pozitív.

$NPV = \sum C_t * e^{\alpha t}$, ahol

- NPV a nettó jelenérték (net present value)
- $C_t (t = 0,...,\infty)$ a t-dik időperiódus végén bekövetkező pénzmozgás
- α egy 1-nél kisebb diszkont faktor,
- t az időszakok száma

A fenti képletből nyilvánvalóan következik, hogy a negatív pénzmozgással rendelkező tevékenységeket minél későbbre, míg a pozitív pénzmozgással rendelkező tevékenységeket minél korábbra érdemes ütemezni, hogy minél magasabb NPV-hez jussunk.

1.6. A projekt ütemezés lehetséges céljai

A projektek ütemezése során a tevékenységeket a fentebb említett korlátok (elsőbbségi feltétel, erőforráskorlát) kielégítése mellett különböző célok szerint ütemezhetjük. A különböző célok közül csupán néhányat emelünk ki az alábbiakban, elsősorban azokat melyek szorosan kötődnek a disszertáció témájához, illetve azokat melyek igen széles körben kutattak. Nem foglalkozunk a több módú esetekkel – néhány kivételtől eltekintve –, a sztochasztikus tevékenység időtartamot feltételező esetekkel, azokkal a modellekkel, ahol nem vég-kezdet elsőbbségi feltételt használnak, illetve azokkal a modellekkel, ahol a tevékenységek megszakíthatóak végrehajtásuk során.

Az alábbiakban a különböző projekt ütemezési célokat (idő-alapú, erőforrás-alapú, pénzügyi, illetve ezek különböző kombinációi) regularitásuk szerint tekintjük át. Az 1.6.1 fejezetben a reguláris, az 1.6.2 fejezetben az irreguláris célfüggvényeket tárgyaljuk. Az osztályozáshoz a következő műveket használtuk fel: Neumann-Schwindt-Zimmermann [2002], Demeulemeester-Herroelen [2002], Kolisch-Padman [2001], Yang és ts. [2001]. A fejezet végén ismertetjük a több célfüggvényes problémák jellemzőit, mivel algoritmusunk is több célfüggvénnyel rendelkezik.

1.6.1. Reguláris célfüggvények

Reguláris célfüggvényről beszélünk a projektütemezés tárgyalása során, ha a célfüggvény értéke az ütemezendő tevékenységek befejezési időpontjaiban monoton

növekvő. A következőkben néhány, a disszertáció témájához szorosabban kötődő reguláris célfüggvényt említünk, melyek mellett természetesen számtalan további reguláris célfüggvény is létezik.

1.6.1.1. Projekt időtartamának minimalizálása

(project duration / makespan minimization)

A legegyszerűbb reguláris célfüggvény, célja a projekt összidejének minimalizálása a tevékenységekre vonatkozó elsőbbségi korlátok kielégítése mellett. A probléma ekvivalensen úgy is megfogalmazható, hogy a tevékenységek befejezési időinek összegét minimalizáljuk. Ebben az esetben minden tevékenység időtartama ismert, a tevékenységek végrehajtása megszakítás mentes. Amennyiben az erőforráskorlátokat nem vesszük figyelembe, a megoldás a háló-beli kritikus út hossza (lásd 1.1).

1.6.1.2. Az erőforráskorlátos projekt időtartamának minimalizálása

(*Resource Constrained Project Scheduling Problem with makespan minimization, RCPSP*)

Az egyik legszélesebb körben kutatott projekt ütemezési probléma. Az egy megvalósítási módú RCPSP célja az erőforráskorlátokat és a tevékenységekre vonatkozó elsőbbségi feltételeket kielégítő ütemezési időket találni, mely ütemezési idők esetén a projekt összideje minimális. A probléma NP-nehéz. (Blazewicz [1983])

1.6.1.3. NPV maximalizálás

Érdekesség kedvéért megemlítjük, hogy mennyiben csak pozitív pénzmozgásokat feltételezünk, a projekt NPV maximalizálása reguláris mérték. (Herroelen és ts. [1999])

Természetesen ez nyilvánvalóan a valóságtól igen távol álló modell, pusztán elvi jelentősége van.

1.6.2. Irreguláris célfüggvények

Ebben az esetben nem teljesül az a feltétel, hogy a célfüggvény értéke a tevékenységek befejezési időpontjaiban monoton növekvő. Az alábbiakban néhány, a témánkhoz legszorosabban kötődő irreguláris célfüggvényt soroljunk fel, melyek mellett természetesen számtalan további irreguláris célfüggvény is létezik.

1.6.2.1. A projekt nettó jelenértékének maximalizálása

(*net present value maximization, Project Scheduling Problem with Discounted Cash Flows, PSPDC, payment scheduling*)

A projekt nettó jelenértéke megfelelő mérték a teljes projekt teljesítményének leírására, amennyiben ismertek a tevékenység indítások költségei (negatív pénzmozgások), illetve a projekt bizonyos részeinek teljesítésekor bekövetkező részteljesítési kifizetések (progress payments, pozitív pénzmozgások). Egyes esetekben késési büntetést (penalty cost) is figyelembe vehet a modell, mely akkor kerül kifizetésre, ha a projekt időtartama meghaladja a betartandó teljesítési időt (due date), illetve figyelembe vehet bónuszt (bonus), ami akkor kerül kifizetésre, ha a projekt a tervezettnél korábban befejeződik. Az NPV használatával a feladat ütemezési probléma pénzügyileg motiválttá válik, igen nagy gyakorlati jelentőségű cél. Habár a probléma már 1970-ben megjelent Russelnél, de bonyolultsága miatt csak a 90-es évek elejétől kapott nagyobb figyelmet a kutatóktól. A cél minden tevékenység számára olyan elsőbbségi feltételeket kielégítő ütemezési időket találni, melyek mellett a projekt NPV maximális. A projekt ütemezés során az NPV maximalizálásának mint célnak figyelembe vétele kifejezett jelentőséget kap, ha a projekt tervezési horizontja hosszú (több hónap, illetve év), és nagy pénzmozgások lépnek fel, mert ilyenkor a pénz időértékét alaposan fontolóra kell vennünk. Ilyen

projekteket találunk például az építőiparban a legtöbb esetben, gyógyszerfejlesztési projektekben.

1.6.2.2. Erőforráskorlátos projekt ütemezési probléma diszkontált pénzmozgásokkal

(*Resource Constrained Project Scheduling Problem with Discounted Cash Flows, RCPSPDC*)

Ez a probléma a fenti probléma erőforráskorlátos változata. A legtöbb NPV projekt ütemezési probléma kutatás súlypontját a pénzmozgásos RCPSP képezi.. Ebben az esetben azonban a megújuló erőforráskorlátok kielégítését is megköveteljük. Az NPV-t tekintve optimális kezdési idők halmaza adja a megoldást.

Kiemelt jelentőségű, az irodalomban igen sokat említett az alábbi, erőforrás felhasználással összefüggő irreguláris célfüggvény, melyet algoritmusunk továbbfejlesztése során felhasználunk:

1.6.2.3. Az erőforrás kiegyenlítési probléma

(resource levelling)

A probléma megoldása során rögzítve a projekt befejezési időpontját, olyan ütemezést keresünk a nem-kritikus tevékenységek ütemezésének késleltetésével, amelyhez tartozó erőforrás felhasználási hisztogram alakja a lehető legjobban megközelít egy „kívánatos alakot”. A „kívánatos alak” többféle módon definiálható, de a modellek jelentős részében, az erőforrás felhasználásban mutatkozó ingadozások csökkentése, vagyis az alak "kisimítása" az elsődleges cél. A modellek másik, kisebb része a maximális erőforrás felhasználást kísérlik meg minimalizálni.

Az alapmodellek több csoportba sorolhatóak aszerint is hogy globálisan, vagy lokálisan tekintenek-e a problémára. Számtalan különböző célfüggvényt használtak a kutatók modelljeik megfogalmazása esetén. Bizonyos modellek például a maximális erőforrás felhasználást minimalizálják, egyéb modellek az átlagos erőforrás felhasználástól való négyzetes eltéréseket minimalizálják, vagy az egymás után következő időintervallumok erőforrás felhasználásnak négyzetes eltérését minimalizálják. Globálisan tekintenek az erőforrás kiegyenlítési problémára az első, és a második modell megoldása során, lokálisan a harmadik modell esetében.

A fenti modelleknek több esetben ismert a több megvalósítási módú változata is. A legkutatottabb a több módú RCPSP. Az alábbiakban néhány többmódú problémát sorolunk fel a témához kötődő jelentősebbek közül:

1.6.2.4. Time/cost tradeoff projekt ütemezési probléma

(Time-Cost Tradeoff Project Scheduling Problem)

Többlet erőforrás felhasználást megengedve a tevékenységek időtartama rövidülhet. A probléma több megvalósítási módú, létezik folytonos és diszkrét változata is, erőforráskorlátos és anélküli változata is. Icmeli és Erenguc [1996/a] részletes leírást ad a problémáról.

1.6.2.5. Time/cost tradeoff projekt ütemezési probléma pénzmozgásokkal

(Time-Cost Tradeoff Project Scheduling Problem with Discounted Cash Flow)

Az előző probléma pénzmozgásos változata, az erőforrásokat korlátlannak tekintjük. Icmeli és Erenguc [1996/a] részletes leírást ad a problémáról.

1.6.2.6. Time/cost tradeoff erőforráskorlátos projekt ütemezési probléma pénzmozgásokkal

(Resource Constrained Time-Cost Tradeoff Project Scheduling Problem with Discounted Cash Flow)

Az előző két probléma ennek a problémának a relaxációja, itt erőforráskorlátokat és pénzmozgásokat is vizsgálunk. Erről a problémáról is Icmeli és Erenguc [1996/a] ad részletes leírást.

1.6.3. Több célfüggvényes (multi-objective) problémák

A fenti projekt ütemezési problémák többsége egy célfüggvényes, a kutatók kezdetben főként ilyen típusú problémákkal foglalkoztak. Az életben azonban a tevékenységek ütemezése során legtöbbször több, párhuzamos cél szerint ütemezünk. Ezeket több célfüggvényes (multi-objective, multi-criteria) problémának nevezzük. Az irodalomban a leggyakrabban a két célfüggvényes (bi-criteria) eseteket vizsgálják, az 1.6.2.4, 1.6.2.5, illetve az 1.6.2.6 pontban ismertetett problémák is ilyenek.

Egyszerű szemléltető példaként tekintsük két párhuzamos célnak a befejezési idők betartását és a készletek alacsony szinten tartását. Amennyiben kizárólag az utóbbira koncentrálunk, előfordulhat, hogy életszerűtlenül nagy befejezési időhöz jutunk, és fordítva, elképzelhető, hogy szűk időkorlátok mellett a készletek elfogadhatatlan mértékben nagyok lennének.

A több célfüggvényes esetekben elfogadható kompromisszumos megoldást szeretnénk kapni. A késő nyolcvanas évek óta jelent meg az erőforráskorlátos projekt ütemezés irodalmában az egyszerre több teljesítmény szempont figyelembe vétele, de bonyolultsága miatt máig igen szűkös az irodalma. Ilyen többcélú ütemezés disszertációban ismertetett megoldás is.

A több célfüggvényes esetek különbözhetnek aszerint, hogy hierarchikusan tekintünk-e a célfüggvényekre, tehát van elsődleges és másodlagos (esetleg harmadlagos, negyedleges) szempont, vagy egyenrangúként kezeljük őket.

Az erőforráskorlátos nettó jelenérték maximalizálása során kiemelt gyakorlati jelentőségű az a két célfüggvényes modell, melyben az elsődleges szempont a projekt időtartamának minimalizálása, másodlagos szempont a nettó jelenérték maximalizálása.

A több célfüggvényes eseteket két csoportba sorolhatjuk aszerint hogy hierarchikusan kezelik-e a célfüggvényeket:

1. Amennyiben két olyan célt tekintünk, melyek közül az egyik fontosabb, mint a másik, akkor nyilvánvalóan a következő eljárást fogjuk követni: A két szempont közül az egyiket elsődleges (primary) szempontnak, a másikat másodlagos (secondary) szempontnak jelöljük ki. Megkeressük ez elsődleges cél szerinti optimális ütemezéseket, és a kapott megoldások halmazán megkeressük a másodlagos szempont szerinti optimumot. Az ilyen megközelítést hierarchikus optimalizálásnak (hierarchical optimization), vagy lexicographical optimalizálásnak nevezzük.

   Ide sorolható a disszertáció-beli ütemezés. Olyan típusú hierarchikus projekt ütemezés, ahol elsődleges cél a projekt időtartamának minimalizálása, másodlagos cél a nettó jelenérték maximalizálása, legjobb tudomásunk szerint eddig ezenkívül még nem készült.

   Megjegyzendő, hogy az NPV maximalizálás természetéből fakadóan másodlagos szempont. Amennyiben megpróbálnánk elsődlegessé tenni, a projekt idejét

valószerűtlenül távolra, a megengedett maximumig tolná ki az ütemezés bizonyos esetekben, így életidegen ütemezéseket kapnánk.

2. Előfordulhat olyan eset, amikor nincs egyértelmű domináns szempont. Ilyen esetben generálhatjuk az összes célfüggvény preferencia sorrendhez tartozó Pareto-megoldást, vagy a célfüggvényeket súlyozzuk az általunk meghatározott fontosságuknak megfelelően, és így kapunk egy közös célfüggvényt. A második típusú megoldás esetén a módszer lehet lineáris kombináció, lehet akár négyzetes kombináció, de bármilyen más függvény kombinációja is előfordulhat.

Súlyozott megoldásra az egyik legjelentősebb munka Al-Fawzan és ts. [2005] tanulmánya, ahol két célfüggvényt – makespan minimalizálás, robosztusság (robustness) maximalizálás- súlyozva kapjuk a közös célfüggvényt. A robosztusság fogalmát a szerzők ebben a tanulmányukban vezették be, a tevékenységek tartalék idejének (floating time) egy függvényeként. Tabu kereső megoldó módszerrel kapták eredményeiket..

Abbasi és ts. [2006] tanulmányában, ugyanezen célfüggvényeket súlyozza. A szimulált hűtő módszert alkalmazták a probléma megoldásához.

Viana és ts. [2000] minimalizálják a projekt összidőtartamát, és a tevékenységek adott tevékenység ütemezési időkhöz viszonyított késési időit (lateness) szintén súlyozva a célfüggvényeket. Szimulált hűtő és tabu kereső eljárást alkalmaztak tanulmányukban.

2. Szakirodalmi előzmények – reguláris modellek

A projektütemezés során reguláris célfüggvényről beszélünk, ha a célfüggvény értéke az ütemezendő tevékenységek befejezési időpontjaiban monoton növekvő. A reguláris modelleket sokkal könnyebb kezelni az irreguláris célfüggvénnyel rendelkezőknél.

Ebben a fejezetben a reguláris modelleken belül csak a legkutatottabbal, és egyben a disszertáció-beli problémához legközelebb állóval, az RCPSP-vel foglalkozunk.

A továbbiakban szükségünk lesz néhány fogalomra:

Lineáris programozási feladat (Linear programming problem): Adott A, egy $n*m$-es (valós) mátrix, egy (valós) m dimenziós b oszlopvektor, és egy (valós) n dimenziós c sorvektor. Keressük $c*x$ optimális értékét - minimumát vagy maximumát, a feladat természetétől függően - , amikor $Ax <= b$, és $x \geq= 0$.

Egészértékű lineáris programozási feladat (Integer linear programming problem): Olyan lineáris programozási feladat, melyben a változók egész értékűek. A projektütemezési feladatok jellemzően ilyenek, sőt a változók általában binárisak.

Mixed integer linear problem: Olyan lineáris programozási feladat, melyben némely változó egész értékű, némelyik nem.

2.1. Az RCPSP

Disszertáció-beli algoritmusunk első körben az erőforráskorlátos projekt időtartamát minimalizálja, ezért tekintjük át ezt a területet.

A nemkorlátos projekt időtartam minimalizálási ütemezési probléma könnyedén megoldható polinomiálisan egy egyszerű előre haladó (forward) rekurziós eljárással, ahol minden tevékenységet a legkorábbi elsőbbségi feltételt kielégítő kezdőidején

hajtunk végre. Az erőforráskorlátos eset azonban NP-nehéz, izgalmas, kalandos természete miatt nagyon népszerű és gyakran tanulmányozott optimalizálási probléma.

A projekt ütemezés korai irodalma elsősorban az RCPSP-re koncentrált. Ebben az esetben mint mondtuk, a probléma NP-nehéz, de mivel a célfüggvény reguláris, így az irreguláris modellek célfüggvényeinél még mindig könnyebben kezelhető. Az utóbbi 20-25 év alatt az RCPSP megoldására irányuló heurisztikák és egzakt megoldási eljárások teljesítményének javulásában jelentős előrelépés történt. Az irodalma igen szerteágazó, ezért csak a lényegesebb tanulmányokat emeljük ki.

Az áttekintő művek közül a következőket emelnénk ki: Herroelen és ts. [1999], Brucker és ts. [1999], Kolisch és ts. [2001], és a könyvek közül a jelentősebbek: Weglarz J. (editor) [1999]: Project scheduling — recent models, algorithms and applications, Demeluemeester és Herroelen [2002], Neumann és ts. [2002].

2.1.1. Az RCPSP heurisztikái

Egzakt megoldások relatív kis méretű problémák esetén használhatóak eredményesen, nagyobb projektek esetén heurisztikákat érdemes használni. Nagy méretű problémák esetében – és a projekt ütemezési problémák az életben főként ilyenek – az egzakt megoldások gyakran nem is alkalmazhatóak, mivel elfogadható számítási idő alatt nem adnak megoldást. Annak ellenére, hogy a heurisztikák általában nem optimális, csak optimális közeli megoldást adnak, a fenti ok miatt a heurisztikákat kiemelten tárgyaljuk a disszertációban.

Kolisch és ts. [2006] osztályozása szerint az RCPSP területén többek között a következő heurisztikákat használták:

- X-pass heurisztikák,
- klasszikus és a szokásos sémákba nem sorolható metaheurisztikák,

- előre-hátra haladó javító heurisztikák,
- egyéb heurisztikák.

A metaheurisztikák esetében a szerzők kiemelik annak jelentőségét, hogy a többi heurisztikával szemben működésük során képesek tanulni, így számos esetben hatékonyabbak.

Tormos és Lova más szempontok alapján osztályozza az RCPSP probléma megoldására született heurisztikákat:

- elsőbbségi szabályt alkalmazó heurisztikák

  Ennél a csoportnál kiemelik annak alacsony számítási és memória igényét, melyek következtében nagyméretű projektek megoldására is jól alkalmazhatóak.

  - single-pass elsőbbségi szabályt alkalmazó heurisztikák

    Az ebbe a csoportba tartozó heurisztikák ütemezés generációs sémákat (schedule generation scheme, SGS), és elsőbbségi szabályokat alkalmaznak együtt. Két ütemezés generációs sémát ismerünk: soros (serial schedule generation scheme, S-SGS) vagy párhuzamos (paralell schedule generation scheme, P-SGS) ütemezési séma. Az S-SGS tevékenység orientált séma és minden ütemezési lépésben egy tevékenységet választ az ütemezendő tevékenységek halmazából, és ütemez. A P-SGS idő orientált séma, és minden ütemezési lépésben egy tevékenység halmazt választ az ütemezendő tevékenységek halmazából, és ütemez. A single-pass elsőbbségi szabályt alkalmazó heurisztikák általában egy SGS-t és egy elsőbbségi szabályt alkalmaznak együtt.

A leggyakrabban alkalmazott elsőbbségi szabályok a területen: minimális legkésőbbi befejezési idő (minimum latest finish time, MLFT), és a minimális teljes késleltetés (minimum total slack, MSLK).

- multi-pass elsőbbségi szabályt alkalmazó heurisztikák

A gyors kiszámíthatóság miatt hozták létre a single pass elsőbbségi szabályt alkalmazó heurisztikák mintájára a multi-pass elsőbbségi szabályt alkalmazó heurisztikákat. Ide sorolják a előre-hátra haladó ütemező eljárást, és a véletlen mintavételezési eljárást.

- truncated branch and bound eljárások,
- diszjunktív arcs eljárások (Alvarez-Valdes és ts. [1994]),
- metaheurisztikák.

Az RCPSP heurisztikáiról áttekintést nyújtanak még Kolisch és Hartmann [1999], Hartmann és Kolisch [2000], Valls és ts. [2005].

Kolisch és Hartmann összehasonlítja a fenti (Kolisch és ts. [2006]) RCPSP heurisztikák teljesítményét, és megállapítja, hogy a legjobban teljesítő eljárások Alcaraz és ts., Debels és ts., Hartmann, Kochetov és Stolyar, és Valls és ts. heurisztikái. Ezeknek a heurisztikáknak a nagy része genetikus algoritmus (lásd később), és legtöbbjük alkalmazza az előre-hátra haladó javító eljárást (forward-backward improvement), melyet Tormos és Lova [2001] fejlesztett ki.

Az előre-hátra haladó javító eljárás több előre és hátra haladó ütemezés (forward és backward schedule) egymás utáni ismétlése. Egy erőforrás korlátos ütemezésből indulunk ki, és célunk a projekt hosszát csökkenteni.

Az eljárás kezdetekor előre ütemezünk minden tevékenységet. Azaz tekintjük sorban a tevékenységeket a befejezési idejük szerinti csökkenő sorrendben. Először a sorrend

szerinti első (azaz a legnagyobb befejezési idejű) tevékenységet toljuk el „jobbra”, azaz úgy, hogy befejezési ideje egybeessen a projekt idő felső korlátával. Ezután vesszük a sorrend szerinti második tevékenységet, és azt is eltoljuk „jobbra” addig, amíg csak el tudjuk tolni az elsőbbségi és erőforrás korlátok megsértése nélkül. Ezt az eljárást megtesszük sorban az összes tevékenységgel.

Az eljárás második felében hasonlóan cselekszünk, csak a tevékenységek előre ütemezéssel kapott, új kezdési idői szerinti sorrend alapján vesszük sorra a tevékenységeket, és sorban „balra” toljuk el őket, kezdve a legkisebb kezdési idővel rendelkező tevékenységgel. Ebben a részben hátra ütemezünk minden tevékenységet.

A fenti előre illetve hátra ütemezést felváltva alkalmazzuk egymás után, általában addig, amíg a tevékenységek „beállnak”, tehát amikor olyan ütemezést kapunk, mely tevékenységei kezdési időin már nem javít egy újabb előre-hátra ütemezés.

Annak ellenére, hogy ez az eljárás maga végtelenül egyszerű, igen eredményes. Tormos és Lova számítási eredményekkel igazolja, hogy ez a technika jóval hatékonyabban oldja meg az RCPSP problémát, mint a többi heurisztika, beleértve a metaheurisztikákat is. Megmutatják, hogy minél több tevékenységből áll a projekt, annál inkább javítja az eljárás hatékonyságát a technika.

Disszertáció-beli algoritmusunk is alkalmazza az előre-hátra haladó javító eljárást. (például Láng [2009/a], Láng [2009/c]), mellyel jelentős teljesítmény javulást értünk el.

Valls és ts. [2005]-ben is leírja ezt az egyszerű új technikát, „double justification” (DJ) néven említve, más kutatók a fenti „előre-hátra haladó javító eljárás” (forward-backward improvement) kifejezést használják (Kolisch és Hartmann [2006]). A „double justification”, vagy „forward-backward improvement” sikerének titka, hogy igen könnyen egyesíthető számos más RCPSP algoritmussal, és alkalmazásával szignifikáns

javulást érhetünk el az ütemezés hatékonyságában. A számítási időt is csak kis mértékben növeli meg. A fentieken kívül is számos kutató számol be az eljárás alkalmazása során tapasztaltakról:

Az ígéretes eredmények késztették például Valls és ts.-t, hogy kifejlesszék hibrid genetikus algoritmusukat (HGA), melyben a fenti double justification eljárást kombinálják az igen hatékony, és így népszerű genetikus algoritmussal (a genetikus algoritmus ismertetését lásd később). Már tanulmányuk címválasztása is igen beszédes: „A technique that pays".

Palpant és ts. [2004] szintén alkalmazzák a forward-backward improvement-t. HGA néven említett algoritmusukat úgy alkották meg, hogy az aknázza ki a problémáról való tudást, azonosítsa és kombinálja a megoldásnak azokat a részeit, amelyek tényleg hozzájárulnak annak hatékonyságához. Hatékony ütemezési tapasztalatokról számolnak be tanulmányukban.

Kolisch és Hartmann [2000] áttekintő jellegű tanulmányukban úgy vélik, az előre-hátra haladó javító eljárás a jövő heurisztikáinak jelentős komponense lesz. Ezt a vélekedésüket az idő azóta igazolta. A szerzők másik lényeges megállapítása, hogy az utóbbi évek során egyre több komponensből állnak a heurisztikák a korábbiakkal ellentétben, és a kutatók általában új technikákat kombinálnak a már ismertekkel, és így jutnak ígéretes fejlesztésekhez.

A heurisztikák más szempontból osztályozhatóak aszerint, hogy konstruktív (constructive), vagy javító (improvement) heurisztikáról van-e szó. Az első típusú heurisztika nem igényel indulásként megvalósítható megoldást, és egy megvalósítható ütemezést ad eredményül, míg az utóbbi egy megvalósítható ütemezésből indul ki, és egy jobb teljesítményű heurisztikát ad.

Az RCPSP-re számtalan heurisztikus eljárást fejlesztettek ki, melyről a szerzők részletes, áttekintést adnak a fenti tanulmányokban. A tanulmányok nagy száma jelzi az NP-nehéz jellegből adódó nehézségeket.

2.1.2. Az RCPSP egzakt modelljei

Annak ellenére, hogy mint korábban leírtuk, a nagyméretű feladatok esetén a heurisztikáknak van létjogosultságuk, mégis említenünk kell az egzakt eljárásokat is. Ennek oka, hogy a heurisztikus megoldások sokszor vesznek át módszertani elemeket a kizárólag kis problémák megoldására alkalmas egzakt megoldásoktól, természetesen módosítva azokat a feladatméretnek megfelelően.

Az RCPSP problémára kidolgozott egzakt megoldási algoritmusok dinamikus programozáson, bináris optimalizálási lineáris programozási feladaton, és implicit leszámláláson alapuló (enumeration) branch and bound eljárás, ez utóbbit alkalmazza a legtöbb egzakt megközelítés. A legtöbb branch and bound eljárás kezdőpontként részleges megvalósítható (partial feasible) ütemezést használ. A branching eljárás a részleges ütemezést kiteljesíti addig, amíg egy teljes ütemezést talál. Különböző branching és metszési (pruning) szabályokat alkalmaznak. Mélységi branching stratégiával oldják meg az erőforrás kapacitási problémákat, tevékenységek halasztásával.

Jelentősebb egzakt megoldások a fenti problémára: egész programozási formulát adott Pritsker és ts. [1969], Patterson és Roth [1976], Cristofides és ts. [1987].

Bell és Park [1990] egy projekt időszükséglet minimalizáló megoldást ad erőforráskorlátok jelenlétében. *A** elnevezésű keresésükben pótlólagos elsőbbségi relációk bevezetésével oldják fel az erőforrás konfliktusokat.

Talbot és Patterson [1978] a tevékenységeket tevékenység listába rendezték, melyek az elsőbbségi kapcsolatokat veszik figyelembe. Ezután előre haladó (forward) rekurzióval $t = 0$-tól kezdve, és hátra haladó (backward) rekurzióval egy felső időtartam korláttól kezdve időablakot származtatnak minden tevékenység számára. A lista első tevékenységével indítanak, a leszámláló eljárás a lista következő tevékenységét kísérli meg ütemezni a legkorábbi intervallumra, amit az elsőbbségi feltételek és az erőforráskorlátok lehetővé tesznek, a tevékenységhez tartozó időablakon belül. Ha ez nem lehetséges, backtrackkelnek, az utolsó tevékenységet egy periódussal későbbre ütemezik. A leszámlálást továbbfejlesztették a keresési fa egyes részeinek lemetszésével.

Stinson és ts. [1978] algoritmusa egy szélességi kereséses branch and bound eljárás, ami Johnson keresési fáján alapul. A fa minden egyes pontja egy megvalósítható részleges ütemezés. A pontokhoz társított ütemezési idő azt az időt méri, ami a szülő pont részleges ütemezéséhez szükséges. A pontok leszármazottai a még nem ütemezett, elsőbbségi feltételeket kielégítő tevékenységek megvalósítható kombinációinak leszámlálásával keletkeznek.

Demeulemeester és Herroelen [1992] algoritmusa Cristofides mélységi kereséses branch and bound eljárásának kiterjesztése. Stinson eljárásától főleg a keresési fában tér el, ahol az új pontok nem a már ütemezett tevékenységek halmazának figyelembe vételével származtatódnak, hanem a késleltetett tevékenységek halmazának figyelembevételével. Két elsőbbségi szabályt használnak a keresési fa metszéséhez. Az első a bal-jobb elsőbbségi szabály egy változata, a másik cutsetet használ fel. A bounding a Stinson által használt elsőbbségi szabály alapú és kritikus sorozat alapú alsó korláttal és a Mingozzi és ts. által használt súlyozott pont pakoló (weighted point packing) korláttal

van végrehajtva. A fenti tanulmány megoldását használja Kolisch és ts. teszthalmazuk új probléma karakterisztikáinak kidolgozásához.

3. Szakirodalmi előzmények – irreguláris modellek

A korábban kutatott irreguláris projekt ütemezési modellek közül a disszertáció témájához szorosabban kapcsolódókat az alábbiak szerint rendszerezve tekintjük át:

Először az erőforráskorlát nélküli nettó jelenérték maximalizáló megoldásokat tárgyaljuk, melyek elsősorban egzakt megoldások. Bár a disszertáció-beli algoritmusunk erőforráskorlátos problémát old meg, az erőforráskorlát nélküli megoldások fontos előzményként kezelendők, mivel sok esetben alkalmazhatóak megfelelő módosításokkal erőforráskorlátos esetekre.

A következő részben az erőforráskorlátos nettó jelenérték maximalizáló megoldásokat vesszük sorra, külön tárgyalva az egzakt, és külön a heurisztikus megoldásokat. Az egzakt modellek kizárólag kis- és közepes méretű projektek esetén alkalmazhatóak, mivel akkor optimális megoldást adnak, azonban problémánk NP-nehéz volta miatt csak korlátozottan alkalmazhatjuk őket. Ezzel szemben a heurisztikus módszerek nagyméretű feladatok esetén is alkalmazhatóak, mivel elfogadható időn belül is képesek jó minőségű megoldást adni.

Annak indoklása, hogy nagyméretű problémák esetén miért említjük mégis az egzakt megoldásokat ugyanaz, mint reguláris modellek esetén: a heurisztikus megoldások sokszor vesznek át módszertani elemeket a kizárólag kis problémák megoldására alkalmas egzakt megoldásoktól, természetesen módosítva azokat a feladatméretnek megfelelően.

A továbbiakban áttekintjük a heurisztikák fejlődését, mivel a fentiek miatt a jövő sikeres megoldásait nagy méretű projektek esetén a heurisztikák között fogjuk találni.

Külön foglalkozunk a harmónia kereső metaheurisztikával, mivel a disszertációban kifejlesztett megoldás is ebbe a heurisztika típusba sorolható.

Az erőforrás kiegyenlítési problémát a továbbfejlesztési irányok szempontjai között fogjuk említeni, így röviden áttekintjük a probléma leglényegesebb korábbi megoldásait.

3.1. Nettó jelenérték maximalizálás erőforráskorlátok nélkül

A projekt ütemezés korai irodalma elsősorban az RCPSP-re koncentrált. Később jelent meg az igény a projekt időtartamának minimalizálása és az erőforráskorlátosság kezelése mellett vagy anélkül a projekt NPV-jének maximalizálására, amely „életközelibbé” tette a modelleket. Az NPV maximalizálásra koncentráló technikák fejlődése a probléma irregularitása, nehezebben kezelhetősége miatt kezdetben jelentősen elmaradt az előbbi problémát megoldó technikák fejlődésétől. Hosszú időn keresztül a kutatások homlokterében az erőforráskorlát nélküli esetek voltak. Népszerűségük abból eredt, hogy lényegesen egyszerűbben megoldhatóak, mint az ebben a fejezetben ismertetett erőforráskorlátos probléma.

Nemkorlátos esetben a projekt ütemezés célja a projekt nettó jelenértékének maximalizálása, amellett hogy az ütemezendő tevékenységeket elsőbbségi feltételek korlátozzák.

Kezdetben az időtartam minimalizálásra koncentráló megközelítések indirekt módon foglalkoztak az ütemezés pénzügyi vonzataival, elsősorban úgy, hogy a modellbe belefoglalták a késlekedési büntetést, és törekedtek annak elkerülésére, vagy törekedtek a hatékony erőforrás felhasználásra. (Davis [1973], Davis és Patterson [1975], Patterson és ts.[1976]). A projekt menedzselés során egyre inkább előtérbe került az időbeli

ütemezésen túlmenően a projekthez kapcsolódó pénzügyi folyamatok kezelésének igénye.

Mivel az egzakt megoldások ebben az esetben jól alkalmazhatóak, csak azokat tekintjük át, mivel elsősorban azokat fejlesztették ki. Az adott területen igen kevés heurisztikus megoldást találunk.

A. H. Russel [1970] vezette be a projekt NPV maximalizálásnak problémáját úttörő jelentőségű, igen sokat idézett tanulmányában. A tevékenységek kezdéséhez negatív pénzmozgásokat társított, és a projekt időtartama alatt bizonyos tevékenység halmazok teljesítéséhez pozitív pénzmozgást társított. A problémát „kifizetés ütemezési problémaként" (Payment Scheduling Problem, PSP) említette. A problémát nemlineáris programként fogalmazta meg. AOA ábrázolási módszert, tehát eseményorientált 0megközelítést használt. Ismertnek feltételezte a tevékenységek időtartamait, az elsőbbségi feltételeket, és nettó pénzmozgásokat társított az esemény-pontokhoz.

Maximalizálandó célfüggvénye a következő:

$$\sum_{i=1}^{n} F_i \exp(-\alpha T_i)$$

ahol

- n az események száma
- az i esemény T_i időpontban következik be
- α a diszkont ráta
- F_i az i tevékenységhez társított pénzmozgás

A nemlineáris célfüggvény elsőrendű Taylor soros megközelítésével lineáris programozási feladathoz jutott, melynek duálisa egy átrakodási háló folyam (transshipment network flow) probléma. A duális feladat megoldása során folyamatosan

felülíródik az események bekövetkezési ideje. Addig ismételte ezt az eljárást, amíg a bekövetkezési idők konvergáltak. Megmutatta, hogy a költség-kritikus út eltér az idő-kritikus úttól.

Grinold [1972] projekt határidő hozzáadásával egészítette ki és tette valóságközelibbé Russel modelljét.

A nemlineáris problémát lineáris elsőbbségi feltételekkel és az exponenciális célfüggvénnyel egy ekvivalens lineáris programmá fogalmazta át, mely így egy súlyozott elosztási probléma (weighted distribution problem) lett. A megoldási eljárás során kihasználta ezt a speciális struktúrát, felkutatva a projekt hálón azokat a lehetséges fákat, melyben minden tevékenységnek nulla tartalékideje (slack time) van. Egy példával illusztrálja az ellentmondást az NPV maximalizálás és a projekt időtartam minimalizálás között. Két egzakt megoldó eljárást javasol.

Sajnálatos módon sem Russel, sem Grinold nem közöltek eljárásukkal kapcsolatban hatékony számítási eredményeket.

Elmeraghby és Herroelen [1990] Russel és Grinold modelljének gyakorlati használhatóságát kérdőjelezte meg, megmutatva, hogy a nem erőforráskorlátos probléma egyszerűségét miként homályosítják el ezek a megoldások, és feleslegesen hosszadalmas megoldási eljárást adnak. Ezenkívül megmutatták, hogy bizonyos esetekben ezek a megoldások értelmetlen eredményt adnak. Russel megoldását tanulmányozva megállapították, hogy projekt határidő hiányában, ha az NPV negatív, akkor a projekt teljesítése határozatlan időre kitolódhat.

A Russel-modellt használva kifejlesztettek ki egy egyszerűsített algoritmust, ami optimális megoldást ad a problémára. Megmutatták, hogy általában azokat az eseményeket, melyekhez pozitív pénzmozgás társul, olyan koránra érdemes ütemezni,

amennyire csak lehet, és a negatív pénzmozgáshoz kapcsolódó eseményeket olyan későre érdemes ütemezni, amennyire csak lehet. Későbbi tanulmányukban Herroelen és Gallens [1993] részletes számítási eredményeket adott a fenti megoldásra.

Demeulemeester és ts. [1996] egy új algoritmust ajánlottak, amely egy rekurzív keresést hajt végre egy részleges fa struktúrán, és használja azt az ütemezési elvet, hogy azokat a tevékenységeket, melyek pozitív pénzmozgással rendelkeznek, a lehető legkorábbra ütemezzük, és azokat, melyek költségekkel járnak, minél későbbre. Számítási eredményekkel demonstrálják Grinold eljárásnál biztatóbb eredményeiket.

Smith-Daniels [1986] kizárólag negatív pénzmozgásokat feltételez, modelljében pozitív pénzmozgás csak a projekt teljesítésekor következik be. A modellben legkésőbbi kezdésű (late start) CPM ütemezést alkalmaz.

Demeulemeester és ts. [1996] eljárása rekurzív keresés a részleges fa struktúrán. Azokat a tevékenységeket, melyek nagy pénzkiadással járnak, lehetőség szerint halasztják, míg a többit lehetőség szerint előre hozzák. Számítási eredményeikkel bizonyítják, hogy az algoritmus hatékonyabb, mint a Grinold eljárás.

Amennyiben megengedjük többlet erőforrás felhasználását, a tevékenységek időtartama rövidülhet. Ekkor az un. pénzmozgásokkal kiegészített idő/költség „tradeoff" projekt ütemezési problémához jutunk. Icmeli és Erenguc [1996/a] részletes leírást ad a problémáról. Modelljükben erőforrás és elsőbbségi korlátokat is feltételeznek.

Igen összetett célfüggvényük, melyben a fenti tradeoffot megfogalmazzák:

$$MAX \sum_{i=1}^{N} \sum_{t=e_i}^{L_i} \left(\sum_{k=1}^{K_i} NF_i^k Y_i^k X_{it} e^{-\alpha t} - \sum_{t=due}^{T} (t - due) X_{Nt} P e^{-\alpha t} \right)$$

ahol

$$NF_i^k = q_i + \sum_{j=1}^{d_i^k} f_{ij}^k e^{\alpha(d_i^k - j)} + c_i^k$$

A képletben szereplő fontosabb jelölések:

- *due* a projekt előre meghatározott teljesítési ideje
- *P* a teljesítési időn felüli periódusonkénti büntetés
- K_i az *i* tevékenység lehetséges hosszainak száma
- d_i^k az *i* tevékenység *k*-adik lehetséges hossza
- q_i az *i* tevékenység teljesítésekor bekövetkező pénzáram
- f_{ij}^k az *i*-edik tevékenység *j* periódusa alatt fellépő pénzáram, ahol, j=1,… d_i^k.
- NF_i^k az *i* tevékenység teljesítésekor bekövetkező nettó pénzáram d_i^k hossz esetén
- e_i az *i* tevékenység legkorábbi teljesítési ideje,
- L_i az *i* tevékenység legkésőbbi teljesítési ideje
- c_i^k a költségcsökkentés, d_i^k hossz esetén

A fenti problémát „P1” néven nevezik. „P2” elnevezéssel konstans hossz esetén vizsgálják a probléma variánsát, míg „P3” néven az erőforrás korlát nélküli variáns esetet vizsgálják.

3.2. Nettó jelenérték maximalizálás erőforráskorláttal

Ebben az esetben a projekt ütemezés célja a projekt nettó jelenértékének maximalizálása erőforráskorlátok és elsőbbségi feltételek megléte mellett. Az alábbiakban áttekintjük ennek a problémának megoldására született egzakt, illetve heurisztikus megoldásokat.

A probléma az erőforráskorlátok figyelembe vételével realisztikusabbá válik. A problémát gyakran RCPSPDC-ként (Resource Constrained Project Scheduling Problem with Discounted Cash Flow) is említik.

Az erőforráskorlátos változat lényegesen bonyolultabb megoldásokkal jár, mint a nemkorlátos eset. A bonyolultság a szűk erőforrások, és a NPV szempont közötti kapcsolatból adódik. Tételezzük fel például, hogy egy tevékenységhez negatív pénzmozgás társul, így Elmeraghby és Herroelen [1990] alapján olyan későre szeretnénk ütemezni azt, amennyire csak lehet, azaz a CPM-ből adódó legkésőbbi kezdési idejére. Azonban elképzelhető, hogy az erőforráskorlátok fennállta miatt egy másik tevékenységet nem tudunk akkorra ütemezni, mint szeretnénk, így annak késleltetése máris csökkenti a projekt NPV-t.

A diszkontált pénzmozgásokhoz kapcsolódó modellekről Kolisch és Padman [2001] ad átfogó leírást.

3.2.1. Egzakt megoldások

A modell megoldása MILP programozási feladat, azaz olyan lineáris programozási feladat, amelyben némely változó egész értékű, némelyik nem. Míg az LP problémák polinomiális idő alatt megoldhatóak, a MILP problémák sajnos nem, azok kivétel nélkül NP-nehéz problémák.

Doersch és Patterson [1977] úttörő tanulmányukban az irodalomban először adott megoldást az erőforráskorlátos NPV maximalizálás problémájára. Tanulmányuk jelentőségére tekintettel az alábbiakban ismertetjük modelljük lényegesebb pontjait.

A célfüggvénybe belefoglalták a tevékenységek teljesítéséhez társított pénzmozgásokat (azaz a részteljesítési kifizetéseket), és a késő teljesítésekből következő büntetéseket is. Modellükbe beleértették a tevékenységek költségeinek tőke korlátait.

A fentiek alapján maximalizálandó célfüggvényük a következő:

$$\sum_{i=1}^{N}\sum_{t=e_i}^{l_i} f_t D_i - \sum_{t=e_{N+1}}^{l_{N+1}} f_t P_t$$

ahol

- P_t a büntetés mértéke t időperiódus alatti késői teljesítés esetén, illetve bónusz korai teljesítés esetén, az előbbi pozitív, az utóbbi negatív szám
- f_t diszkont faktor t időperiódus esetén
- D_i az i tevékenységhez tartozó pénzmozgás, melyet a modellben a tevékenység befejezési idejéhez társítanak

Az i tevékenységhez tartozó pénzmozgás több tagból adódik össze, a tevékenységhez társított pozitív és negatív pénzmozgásoknak megfelelően:

$$D_i = \sum_{j=1}^{d_i} F_{ij} \exp\alpha(d_i - j) - C_i \exp(\alpha d_i) + C_i$$

ahol

- F_{ij} a j időperiódusban az i tevékenységhez társított pénzmozgás
- α a diszkont ráta
- C_i az i tevékenység végrehajtásához szükséges tőkeberuházás
- D_i az i tevékenység hossza

Eredményeik azt jelezték, hogy magas tőkeköltség vagy nagy időtartamú projekt esetén a tevékenység ütemezés során igen fontos megbecsülni a késési büntetéseket és a tőke korlátokat. A probléma megoldására bináris egész programozási modellt fejlesztettek ki. Annak okát, hogy korábban nem foglalkoztak a problémával a szakirodalomban, a kutatók elsősorban abban látják, hogy igen nehéz megtalálni a problémához az alkalmas

célfüggvényt, és nehéz kezelni az így felállított modellt. Kis méretű, 15-20 tevékenységből álló problémákra adnak számítási eredményeket.

Tanulmányukban megmutatják, hogy ha a projekt menedzselés során megfontoljuk a részteljesítési kifizetéseket is, akkor a projekt időtartamának minimalizálására törekvő ütemezések nem a legmagasabb NPV-t adó ütemezések is egyben. Ezt a megállapításukat számtalan esetben idézik a későbbi irodalmakban. Korábban nemkorlátos ütemezés esetén Russel [1970] és Grinold [1972] tett hasonló megállapítást.

Lényeges meglátásuk még a gyakorlati projekt ütemezéssel kapcsolatban a cikkben az, hogy a projekt menedzserek a gyakorlatban mindig is megkísérelték javítani a projekt NPV-jét az előreütemezéses ajánlatokkal (front loading bid. Ez a gyakorlat azt jelenti, hogy a korán teljesített munkákat túlárazzák, a később teljesített munkákat pedig alulárazzák, így kísérelve meg javítani a projekt NPV-t.

Bey, Doersch és Patterson [1981] cikkükben a nettó jelenérték vizsgálatát állandó pénzmozgások esetében ajánlják. Megmutatják, hogy a nettó jelenérték célfüggvény használata kezeli a tradeoffot az (a) nagy tőke igénynek és / vagy pénzkifizetésnek kitett tevékenységek késleltetése (b) a pénzbevételhez vezető tevékenységek korai teljesítése miatti korai pénzbevétel és (c) késedelmi büntetés csökkentése vagy elkerülése vagy a projekt tevékenységek késedelmes teljesítéséhez kapcsolódó bevétel csökkenés között. Tanulmányukban 15-20 tevékenység esetén sikeresen futó megoldásról számolnak be, de módszerükkel a 25-30 tevékenységből álló projektek ütemezése már nem oldható meg elfogadható számítási időn belül.

Tavares [1986] egy új dinamikus programozási formulát és megoldási eljárást javasolt, amelyben a célfüggvénybe belefoglalta a projekt során fellépő hasznokat, és az

időtúllépések esetén bekövetkező büntetéseket. Programját Portugáliában nagy vasúti építkezési projektekben használták. A célfüggvény a program alatt keletkezett hasznok diszkontált (nettó) összegét, a projekt költségek diszkontált összegét és a költségek változásának a büntetését maximalizálja.

Smith-Daniels és Smith-Daniels [1987] Doersch és Patterson bináris formuláját kiterjesztve optimalizációs eljárást adnak. Anyag és tőke korlátokat is feltételeznek. Eljárásukat kis méretű illusztrációs példákkal mutatják be, számítási eredményekről nem számolnak be.

Patterson és ts. [1990] egy 0-1 programozási modellt mutattak be, és egy backtrack algoritmust, mely maximalizálja az erőforráskorlátos projektek NPV-jét. Megoldásaik a projekt időtartamának minimalizálására is alkalmazhatóak. 10 - 500 tevékenységből álló projektekre tesztelték megoldásaikat, de optimális megoldást csak a kisebb problémákra találtak. A megoldásuk hasznosítja azt a tényt, hogy a minimális időtartam problémát könnyebb megoldani, mint a NPV maximalizálási problémát, és használja azt, mint egy heurisztikát, hogy kezdő megoldásokat generáljon, melyen a pénzmozgások jobbra eltolását alkalmazták az NPV javítása érdekében. Tanulmányukban ismertetett új szabály szignifikánsan magasabb NPV-t adott, mint a véletlen szabály.

Yang és ts. [1992] egy egész programozási eljárást fejlesztettek ki a problémára, mélységi branch and bound megoldást használnak. Modelljük és célfüggvényük apró eltérésektől eltekintve megegyezik Doersch és Patterson korábban ismertetett modelljével és célfüggvényével. Eljárásuk Talbot és Patterson [1978] megoldási eljárásán alapult, melyet a projekt időtartam minimalizálási céllal fejlesztettek ki. Yang és ts. számítási eredményei jelzik, hogy algoritmusuk hatékonysága nagy mértékben függ az erőforráskorlátok szorosságától, és a projekt határidejétől.

Icmeli és Erenguc [1996b] tanulmányukban egy mélységi branch and bound algoritmust mutatnak be, mely hasznosítja azt az ismert tényt, hogy az erőforrás konfliktusok kiküszöbölhetőek néhány projekt tevékenység közötti pótlólagos elsőbbségi viszony bevezetésével.

A „minimális késleltetési alternatíva” (minimal delaying alternatives) elvet használták, hogy megoldják az erőforrás konfliktusokat. A minimális késleltetési alternatíva elvet eredetileg Demeulemeester és Herroelen [1992] vezette be a branching probléma megoldására. Ezt az elvet a szabállyal együtt, ami meghatározza a branch csomópontjait, a fa méretének korlátozásához használják. A korlátokat az eljárásban a Payment Scheduling Problems segítségével számítják, így lineáris programozási feladattal dolgoznak, amely „könnyen megoldható”. Az algoritmusuk véges számú lépésben optimális megoldást ad. Az eljárást a Patterson halmazból vett 50 problémán és 40 PROGEN problémán tesztelték, és megmutatták, hogy a tanulmány megjelenésének időpontjában a szakirodalmi eredményekkel összehasonlítva a leghatékonyabb algoritmusnak bizonyul.

Baroum és Patterson [1999] egy branch and bound eljárást tervezett kifejezetten erre a problémára.

Csébfalvi [2002] bemutat egy új egzakt implicit leszámlálási eljárást hatékony metszési szabállyal, amely a diszkontált pénzmozgásos erőforráskorlátos projekt ütemezési problémát oldja meg. Ellenpélda adásával bizonyítja, hogy a hagyományos „minimális késleltetési alternatíva” elképzelés nem feltétlenül ad optimális erőforrás felhasználási megoldást mivel a rejtett erőforrás konfliktusok láthatatlanok maradhatnak a keresési eljárás során. Eljárása a „minimális erőforrás összeférhetetlenségi halmaz” elképzelésen alapul hatékony új metszési eljárással, és kijavítja mind a látható, mind a rejtett

erőforrás konfliktusokat. A probléma megoldásához egy primal-dual interior pont algoritmust alkalmaz. Számítási próbákkal igazolja, hogy az ajánlott új megközelítés nagyon gyors és hatékony. Az egzakt keresési algoritmust sugárkeresés (beam search) algoritmussal helyettesítve nagyméretű problémákat heurisztikusan megfelelő időn belül oldhatunk meg.

3.2.2. Heurisztikus megoldások

Az erőforráskorlátos NPV maximalizálás probléma egzakt megoldása már kevés tevékenységből álló projektek esetén is igen időigényes. Mivel egy projekt ütemezési probléma az életben gyakran több száz, vagy több ezer tevékenységből áll, a közepes és nagyméretű projektek megoldásához szükséges a heurisztikus megoldások igénybevétele, mivel az optimalizálási technikák alkalmazása ekkor lehetetlen. Megfelelő heurisztikák megtalálásával elfogadható futási idejű „jó minőségű" megoldáshoz juthatunk. A továbbiakban a terület eddigi fontosabb heurisztikus fejlesztéseit tekintjük át.

Kolisch és Padman [2001] az NPV maximalizálás problémára kifejlesztett heurisztikákat három osztályba sorolja. Az első az *optimalizálás alapú heurisztika*, amely a nemkorlátos NPV modellből veszi a megoldási értékeit, ami az RCPSP egy relaxációja. A második a *paraméter alapú heurisztika*, amely ezzel szemben a kritikus út és a pénzmozgás értékek információkat használja elsőbbségi szabályok fejlesztéséhez, ilyen például az utód tevékenységek pénzmozgásainak az összegét használó elsőbbségi szabállyal dolgozó heurisztika. A harmadik a *metaheurisztikus stratégiák*, melyekről később bővebben beszélünk a 3.3 és 3.4 pontban.

Smith-Daniels és Aquilano [1987] azokat az eseteket tekintik, amikor csak negatív pénzmozgások következnek be a projekt alatt, és pozitív pénzmozgás csak az utolsó

tevékenység befejeztekor következik be. Azt tapasztalták, hogy az ilyen esetekben az időtartam minimalizáló heurisztikák nagy valószínűséggel adnak jó NPV eredményt. Legkésőbbi kezdésű (late start) ütemezést használnak, és elsőbbségi szabályokat alkalmaznak, hogy válasszanak az egymással versengő munkák között.

Smith-Daniels és Aquilano [1987] összehasonlították a legkésőbbi kezdésű út (late start path schedule) ütemezést a korai kezdésű kritikus út ütemezéssel az időtartam és az NPV szempontjából. Feltételezték, hogy a negatív pénzmozgások a periódusok kezdetekor következnek be, és a projekt teljesítésekor egyszeri pénzbevétel történik. A feltevésüket a közismert Patterson problémákon tesztelték. Úgy találták, hogy inkább a késői kezdésű ütemezések esetén javult az átlagos NPV és az átlagos időtartam, mint a korai kezdésű ütemezések esetén. Heurisztikájuk paraméter alapú megközelítés.

Pinder és Marucheck [1996] diszkontált pénzmozgás súlyozó heurisztikus megoldást adnak.

R. A Russel [1986] adja az erőforráskorlátos NPV maximalizáló heurisztikák első összehasonlítását. A heurisztikákat 80 problémán tesztelte, a kis méretű, 30 tevékenységből álló problémáktól kezdve a nagy méretű, 1461 tevékenységből álló problémákig bezárólag. 5 heurisztikus szabályt értékel ki egy hatodik, véletlen szabály mellett. Három heurisztika A. H. Russel nemkorlátos NPV maximalizálási formulájából veszi az információkat. Másik két heurisztika már bizonyított a projekt időtartamának minimalizálásában. Elsőbbségi szabályokat vezet be az ütemezendő tevékenységek kiválasztásához. Nem volt olyan heurisztika, amely legjobbnak bizonyult az összes környezetben. A kis problémák esetén a heurisztikák hasonlóan teljesítettek, az optimális megoldástól 5-10%-ban tértek el, a „véletlen szabály” heurisztika teljesített legjobban. Ahogy a probléma mérete nő, a heurisztikák hatékonyságát az

erőforráskorlátosság mértéke határozza meg. Úgy találta, hogy a minimális időtartam problémához kidolgozott hatékony szabály, nagy projektek esetén, és amikor az erőforráskorlátok nem szűkek, a legjobban teljesített. Ellenben, amikor az erőforráskorlátok szorosak, egy RCPSP relaxáción alapuló, azaz a nemkorlátos esetből származó információkra épülő szabály bizonyult legjobbnak, ami megerősítette, hogy az erőforráskorlátos NPV maximalizálási probléma az erőforráskorlátos időtartam minimalizáló problémával szemben új megközelítéseket kíván.

Két kapcsolódó tanulmányban Padman és Smith-Daniels [1993] és Padman és ts. [1990] egy optimalizáción alapuló heurisztikát fejlesztettek ki, és összehasonlították meglévő heurisztikákkal. Egyszeres és többszörös prioritási sorokat használtak. Az előbbi tanulmányban feltételeznek koraisági (earliness) büntetést, az utóbbiban nem.

Amennyiben megengedjük, hogy többlet erőforrást használva a tevékenységek időtartama rövidülhet, az un. idő/költség tradeoff erőforráskorlátos projekt ütemezési probléma pénzmozgásokkal problémához jutunk. Erről a problémáról Icmeli és Erenguc [1996/a] ad részletes leírást.

Padman és ts. [1997] úgy találták, hogy a beágyazott elsőbbségi szabályon alapuló heurisztikák, amelyek egy relaxált optimalizációs modellnek az ismételt megoldásából veszik az információkat - eltérően Russel megoldásától, amely a relaxált modellt egyszer oldja meg - növelik a projekt NPV-t. Az optimalizáció vezérelt heurisztikus eljárást és a kilenc különböző beágyazott elsőbbségi szabályt tesztelték számos projekt környezetben, különböző háló struktúrákat, erőforráskorlátossági szinteket, pénzmozgás paramétereket feltételezve. PSD adathalmazon teszteltek. Úgy találták, hogy az új heurisztika a hatékonyságot tekintve felülmúlja a CPM-ből származó információkat használó heurisztikákat, és a legtöbb esetben a korábban a szakirodalomból ismert

heurisztikákat is. A legjobban teljesítő heurisztika a tevékenységeket elsődleges és másodlagos sorokba osztályozza, aszerint, hogy származik-e belőlük részteljesítési kifizetés, majd előreütemez.

Padman és Smith Daniels [1993] az előző munkát fejlesztik tovább, a relaxált optimalizációs modellt használva, kiértékelve a projekt ütemezés során a korai és a késői teljesítésből adódó büntetésekből származó veszteséget. Mohó eljárásba (greedy procedure) ágyaznak nyolc heurisztikát, melyet az előbbi tanulmányban tárgyaltak, hogy teszteljék, vajon a tevékenységek ütemezési sorba adása olyan hamar, ahogy az elődtevékenységek teljesültek, eredményez-e jobb projekt NPV-t. PSD adathalmazokon végzett tesztek bizonyítják a megközelítésük hatékonyságát.

Özdamar és Ulusoy [1996] egy iteratív ütemezési algoritmust mutattak be, a projekt időtartam és NPV javításának céljával. Megközelítésük paraméter alapú, modelljükben a tevékenységek költségei a kezdésükkor következnek be, és a pénzbevétel a projekt teljesítésekor következik be. Megoldásukat két irodalomból ismert problémahalmazon tesztelték. A tesztelés során megmutatták, hogy mindkét fenti célt sikerült javítani.

Pinder és Maruchek [1996] tanulmányukban 17 heurisztika teljesítményét hasonlítják össze, melyből 10-et először ebben a munkájukban ismertetnek. Ebből kettő szignifikánsan jobb teljesítményt nyújt számítási eredményeik szerint, mint az addig ismert heurisztikák. Ez a két heurisztika egyben a projekt időszükségletének minimalizálásának problémáját is eredményesen oldja meg.

Icmeli és Erenguc [1994], Zhu és Padman [1999] ütemezés javító eljárást (schedule-improvement procedure) ad.

Baroum és Patterson [1996] tanulmányában megállapítja, hogy nagy problémák esetén az optimalizálási technikák nem voltak eddig eredményesek, és leginkább egyszerű

„ökölszabályon” alapuló heurisztikákat alkalmaztak a kutatók. A tanulmány írói által kifejlesztett heurisztika a tevékenységek és logikai utódaik kumulatív pénzmozgásán alapuló súlyokat társít a tevékenységekhez, modelljük paraméter alapú megközelítés. Egy teljes faktoriális kísérleti tervet használtak, hogy felmérjék a heurisztikus eljárás teljesítményét, és összehasonlítsák azt a hagyományos eljárások hatékonyságával. Számítási eredményekkel mutatják be az eljárásuk hatékonyságát, és azt, hogy az a hagyományos eljárásoknál jobb eredményeket szolgáltat..

A fenti heurisztikák nem bizonyultak elég hatékonynak, jobb minőségű megoldásokra törekedtek a kutatók, amit a metaheurisztikák között találtak meg. A metaheurisztikus megoldások hatékonyságát számos kutató támasztja alá, - elsősorban a PSPLIB teszthalmazon végzett - futtatási eredményekkel. (Csébfalvi [2007], Kolisch-Padman [2001]).

3.3. Metaheurisztikák

A metaheurisztikák kifejlesztése, és alkalmazásuk egyre gyorsabban fejlődő tudományterület. Ez nagy részben a tudományban és az iparban egyaránt egyre fontosabbá váló kombinatorikai optimalizálási problémáknak köszönhető.

A $P = (S, f)$ *kombinatorikai optimalizálási (CO) probléma* esetén adott:

- A változók halmaza: $X = \{x_1,...,x_n\}$;
- Az értékkészletek: $D_1,...,D_n$;
- A változók között fennálló összefüggések;
- Egy $f : D_1 \times ... \times D_n \to R^+$ célfüggvény (goal function, objective function), amit minimalizálunk (vagy maximalizálunk).

Legyen $S = \{s = \{(x_1, v_1),...,(x_n, v_n)\} \mid v_i \in D_i, s$ kielégíti a fenti korlátokat$\}$. Ekkor S-t *keresési tér*nek (search space) hívjuk, és keressük az(oka)t az S-beli s^* értéke(ke)t, mely(ek)re a célfüggvény értéke minimális (maximális). Az S-beli s^* értékeket, diszkrét matematikai objektumokat *állapoto*knak (state) nevezzük, azt az S-beli értéket pedig, melyre a függvény érték minimális (maximális), globális minimumnak (maximumnak).

Számos gyakorlati fontossággal bíró probléma tartozik a CO problémák körébe, például utazó ügynök (travelling salesman) probléma, és ide tartoznak a témámba vágó ütemezési problémák közül a diszkrét esetek. A CO problémák általában NP-nehézségűek, tehát nincs algoritmus amely polinomiális idő alatt megoldja. Így aztán a legutóbbi 30 évben a közelítő megoldások egyre nagyobb és nagyobb figyelmet kaptak a CO problémák megoldásában. Ezeknek a közelítő megoldásoknak az alkalmazása során a garantáltan optimális megoldás megtalálását feláldozzuk a csupán „elég jó”, cserébe azonban kielégítő idő alatt megkapható megoldás kedvéért.

Az utóbbi 20 évben egy újfajta közelítő algoritmus fejlődött ki, amely megkísérelte az alapvető heurisztikus algoritmusokat egy magasabb szintű keretbe foglalni, annak érdekében hogy hatékony(abb)an és eredményes(ebb)en térképezze fel a keresési teret, ezeket az algoritmusokat metaheurisztikáknak nevezték el.

Mielőtt a metaheurisztika definícióját tárgyalnánk, tekintsünk át még néhány fogalmat, melyet a fenti fogalmak mellett a metaheurisztikák használnak. A különböző állapotok, megoldások halmazát, melyekkel a metaheurisztika dolgozik, *populáció*knak nevezzük. Ezek a populációk lehetnek egyeleműek, vagy többeleműek, és minden lépésben felülíródnak. Az állapotokat *egyed*eknek is nevezik.

Kulcs lépés a metaheurisztika során az *állapotátmenet* (state transition), amikor az adott állapot variánsát állítjuk elő. Az előállított új megoldást az egyed *utód*jának is hívjuk. Ez a lépés növelheti, vagy csökkentheti a célfüggvény értéket.

Az operátorok két csoportba sorolhatóak. Az első csoportba tartoznak azok, melyek egy megoldásból állítanak elő egy újabbat, az eredeti egy variánsát, mutációját, ezek a *mutációk*. Ebben az esetben az új állapotot az előző állapot egy szomszédjának hívjuk (neighbour). A másik csoportba tartoznak azok, melyek két vagy több megoldásból (szülőből) állítanak elő új megoldást, a szülők kódjának valamilyen kombinációja segítségével, lehetőség szerint a „jó" tulajdonságok megtartásával, ezeket összefoglaló néven *rekombináció*knak nevezzük. Ilyen például a *szelekciós operátor*, mely több állapot közül egyet kiválaszt, a *generációs operátor*, amely új véletlen állapotot hoz létre egy vagy több megoldásból, a *kereszteződés* (crossover operátor), mely a megoldás kódok darabjaiból rak össze egy új megoldást különböző szabályok szerint, az az operátor, amelyik felső és alsó korlátokat ad a célfüggvényhez.

Kifinomultabb metaheurisztikák az egyelemű pillanatnyi állapot helyett pillanatnyi állapotok halmazát tartják fent egyidejűleg több különböző jelölt állapottal. Ekkor az állapot átmenet lépésben a metaheurisztika törölhet vagy hozzáadhat a halmazhoz állapotokat.

A metaheurisztika nyomon követheti az aktuális optimum alakulását, a jelenlegi és a már kiértékelt optimumokat, egy vagy többelemű memóriát tarthat fent. Mivel a lehetséges állapotok halmaza igen nagy, a metaheurisztikák futását általában egy időkorláthoz (time budget) kötjük.

A metaheurisztikus algoritmusok közös jellemzője, hogy a véletlenszerűséget és a szabályszerűséget egyesítik, hogy így utánozzák a természeti jelenségeket. Ha a

biológiai jelenség a fizikai hűtési eljárás, akkor *szimulált hűtés*ről (simulated annealing) beszélünk, ha az állati viselkedésmód, akkor *tabu keresés*ről, ha a biológiai evolúció, akkor az *evolúciós algoritmus*okról, hogy csak néhány példát említsünk.

Ezeket az eljárásokat metaheurisztikáknak nevezték el, azonban a metaheurisztika fogalmának máig nincs egy közösen elfogadott definíciója. Az utóbbi néhány évben számos kutató próbált meg definíciót adni a fogalomra, melyek részben megegyeznek, részben eltérnek, mert más oldalról közelítik meg a fogalmat. Néhány ezek közül a definíciók közül:

- „A metaheurisztika egy iteratív vezérlő eljárás, amely az alsóbb szintű (subordinate) heurisztikák műveleteit vezérli, és módosítja azokat, hogy hatékony, magas minőségű megoldást állítson elő. Dolgozhat egyedi (single) megoldással, vagy a megoldások egy halmazával minden iterációs lépésben. Az alsóbb szintű heurisztikák lehetnek magas (vagy alacsony) szintű eljárások, vagy egyszerű lokális kereső eljárások, vagy épp egy konstrukciós eljárás.” (Volf és ts. [1999])
- „A metaheurisztika formálisan definiálható egy iteratív generációs eljárásként, amely alsóbb szintű heurisztikákat vezérel, különböző elképzelések intelligens kombinálásával, a keresési tér feltérképezéséhez és hasznosításához. A tanuló stratégiákat az információ strukturálásához használjuk, annak érdekében hogy hatékonyan találjunk optimális közeli megoldást.” (Osman és ts. [1996])
- „Magas szintű algoritmikus keret, vagy szemléletmód, mely optimalizációs problémákra specializálódott.” (Forrás: National Institute of Standards and Technology, http://www.nist.gov/)

- „A metaheurisztika az elgondolások azon gyűjteménye, melyek olyan heurisztikus eljárások definíciójához használhatóak, melyek alkalmazhatóak különböző problémák széles halmazára. Más szóval, a metaheurisztika egy általános algoritmikus keretként tekinthető, amely alkalmazható különböző optimalizációs problémákra, viszonylag kisszámú módosítással illeszthető az adott speciális problémához.” (Forrás: http://www.metaheuristics.net)

A szakirodalmat tanulmányozva felvázolhatóak azok az alapvető tulajdonságok, melyek jellemzik a metaheurisztikákat:

- A metaheurisztikák olyan stratégiák, melyek a kereső eljárásokat vezérlik.
- Céljuk a keresési teret hatékonyan feltérképezni a közel optimális megoldás megtalálása érdekében.
- A metaheurisztikus algoritmusokat alkotó technikák az egyszerű kereső eljárásoktól a komplex tanuló eljárásokig terjednek.
- A metaheurisztikus eljárások közelítőek, és rendszerint nem determinisztikusak.
- A metaheurisztika akkor is használható, ha a feladat struktúrája kevésbé ismert, mivel nem igényel területfüggő tudást, tehát használhatjuk abban az esetben is, ha nem tudjuk a problémát leírni, de a lehetséges megoldások minőségét valamilyen szempont szerint értékelni, rangsorolni tudjuk.
- A metaheurisztika alapcélkitűzéseit tekintve globális optimumot keres, ezért tartalmaz olyan mechanizmusokat, melyek megóvnak attól, hogy a keresési térnek csupán egy elzárt területét térképezzük fel, így az esetek jelentős részében globális maximumhoz (minimumhoz) közeli értéket adni megoldásként a kezdőpont választásától függetlenül, ellentétben korábban használt algoritmusokkal, melyek gyakran lokális optimumot adtak eredményül.

- A metaheurisztikák a felsőbb szintű stratégia által kontrollált heurisztikákon keresztül hasznosíthatnak területfüggő tudást.
- A fejlettebb metaheurisztikák a keresési eljárás alatt a korábbi lépések során szerzett tapasztalatokat használják a további lépések vezérléséhez.

A metaheurisztikákat több szempont szerint is osztályozhatjuk. Néhány ezek közül az osztályozások közül:

- *Egy vagy több elemű populációval rendelkező algoritmus.*

  Osztályozhatunk az alapján, hogy a metaheurisztika egy időben egyszerre hány megoldást kezel, azaz a megoldások egy többelemű populációjával, vagy egyedi megoldással dolgozik egy időben. Az utóbbi csoportba sorolható például a tabu keresés, a sztochasztikus hegymászó, és a szimulált hűtés (a konkrét metaheurisztikákat a 3.4 fejezetben ismertetjük), amelyek egyelemű populációval dolgoznak, az előzőbe tartozik például a genetikus algoritmus, és a disszertáció algoritmusa.

- *Memória használat*

  Talán a legfontosabb tulajdonság, melyet az osztályozásnál figyelembe vehetünk az, hogy használ-e memóriát a metaheurisztika, vagy sem. A Markov eljárás például nem használ memóriát, a tabu keresés és a disszertáció algoritmusa pedig igen. A keresés során felhalmozott tárolt ismeretanyag folyamatosan felülíródik a keresési folyamat előrehaladása alatt. Eltérés mutatkozik az eljárásoknál abban is, hosszú, vagy rövid távú memóriát használnak, illetve milyen nagyságú a memória.

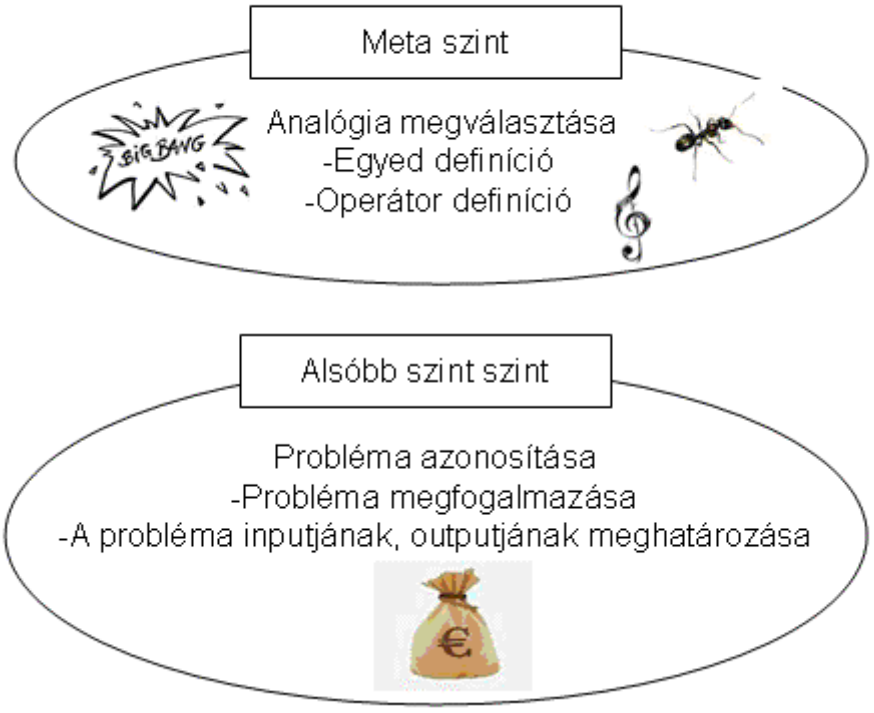
- *Dinamikus, vagy statikus a célfüggvény*

  A metaheurisztikákat osztályozhatjuk az alapján, amilyen módon a célfüggvényt kezelik. Míg a legtöbb algoritmus a célfüggvényt adottnak tekinti, változatlanul tartja az eljárás alatt, például a mi algoritmusunk is ilyen, némely metaheurisztika, például a Guided Local Search a keresés során változtatja azt. Ez esetben a keresési eljárás során folyamatosan belevesszük a célfüggvénybe az keresési eljárás alatt összegyűjtött információkat, és így nő az esélye, hogy nem lokális, hanem globális optimumot kapunk.

A metaheurisztika egy kétszintű heurisztika. A felső, *meta szinten* (meta level, vagy control level) – innen az elnevezés-, az eljárás fejlesztője választ egy analógiát, általában egy valós életből vett párhuzamot. Ilyen például a hangyaboly-analógia. Definiálja az egyedeket (species), ebben az esetben a hangyákat, azok tulajdonságait, például, hogy feromonnal rendelkeznek, majd meghatározza az egyedeken működő operátorokat, például mutációs, szelekciós, keresztező operátorokat.

Ezután az alsóbb, *második szinten* (object level, vagy solution representation) keresünk egy problémát, például a projekt NPV maximalizálást, és a megoldandó problémát „lefordítjuk” az adott metaheurisztika nyelvére. Ezen a ponton az a kérdés merül fel, hogy az adott keretben (például hangyaboly-analógia, vagy zenei harmónia keresés analógia) milyen formában tudjuk megfogalmazni a problémát, és annak megoldását. Ezen a szinten határozzuk meg a probléma inputját és outputját.

4. ábra A kétszintű metaheurisztika



A ábrázolási módszerek három féle típusba sorolhatóak.

a) *Bináris ábrázolási módszer*

Mivel a projekt ütemezési problémák esetében a változók általában binárisak, jogosan felvetődő gondolat, hogy az ábrázolási módszert is válasszuk binárisnak. Ilyen például a darwini evolúció analógiáját használó génreprezentáció, ahol az egyedeket bitsorozatokkal ábrázoljuk. Egy-egy ilyen bitsorozatot kromoszómának hívunk, a biteket pedig a biológiában is használt „allél” néven nevezzük.

b) *Egész ábrázolási módszer*

A másik felvetődő gondolat, hogy ne törekedjünk szigorúan a binaritásra, válasszunk egész ábrázolási módszert. Ilyen ábrázolási módszer például a tevékenység lista, melyben felsoroljuk a tevékenységek végrehajtási sorrendjét: Például: {4, 5, 1, 2, 3}.

Mindkét kategóriába tartozó heurisztikákkal az a probléma, hogy túl merevek, sok esetben „elrontják” a megoldást. Gyakran előfordul például, hogy a kapott eredmény nem „network feasible”, tehát sérülnek a tevékenységek között fennálló elsőbbségi feltételek, ekkor „elsőbbségi feltételeket helyreállító vektort” alkalmaznak a felhasználók. Abban az esetben pedig, amikor sérülnek az erőforráskorlátok, „erőforráskorlátot helyreállító vektort” alkalmaznak. Az ilyen - legtöbbször többszörösen is - javított heurisztikák ritkán sikeresek, kevéssé hatékonyak.

c) *Folytonos ábrázolási módszer*

A harmadik kategóriába sorolhatjuk a folytonos ábrázolási módszereket. Ilyen például a Csébfalvi [2007] által kifejlesztett Sounds of Silence algoritmus is, melynek változói a [-1, 1] intervallum elemei.

Általános szabályként megfogalmazható, hogy amikor metaheurisztikával problémát oldunk meg, keresünk egy meta szintű keretet, majd megkeressük azt ábrázolási módszert, amely legjobban illeszkedik hozzá. Akkor lesz igazán sikeres a megoldás, ha e kettő minél inkább illeszkedik egymáshoz és az adott problémához.

3.4. A metaheurisztikus algoritmusok történeti áttekintése

A hatvanas években merült fel először az a gondolat a tudósokban, hogy az evolúciós folyamatok mintájára globális optimalizációs eljárást alkossanak. Egymástól függetlenül, párhuzamosan több próbálkozás született. Kezdetben evolúció kutatással foglalkozó biológusok dolgoztak ki evolúció által inspirált algoritmusokat, de ezek a munkák nem váltottak ki nagy figyelmet.

1965-ben született az *evolúciós stratégia*, melyet repülőgép-szárnyak valós paramétereinek optimalizálására fejlesztette ki Rechenberg [1965]. Az evolúciós

stratégiák már használják a populáció és a mutáció fogalmát. Az eljárás evolúciós analógiát használva a viselkedési kapcsolatot hangsúlyozza egyed és utódai között.

Az *evolúciós programozást* (evolutionary programming) előrejelzési feladatok megoldására használt véges automaták automatikus kifejlesztésére dolgozta ki Fogel, Owens, Walsh [1966]. Az algoritmus szelekciós és mutációs operátort használ.

A *genetikus algoritmus* (GA) a természetes genetika és a darwini „természetes kiválasztódás" elvén alapul, egy rátermettségi célfüggvénnyel (fittness function) keresi a sok lehetséges megoldás közül a legjobbat. Az egyedeket legtöbbször bitsorozattal ábrázoljuk.

Több oka is van, hogy ezt a metaheurisztikát kiemelten lényegesnek tartjuk. Egyrészt ez a kutatók által talán leggyakrabban használt metaheurisztika. Másrészt az összes metaheurisztika közül a GA mutat legtöbb rokonságot a harmónia kereső metaheurisztikával (részletesen lásd később), és a disszertáció-beli algoritmus egyfajta harmónia kereső heurisztika.

A genetikus algoritmus legfőbb jellemzője hogy a megoldásokat populációba szervezi, így egyszerre több megoldással dolgozik, mely módszer tág keresési teret eredményez, és elősegíti hogy globális optimumot kapjunk. Az algoritmus az aktuális populációból minden lépésben egy új populációt állít elő három operátor – a szelekciós, a rekombinációs, és a mutációs operátor – segítségével. A szelekciós operátorral kiválasztott legrátermettebb egyedeken a rekombinációs és mutációs operátorokat alkalmazza.

A rekombinációs (recombination) operátor két egyedből hoz létre új egyedet, vagy egyedeket. A biológiai analógiában ez egy biológiai utód létrehozásának felel meg. Egyik leggyakoribb kombináció az egypontos kereszteződés (one-point crossover).

Ennek során a műveletben résztvevő egyedeket egy véletlenszerűen kiválasztott bitnél elvágjuk. A vágás előtti bittulajdonságokat örökli az új egyed a szülőtől, a vágás utániakat a másik szülőtől. Létrehozhatunk testvér egyedet is: ez esetben megcseréljük a két szülőt az eljárásban, és most a másiktól vesszük el a vágás előtti biteket, illetve a vágás utáni biteket, mint az eredeti lépésben.

A másik gyakori kombináció az egyenletes kereszteződés (uniform crossover). Ez esetben az utód az egyes génjeit adott ω valószínűséggel örökli az egyik szülőtől, és $1-\omega$ valószínűséggel a másik szülőtől. (ω egy 0 és 1 közé eső valós érték).

A mutációs operátor leggyakoribb alkalmazása során egyes géneket bizonyos valószínűséggel megváltoztatunk. Bitsorozat esetén a megváltoztatás nyilvánvalóan negálást jelent. (Valós vektor esetén véletlen értékkel helyettesítjük az eredetit.)

Mivel minden populáció az előzőnél rátermettebb egyedeket tartalmaz, így egyre jobb megoldásokat kapunk, ha elérünk egy kívánt célértéket, akkor az algoritmus befejeződik. A GA kifejlesztése Holland [1975] nevéhez fűződik, Goldberg [1989] és mások fejlesztették tovább. Az evolúciós stratégiával szemben a genetikus, és nem a viselkedési kapcsolatot hangsúlyozza egyed és utóda között, és eltér attól abban is, hogy elsősorban a rekombinációs operátorra támaszkodik, míg az evolúciós stratégiánál a mutáció a domináns operátor.

A *„genetikus programozás”* (genetic programming), a genetikus algoritmus egy speciális területe, a cél meghatározott feladatokat végrehajtó számítógépes programok automatikus kifejlesztése, Koza [1992] nevéhez fűződik.

A *„tabu keresés”* egyelemű populációt használ, és a kereső operátor csak a mutáció. A keresés során egy tabulistát tartunk fent, amely a legutóbb megvizsgált néhány megoldásból áll. Az algoritmus paramétere a tabulista mérete. Minden új elem esetében

megvizsgáljuk, hogy az benne van-e a tabulistában, ha igen, eldobjuk, ha nem, akkor amennyiben nem rosszabb, mint a régi megoldás, akkor elfogadjuk. (Ehhez a lépéshez hasonló lépés szerepel a disszertáció-beli algoritmusban is.) A régi megoldás tabulistára kerül. Ha a lista mérete elér egy bizonyos elemszámot, akkor a legrégebbi elemet töröljük. Ha a tabu keresés nem konvergál, a keresést véletlenszerűen állítjuk alaphelyzetbe. Kifejlesztése Glover [1986] nevéhez fűződik. A metaheurisztika fogalmát ő említi először ebben a munkájában.

Előfordulhat olyan probléma, amikor a szokásos operátorok (szelekciós, rekombinációs) nem eléggé hatékonyak az optimális megoldás keresésekor, mert fontos jellemzőket, motívumokat nem őriznek meg. Az építőkövek megóvásának technikája egy új algoritmus osztályhoz vezet, egy új evolúciós számításhoz, amit „probabilistic model building genetic algorithm" (PMBGA) néven is említenek, de ismert az „estimation of distribution algorithm" (EDA) elnevezés is. Az EDA-t Mühlenbein és Paaß [1996] dolgozta ki. Ez a heurisztika hasonlít a GA-ra, de annak operátorai helyett valószínűségi mintavétellel kapott valószínűségi becsléssel dolgozik. A PMBGA kezdeti populációját először véletlenszerűen generáljuk. Később minden iterációban kiválasztjuk az ígéretes megoldásokat a jelölt megoldások jelenlegi populációjából, és ezeknek a választott megoldásoknak a valószínűségi eloszlását becsüljük. Az új jelölt megoldásokat aztán a becsült valószínűségi eloszlás mintavételével generáljuk. Az új megoldásokat ezután egyesítjük az eredeti populációval, lecserélve néhány régit vagy az összeset. Ezt az eljárást addig folytatjuk, amíg a leállási feltétel bekövetkezik.

A „*részecske rajzás"* (particle swarm) egy evolúciós számítási eljárás Kennedy és Eberhart [1995] által fejlesztve, a madarak, méhek illetve a halak együttes viselkedése ihlette. A kutatók az élelmet kereső madárrajok, méhrajok, illetve halrajok

csapatmozgását tanulmányozták és kísérelték meg szimulálni. A GA-hoz hasonlóan, a „részecske rajzás” is egy populáció alapú optimalizációs eljárás globális optimum kereséshez. A rendszer véletlen megoldások populációjával indul, és keresi az optimumát újabb generációkon keresztül. A GA-tól eltérően azonban, a „részecske rajzásnak” nincs evolúciós operátora, mint a mutáció, vagy a rekombináció. Míg a genetikus módszereknél egyedekkel szimbolizálták az egyes beállításokat, úgy itt a paraméterértékek terében kis részecskék szimbolizálják azokat. A többdimenziós térben az optimálás során minden részecskének irányvektort és sebességet adnak. Ezután minden részecske „repül” a vizsgálati térben az adott irányba az adott sebességgel. Ezt az irányt és sebességet módosíthatja a többi részecske a környezetében. A heurisztika célfüggvényének kiértékelése után az összes részecske az éppen legjobb megoldást szimbolizáló részecske felé mozdul el. Ezek a helyi hatások végigterjednek a teljes csoporton, így kerül a csoport egyre kedvezőbb helyzetbe, egyre közelebb az optimumhoz. Néhány kiértékelés - mozgatás pár után az összes részecske az optimum közelében fog tartózkodni.

Ezeknek a fenti algoritmusoknak (evolúciós stratégia, evolúciós programozás, genetikus programozás, GA, részecske rajzás, PMBGA) a gyűjtőneve *evolúciós számítások* (evolutionary computation).

A „*szimulált hűtés”* (simulated annealing) algoritmus a fémek edzési folyamatának analógiájára működik. Egyelemű populációt használ, és a mutációs operátorral dolgozik, ebben eltér a GA-tól, viszont ebben egyezik meg a tabukereséssel. A visszahelyezési függvényben azonban eltér ez utóbbitól is. Az fémek edzése során a fém lassú hűtés során közel optimálisan rendezett atomi szerkezetet vesz fel. Az algoritmusban ha az új megoldás rosszabb, mint a régi, akkor elfogadjuk egy bizonyos

valószínűséggel, melyet a „hőmérséklet” paraméter szabályoz. A hőmérséklet a keresés folyamán fokozatosan csökken. Kirkpatric és ts. [1983] alakították ki a kombinatorikus (és más) problémákra vonatkozó szimulált hűtésen alapuló optimalizálási technikák alapjait.

A *„sztochasztikus hegymászó”* (stochastic hill climbing) algoritmus esetében szintén egyelemű a populáció, és a kereső operátor a mutáció, a fentiek közül ez a legegyszerűbb algoritmus. Az új megoldás felváltja a régit, ha az legalább olyan „jó”.

Mérnöki problémák megoldása során gyakran kell megkeresnünk gráfban két adott csúcspont közötti legrövidebb utat, ami különösen akkor nehéz, ha nem ismerjük előre a gráf szerkezetét. Ennek a problémának a megoldására is alkalmas *„hangyaboly”* (ant colony) algoritmust Dorigo [1992] fejlesztette ki Phd dolgozatában. Az algoritmus analógiáját szintén a természetből veszi. Az eljárás azon a felismerésen alapul, hogy a hangyák a táplálékszerzési folyamatban a lehetséges útvonalak közül az általuk lerakott feromon információk alapján rátalálnak a legrövidebbre. A természetben a legrövidebb út megtalálása létfontosságú, például egy hangyaboly esetében, hiszen nem mindegy, milyen hamar jutnak el a dolgozók a táplálékforrásig. Gyorsabb eljutás esetén más állatok előtt találják meg a táplálékot, illetve ezzel együtt kevesebb energiát, azaz táplálékot használnak fel. A probléma nehézsége, hogy az útvonaloptimalizálást irányító nélkül kell megoldani, illetve hogy a dolgozók nem képesek raktározni tapasztalataikat. Az útvonal választás szabálya némileg leegyszerűsítve: minden dolgozó a táplálékforrástól hazatérve feromon illatnyomot hagy, ezzel jelezve a táplálékforrás irányát. Ezt a nyomot a többi hangya valamilyen valószínűséggel követi. Ezen felül a rövidebb úton erősebb illatnyomot hagynak a dolgozók, így a követés során a nyom erősségét is figyelembe veszik a társak, illetve az idő múlásával arányosan a feromon

párolog. A feromon párolgás segíti elő, hogy az algoritmus leragadjon egy lokális optimumnál.

Az optimalizálási eljárás lényege:

Kezdetben beállítjuk a paramétereket, inicializálunk. Megadjuk például a hangyák számát, a minimális feromon értékeket a gráf élein, a feromon párolgási állandót, az iterációk számát. Az iterációs lépések mindegyikében kiszámítjuk a hangyák következő lépéseit, és aktualizáljuk a feromon értékeket. Végül eljutunk a „táplálékhoz", azaz az optimumhoz.

Az algoritmus előnye például a szimulált hűtéssel és a genetikus algoritmussal szemben az, hogy az állapotgráf az optimálás folyamán dinamikusan változhat, így az algoritmus valós időben képes alkalmazkodni a változáshoz.

Az egyik újabb fejlesztésű igen ígéretes metaheurisztika a „*Big Bang Big Crunch*" (BBBC) algoritmus. Ezt az evolúciós algoritmust Erol és Eksin [2006] fejlesztette ki a világegyetemben 15-20 billió évvel ezelőtt lezajlott ősrobbanás, és a világegyetem a jövőben talán valamikor bekövetkező fekete lyukká való „összeomlása" analógiájára. Az ősrobbanás megfelelője a metaheurisztikában a „Big Bang" fázis, az azt követő összeomlás megfelelője pedig a „Big Crunch" fázis. Ezek a fázisok felváltva ismétlődnek folyamatosan egymás után a metaheurisztika futása során, amíg az általunk megválasztott leállási kritérium be nem következik.

Az algoritmus a GA-hoz hasonlóan véletlenszerűen generált kezdeti populációval indul. A véletlen populáció létrehozásakor természetesen figyelembe vesszük a keresési tér korlátait. A következő lépésben minden populációbeli elemre számoljuk a célfüggvény értéket. Ezt követően megkeressük a keresési tér azon pontját, amely leginkább

illeszkedik ezekre az értékekre. Ezt a pontot hívjuk „tömegközéppontnak”, „fekete lyuknak”. Eddig tartott a Big Bang fázis.

A Big Crunch fázisban ezt követően egy képlet segítségével létrehozzuk az új populáció elemeit.

A képlet:

$x^{new} = x^c + lr / k$, ahol

- x^c a Big Bang fázis végén kapott tömegközpont
- l egy alkalmasan megválasztott szám. Úgy választjuk, hogy a származtatott pontok a keresési térbe essenek.
- r egy véletlen szám
- k az ismétlés szám

A Big Crunch fázis után a generált új tagok a következő Big Bang fázis bemenetei. Az eljárás addig folytatódik, amíg a leállási kritérium be nem következik. Az algoritmus végén a populáció elemei a tér egy pontjába „zsugorodnak össze”. Ez a pont lesz a remélt optimum.

A fejlesztők számítási eredményei szerint az új algoritmus teljesítménye felülmúlja a genetikus algoritmus, és a többi jelenleg ismert metaheurisztika teljesítményét.

Ez a metaheurisztika bizonyos problématípusok esetén igen hatékonynak bizonyul, de a disszertáció-beli probléma természetétől távol áll, arra nem érdemes használni. Ennek oka, hogy alkalmazása során gyorsan zsugorodik a keresési tér, így gyakran nem globális, hanem lokális optimumhoz konvergálnak a megoldások.

Egy másik új fejlesztésű populáció alapú metaheurisztika a *„méh algoritmus”* (bee algorithm), melyet Pham és ts. [2006] publikáltak. Az eljárást a méhek táplálékkeresés során ellejtett tánca inspirálta. Ezzel a jellegzetes tánccal közlik a kaptárba visszatérő

méhek társaikkal a táplálék irányát, a távolságát, minőségét, és mennyiségét. A méhtánc jelenségét Karl von Frisch osztrák etológus fejtette meg több évtizedes munkával, amiért 1973-ban Konrad Lorenz-el és Niko Tinbergennel megosztva orvosi és fiziológiai Nobel-díjban részesült.

A kolónia célja tagjainak minél optimálisabb „használata" az élelemgyűjtés során, azaz minél gazdagabb és minél közelibb lelőhelyekre szeretné küldeni a méheket.

Az élelemgyűjtés a felderítő méhek véletlenszerű kereső tevékenységével indul. Amikor visszatérnek, a tánccal beszámolnak az eredményeikről: a táplálék irányáról, a távolságáról, minőségéről, és mennyiségéről. Az eljárás során folyamatosan vizsgáljuk a felderített területek minőségét. A legértékesebb információkat szerző méheket választjuk, és az általuk látogatott területeket választjuk a szomszédsági keresés során. Az algoritmust kombinatorikai optimalizálási problémák, függvény érték optimalizálási feladatok megoldására, illetve bio-robot tervezésben és fejlesztésben használják.

Szintén populáció alapú metaheurisztika a „*tűzlégy algoritmus*" (firefly algorithm), melyet Yang [2008] fejlesztett ki a tűzlegyek fénylő képességének analógiáját használva. A tűzlegyek egynemű egyedek, így bármely egyed vonzó lehet bármely másik számára. Az egymásra gyakorolt vonzerőt fényléssel érik el, a vonzás erőssége arányos a kibocsátott fényerővel. Bármely két egyed közül a kisebb fényerejű fog elmozdulni a fényesebb egyed felé. A fényesség csökkenhet, ha az egyedek távolsága nő. Ha nincs egy adott egyednél fényesebb társ, akkor az véletlenszerűen fog elmozdulni. A cél maximalizálni a populáció egyedei fényerejének összegét.

Erőforrás korlátos projekt ütemezési problémák megoldására fejlesztették ki Agarwal és ts. [2007] a „*mesterséges immunrendszer"* (artificial immun system) metaheurisztikát. Eljárásukat az immunrendszer működésének analógiáját felhasználva hozták létre. A

többi metaheurisztikával szemben, melyeknél az analógia megértéséhez általában elég az adott terület felszínes - átlagember számára rövid idő alatt is elsajátítható - ismerete, ahhoz hogy ennek a heurisztikának a működését maradéktalanul átlássuk, az immunrendszer alapfogalmainak – antigén, antitest, immuncellák, osztódás, stb. - , és működésének részletes megértése szükséges.

Analógiájukban a rendszer – azaz az adott ütemezés - beteg, ha bizonyos korlátok sérülnek, például az elsőbbségi feltételek, vagy az erőforrás korlátok. Az eljárás során célunk a rendszer immunizálása, meggyógyítása, azaz egy olyan hosszminimalizáló ütemezés megtalálása, mely nem sérti a korlátokat.

Egyik legújabb fejlesztésű metaheurisztika az „*imperialist competitive algorithm*". Kaveh és Talatahari [2010] fejlesztette ki politikai, stratégiai analógiát használva. A többi metaheurisztikától eltérően ez az eljárás a társadalomtudományból veszi az analógiáját, az ismert Monopoly játékhoz hasonló egyszerűsítéseket használ. A keresési tér területei az országok, melyek vagy hódítók, vagy azok gyarmatai. Az eljárás véletlenszerűen generált országokkal indul. A legjobb értékű országokat hódítónak, míg a többit azok gyarmatainak nevezzük ki. Az eljárás alapja a gyarmatok mozgása a hozzájuk tartozó hódító ország felé, és a területekért folytatott verseny. Az eljárás során az erős szereplők folyamatosan erősödnek, a gyengék folyamatosan gyengülnek, végül összeomlanak. Ekkor egy hódító ország birtokába kerülnek. A szerzők tanulmányukban egy rácsszerkezet optimalizálási problémára alkalmazzák a metaheurisztikát.

Ha tüzetesebben megnézzük, a fenti algoritmusok mind egymás módosításai valamilyen értelemben, ami elég érdekes, tekintve hogy különböző természeti jelenségek analógiájára épülnek! A sztochasztikus hegymászó egy állandó nulla hőmérsékleten futó szimulált hűtés, vagy egy nulla elemű tabulistát használó tabu keresés, vagy

egyelemű populációt használó ún. elitista[2] metaheurisztika. A hangyaboly módszer tekinthető úgy, mint egy hosszú távú memóriát alkalmazó evolúciós algoritmus.

Mi ennek a hasonlóságnak az oka? A metaheurisztikák jellemzői, hogy mind a természetből, a körülöttünk lévő világból veszik az analógiát. Mivel ez a valós világ optimalizáció centralizált, ezért van, hogy a metaheurisztikák érdekes rokon vonásokat mutatnak egymással, hiába veszi az egyik a fémek hűtéséből, a másik a genetikából a párhuzamot, az összes jelenség mögött ugyanaz az optimalizáció centralizáltság húzódik.

Általánosságban megállapítható, hogy jelenleg egyre élénkebb az információcsere a fenti területek között, kihasználva, hogy a módszerek alapelvei, és főbb komponensei megegyeznek. Így aztán a módszereket gyakran kombinálják, így jön létre például a tabu listát alkalmazó szimulált hűtés, vagy az „angyalkar” (angel) heurisztika is, mely az *ant* colony, *ge*netic algoritmus, és *l*ocal search eljárások egyesítése.

Ezek a metaheurisztikák népszerű megoldások, de az életből vett komplikált, nehéz optimalizációs problémákhoz újabb, erőteljesebb algoritmusokra van szükség.

3.5. A harmónia kereső algoritmus

Az utóbbi években Lee és ts. [2005] kifejlesztettek egy új harmónia kereső (harmony search) igen hatékony metaheurisztikus algoritmust.

A harmónia keresés egyesíti az ismert metaheurisztikák struktúráját. A tabu keresésből átveszi a „tabulistát” (a Harmony Search-ben harmony memory, HM), a szimulált hűtésből a hőmérséklet/ráta változtatását (a Harmony Search-ben harmony memory consideration, HMCR). A genetikus algoritmushoz hasonlóan szimultán kezel egyszerre

[2] Elitist: a régi populáció legrátermettebb elemét is az új populációba helyezzük, akár ki lett választva, akár nem, feltéve hogy az új populációban egyébként minden elem rosszabb lenne.

több vektort. Ez a vektor kezelési mód tág keresési teret eredményez, megkönnyíti a globális optimumhoz való konvergálást, így elkerüljük korábbi eljárások hibáját, azt, hogy lokális optimumot adnak eredményül. Azonban a genetikus algoritmustól jelentősen eltér abban, hogy nem két szülőtől származtat egy új vektort, hanem több vektortól. A PMBGA-hoz (probabilistic model building genetic algorithm) hasonlóan az új vektor elemeit függetlenül kezeli.

A Harmony Search eljárás annak a keresésnek a folyamatát igyekszik megragadni, amelynek segítségével egy improvizációra képes zenekar eljut az általa tökéletesnek gondolt teljes harmóniáig. A zenei világ-beli harmónia analóg az optimalizációs megoldás-vektorral, a zenei rögtönzések analógok az optimalizációs technikák keresési sémáival.

Miben rejlik a harmónia keresés hatékonyságának titka? A Harmony Search algoritmus előnyös tulajdonsága, hogy nem igényel kezdeti értékeket a döntési változókhoz. Továbbá, a gradiens keresés helyett a Harmony Search algoritmus sztochasztikus véletlen keresést használ, így a gradiens keresés, mely nehézzé és bizonytalanná válhat, amikor a célfüggvénynek és a korlátoknak többszörös és/vagy éles csúcsai vannak, szükségtelen. A véletlen keresés harmonikus memóriát feltételező rátán (HMCR, harmony memory consideration rate) és hangmagasság szabályozó rátán (PAR, Pitch Adjusting Rate, vagy más néven SAR, Sound Adjusting Rate) alapul. Korábbi metaheurisztikus optimalizációs algoritmusokkal összehasonlítva a Harmony Search algoritmus kisebb matematikai igényű, és jobban alkalmazkodik, könnyebben adoptálható az optimalizációs problémák különböző típusai számára. A fenti tulajdonságainak köszönhetően a Harmony Search jobb megoldásokat ad, mint az eddig ismert algoritmusok. Ezt tényt számos kutató számítási próbákkal, esettanulmányokkal

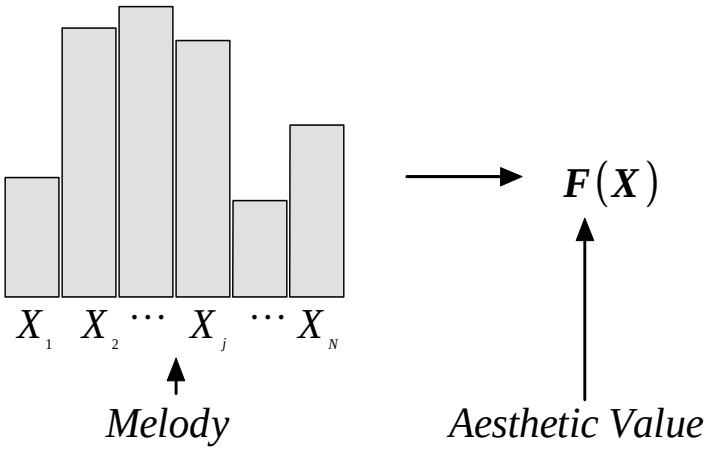
is alátámasztja (Lee és ts. [2005], Mahdavi és ts. [2007], Vasebi és ts. [2007], hogy csak néhány példát említsünk). A Harmony Search nagyon jól azonosítja a keresési tér magas teljesítményű részterületét, és elfogadható futási idővel rendelkezik.

A Harmony Search a zenei világból vett improvizáció és az optimalizáció közötti analógián alapul. A zenei improvizációban minden zenész ötletszerűen játszik valamilyen dallamot a hangszerén, és az együttes hangzás, a „harmónia-vektor” jó esetben valamilyen harmonikus, szép dallam lesz.

Az alábbi ábra szemlélteti az optimális függvényérték keresés zenei analógiáját. A változóknak a zenészek, a dallamok, míg a maximális függvényértéknek a harmonikus hangzás felel meg.

5. ábra Az optimumkeresés zenei analógiája

$$\boldsymbol{max}\left\{\boldsymbol{F}(\boldsymbol{X}) \mid \boldsymbol{X}=\left\{X_j \mid \underline{X}_j \le X_j \le \overline{X}_j,\ j \in \{1,2,\ldots,N\}\right\}\right\}$$

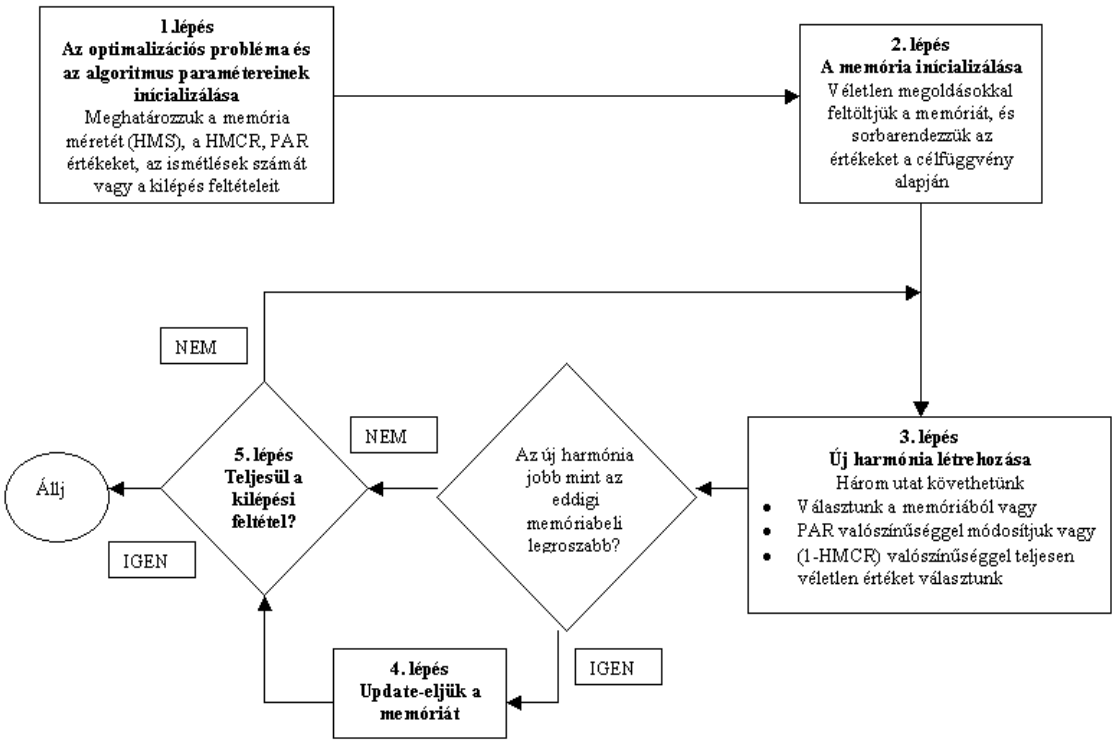


Forrás: Lee és Geem [2005]

Amennyiben szép az eredményül kapott hangzás, a zenész megjegyzi, „eltárolja” a memóriájában. Az optimalizáció világában az *n* zenész helyett *n* döntési változónak adunk valamilyen értéket a lehetséges tartományokban, így kapunk egy vektort. Az együttes hangzás „megfelelője” egy célfüggvényérték, amelyet megjegyzünk, amennyiben kedvezőnek ítéljük. A szebb hangzásnak az optimalizációban a kedvezőbb érték felel meg.

Az alábbi ábrán szemléltetjük a Harmony Search algoritmus lényegét:

6. ábra A harmónia kereső algoritmus



Forrás: Lee és Geem [2005], magyarra fordítva

1. Meghatározzuk a memória méretét (HMS), a HMCR, PAR, *bw* értékeket, és az ismétlések számát, vagy a kilépési feltételt. A PAR értéknek tetszés szerinti (-1,1)

intervallumba eső értéket választunk, *bw* (bandwith, sávszélesség) értéknek egy tetszés szerinti értéket. A futás során a PAR és a bw érték nem változik.

2. Ezután véletlenszerű megoldásokkal felöltjük a memóriát, és sorbarendezzük az értékeket a célfüggvény alapján.

3. Majd új harmóniákat hozunk létre. Amikor egy zenész improvizál, rendszerint három utat követhet:

- a memóriában tárolt dallamok közül választ, vagy
- a memóriában tárolt dallamok közül választ, és azt módosítja kissé, vagy
- egy teljesen véletlen hangzást választ.
- A Harmony Search követi ezt az eljárást, amikor döntési változókat hoz létre:
- HMCR valószínűséggel választunk a Harmony Search memóriából (memory consideration),
- melyet PAR valószínűséggel módosítunk, (1-PAR) valószínűséggel változatlanul hagyjuk. (pitch adjustment) A módosítás egy $bw \times u(-1,1)$ érték hozzáadásával történik. Az $u(-1,1)$ egy egyenletes eloszlás -1 és 1 között.
- vagy egy teljesen véletlen értéket választunk (1-HMCR) valószínűséggel (randomization).

4. Frissítjük a memóriát: Ha az így kapott harmónia jobb, mint az eddigi memória-beli legrosszabb, akkor az eddigi legrosszabb megoldást erre az újra cseréljük, egyébként folytatjuk az eljárást. A memóriabeli megoldásokat mindig sorba rendezzük a kiértékelő függvény segítségével. Így az eljárás során a megoldások minősége folyamatosan javul.

5. Megismételjük a (3) és (4) lépéseket mindaddig, amíg egy előre meghatározott leállítási feltétel (*termination criterion*) nem teljesül, például elérünk egy bizonyos lépésszámot, vagy futási idő korlátot.

A Harmony Search algoritmus egy továbbfejlesztése, a Mahdavi és ts. [2006] nevéhez fűződő *improved harmony search algorithm* (IHS). Az Improved Harmony Search a Harmony Search finomhangolása, így annál jobb teljesítményt nyújt. Az új elem ebben az algoritmusban csupán annyi, hogy a PAR érték a futás során nő, míg a módosítás mértéke (bw) a futás során csökken. Ez a módosítás jelentős javulást eredményez a hatékonyságban, melyet a szerzők számítási eredményekkel igazolnak a tanulmányban.

Az ebben a disszertációban kifejlesztett heurisztika a harmónia keresés egyik első többszempontos kiterjesztése. A harmónia kereső algoritmust újdonsága miatt elsősorban még nem a projekt ütemezés terén alkalmazták, hanem műszaki, mérnöki problémák megoldására. Ilyen példát számosat találunk: (Lee és ts. [2005], Mahdavi és ts. [2006], Vasebi és ts. [2007].

3.6. Erőforrás kiegyenlítés

Az erőforrás kiegyenlítési problémát az 5.3 fejezetben fogjuk említeni a továbbfejlesztési irányok szempontjai között, így röviden áttekintjük a probléma leglényegesebb korábbi megoldásait.

Egzakt megoldó algoritmus a problémára igen kevés született, ennek oka a probléma NP-nehéz volta. Ezen kevés megoldás közül a legismertebb eredmény Neumann és Zimmermann [1999] egzakt algoritmusa.

Számos heurisztikus megoldás született a problémára. Ezen eljárások többsége a legkorábbi ütemezésből indul ki, és bizonyos tevékenységek ütemezését késleltetik

valamilyen „ökölszabály” alkalmazásával. A legelső eredmény Burgess és Killebrew [1962] heurisztikája volt.

Ahuja [1976] explicit leszámlálási algoritmust, Younis és Saad [1996] egész programozási megoldást adott a problémára. Implicit leszámlálási algoritmust fejlesztett ki Zimmermann és Engelhart [1998]. Ezen megoldások mindegyike egy módú és hagyományos erőforrás kiegyenlítési mértéket használ. A hagyományos erőforrás kiegyenlítési mértékek a következők: maximális erőforrás felhasználás, az átlagos erőforrás felhasználástól való eltérés, az egymás után követő időperiódusok erőforrás felhasználás eltérése.

A „konkáv erőforrás felhasználást” fogalmazták meg célként Tavares [1987], Hajdu [1997], Lova és ts. [2000] tanulmányukban. Konkáv erőforrás felhasználásról beszélünk, ha bármely három tetszőlegesen kiválasztott időpontban, a középső ponthoz tartozó erőforrás felhasználás nagyobb vagy egyenlő, mint szélső pontokhoz tartozó erőforrás felhasználás minimuma.

4. Egy új harmónia kereső algoritmus

A 3. fejezetben említettük, hogy a harmónia kereső metaheurisztikát leginkább mérnöki problémák megoldására használják. Akad azonban olyan alkalmazás is, mely esetén a harmónia kereső algoritmust projekt ütemezési probléma a megoldására alkalmazták.

Ilyen megoldás például Csébfalvi és ts. [2008], Láng [2009/a, 2009/b, 2009/c, 2010, 2010/b] – ezek a megoldások a disszertáció-beli algoritmus fejlesztésének különböző állomásai - , és Szendrői [2009] heurisztikái. Szendrői a több módú erőforráskorlátos projekt ütemezési probléma megoldására, Láng az erőforráskorlátos projektütemezés során a nettó jelenérték maximalizására ad harmónia kereső algoritmuson alapuló megoldást. Ezekben a tanulmányokban a szerzők Csébfalvi [2007] erőforráskorlátos projekt ütemezési problémákra adott Sounds of Silence metaheurisztikus algoritmusát fejlesztik tovább a fenti problémák megoldására.

Csébfalvi és ts. [2008], és Láng megközelítésének újdonsága a diszkontált pénzmozgásos erőforráskorlátos projekt ütemezés irodalmában azok hierarchikus célfüggvény szemlélete, azaz a heurisztika elsődleges és másodlagos szempontot egyszerre kezel. Az erőforráskorlátos projekthez annak „legjobb” ütemezését keressük, ahol a „legjobb” alatt egy minimális időtartamú erőforráskorlátos ütemezést értünk, melynek nettó jelenérték mértéke maximális. Ilyen elsődleges - másodlagos szempont szerinti (bi-criteria) optimalizálás korábban a szerző tudomása szerint nem született. Ez a hierarchikus megközelítés reálisabb, mint a hagyományos egy szempont szerinti optimalizálás. Az utóbbiban ugyanis elméletileg előfordulhat olyan eset, hogy egy negatív pénzmozgás a projekt befejezési idejét kitolja a felső korlátig („amennyire csak

lehet”), így aztán a modell szerint „optimális” ütemezést kapunk, azonban nyilvánvalóan nem életszerűen optimális ez az ütemezés.

Az optimális ütemezés elméletileg két egészértékű (nulla-egy típusú) programozási feladat megoldását jelenti, ahol az első lépésben meghatározzuk a minimális időtartamú erőforráskorlátos ütemezések időtartamát, majd a második lépésben az optimális időtartamot feltételként kezelve megoldjuk a nettó jelenérték maximalizálási problémát minimális időtartamú erőforráskorlátos ütemezések halmazán. A probléma NP-nehéz jellege miatt az egzakt megoldás elfogadható idő alatt csak kisméretű projektek esetében képzelhető el. Az alkalmazott hibrid eljárás egy konfliktus javító, harmónia kereső metaheurisztika és egy lineáris programozáson alapuló lokális kereső algoritmus kombinációja. A hibrid eljárás kihasználja azt a tényt, hogy nemkorlátos nettó jelenérték maximalizálási probléma polinomiális idő alatt megoldható, ha a megelőző-követő relációkat teljesen unimoduláris[3] formulában írjuk le. Az ajánlott metaheurisztika hatékonyságának szemléltetésére számítási eredményeket adunk a jól ismert és népszerű PSPLIB tesztkönyvtár J30 részhalmazán futtatva. Az egzakt megoldás generálásához egy korszerű MILP szolvert (CPLEX) alkalmaztunk.

Ebben a fejezetben először ismertetjük a probléma megoldásához felhasznált modellünket, majd részletezzük algoritmusunkat. Ismertetjük a metaheurisztika zenei analógiáját, kiemeljük a fontosabb lépéseket – az improvizálást, a rejtett erőforrás felhasználási konfliktusok javítását, a teljesen unimoduláris mátrixok alkalmazását. Ezek után végigkövetjük a lépéseket egy kisméretű projekt segítségével. A továbbiakban ismertetjük az algoritmus pszeudokódját, majd részletes számítási eredményeket adunk, mellyel igazoljuk algoritmusunk hatékonyságát és életképességét.

[3] Egy mátrix teljesen unimoduláris ha minden aldeterminánsa +1, 0, vagy –1.

4.1. A modell

A modell megoldása során az erőforráskorlátos feladatütemezési problémák nettó jelenértékének maximalizálására keresünk minél hatékonyabb heurisztikus algoritmust. Célunk a tevékenységeket úgy ütemezni, hogy az elsőbbségi és az erőforráskorlátok teljesüljenek, és a projekt időtartamának minimalizálása mellett a projekt nettó jelenértékét maximalizáljuk. Modellünkben a következő feltevésekkel élünk az 1. táblázat jelöléseinek megfelelően:

- A projekt N darab valós tevékenységből áll, $i \in \{1,2,...N\}$.
- A tevékenységek nemmegszakíthatóak végrehajtásuk során, a időtartamuk D_i.
- X_0 és X_{N+1} 0 időtartamú áltevékenységek, mellyel a projekt kezdetét és végét jelezzük.
- A tevékenységek között megelőző-követő relációkat feltételezünk. Tehát egy tevékenység végrehajtását nem kezdhetjük meg addig, amíg az összes őt megelőző tevékenység be nem fejeződött.
- Jelölje a $\boldsymbol{PS} = \{i \to j \mid i \neq j, i \in \{0,1,\ldots,N\},\ j \in \{1,2,\ldots,N+1\}\}$ halmaz a közvetlen megelőző-követő relációk halmazát.
- A tevékenységek végrehajtásához erőforrásokat igényelnek, az i tevékenység R_{ir} egységet igényel a $r \in \{1,...,R\}$ erőforrásból végrehajtása során minden időperiódusban. Mivel az r erőforrásból, ($r \in \{1, ...,R\}$) csak R_r egység áll rendelkezésre minden időperiódusban, a tevékenységek nem ütemezhetők a legkorábbi (tehát a nemkorlátos, csak elsőbbségi korlátokat kielégítő esetben fellépő) kezdési időpontjukra, csak későbbre.

- Jelöljük a tevékenységek időtartamának összegét $\overline{T}$-al, és rögzítsük a $\overline{T}+1$ időperiódusba a projekt végét jelképező $N+1$-ik áltevékenységet.
- Minden tevékenységhez pénzmozgást társítunk, mely feltételezésünk szerint a tevékenység befejeződésekor lép fel, és a projekt kezdetére diszkontáljuk vissza.

A megoldandó modell:

$$\textbf{\textit{max}}\left[NPV = \sum_{i=1}^{N} \sum_{t \in T_i} C_{it} * X_{it} \right] = NPV^* \tag{1}$$

$$X_i + D_i \le X_j,\ i \to j \in \textbf{\textit{PS}} \tag{2}$$

$$X_{N+1} = \overline{T} + 1 \tag{3}$$

$$X_i = \sum_{t \in T_i} X_{it} * t,\ T_i = \left\{ \underline{X}_i, \underline{X}_i + 1, \ldots, \overline{X}_i \right\},\ i \in \{1,2,...,N\} \tag{4}$$

$$\sum_{t \in T_i} X_{it} = 1,\ X_{it} \in \{0,1\},\ i \in \{1,2,...,N\} \tag{5}$$

$$A_t = \left\{ i \,\middle|\, X_i \le t < X_i + D_i, i \in \{1,2,...,N\} \right\},\ t \in \{1,2,...,T\} \tag{6}$$

$$U_{tr} = \sum_{i \in A_t} R_{ir},\ t \in \{1,2,...,T\},\ r \in \{1,2,...,R\} \tag{7}$$

$$U_{t\,r} \le R_r,\ t \in \{1,2,...,T\},\ r \in \{1,2,...,R\} \tag{8}$$

$$C_{it} = C_i * e^{-\alpha(t+D_i-1)},\ i \in \{1,2,...,N\},\ t \in T_i \tag{9}$$

$$X_{it} \in \{0,1\},\ t \in T_i,\ i \in \{1,2,...,N\} \qquad (10)$$

- A (10) egyenletben leírt bináris változók meghatározzák az összes projektbeli tevékenység kezdési idejét. $X_{i,t} = 1$, ha az i-edik tevékenység t időperiódusban kezdődik, egyébként $X_{i,t} = 0$.
- A (9) egyenlet a t időperiódusban kezdődő, i-edik tevékenység pénzmozgását adja meg a projekt kezdetére visszadiszkontálva, a tevékenység befejezési idejének függvényében. Definíció szerint az i tevékenységhez ($i \in \{1,2,...,N\}$) társított C_i pénzmozgás lehet pozitív, negatív, és 0, és feltevésünk szerint tevékenység befejezési időpontjában következik be. A diszkont rátát jelöljük α-val.
- Az (1) egyenlet a modell célfüggvényét írja le, mely maximalizálja a projekt ideje alatt bekövetkező összes pénzmozgás diszkontált értékének összegét. Megjegyezzük, hogy a legkorábbi ütemezés nem maximalizálja szükségszerűen a pénzmozgások nettó jelenértékét.
- A (2) egyenlet a tevékenységek között fennálló elsőbbségi feltételeket írja le. A tevékenységek között megelőző-követő relációkat feltételezünk. Tehát egy tevékenység végrehajtását nem kezdhetjük meg addig, amíg az összes őt megelőző tevékenység be nem fejeződött. Ha a j tevékenységnek az i tevékenység közvetlen előzménye, akkor a j tevékenység nem kezdődhet el mindaddig, amíg az i tevékenység be nem fejeződik.

- A (3) egyenlet szerint a projekt végét jelző X_{N+1} áltevékenységet a $\overline{T}+1$ időpontban rögzítjük.
- A (4) egyenlet biztosítja, hogy az minden tevékenység a legkorábbi és a legkésőbbi kezdési időpontjai közötti időtartamban kezdődjön el. $X_{i,t}=1$, ha az i-edik tevékenység t időperiódusban kezdődik, egyébként $X_{i,t}=0$.
- Az (5) egyenlet biztosítja, hogy minden tevékenység pontosan egyszer kezdődjön el.
- A (6) egyenlet megadja a t időperiódusban aktív tevékenységek halmazát.
- A (7) egyenlet a t időperiódusban az r-edik erőforrásból fogyasztott mennyiséget adja meg.
- A (8) egyenlet az erőforráskorlát kielégítését követeli meg. Az erőforráskorlátokat állandónak feltételezzük a projekt időtartama alatt.

Tekintsük a PS halmaz és a konfliktus javító reláció halmaz úniójának nem redundáns részhalmazát, így kapjuk a PS^+ halmazt. Írjuk fel a (2) egyenletben ismertetett szokásos formában leírt elsőbbségi feltételeket egy másik alakban:

Az új modell:

$$\boldsymbol{max}\left[NPV=\sum_{i=1}^{N}\sum_{t\in T_i}C_{it}*X_{it}\right]=NPV^* \tag{11}$$

$$\sum_{p=T_i}^{\overline{X}_i}X_{ip}+\sum_{q=\underline{X}_p}^{T_i+D_i-1}X_{jq}\le 1,\ T_i\in\left\{\underline{X}_i,\underline{X}_i+1,\ldots,\overline{X}_i\right\},\ i\to j\in \mathrm{PS}^+ \tag{12}$$

$$X_{N+1}=\overline{T}+1 \tag{13}$$

$$X_i = \sum_{t \in T_i} X_{it} * t\ ,\ T_i = \left\{ \underline{X}_i, \underline{X}_i + 1, \ldots, \overline{X}_i \right\},\ i \in \left\{ 1,2,...,N \right\} \tag{14}$$

$$\sum_{t \in T_i} X_{it} = 1\ ,\ X_{it} \in \left\{ 0,1 \right\},\ i \in \left\{ 1,2,...,N \right\} \tag{15}$$

$$C_{it} = C_i * e^{-\alpha\left(t + D_i - 1\right)}\ ,\ i \in \left\{ 1,2,...,N \right\},\ t \in T_i \tag{16}$$

$$X_{it} \in \left\{ 0,1 \right\},\ t \in T_i\ ,\ i \in \left\{ 1,2,...,N \right\} \tag{17}$$

Az új modellben a fent említett mellett még egy lényeges változtatást tettünk. Elhagytuk a (7) és (8) egyenletben szereplő erőforrás korlátokra vonatkozó feltételeket. Ezt azért tehettük meg, mert mint később látni fogjuk, az algoritmus ezen pontján a konfliktusjavítás eredményeképpen olyan ütemezések fognak szerepelni, melyekben minden tevékenység, mely az elsőbbségi feltételek szerint mozgatható, mozgatható az erőforrás korlátok szerint is, így azokat itt már nem kell figyelembe vennünk.

Ezt az új modellt alkalmazva a nemkorlátos nettó jelenérték probléma (NPVP) polinomiális idő alatt oldható meg, mint egy LP probléma. Ezt a megoldást a 4.2.4 fejezetben fogjuk részletezni.

Az erőforráskorlátos nettó jelenérték maximalizálási probléma (RCNPVP) ábrázolásához tekintsünk példaként egy projektet 30 valódi tevékenységgel, melyek négy megújuló erőforrást igényelnek. A J30-1-1 projekt példával dolgozunk, a Kolisch Sprecher [1996]-féle PSPLIB teszt könyvtár (http://129.187.106.231/psplib/) J30 „legkönnyebb” egymódú teszthalmazának első példájával.

A példa legfontosabb paramétereit a következő táblázatokban foglaljuk össze:

Táblázat 2 – J30 példa paraméterei

Tevékenységek száma	30
Erőforrások száma	4
Legkorábbi ütemezés hossza:	39
Tevékenységek hosszának összege:	159
A legrövidebb erőforrás korlátos ütemezés hossza	44

Táblázat 3 – J30 példa paraméterei

Erőforrás korlátok	
Erőforrás száma	**maximuma**
R1	12
R2	13
R3	4
R4	12

Táblázat 4 – J30 példa paraméterei

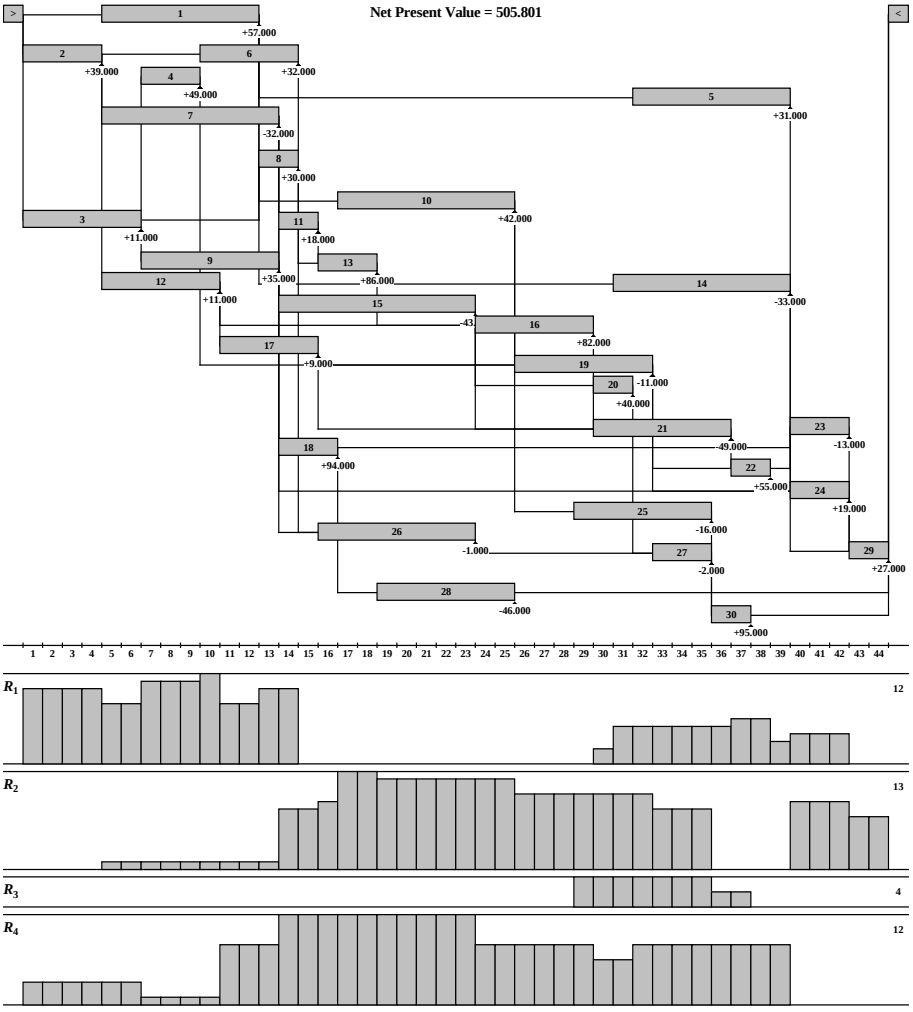
Tevékenység				**Tevékenységek erőforrás igénye**						
sorszám	**hossz**	**legkorábbi kezdési ideje**	**legkésőbbi kezdési ideje**	**R1**	**R2**	**R3**	**R4**	**Elsőbbségi feltételek**		
0	1	0	0	0	0	0	0	0->1	0->2	0->3
1	8	1	128	4	0	0	0	1->5	1->10	1->14
2	4	1	121	10	0	0	0	2->6	2->7	2->12
3	6	1	122	0	0	0	3	3->4	3->8	3->9
4	3	7	142	3	0	0	0	4->19		
5	8	9	149	0	0	0	8	5->29		
6	5	5	141	4	0	0	0	6->26		
7	9	5	125	0	1	0	0	7->11	7->18	7->26
8	2	7	134	6	0	0	0	8->13		
9	7	7	128	0	0	0	1	9->15	9->24	
10	9	9	136	0	5	0	0	10->19	10->25	
11	2	14	134	0	7	0	0	11->13		
12	6	5	133	4	0	0	0	12->16	12->17	
13	3	16	136	0	8	0	0	13->16		
14	9	9	145	3	0	0	0	14->24		
15	10	14	135	0	0	0	5	15->20	15->21	
16	6	19	139	0	0	0	8	16->21		
17	5	11	140	0	0	0	7	17->19	17->21	
18	3	14	149	0	1	0	0	18->23	18->28	
19	7	18	145	0	10	0	0	19->22	19->24	
20	2	24	152	0	0	0	6	20->27		
21	7	25	145	2	0	0	0	21->22		
22	2	32	152	3	0	0	0	22->23		
23	3	34	154	0	9	0	0	23->29		
24	3	25	154	4	0	0	0	24->29		
25	7	18	150	0	0	4	0	25->30		
26	8	14	146	0	0	0	7	26->27		
27	3	26	154	0	8	0	0	27->30		
28	7	17	152	0	7	0	0	28->31		
29	2	37	157	0	7	0	0	29->31		
30	2	29	157	0	0	2	0	30->31		
31	1	159	159	0	0	0	0			

A 7. ábrán a J30-1-1 példa egy elsőbbségi és erőforráskorlátokat kielégítő ütemezését láthatjuk, véletlenszerűen generált pénzmozgásokkal:

$$C_i = \textbf{\textit{RandomUniform}}(-50,100),\ i \in \{1,2,...,N\}$$

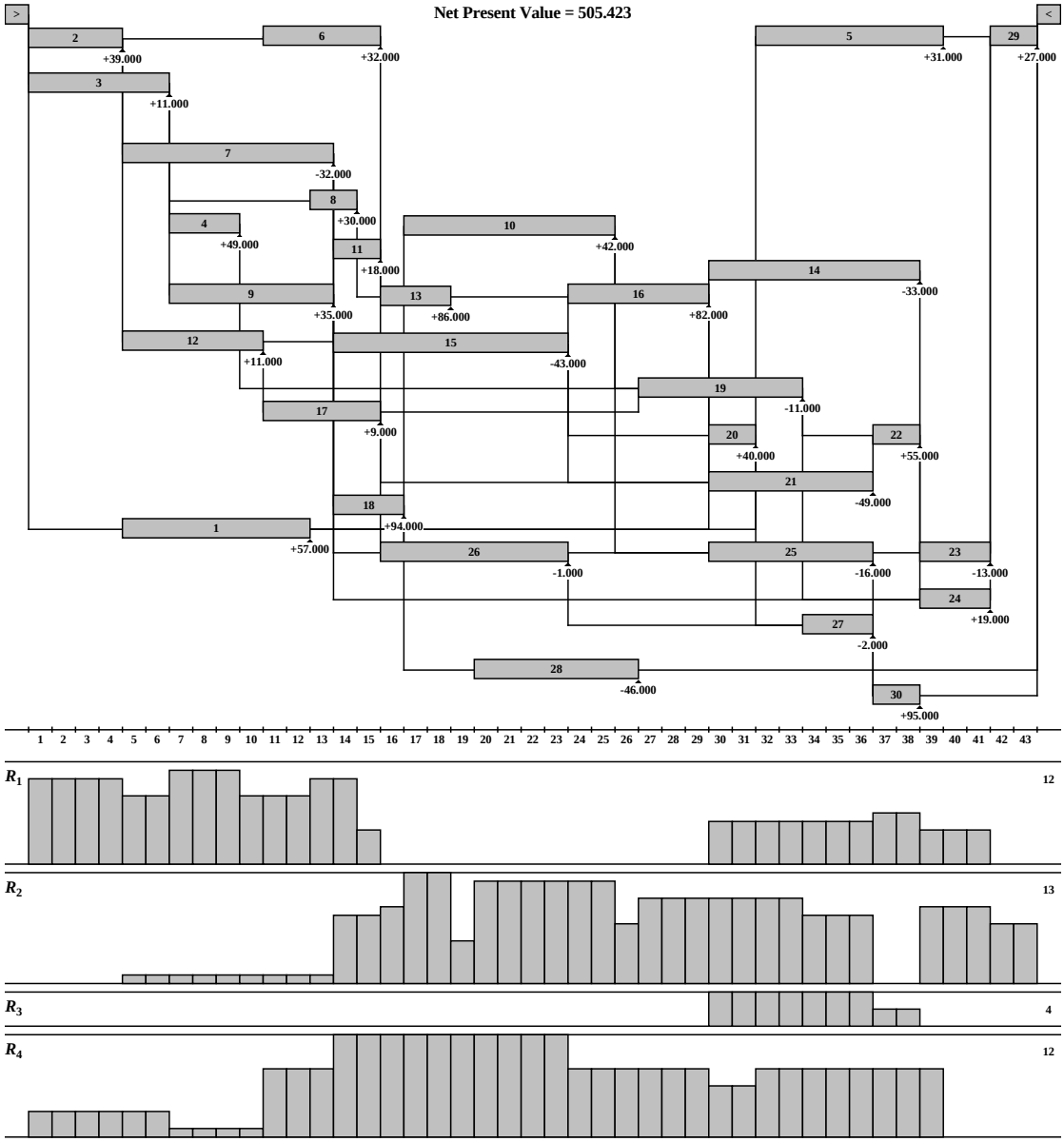
A tevékenységeket téglalapokkal, az elsőbbségi feltételeket vonalakkal ábrázoljuk. Az egységnyi áltevékenységeket a $>$ $(<)$ szimbólummal jelöltük.

7. ábra: A J30-1-1 példa egy erőforráskorlátos ütemezése



A 8. ábrán a „legjobb” ütemezést mutatjuk be, ahol a „legjobb” azt az időtartam minimalizáló ($\overline{T} = 44$), erőforráskorlátos ütemezést jelenti, melyre a generált pénzmozgásokból számolt nettó jelenérték (NPV) mérték maximális ($NPV^* = 508.0323$). Az ütemezés annak az algoritmusnak az eredménye, melyet a 4.2 fejezetben fogunk ismertetni.

8. ábra: A J30-1-1 példa "legjobb" ütemezése



4.2. Az algoritmus

Ebben a fejezetben hibrid algoritmusunkat fogjuk ismertetni. Először a modellnek megfelelő zenei analógiát részletezzük. Részletesen tárgyaljuk az algoritmus kulcsfontosságú pontjait: az improvizálást követő részt, a rejtett erőforrás felhasználási konfliktusok javítását, a teljesen unimoduláris mátrixok alkalmazását. Az algoritmus könnyebb megértése céljából végigkövetünk egy egyszerű példát: egy kisebb projektre alkalmazzuk algoritmusunkat, részletesen bemutatva a lépéseket. Legvégül eljárásunk pszeudókódját ismertetjük.

4.2.1. A modellnek megfelelő harmónia keresési analógia

Megoldásunk során az eredeti – 3.5 pontban ismertetett – harmónia kereső eljárás erőforrás korlátos esetre átdolgozott változatát – a Sound of Silence algoritmust - fejlesztjük tovább, tesszük alkalmassá másodlagos kritérium kezelésére.

Az eredeti HS (IHS) algoritmus explicit, mivel direkt dolgozik a hangokkal. A cikkbeli algoritmus – mint „elődje”, a Csébfalvi-féle Sounds of Silence (SoS) algoritmus is - azonban implicit kezeli a hangokat, és a probléma megoldásához „karmester” szükséges. Az eredeti HS algoritmusban a zenészek maximális szabadságot kapnak, az „improvizáció” is távol áll a valós életbelitől. Míg a HS-beli improvizáció véletlenszerűen választott hangok véletlenszerű módosítása, addig az SoS-ben és így a disszertáció-beli algoritmusban is az improvizáció egy - a karmester által választott - többé-kevésbé harmonikus dallam módosítása. Ezen felül mindkét algoritmusban minden döntésért a karmester a felelős, bevezetése nélkül nem tudjuk hatékonyan

menedzselni a problémát, a zenészek ilyenformán csupán egy „döntés támogató rendszert” alkotnak.

A zene világában az erőforrás profil egy „polifonikus dallam” formájában jelenik meg. Így – feltételezve, hogy minden szólamban csak a „magas” hangok hallhatóak – az erőforrás korlátos projekt hossz minimalizáló probléma a következőképpen fogalmazható meg: egymást követő improvizációk során keressük a legrövidebb „Sounds of Silence” dallamot, azaz a „leghosszabb csendet”. Természetesen a zenei „magas” hang megfelelője a projekt ütemezésben az erőforráskorlát átlépése.

A zenei analógia nyelvén fogalmazva az RCPSP probléma a következőképpen fogalmazható meg:

- a zenekar N zenészből áll;
- a polifonikus hangzás N hangból (sound) áll és R szólamból (phrase);
- minden $i = 1,2,...,N$ zenész pontosan egy polifonikus hangért felel;
- minden $i = 1,2,...,N$ polifonikus hangot a következő halmazzal tudunk jellemezni: $\{X_i, D_i, \{R_{i,r} \mid r = 1,2,...R\}\}$;
- minden $r = 1,2,...,R$ szólamban a hangok a párhuzamos hangokhoz adódnak hozzá;
- minden $r = 1,2,...,R$ szólamban csak a „magas” hangok hallatszanak, melyekre $\{U_{t,r} \mid U_{t,r} > R_r, t = 1,2,...,T\}$;
- Az improvizáció során minden $i = 1,2,...,N$ zenész egy $IP_i \in [-1,1]$ értéket (idea about position) ad meg „elképzelésként”, mellyel annak a szándéknak az erősségét jelzi, mellyel szeretne belépni a melódiába. Ha az érték 1, akkor minél

hamarabb, ha -1, akkor minél később szeretne belépni, ha 0, akkor tanácstalan a belépési szándékát illetően.

- A repertoár feltöltő fázisban minden zenész szabadon improvizál, $IP_i \leftarrow RandomGauss(0{,}1{,}-1{,}1)$, ahol a $\eta \leftarrow RandomGauss(\mu, \sigma, \text{MinX}, \text{MaxX})$ függvénnyel generálunk véletlen számokat, a $\text{MinX} \le \eta \le \text{MaxX}$ halmazon ($MinX = -1; MaxX = 1$), normális eloszlással, μ várható értékkel, és σ szórással.
- Az improvizációs fázisban a Mahdavi-féle improved harmónia kereső eljárásnak megfelelően az „elképzelések" szabadsága lépésről lépésre csökken, $IP_i \leftarrow \text{RandomGauss}(IP_i, \sigma, MinX, MaxX)$, ahol a σ szórás folyamatosan csökken.
- Az improvizációs eljárás során az összes döntés a karmester felelőssége, ideértve a dallam választását, és a zenészek elképzeléseinek összehangolását is.
- Az eljárás során a zenészek a legrövidebb „Sounds of Silence" dallamot, azaz a legrövidebb csendet keresik improvizációkon keresztül.

4.2.2. Az improvizálást követő lépések

Az eljárás során a karmester egy lineáris programozási problémát (LP) old meg, amikor összehangolja a zenészek többé-kevésbé ellentétes elképzeléseit a legrövidebb „Sounds of Silence" dallam felől. Ezt az LP problémát a következőképpen fogalmazhatjuk meg:

$$\boldsymbol{min}\left[\sum_{i=1}^{N} IP_i * X_i\right] \tag{18}$$

$$X_i + D_i \leq X_j, i \rightarrow j \in \boldsymbol{PS}\ , \tag{19}$$

$$\underline{X}_i \leq X_i \leq \overline{X}_i,\ i \in \{1,2,\ldots,N\} \tag{20}$$

Az optimalizáció eredménye egy ütemezés (dallam), melyet karmester arra használ, hogy hangok végső sorrendjét (vagy a zenészek belépésének a végső sorrendjét) meghatározza. A karmester egy „hangnélküli" dallamot generál, egyenként véve a tevékenységeket az adott sorrendben, és ütemezve őket a legkorábbi kezdési időpontra, amit az elsőbbségi és erőforráskorlátok megengednek.

Ezután, felhasználva a forward-backward javító (FBI) eljárást (Tormos és Lova [2001]), a karmester megpróbálja növelni a generált dallam minőségét. A szerző korábbi algoritmusában (Láng [2009b]) az eljárás ezen pontján egy egyszerű forward-backward ütemezés szerepelt. Ezt cseréltük le az FBI-ra, amely egy jóval robosztusabb eszköz, a korábbiaknál sokkal jobb futási eredményeket ad.

Az FBI után a karmester a dallamban szereplő rejtett konfliktusokat oldja fel.

4.2.3. A rejtett erőforrás felhasználási konfliktusok feloldása

A „Sounds of Silence" algoritmus konfliktusjavító változata a tiltott halmazok elvén alapul. Ebben a fejezetben azt a módosítást ismertetjük – mely algoritmusunk egyik leglényegesebb újítása – amely lehetővé teszi az SoS alkalmazását másodlagos szempont megléte mellett is.

Tiltott halmazról beszélünk, ha

- minden halmazbeli tevékenységet végrehajthatunk párhuzamosan, párhuzamos végrehajtásuk nem sérti a tevékenységek között fennálló elsőbbségi feltételeket;

- van azonban olyan erőforrás, melyből kevés van ahhoz, hogy minden tevékenységet végrehajtsunk a halmazból párhuzamosan;
- a halmaz nem tartalmaz valódi tiltott halmaz részhalmazt.

Az erőforrás konfliktus explicit javítható, ha elsőbbségi feltételeknek nem ellentmondó elsőbbségi relációkat illesztünk két tiltott halmaz-beli elem közé, így garantálható, hogy ne kerüljön végrehajtásra minden tiltott halmazbeli tevékenység párhuzamosan, és így nem sérülnek az erőforráskorlátok. Egy beillesztett explicit konfliktus javító reláció mellékhatásként javíthat egy vagy több más konfliktust, ugyanabban az időben.

A konfliktus javító változatban az elsődleges változók a konfliktus javító relációk, a megoldás egy projekt időtartam minimalizáló, és egyben erőforráskorlátokat kielégítő megoldás halmaz lesz, melyben minden elsőbbségi feltétel szerint mozgatható tevékenység eltolható anélkül hogy az erőforráskorlátokat megsértenénk. A hagyományos „idő alapú” modellekben (lásd például Pritsker és ts. [1969]) az erőforráskorlátokat kielégítő ütemezéseket a tevékenységek kezdési idejével jellemezzük. Ebben a modellben az explicit erőforráskorlát kezelésnek megfelelően az elsőbbségi feltételek szerint eltolható tevékenységek mozgatása sértheti az erőforráskorlátokat, azaz előfordulhat olyan eset, amikor egy tevékenység eltolásának eredményeként az ütemezés már nem elégíti ki az erőforráskorlátokat. A konfliktus javító modell időtartam minimalizáló megoldásai védettek a tevékenységek mozgatása ellen, így bevezethetünk egy nem szükségszerűen reguláris másodlagos teljesítmény mértéket, hogy a generált megoldás halmazból a „legjobb” időtartam minimalizáló erőforráskorlátos megoldást válasszuk.

Jól ismert tény, hogy a konfliktus javító modell döntő jelentőségű pontja a tiltott halmazok számítása annak igen nagy számítási időigénye miatt, ezért lényeges, hogy csak akkor alkalmazzuk, ha feltétlenül szükséges.

A „Sound of Silence” algoritmus alkalmazása során a karmester először egy egyszerű, bár igen hatékony és gyors hüvelykujj szabályt alkalmaz, hogy a tiltott halmazok számításának az időigényét csökkentse. A forward-backward javító eljárás után a karmester - explicit tiltott halmaz számítás nélkül - $i \rightarrow j$ elsőbbségi relációt illeszt egy már ütemezett i tevékenység és egy éppen akkor ütemezendő j tevékenység közé, oly módon, hogy a kettő között ne legyen tartalékidő ($X_i + D_j = X_j$). Ezt azokban az esetekben teszi, amikor egyébként sérülnének az erőforrás korlátok. Az eljárás eredménye egy látható konfliktus nélküli ütemezés lesz.

Ebben az ütemezésben azonban még előfordulhatnak rejtett konfliktusok, azaz előfordulhat, hogy egy ütemezésben nincs látható konfliktus, de van olyan tevékenység, melyet bár mozgathatnánk az elsőbbségi feltételek megsértése nélkül, de ez a mozgatás megsérti az erőforrás korlátokat. Célunk az, hogy az ütemezésekben minden elsőbbségi feltételt kielégítő tevékenység mozgatás egyben az erőforrás korlátokat is elégítse ki.

A látható konfliktusok javítása után a karmester pontosan egy lépésben javítja az összes rejtett konfliktust. Ehhez először számítja az összes tiltott halmazt, és kiszámítja minden tiltott halmaz esetén az összes lehetséges javító relációt is.

Ezután megkeresi minden tiltott halmaz esetén a „legjobb” konfliktus javító relációt. Ebben az esetben a „legjobb” egy olyan $i \rightarrow j$ relációt jelent két tiltott halmaz-beli elem között, melyekre az ($X_j - X_i - D_i$) tartalékidő maximális. Azért a maximális tartalékidejűt választja, hogy olyan ütemezést kapjon, amelyben a tartalékidők összege a lehető legnagyobb, így ezt követően minél jobban mozgathatjuk a tevékenységeket,

azaz nagyobb valószínűséggel kapunk a másodlagos szempontnak minél inkább megfelelő ütemezést

Tehát az eljárás ezen pontján a karmester az összes látható és az összes rejtett konfliktust kijavította az ütemezésben.

4.2.4. A teljes unimodularitásból adódó egyszerűsítés lényege

Ebben a fejezetben a konfliktus javító modell alkalmazása melletti másik algoritmusbeli újításunkat, a teljesen unimoduláris halmazoknak a nettó jelenérték maximalizálás esetére való alkalmazását részletezzük, mellyel a probléma exponenciális megoldási idejét fogjuk lényegesen csökkenteni.

Hibrid eljárásunk egy pontján szükségünk lesz egy adott ütemezéshalmaz minden ütemezéséhez tartozó nettó jelenérték kiszámolására. Ezt a számítást a 4.2.6 pontban ismertetett pszeudokódban szereplő *NPV*( ) függvényünk fogja megvalósítani.

Tekintsük a szokásos formában leírt elsőbbségi feltételeket:

$X_i + D_i \leq X_j$, $i \rightarrow j \in \boldsymbol{PS}$

Első megközelítésben változóink a tevékenységek kezdési idői lennének. Amennyiben a maximalizálandó nettó jelenérték függvényt a kezdési idők függvényében írnánk fel, nemlineáris célfüggvényhez, így NP-nehéz feladathoz jutnánk.

$X_{it} \in \{0,1\}$ bináris indikátor változók bevezetésével azonban a tevékenységeket a változók lineáris kombinációjaként írhatjuk fel: $X_i = \sum_{t \in T_i} X_{it} * t$

Ezeket az indikátor változókat alkalmazva célfüggvényünk felírásához, azt lineáris alakra hozzuk:

$$\boldsymbol{max}\left[NPV = \sum_{i=1}^{N} \sum_{t \in T_i} C_{it} * X_{it} \right] = NPV^*$$

Amennyiben az így kapott (1-10) modellt oldanánk meg, együttható mátrixunk sajnos - pozitív, negatív, és nulla - egész számokból állna, így MILP feladathoz jutnánk. Amennyiben egy egészértékű lineáris programozási feladat kisméretű, egy megfelelő szoftverrel (például LINDO, LINGO) is könnyen, gyorsan megoldható. Azonban ha az ismeretlenek száma nagy – mint a projekt ütemezési problémák esetében általában, így a fenti probléma esetén is – , a megoldás igen nehézkessé válik, elfogadható időn belül nem kapunk megoldást. Egy alkalmas megoldást kell keresnünk, mellyel az exponenciális megoldási időt jelentősen csökkenthetjük.

Relaxáljuk a feladatot, és a bináris változó feltételeket a $0 \le X_{ij} \le 1$ feltételekre cseréljük. Ekkor az eredeti MILP feladat helyett LP feladathoz jutunk. Egy LP feladatot már meg tudnánk oldani polinomiális idő alatt akár például egy szimplex módszert alkalmazva. Azonban amennyiben a megoldás során az Xij változók értékeire 0/1-től eltérő tört értékeket kapnánk, természetesen értelmezhetetlen lenne az eredmény. Szerencsére ez nem fordulhat elő:

Helyettesítsük a szokásos formában leírt elsőbbségi feltételeket az alábbi (12) formulával, így kapjuk a (11 - 17) új egyenletrendszerrel leírt modellt.

Megjegyezzük, hogy az erőforrás korlátok figyelmen kívül hagyhatóak az eljárás ezen pontján, mivel konfliktusjavító relációk beillesztésével mind a látható, mind a rejtett konfliktusokat feloldottuk.

$$\sum_{p=T_i}^{\overline{X}_i} X_{ip} + \sum_{q=\underline{X}_p}^{T_i+D_i-1} X_{jq} \le 1,\ T_i \in \{\underline{X}_i, \underline{X}_i+1, \ldots, \overline{X}_i\},\ i \to j \in \boldsymbol{PS} \qquad (12)$$

Ekkor az új együttható mátrixunk teljesen unimoduláris lesz. Ennek az állításnak a bizonyítását Cavalcante és ts. [2001] tanulmányban találjuk.

Ezután felhasználjuk azt, hogy kapcsolat mutatható ki a teljesen unimoduláris mátrixok, és az egészértékű lineáris programozási feladatok között. Felhasználhatjuk Schrivjer [1987] tételét:

Legyen A teljesen unimoduláris mátrix, és b egész vektor.

Ekkor

$P := \{x \mid Ax <= b\}$, P poliéder egész.

Felhasználva mátrixunk unimodularitását, a tételből következik, hogy a (11-17) egyenletrendszerrel leírt feladatot relaxálva, majd LP feladatként megoldva, egész, azaz esetünkben bináris megoldáshoz jutunk.

Ha a lineáris programozási probléma megoldására a jelenleg élvonalat képviselő belső pontos módszerek valamelyikét használjuk (például Mészáros [1996] – féle BPMPD), akkor még egy nagyméretű probléma is „szinte azonnal”, a hagyományos szimplex módszerhez képest nagyságrendekkel gyorsabban megoldható. A megoldás során kihasználtuk hogy a célfüggvény minden komponensében monoton, azaz ha egy mozgatható tevékenységet egy adott irányba mozgatunk – a többi tevékenységet rögzítettnek tekintve - , a célfüggvény értéke monoton nő, vagy csökken, ebből következően az optimális megoldás egészként fog adódni.

4.2.5. Egy ütemezési feladat bemutatása

Tekintsünk egy ütemezési feladatot, mellyel szemléltethetjük eljárásunk lényegét. A 9 ábrán egy kisméretű projektet láthatunk, mely 10 tevékenységből áll, két erőforrás típust igényel. Keressük a legrövidebb erőforráskorlátos ütemezések halmazán a maximális NPV-vel rendelkező ütemezést.

A példa legfontosabb paramétereit a következő táblázatokban foglaljuk össze:

Táblázat 5 – A példa paraméterei

Tevékenységek száma	10
Erőforrások száma	2
Legkorábbi ütemezés hossza:	16
Tevékenységek hosszának összege:	32

Táblázat 6 – A példa paraméterei

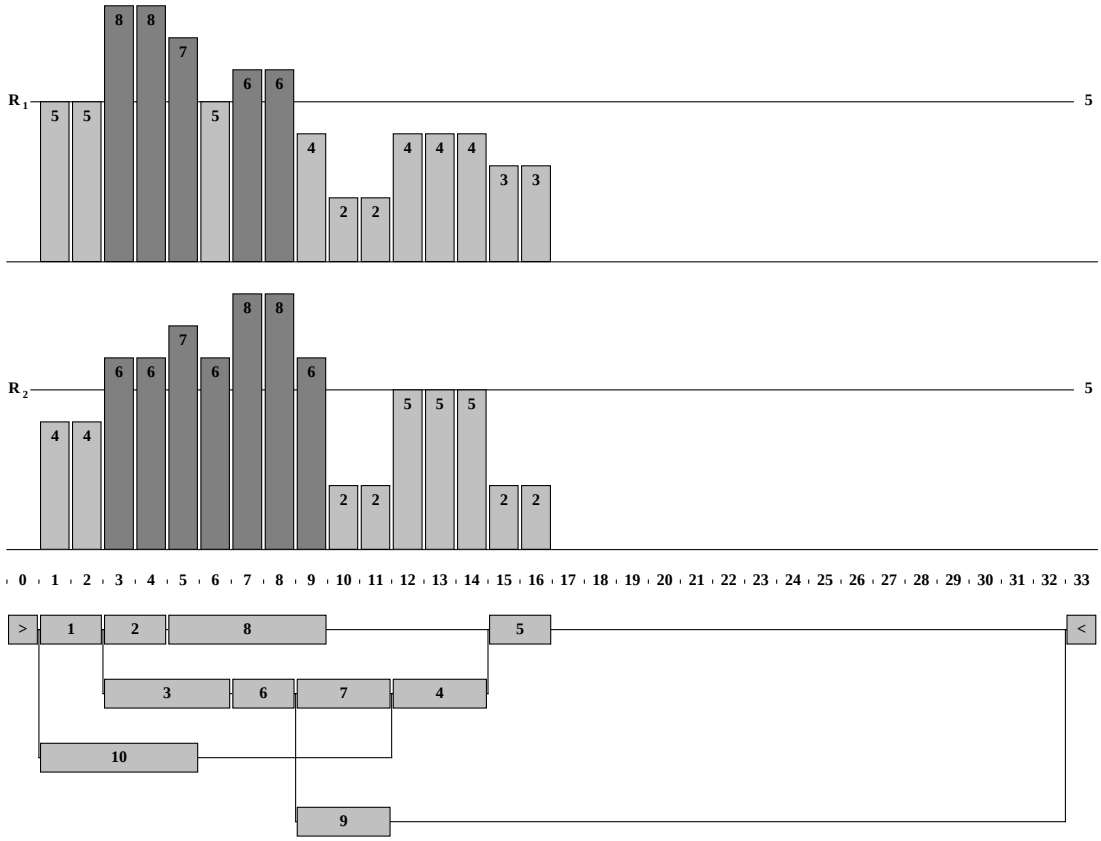
Erőforrás korlátok	
Erőforrás száma	**maximuma**
R_1	5
R_2	5

Táblázat 7 – A példa paraméterei

Tevékenység		**Tevékenységek erőforrás igénye**			
sorszám	**hossz**	**R_1**	**R_2**	**Elsőbbségi feltételek**	
0	1			0->1	0->10
1	2	3	3	1->2	1->3
2	2	3	3	2->8	
3	4	3	2	3->6	
4	3	4	5	4->5	
5	2	3	2		
6	2	4	4	6->7	6->9
7	3	1	1	7->4	
8	5	2	4	8->5	
9	3	1	1		
10	5	2	1	10->4	

Az algoritmus első lépéseként megállapítjuk, hogy a legkorábbi CPM ütemezés nem erőforráskorlátos, így továbbmegyünk. Amennyiben erőforráskorlátos lett volna, akkor egyszerűen kiszámoltuk volna a nettó jelenértékét, és így állt volna le az algoritmus.

9. ábra: Nem erőforráskorlátos kiindulási ütemezés



Tekintsük tehát az elsőbbségi feltételeket kielégítő ütemezést a 9. ábrán. Az ütemezést vizsgálva két problémát fogalmazhatunk meg:

1. Az erőforráskorlátok miatt bizonyos tevékenységeket nem ütemezhetjük a legkorábbi elsőbbségi feltételeket kielégítő kezdőidejükre, csak későbbre. A kapott ütemezés nem erőforráskorlátos.

2. Van egy másodlagos szempontunk, a NPV maximalizálás. Így a hagyományos idő alapú modellt nem alkalmazhatjuk. Abban a modellben az elsődleges változók a tevékenységek kezdési idői. A Csébfalvi-féle SoS algoritmus erőforrás konfliktus javító verzióját fogjuk használni, konfliktusjavító relációkat alkalmazva.

Az eljárás kezdetekor - az első generáció létrehozásakor - karmester összegyűjti az elképzeléseket. Teljesen véletlenszerű megoldásokkal tölti fel az *IP* halmazt, a repertoárt, az első generációt.

$$I(i) = RandomGauss(0, MaxS\tan dardDeviation, -1,1)$$

Legyenek a zenészek így kapott elképzelései a következők:

$$\mathrm{IP} = \{\, 0.359,\ 0.057,\ 0.136,\ -0.368,\ -0.346,\ 0.489,\ -0.961,\ 0.462,\ 0.567,\ 0.366 \,\}$$

Ekkor a fenti (18) minimalizálandó képlet a következő lesz:

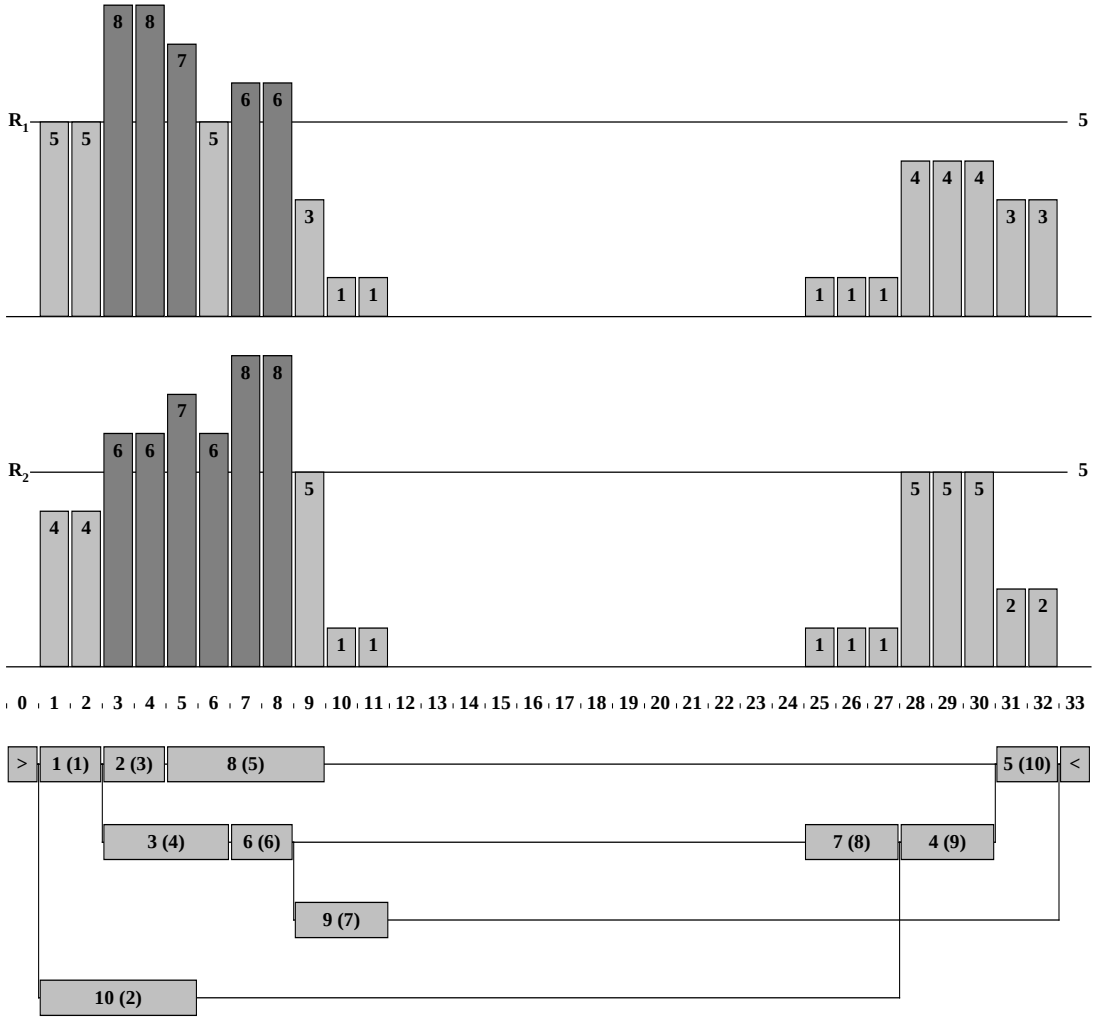
$$0.359X_1 - 0.057X_2 + 0.136X_3 - 0.368X_4 - 0.346X_5 + 0.489X_6 - 0.961X_7 + 0.462X_8 + 0.567X_9 + 0.366X_{10}$$

A későbbiekben a következő generációk létrehozásakor nem a fenti véletlenszerű feltöltést fogja alkalmazni a karmester, hanem levezényeli a zenekar **improvizáció**ját. Az improvizáció során a harmónia kereső eljárásnak megfelelően először egy bizonyos valószínűséggel minden zenész választ egy dallamot a repertoárból, minél rövidebb, annál nagyobb valószínűséggel. Egy bizonyos valószínűséggel módosítja az értékeket, egyébként változatlanul hagyja. A másik lehetőség a HS-nek megfelelően egy bizonyos valószínűséggel a véletlenszerű hang generálás. Az improvizációkkal létre jöttek az új értékek. Ezekben az esetekben az így kapott „elképzelések” alapján hozza létre a karmester a minimalizálandó képletet.

Következő lépésben a karmester megoldja a fenti ILP problémát LP problémaként, alkalmazva a belső pontos módszerek valamelyikét (például Mészáros [1996] – féle BPMPD), a **tevékenység lista generátor eljárás**ban. Ennek az optimalizációnak az eredménye rendszerint gyakorlati szemszögből nézve érdektelen, az analógiát tekintve azonban érdekes ütemezés. A fenti példa esetében például a 10. ábrán figyelhetjük meg az eredményt, az ábrán egy extrém hosszú szünet van a két tevékenység-csoport között,

megfelelően a zenészek minél korábbi és minél későbbi belépési szándékának. A téglalapokon szereplő első szám a tevékenység száma, zárójelben az eredményképpen kapott fontossági sorrendben betöltött helye. Ezt a sorrendet úgy kaptuk, hogy vettük rendre a tevékenységeket kezdési idejük sorrendjében. Abban az esetben, amikor két tevékenységnek ugyanaz a kezdőideje, véletlenszerűen állapítja meg a sorrendet. A ábrán az erőforráskorlátot megsértő tevékenységek sötétebb színűek.

10. ábra A tevékenység lista generátor eljárás egy lehetséges eredménye



Azonban a karmester ezt az ütemezést csak arra használja, hogy a zenészek, azaz a hangok végső sorrendjét határozza meg.

A továbbiakban a tevékenység sorrendhez egy megfelelő ütemezést próbál meg a karmester társítani, egyenként véve a tevékenység listáról a tevékenységeket, és ütemezve őket a legkorábbi időpontra, melyet az erőforrás korlátok szerint választhat.

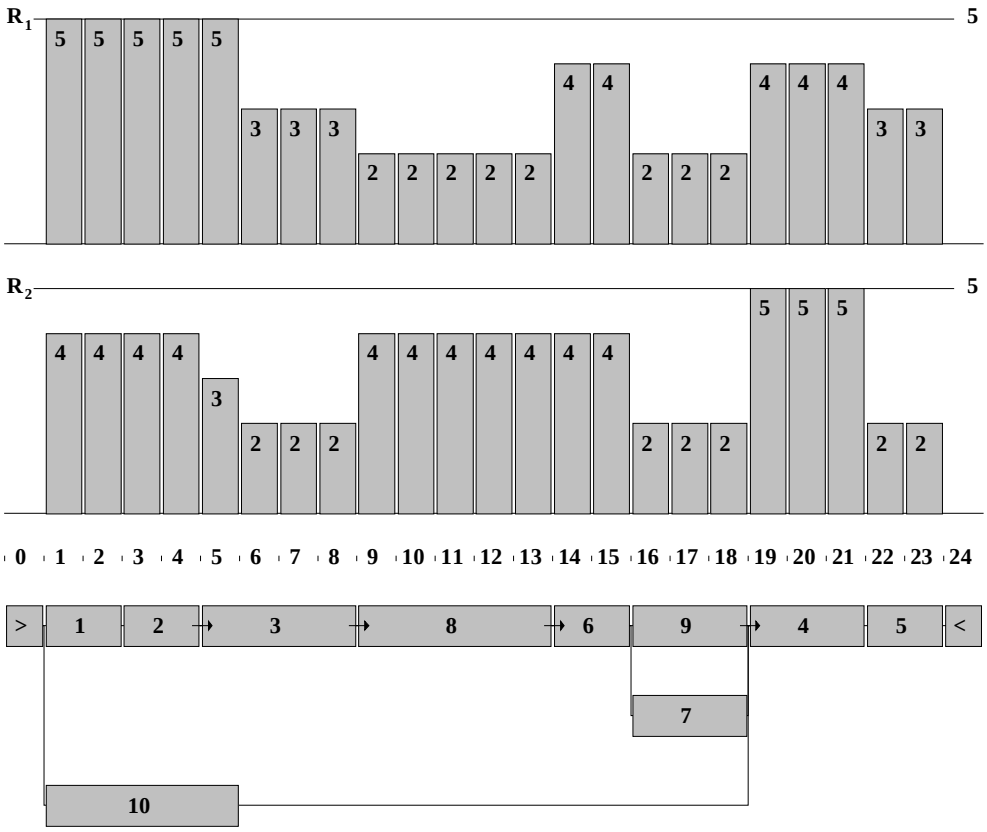
Ezután, felhasználva a Tormos-Lova-féle **forward-backward improvement** (FBI) eljárást a karmester megpróbálja növelni a generált dallam minőségét.

Ezt követően a karmester alkalmazza a korábban említett „ökölszabályt”, először javítja a látható konfliktusokat.

Esetünkben ezek a javító relációk: {2→3, 3→8, 8→6, 9→4}. Ezek a tevékenységpárok a javító relációk alkalmazása nélkül párhuzamosan ütemezve rendre a R_1, R_2, R_1 és R_2, illetve az R_2 erőforráskorlátokat sértenék.

A 11. ábrán a fenti lépések eredményeként kapott ütemezés látható.

11. ábra Az ütemezési sorrendet meghatározó eljárás egy lehetséges eredménye

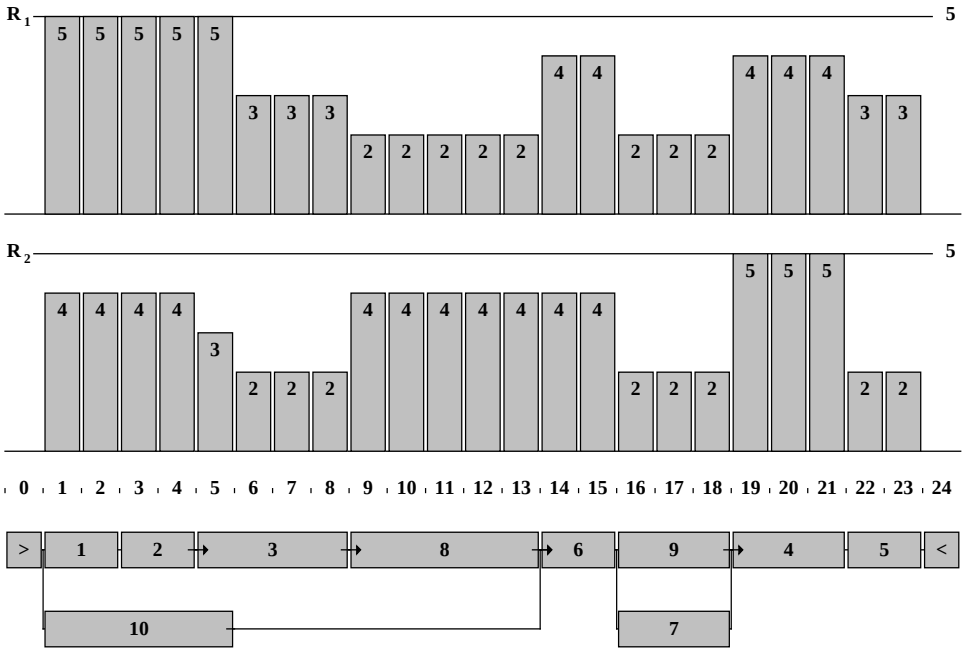


Az eljárás eredménye egy aktív ütemezés lesz, látható konfliktus nélkül, azonban előfordulhatnak rejtett konfliktusok, melyeket nem engedhetünk meg a másodlagos szempont léte miatt, mivel így nem tudnánk mozgatni az elsőbbségi feltételek szerint mozgatható tevékenységeket. Figyeljük meg a fenti ábrán a 10. és a 6. tevékenységet. Az elsőbbségi feltételeket tekintve ütemezhetnénk őket párhuzamosan, de ez esetben megsértenénk az erőforráskorlátokat. A rejtett erőforrás felhasználási konfliktusok megoldása a karmester következő feladata.

A továbbiakban a karmester a korábban ismertetett módon minden tiltott halmaz esetében választja a „legjobb” , azaz maximális tartalékidejű konfliktusjavító relációt.

A 12. ábrán kapott ütemezésben látható, hogy a 10 → 6 javító relációt beillesztve már egy olyan ütemezést kaptunk, melyben sem látható, sem rejtett konfliktus nem fordul elő.

12. ábra A rejtett erőforrás felhasználási konfliktusok javítása után



A zenei analógia nyelvén fogalmazva, a konfliktus javító eljárás eredménye egy robosztus, rugalmas „Sounds of Silence” dallam, melybe a zenészek némi szabadsággal léphetnek be anélkül, hogy a kompozíció esztétikai értékét csökkentenék. Természetesen, ha bevezetünk egy másodlagos szempontot, melynek az esztétikai értéke a kezdőidők függvénye, ez a szabadság teljesen eltűnik.

Az eljárás ezen pontján a populáció elemei olyan ütemezések, melyekben minden látható és rejtett konfliktust kijavítottunk explicit vagy implicit relációk beépítésével. A mozgatható tevékenységek különböző helyzeteit tekintve ütemezéshalmazokat kapunk, melyek minden elemére számítjuk a nettó jelenértéket, és kiválasztjuk a legkedvezőbb ütemezést, azaz a legnagyobb nettó jelenértékűt.

A továbbiakban a harmónia kereső alapgondolatnak megfelelően mindig azt az ütemezést cseréljük az eljárás során, melynél van jobb, azaz rövidebb ütemezés, vagy ha a populáció-beli legrosszabb vele egyező időtartamú, akkor azt választjuk, amelynek az NPV-je jobb. Így tehát az eljárás alkalmazása során a populáció minősége lépésről lépésre javul, mert egyre több benne a rövid erőforráskorlátos ütemezés. Minél több ilyen van, annál nagyobb eséllyel kapunk a másodlagos feltételnek megfelelő jó megoldást. Az eljárás eredményeként a repertoár közel optimális ütemezésekkel lesz végül tele.

4.2.6. Az algoritmus pszeudokódja

Ebben a fejezetben algoritmusunk leglényegesebb részeinek pszeudokódját ismertetjük. A kódban a halmazok nevét, a függvényneveket, az eljárások nevét vastagon szedtük.

Az ajánlott algoritmus két globális, beállítható paramétert használ:

- `PopulationSize`, a populáció mérete,
- `Generations`, a generációk száma,

és három globális beállítható paraméter párt:

- {*MinHMCR, MaxHMCR*}, ezen értékek között változik a HMCR paraméter értéke,
- {*MinPAR,MaxPAR*}, ezen értékek között változik a PAR paraméter értéke,
- {*MinDeviation, MaxDeviation*} ezen értékek között változik a *StandardDeviation* paraméter értéke,

T*-al fogjuk jelölni a mindenkori „legjobb” ütemezést. „Legjobb” ütemezésen a mindenkori aktuális ütemezéshalmazból a minimális időtartamút értjük, a minimális időtartamú ütemezések részhalmazán belül pedig a maximális nettó jelenértékű ütemezést. Ennek az ütemezésnek a nettó jelenértékét NPV^*-al, magát az ütemezést S^*-al jelöljük.

Tekintsük a kódban szereplő függvényeket sorban.

A kódban először szereplő függvény, a véletlenszám generátor: ***RandomGauss(***$\mu, \sigma, MinX, MaxX)$. A függvény egy (X) véletlenszám értéket generál a ($MinX \leq X \leq MaxX$) halmazon, (μ) középértékkel és (σ) szórással. A függvényt a repertoár kezdeti feltöltésekor és az improvizációs eljárásban is alkalmazni fogjuk. Algoritmusunkban a ***RandomGauss*** függvény bemeneti értékei a következőképpen alakulnak:

Táblázat 8

A ***RandomGauss*** függvény paraméter értékei

μ	***σ***	***MinX***	***MaxX***
0	*MaxDeviation*	-1	1

Az algoritmus központi eleme a ***ActivityListGeneration(I)*** függvény, mely az (***I***) elképzelések halmazából egy (***L***) tevékenység listát generál. A relaxált egész lineáris programot a BPMPD-vel (Mészáros[1996]) oldjuk meg.

A ***ListScheduling(L)*** eljárás sorban veszi az (***L***) tevékenység listáról a tevékenységeket, és ütemezi azokat a legkorábbi kezdési időpontra, amit az elsőbbségi és erőforráskorlátok megengednek. Így egy látható konfliktus nélküli ütemezést ($\{T_1, \mathbf{S}_1\}$) hoz létre, melyben természetesen rejtett erőforrás felhasználási konfliktusok még lehetnek.

Ezután alkalmazzuk a forward-backward improvement (FBI) eljárást (Tormos és Lova [2001]), mellyel a karmester megpróbálja növelni a generált dallam minőségét. Kimenete egy – a korábbinál remélhetőleg jobb minőségű – ütemezés ($\{T_2, \mathbf{S}_2\}$).

A továbbiakban a ***ForbiddenConflictRepairing*** $\{T_2, \mathbf{S}_2\}$ függvény megelőző-követő relációt helyez el az összes olyan tevékenységpár közé, melyek az FBI eljárás után időben közvetlenül egymás után ütemeződtek, eredeti megelőző-követő reláció azonban nincs közöttük. Ez után az eljárás tiltott halmazok számításával egy lépésben javítja az összes rejtett konfliktust. Kimenete egy ütemezés ($\{T_3, \mathbf{S}_3\}$), melyben sem látható, sem rejtett erőforrás felhasználási konfliktusok nincsenek. Megjegyzendő, hogy $T_2 = T_3$, az összetartozó bemeneti és kimeneti ütemezések csak a tevékenységek kezdési időpontjaiban tér(het)nek el.

Konfliktusjavító eljárásunk jelenlegi formájában egy lépésben kezeli az összes rejtett konfliktus javítását. Mivel egy javító reláció beillesztése indirekt javíthat egy másik konfliktust is, előfordulhat, hogy az eljárás felesleges javító relációkat is beilleszt, így a jövőben érdemes az algoritmus ezen pontján megkísérelni hatékonyságot javítani.

Az ***NPV*** függvény bemenete ez a konfliktus nélküli ütemezés. Az ütemezésben az összes eltolható tevékenységet eltolva az összes lehetséges módon, ennek az egyetlen ütemezésnek egy ütemezéshalmaz fog megfelelni. Az ütemezéshalmaz minden elemére számolva a nettó jelenértéket, kiválasztjuk a maximális nettó jelenértékűt, ez lesz a függvény kimenete (*{T,NPV,**S**}*).

A ***RandomMelodySelection*** függvény egy véletlen ütemezést választ a repertoárból, oly módon, hogy minél kisebb az ütemezés időtartama, annál nagyobb valószínűséggel választja.

A ***Improvisation(I,G)*** függvény a javított harmónia keresésnek (Mahdavi [2007]) megfelelően a bemeneti elképzelés halmazból egy „improvizált” kimeneti elképzelés halmazt ad vissza. A *G* index – az aktuális generációszám – jelzi az algoritmusunk előrehaladtát. Ettől az indextől fog paramétereink nagyságának alakulása függni az eljárás során. A 13. ábrán találjuk az improvizáció részletesebb pszeudokódját.

A ***WorstCaseSelection*** függvény kiválasztja a „legrosszabb” ütemezést az addigi repertoárból. „Legrosszabb” ütemezésen a legnagyobb időtartamú ütemezést értjük, és azon belül – amennyiben több ilyen ütemezés van – a legkisebb nettó jelenértékűt.

A ***WorstCaseSelection*** függvény meghívása után vizsgáljuk az új ütemezést. Ha jobb, mint az addigi legrosszabb, akkor az új ütemezést beemeljük a repertoárba, a régit eldobjuk a repertoárból. „Jobb” alatt azt értjük, hogy vagy rövidebb, mint az addigi legrosszabb, vagy azonos időtartamú, mint az addigi legrosszabb, azonban a nettó jelenértéke nagyobb. Így a repertoár minősége lépésről lépésre javul.

13. ábra A hibrid algoritmus pszeudokódja

```
T* ← T̄
/* T*-al jelöljük a mindenkori „legjobb” ütemezést */
For G = 1 to Generations
      If G = 1 Then
      /* az első generációt véletlen elemekkel töltjük fel */
            For Member = 1 to PopulationSize
                  For i = 1 to N
                        I(i) ← RandomGauss(0,MaxDeviation,-1,+1)
                  Next i
                  L ← ActivityListGeneration(I)
                  /* az elképzelésekből a karmester egy tevékenység
                  sorrendet ad, BPMPD-t hív meg */
                  {T1,S1} ← ListScheduling(L)
                  /* A tevékenység sorrendből egy erőforráskorlátos
                  ütemezést hoz létre */
                  {T2,S2} ← FBI {T1,S1}
                  /* alkalmazza az FBI-t*/
                  {T3,S3} ← ForbiddenConflictRepairing {T2,S2}
                  /* egy lépésben javítja a rejtett konfliktusokat*/
                  {T,NPV,S} ← NPV {T3,S3}
                  If T = T Then Exit With {T*=T, NPV*=NPV, S*=S}
                  /* Ha elértük a minimális hosszú ütemezést, nem
                  érdemes tovább számolni, kiugrunk az eljárásból az
                  addigi „legjobb” ütemezéssel */
                  I(Member) ← I
                  T(Member) ← T
```

```
            S(Member) ← S
            /* az ütemezést behelyezzük a repertoárba*/
            If (T* > T) or (T* = T and NPV* > NPV) Then {T*=T,
            NPV*=NPV ,S*=S}
                  /* ha az új ütemezés időtartama kisebb, mint
                  az addigi legjobb/legrövidebb időtartama, vagy
                  az időtartama egyenlő mint az addigi
                  legjobb/legrövidebb időtartama, de a NPV-je
                  nagyobb, akkor őt nevezzük ki „legjobbnak” */
      Next Member
Else
/* létrehozzuk a többi generációt*/
      For Member = 1 to PopulationSize
            Melody ← RandomMelodySelection
            /* a memóriából választunk dallamot, minél
            rövidebb, annál nagyobb valószínűséggel*/
            I ← I(Melody)
            /* a dallamból elképzelések halmazát képezem*/
            I ← Improvisation(I,G)
            /* az improvizáció lépése, 14. ábrán részletezve */
            L ← ActivityListGeneration(I)
            /* az elképzelésekből a karmester egy tevékenység
            sorrendet ad */
            {T1,S1} ← ListScheduling(L)
            /* A tevékenység sorrendből egy erőforráskorlátos
            ütemezést hoz létre */
            {T2,S2} ← FBI {T1,S1}
            /* alkalmazza az FBI-t*/
            {T3,S3} ← ForbiddenConflictRepairing {T2,S2}
```

```
                /* egy lépésben javítja a rejtett konfliktusokat*/
                {T,NPV,S} ← NPV {T3,S3}
                If T = T Then Exit With {T*=T, NPV*=NPV, S*=S}
                /* Ha elértük a minimális időtartamú ütemezést, nem
                érdemes tovább számolni, kiugrunk az eljárásból az
                addigi „legjobb” ütemezéssel */
                {WorstMember,WorstDuration,WorstNPV}          ←
                WorstCaseSelection
                If (WorstDuration > T ) or (WorstDuration = T and
                WorstNPV < NPV ) Then
                /* amennyiben az új ütemezés jobb mint az addigi
                legrosszabb, akkor a legrosszabbat eldobjuk a
                repertoárból, az újat bevesszük a repertoárba */
                      I(WorstMember) ← I
                      T(WorstMember) ← T
                      S(WorstMember) ← S
                      NPV(WorstMember) ← NPV
                End If
                If (T* > T) or (T* = T and NPV* > NPV) Then {T*=T,
                NPV*=NPV ,S*=S}
                /* ha az új ütemezés időtartama kisebb, mint az
                addigi legnagyobb időtartamú, vagy az időtartama
                egyenlő mint az addigi legnagyobb időtartamúé, de a
                NPV-je nagyobb, akkor őt nevezzük ki „legjobbnak”
                */
          Next Member
      End If
Next G
Exit With {T*=T,NPV*=NPV,S*=S}
```

Az alábbiakban az improvizáció pszeudokódját részletezzük.

Az improvizáció során először beállítjuk a megfelelő paraméter értékeket:

- A HMCR-t, a harmony memory consideration rate-t. Ekkora valószínűséggel fogunk a memóriából hangot választani. A valószínűség az eljárás során fokozatosan nő, mivel a memóriában egyre jobb minőségű dallamokat / ütemezéseket találunk, egyre inkább érdemes onnan választanunk. Zenei analógiával fogalmazva kezdetben a repertoár minősége nem jó, ekkor még nem támaszkodunk rá, hagyjuk kísérletezni a zenészeket, később már nem engedjük őket olyan „szabadjára".
- A PAR-t, a hangmagasság szabályozó rátát. Ekkora valószínűséggel fogunk módosítani a memóriából választott hangon. A módosítás mértéke az eljárás során fokozatosan csökken, kezdetben kevésbé módosítunk, később nagyobb mértékben.
- A `StandardDeviation` értékét. A `StandardDeviation` értéke csökken az eljárás során, így egyre kisebb környezetben módosítunk az értékeken ahogy egyre inkább „közeledünk" az idő előrehaladtával a megoldáshoz.

Mindhárom paraméter érték beállítását a

`Interpol(``StartingX,StartingY,FinishingX,FinishingY,CurrentX`***`)`*** függvény végzi, mely egy egyszerű lineáris interpolációt valósít meg.

Az improvizáció során meghívódik a ***`RandomUniform`*** véletlenszám generátor függvény, mely egy (`X`) véletlenszám értéket generál a (`-1 ≤ X ≤1`) halmazon, és a már ismertetett ***`Interpol(`***`StartingX,StartingY,FinishingX,FinishingY,CurrentX`***`)`*** függvény.

Az improvizáció kimenete egy új elképzelés halmaz, egy új dallam alapja.

14. ábra Az improvizáció pszeudokódja

```
Function Improvisation(I,G)
HMCR = Interpol(2,MinHMCR,Generations,MaxHMCR,G)
/* ekkora valószínűséggel választunk a memóriából*/
PAR = Interpol(2,MinPAR,Generations,MaxPAR,G)
/* ekkora valószínűséggel módosítunk*/
StandardDeviation=Interpol(2,MaxDeviation,Generations,MinDeviation,G)
/* értéke az eljárás során fokozatosan csökken*/
      For musician = 1 to N
            If RandomUniform < HMCR Then
            /* ekkora valószínűséggel választunk a memóriából*/
                  If RandomUniform < PAR Then
                  /* ekkora valószínűséggel módosítunk*/
                        Mean = I_musician
                        I_musician=RandomGauss(Mean,  StandardDeviation,-
                        1,+1)
                        /* egyre kisebb mértékben módosítunk*/
                  End If
            Else
                  I_musician = RandomGauss(0,1,-1,+1)
                  /* véletlen új egyedet generálunk*/
            End If
      Next musician
Return With I
/* egy új elképzelés halmazt generáltunk */
```

4.3. Számítási eredmények

Olyan megoldás, amely a projekt időtartam minimalizálását elsődleges - és a projekt nettó jelenértékét másodlagos szempontnak tekinti, legjobb tudomásunk szerint korábban még nem született, így eljárásunk eredményeit nem tudjuk összehasonlítani korábbi eredményekkel.

Így az egyetlen, amit tehetünk az, hogy algoritmusunk életképességének és hatékonyságának illusztrálására teszthalmaz-beli eseteken futtatott számítási eredményeket adunk. A 4.2 pontban ismertetett megoldást a jól ismert PSPLIB (Kolisch [1996]) teszt könyvtár (http://129.187.106.231/psplib/) J30 alkönyvtárának projektjein teszteltük. Az elsődleges szempontot, az időtartam minimalizálást tekintve, az alkönyvtár minden esetére ismert az optimális megoldás. A másodlagos szempontra, a nettó jelenérték maximalizálásra az egzakt megoldásokat egy korszerű MILP szolverrel (CPLEX 8.1) generáltuk, alapértelmezett módban használva, 900 másodperces időkorláttal. Az időkorlát nagyságának választását a későbbiekben indokoljuk.

A PSPLIB tesztkönyvtárat korábban az 1.4 pontban ismertettük. A J30 halmazban a valós tevékenységek száma 30, és négy megújuló erőforrást igényelnek. Az eseteket teljes faktoros kísérleti tervvel generáltuk, három független probléma paraméterrel:

- háló bonyolultság (network complexity): a projekt hálóban a nem-redundáns irányított élek és a kapcsolódó csúcspontok hányadosának átlaga, a számításba beleértve az áltevékenységeket is.
- erőforrás faktor (resource factor), mellyel a különböző erőforrástípusokat jellemezhetjük: mekkora részt használunk fel a rendelkezésre álló erőforrás mennyiségekből.
- erőforrás erősség (resource strength), mely az erőforráskorlátok erősségét méri.

A J30 halmaz 480 esetből áll, (48 eljárás 10 ismétlése). A pénzmozgásokat véletlenszerűen, de reprodukálhatóan generáltuk. Így a jövőben kifejlesztett egzakt és heurisztikus megoldások minőségének mérésére is használhatjuk a teszthalmazokat az optimális megoldásaikkal. Ezt biztosítandó, a ***Rand 2000*** véletlenszám generátort (Microsoft Q86523) használtuk 0.5 kezdőértékkel minden esetnél:

$$C_i = -50 + \mathbf{Int}\left(\mathbf{Rand\ 2000} * 150 + 0.5\right),\ i \in \{1,2,...,N\} \tag{21}$$

A ***Rand 2000*** véletlenszám generátor sajátossága, hogy a legtöbb véletlenszám generátorral szemben különböző hardver és szoftver esetén is ugyanazt az eredményt adja, így az esetleges jövőbeli megoldásokkal heurisztikánk összehasonlítható lesz.

Ezzel az időkorláttal 355 esetet oldottunk meg optimálisan a CPLEX-el. Az ajánlott algoritmust Visual C++® 6.0 – ban és Visual Basic® 6.0 – ban programoztuk. Az algoritmust 1.8 GHz Pentium IV IBM PC 256 MB memóriával, Microsoft Windows XP® operációs rendszerrel rendelkező számítógépen futtatva kaptuk a számítási eredményeket.

A J30 teszthalmaz eseteit 3 csoportra bontottuk az alábbiak szerint:

1. Az első csoportba azok az esetek tartoznak, melyekre a 900 másodperces időkorláttal leállítva a futást, a CPLEX-et használó eljárás optimális megoldást ad. Erre a „*könnyű*” csoportra adunk számítási eredményeket, mellyel igazoljuk megközelítésünk gyorsaságát és hatékonyságát.
2. A második csoportba azok az esetek tartoznak, melyekre 900 másodperces időkorláttal leállítva a futást, optimális megoldást nem kapunk, megvalósítható,

„feasible” megoldást azonban igen. A továbbiakban megmutatjuk, hogy ezekre a „*nehéz”* esetekre is hatékonynak és gyorsnak bizonyul algoritmusunk.

3. A harmadik csoportba azok az esetek tartoznak, melyekre a 900 másodperces időkorláttal leállítva a futást, CPLEX-et használó eljárás sem tudott a fenti időkorlát alatt megvalósítható megoldást adni. Az ebbe a csoportba tartozó eseteknél nemhogy az optimális megoldás megtalálása a nehéz, hanem egyáltalán egy megvalósítható megoldás megtalálása. Ezekkel az esetekkel ezért nem foglalkozunk disszertációnkban.

Az első két csoportba tartozó eseteket külön-külön fejezetekben fogjuk vizsgálni.

4.3.1. A „könnyű” esetek futási eredményei

Ebben a fejezetben azokkal az esetekkel foglalkozunk, melyekre 900 másodperces időkorláttal leállítva a futást, a CPLEX-et használó eljárás optimális megoldást ad. A 480 J30-beli esetből 355 eset bizonyult ilyennek. Ez az eredmény is jelzi a probléma „NP-nehézségét”. Futtatásaink során a generáció értékét 10-nek, a populáció értékét 10-nek, illetve 100-nak választottuk. Így az ismétlések (iteráció) értékére 100-at, illetve 1000-t kaptunk.

A „könnyű” halmazra vonatkozó paraméter beállításokat a 9. Táblázatban tekinthetjük meg összesítve.

Táblázat 9

Paraméter értékek a „könnyű” halmazra

Paraméter neve	**Érték a "könnyű" halmazra**
Populáció mérete	10, 100
Generáció	10
Iterációk száma	100, 1000
Min. Repertoire Rate, Max. Repertoire Rate	0.1; 0.9
Min. Adjusting Rate, Max. Adjusting Rate	0.1; 0.9

A 10. Táblázatban hibrid algoritmusunkra vonatkozó futási eredményeket láthatjuk két különböző ismétlésszámra.

Táblázat 10

Sounds of Silence eredmények

Ismétlésszám	**Elsődleges szempont eltérése (%)**	**Másodlagos szempont eltérése (%)**	**Megoldási idő**	
			μ (sec)	**σ (sec)**
100	0.02	1.06	0.111	0.045
1000	0.00	0.98	1.080	0.246

A megoldások minőségét az elsődleges szempont érték optimális elsődleges szempont értéktől való százalékos eltérésében mérjük. A másodlagos szempont százalékos eltérését az elsődleges eltérés nélküli megoldással számítottuk.

100 ismétlésszám esetén a legtöbb esetben elértük az optimális elsődleges szempontot - a minimális projekt időtartamot. Értelemszerűen 1000 ismétlésszám során jóval kedvezőbb eredményeket kaptunk, ez esetben minden esetben megkaptuk a minimális időtartamú projektet.

A másodlagos szempontot – a nettó jelenérték maximalizálást – tekintve, 100 ismétlésszám esetén is elég alacsony értékeket kaptunk, míg ez az érték 1000 ismétlésszám esetén jelentősen csökkent.

A 11. Táblázatban a CPLEX 8.1-el futó algoritmus megoldási időinek átlagát, szórását, a minimális és maximális futási idő értékeket figyelhetjük meg.

Táblázat 11

Megoldási idők CPLEX 8.1-el (sec)

Megoldási idők			
Átlag	**Szórás**	**Minimális idő**	**Maximális idő**
19,94	88,98	0,01	770,00

A „könnyű” csoport futási idői nagy szórást mutatnak, „egészen kicsi”, szinte azonnali leállástól 770 másodpercig tartanak.

Érdekes megfigyelés, hogy 900 másodperc „közelében” nem fejeződött be futás. Tehát vagy jóval korábban (maximum 770 másodperc) leállt a futás magától, vagy az eset a 2. illetve 3. csoportba került, mivel mi állítottuk le a futást 900 másodpercnél. Úgy tűnik, a

900 másodpercet érdemes határnak választani, vagy leáll a futás ez idő alatt, vagy sokkal tovább fut.

A két táblázat eredményeit összevetve figyeljük meg, hogy hibrid algoritmusunk az esetek nagy részében nagyságrendekkel gyorsabbnak bizonyul a másik eljárásnál a „könnyű” esetekre.

4.3.2. A „nehéz” esetek futási eredményei

Ebben a fejezetben azokkal a J30-beli esetekkel foglalkozunk, melyekre 900 másodperces időkorláttal leállítva a futást, a CPLEX-et használó algoritmus optimális megoldást nem ad, megvalósítható, „feasible” megoldást azonban igen. Futtatásunk során a generáció és a populáció értékét is 10-nek választottuk. Így az ismétlések értékére 100-at kaptunk. Minden futtatást 30-szor futtatunk le egymás után egymástól függetlenül. Ezzel a módszerrel az eljárás robusztus jellegével kapcsolatban megbízható (szignifikáns) statisztikai információkat kaptunk.

A „nehéz” halmazra vonatkozó paraméter beállításokat a 12. Táblázatban tekinthetjük meg összesítve.

Táblázat 12

Paraméter értékek a „nehéz” halmazra

Paraméter neve	**Érték a „nehéz” " halmazra**
Populáció mérete	10
Generáció	10
Iterációk száma	100
Független futások száma	30
Min. Repertoire Rate, Max. Repertoire Rate	0.1; 0.9
Min. Adjusting Rate, Max. Adjusting Rate	0.1; 0.9

Annak érdekében, hogy hibrid eljárásunkat összehasonlíthassuk a CPLEX-et használó eljárás eredményeivel, a minőség mérésére bevezettük a *QualityMeasure* mértéket.

A *QualityMeasure* definíció szerint a következő:

If NetPresentValueApproach Then

QualityMeasure=

=(PrimaryMeasure -OptimalPrimaryMeasure) / (OptimalPrimaryMeasure-1)

If QualityMeasure = 0 Then

QualityMeasure = (CplexSecondaryMeasure - SecondaryMeasure) / CplexSecondaryMeasure

Else

QualityMeasure = QualityMeasure + 1

End If

End If

Mint korábban említettük, az *OptimalPrimaryMeasure* értéke ismert a J30-beli esetekre. A J30 halmaz ezen „nehéz” részhalmaza a következő 17 esetet tartalmazza:

{ 1:6, 5:5, 6:1, 10:2, 10:3, 10:8, 14:1, 15:5, 17:1, 26:5, 31:9, 33:3, 37:2, 37:9, 38:5, 42:3, 46:1},

ahol a kettőspont előtt szereplő szám a 48 eset valamelyikét jelöli, a kettőspont után szereplő szám pedig az azon belüli 10 ismétlés valamelyikét.

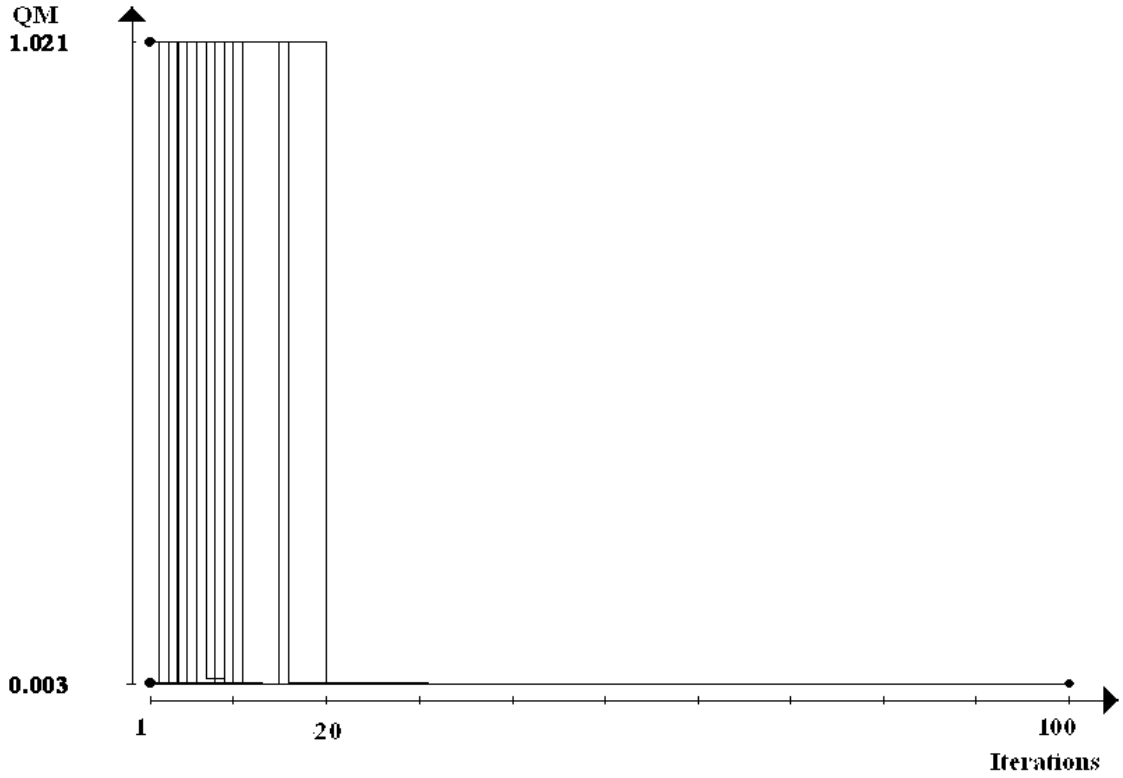
4.3.2.1. A J30-1-6 „nehéz” eset futási eredményei

Először kiemelten foglalkozunk egy „nehéz” halmazbeli esettel, a J30-1-6-al. Fontosnak tartjuk megjegyezni, hogy bár a J30-1-6 kétségtelenül a „legjobb” esetek közül való, azaz a leglátványosabb eredményeket adja, a többi esetre is hasonlóan jó eredményeket

kapunk, mint azt a 4.3.2.2 pontban látni fogjuk. Az egy eset kiragadásának és külön tárgyalásának az oka az volt, hogy így hatásosan tudjuk szemléltetni az eredményeket, például grafikonnal. A továbbiakban erre az esetre adunk futási eredményeket.

A 15. ábrával szemléltetjük a futások eredményét. A grafikon a 100 iterációs lépés alakulását mutatja a 30 független futás esetén. A függőleges tengelyen a QualityMeasure értékét (QM) ábrázoltuk, vízszintes tengelyen az ismétlések (*generation * population*) számát .

15. ábra 30 független futás eredménye a J30-1-6 esetre



Tekintsük a QualityMeasure fenti definícióját. Mivel ebben az esetben kezdetben *PrimaryMeasure= OptimalPrimaryMeasure* (49), *QM = 0* értéket kapunk.

Később, mivel *CplexSecondaryMEasure = 511,3021* és

SecondaryMeasure = 509,9756 , kerekítve a *QM = 0*,003 értéket kapjuk.

Az ábrán szereplő, QualityMeasure értékek kezdetben magasak, majd igen gyorsan konvergálnak felülről a *QM = 0*,003 értékhez, tehát ebben az esetben hibrid algoritmusunk robosztusnak, stabilnak és gyorsnak bizonyult.

A halmaz többi elemére hasonló ábrákat kapunk.

Vizsgáltuk a 30 független futás QM eredményeit a J30-1-6 esetre. Az elsődleges szempontra optimalizálás esetére vonatkozó eredményeket a 13. Táblázatban találjuk:

Táblázat 13

Futási eredmények a J30-1-6 esetre elsődleges szempontra optimalizálás esetén

Futások száma	**A legjobb megoldás**	**Hibrid eljárás**	**QM (%)**
1	49	49	0
…	…	…	…
30	49	49	0

Mivel azt tapasztaltuk, hogy az összes futásra ugyanazokat az eredményeket kapjuk, így a táblázat összes sora azonos lett, ezért nem közöljük külön az összes sort.

A J30-1-6 esetre mind a 30 futás során elértük a legjobb megoldást, így a QM érték nyilvánvalóan 0, maximális lett.

A másodlagos szempontra optimalizálás esetére vonatkozó eredményeket a 14. Táblázatban találjuk

Táblázat 14

Futási eredmények a J30-1-6 esetre másodlagos szempontra optimalizálás esetén

Futások száma	**CPLEX megoldás**	**Hibrid eljárás**	**QM (%)**
1	509,9756	511,3021	0,26
…	…	…	…
30	509,9756	511,3021	0,26

Ebben az esetben is azt tapasztaltuk, hogy az összes futásra – szinte - ugyanazokat az eredményeket kapjuk, így a táblázat összes sora azonos lett, ezért nem közöljük külön az összes sort.

Mind a 30 futás esetén ugyanazt a minimális eltérést kaptuk a CPLEX megoldás és a hibrid algoritmus megoldása között: 0.26.

Az, hogy az összes futás esetén ugyanazokat a végeredményeket kaptuk, eljárásunk robosztusságát, stabilitását jelzi.

A 15. Táblázatban a 30 független futás megoldási időinek átlagát, szórását, a minimális és maximális futási idő értékeket figyelhetjük meg a J30-1-6 esetre.

Táblázat 15

Megoldási idők (sec) a J30-1-6 esetre

Átlag	**Szórás**	**Minimum Idő**	**Maximum Idő**
0.10	0.04	0.02	0.17

Vegyük észre, hogy a szórás igen alacsony – szinte 0 - , ami hibrid algoritmusunk stabilitásáról tett állításunkat támasztja alá. Az alacsony maximális futási idő érték megközelítésünk gyorsaságát mutatja.

4.3.2.2. Az összes „nehéz” eset futási eredményei

A továbbiakban az egész „nehéz” részhalmazra vonatkozólag adunk futási eredményeket. Célunk, hogy bebizonyítsuk, hibrid eljárásunk ezekre a „nehéz” esetekre is gyorsnak, hatékonynak és robosztusnak bizonyul.

A 16. Táblázatban a QualityMeasure értékek átlagát, szórását, minimális és maximális értékeit figyelhetjük meg a fenti halmaz összes elemére. Ebben az esetben a QM értékeket az elsődleges szempontra számoltuk.

Táblázat 16

A QM (%) értékek az elsődleges szempontra

	Átlag	**Szórás**	**Minimum**	**Maximum**
J30(1:6)	0.00	0.00	0.00	0.00
J30(5:5)	0,04	0,24	0.00	0,04
J30(6:1)	0.00	0.00	0.00	0.00
J30(10:2)	1,19	0,86	0.00	1,19
J30(10:3)	0,38	0,69	0.00	0,38
J30(10:8)	0.00	0.00	0.00	0.00
J30(14:1)	0.00	0.00	0.00	0.00
J30(15:5)	1,09	0,84	0.00	1,09
J30(17:1)	0,31	0,86	0.00	0,31
J30(26:5)	0.00	0.00	0.00	0.00
J30(31:9)	0.00	0.00	0.00	0.00
J30(33:3)	0.00	0.00	0.00	0.00
J30(37:2)	0.00	0.00	0.00	0.00
J30(37:9)	0,35	0,71	0.00	0,35
J30(38:5)	0.00	0.00	0.00	0.00
J30(42:3)	0,50	0,78	0.00	0,50
J30(46:1)	0,39	0,73	0.00	0,39
Összesítve	**0.25**	**0.34**	**0.00**	**1.19**

A táblázatban szereplő időtartam értékek közötti eltérés igen kicsi – a legtöbb esetben nulla, ami azt mutatja, hogy a legtöbb esetben elértük az optimális elsődleges

szempontot, vagy jelentősen megközelítettük azt, tehát eljárásunk igen hatékony. Az alacsony – legtöbb esetben 0 - szórás értékek algoritmusunk robosztusságát mutatják.

Táblázat 17

A QM (%) értékek az másodlagos szempontra

	Átlag	**Szórás**	**Minimum**	**Maximum**
J30(1:6)	0.26	0.00	0.26	0.26
J30(5:5)	4.12	0.09	4.07	4.12
J30(6:1)	1.27	0.33	1.00	1.27
J30(10:2)	-0.31	2.10	-1.80	-0.31
J30(10:3)	-0.05	1.39	-3.98	-0.05
J30(10:8)	1.83	1.69	0.56	1.83
J30(14:1)	1.47	0.62	0.37	1.47
J30(15:5)	1.61	0.85	0.28	1.61
J30(17:1)	2.29	1.86	-1.98	2.29
J30(26:5)	0.41	0.00	0.41	0.41
J30(31:9)	0.07	0.00	0.07	0.07
J30(33:3)	0.73	0.00	0.73	0.73
J30(37:2)	-0.09	0.29	-0.29	-0.09
J30(37:9)	0.02	1.69	-1.73	0.02
J30(38:5)	0.81	0.00	0.81	0.81
J30(42:3)	1.06	0.83	-0.31	1.06
J30(46:1)	-0.32	0.30	-0.91	-0.32
Összesítve	**0.89**	**0.71**	**-3.98**	**4.12**

A 17. Táblázatban a QualityMeasure értékek átlagát, szórását, a minimális és maximális értékeit figyelhetjük meg a „nehéz” részhalmaz összes elemére. Ebben az esetben a QM értékeket az másodlagos szempontra számoltuk. A táblázatban előforduló negatív eredmények azt jelentik, hogy abban az esetben hibrid algoritmusunk bizonyult eredményesebbnek, azaz adott magasabb nettó jelenértéket! Ilyen eset a 17-ből több is, például a J30-10-2, J30-10-3. A nemnegatív értékek az értékek közötti eltérés igen kicsi, ami azt mutatja, hogy a legtöbb esetben jelentősen megközelítettük az optimális másodlagos szempontot. Az alacsony szórás értékek algoritmusunk robosztusságát mutatják. Figyeljük meg, hogy több esetben is 0 szórás értéket kapunk.

A 18. Táblázatban a futási idők átlagát, szórását, a minimális és maximális értékeit figyelhetjük meg a „nehéz” részhalmaz összes elemére, amikor célunk a időtartam minimalizáló erőforráskorlátos ütemezések halmazán a nettó jelenérték maximalizáló ütemezések megtalálása.

A táblázatban szereplő szórás értékek igen alacsonyak, tehát az eredmények algoritmusunk robosztus voltát igazolják. Az alacsony maximális megoldási idő – minden esetben jóval 1 másodperc alatti értéket kaptunk - megközelítésünk gyorsaságát igazolja. Hibrid algoritmusunk minden „nehéz” esetben nagyságrendekkel gyorsabbnak bizonyul a másik eljárásnál.

Táblázat 18

Megoldási idők (sec) a „nehéz” részhalmaz eseteire

	Átlag	**Szórás**	**Minimum idő**	**Maximum idő**
J30(1:6)	0.04	0.09	0.02	0.17
J30(10:2)	0.03	0.09	0.05	0.16
J30(10:3)	0.04	0.11	0.03	0.20
J30(10:8)	0.04	0.10	0.03	0.17
J30(14:1)	0.03	0.09	0.03	0.17
J30(15:5)	0.03	0.08	0.02	0.14
J30(17:1)	0.04	0.12	0.05	0.24
J30(26:5)	0.03	0.12	0.05	0.17
J30(31:9)	0.05	0.11	0.03	0.23
J30(33:3)	0.06	0.16	0.06	0.33
J30(37:2)	0.05	0.16	0.08	0.27
J30(37:9)	0.04	0.13	0.06	0.23
J30(38:5)	0.04	0.14	0.06	0.24
J30(42:3)	0.05	0.15	0.08	0.25
J30(46:1)	0.04	0.12	0.05	0.22
J30(5:5)	0.03	0.09	0.03	0.16
J30(6:1)	0.04	0.10	0.05	0.22
Összesítve	**0.04**	**0.12**	**0.02**	**0.33**

Összegezve a fenti – a „könnyű” és a „nehéz” esetekre vonatkozó - számítási eredményeket:

Mivel korábbi algoritmusokkal nem tudtuk összehasonlítani hibrid eljárásunkat, számítási eredményeket közöltünk a közismert J30 teszthalmaz eseteire, vizsgálva, eredményeink mennyire térnek el az optimális értékektől. Az eredmények a „könnyű” és a „nehéz” esetekre is algoritmusunk hatékonyságát, gyorsaságát, és még a „nehéz” esetekre is annak robosztusságát bizonyították.

5. Továbbfejlesztési irányok

Az alábbiakban a korábban ismertetett algoritmus lehetséges jövőbeli továbbfejlesztési irányait adjuk meg, melyből időközben az 5.3 pontban bemutatott fejlesztési ötlet már megvalósításra is került (Láng [2010/a]). A sikerrel járt ötletek mellett megemlítjük, az ígéretesnek ígérkező, de eredményt nem adó ötleteket is.

5.1. NPV sajátosságainak kihasználása

Hibrid algoritmusunk másodlagos szempontja a nettó jelenérték maximalizálása. Nyilvánvalóan adódik az ötlet, hogy érdemes lenne kihasználni az NPV természetéből adódó sajátosságokat ütemezésünk során.

Az NPV természetéből fakadó sajátosságot figyelembe vettük már ott, amikor azt említettük, hogy az NPV „természetétől fogva” másodlagos szempont.

A továbbiakban ismertetjük ez irányú próbálkozásainkat: az algoritmusunk több pontján próbáltunk módosítani, úgy hogy a fenti sajátosságokat vettük figyelembe.

5.1.1. Az ütemezési sorrend

A Sounds of Silence algoritmus egyik kulcslépése a „tevékenység lista generálás”. A *ListScheduling()* függvényünk alkalmazása során rendre kezdési idejük sorrendjében vesszük a tevékenységeket a listáról. Előfordulhat, hogy két tevékenység kezdési ideje azonos. Ekkor több lehetőség is áll előttünk a sorrendjük eldöntésére. Jelenleg az algoritmusban index szerinti sorrendet állapítunk meg ilyen esetben.

Felmerült az ötlet, hogy ezekre a pontokra vonatkozólag tekinthetjük az összes lehetséges sorrendet, és egy lokális keresési fával végigjárhatjuk őket. Azonban ennek a megoldásnak túl nagy lett volna a számítási ideje, elvetettük.

Másik lehetséges megoldás lehetne az, hogy a tevékenységeket a hozzájuk társított pénzmozgások nagyságának sorrendjében vesszük rendre. Ebben az esetben azt a már többször említett jól ismert szabályt használnánk ki, hogy a pozitív pénzmozgású tevékenységeket olyan hamarra érdemes ütemezni, ahogy csak tudjuk, a negatív pénzmozgásúakat pedig olyan későre, ahogy csak lehetséges. Hiába alkalmaztuk azonban ezt a megoldást, nem adott semmilyen javulást sem az eljárás számítási idejében, sem hatékonyságában, így elvetettük, bármilyen ígéretesnek ígérkezett.

5.1.2. Intelligens kezdőrepertoár

Másik adódó ötlet, hogy nem teljesen véletlen kezdőrepertoárt generálunk a *RandomGauss()* függvénnyel, hanem a pénzmozgásokat lenormáljuk a [-1;1] intervallumra, és ezek az értékek adnák a kezdeti elképzeléseket. Itt is a fent említett szabályt használnánk ki. Azonban ez a megoldás sem hozott semmilyen javulást sem az eljárás számítási idejében, sem hatékonyságában, így ezt is elvetettük.

Az 5.1.1 és 5.1.2 továbbfejlesztési irányok ugyan nem voltak sikeresek, de érdemesnek láttuk további módosításokon gondolkozni, melyekben a nettó jelenérték sajátosságait kíséreltük meg kihasználni. A következő továbbfejlesztés sikeresnek ígérkezik.

5.1.3. Nettó jelenérték növelése a rejtett konfliktusok javítása során

Az eredeti algoritmusunkban a rejtett konfliktusok javítását egyfajta „ökölszabállyal” valósítottuk meg. Továbbfejlesztésünk lényege, hogy oly módon fogjuk javítani a rejtett konfliktusokat, hogy a javítással egyben lehetőleg a projekt nettó jelenértékét is növeljük. Kihasználjuk azt a tényt, hogy rövid szakaszon az exponenciális nettó jelenérték függvény lineáris függvénnyel jól közelíthető, így gyorsan számolhatjuk. Az

eljárás azon pontján, ahol a rejtett konfliktusokat javítjuk, már csak kis számú tevékenység mozgatható, azok is csak rövid szakaszon. Az elkövetkező időszak egyik feladata ezen továbbfejlesztés kidolgozása, megvalósítása. (Csébfalvi Anikó, Láng [megjelenés alatt]).

5.2. A konfliktus javító lépés javítása

A konfliktusjavítást végző ***`ForbiddenConflictRepairing`*** algoritmus jelenlegi formájában egy lépésben kezeli az összes rejtett konfliktus javítását. Mivel egy javító reláció beillesztése indirekt javíthat egy másik konfliktust is, előfordulhat, hogy az eljárás felesleges javító relációkat is beilleszt. Egy esetleges jövőben kifejlesztett megoldás esetén érdemes megfontolni, van-e mód arra, hogy elkerüljük azokat az eseteket, amikor egy konfliktust többször is javítunk, illetve ha elkerüljük a többszörös javításokat, tapasztalunk-e mérhető javulást a heurisztika megoldási idejét és hatékonyságát tekintve.

5.3. Finomítási lehetőségek az erőforrás felhasználás modellezésében

Hibrid eljárásunk egyik lehetséges továbbfejlesztési iránya az erőforrás felhasználás módjának megfontolása. Új algoritmusunk két ponton is tartalmaz erőforrás felhasználással kapcsolatos fejlesztést: alkalmassá tettük az „erőforrás kiegyenlítés”, és a „dedikált hozzárendelési probléma” kezelésére.

Amennyiben az ábrázolási módszerünket megváltoztatjuk, modellünk alkalmassá válik egy harmadlagos szempont szerinti optimalizálásra az eddigi elsődleges és másodlagos szempont mellett, így eljárásunk egy még inkább valósághűbb problémát old meg. Ez az új szempont választásunk szerint az „erőforrás kiegyenlítés” problémája:

minimalizáljuk az erőforrás egységek indítási - újraindítási eseményeinek számát az erőforráskorlátot kielégítő tevékenység mozgatások halmazán, rögzítve a projekt időtartamát és a pénzmozgás események helyét a lokális legjobb megoldásnak megfelelően. Az új szempont bevezetésével a folyamatos munkát részesítjük előnyben, az alkalmazott mérték jó indikátora a hatékony erőforrás felhasználásnak.

Az új modellben a pénzmozgásokra nulla hosszúságú áltevékenységként tekintünk, és az elsőbbségi feltételeket kiterjesztjük a pénzmozgás tevékenységekre is. Ez a modell figyelembe veszi azt a tényt, hogy amíg az erőforrás felhasználás tevékenység orientált, addig a pénzmozgás eseményorientált. Negatív pénzmozgás általában egy tevékenységhalmaz megkezdése előtt következik be - ilyen például az anyagvásárlás -, pozitív pénzmozgás pedig tevékenység halmaz befejezése után, például részhatáridő teljesítésekor.

Továbbfejlesztett megoldásunk egy másik – a harmadlagos szemponttól független – újítása a „dedikált hozzárendelési probléma” kezelése, azaz annak megkövetelése, hogy egy tevékenység egységnyi erőforrás igényét pontosan egy erőforrás elégítse ki. Azt az ütemezést, amely képes erre, röviden „erős erőforrás hozzárendelési feltételekkel” rendelkező ütemezésnek fogjuk nevezni. A probléma gyakorlati jelentőséggel bír, mivel megoldásakor minimalizáljuk a munkások munkára és munka típus váltásra való ráhangolódáshoz szükséges energiaigényét. Ez a minimalizálás minden projekt menedzser érdeke.

A fentiek alapján az ebben a fejezetben ismertetett továbbfejlesztett algoritmus keresi erőforráskorlátos projektünk „legjobb” ütemezését, ahol „legjobb” alatt azt az erős erőforrás hozzárendelési feltételekkel rendelkező, minimális időtartamú ütemezést

értjük, melyre a nettó jelenérték mérték maximális, és az erőforrás felhasználási profil alakja leginkább megközelíti az ideális téglalap alakot.

Az alkalmazott modellben a pozitív és negatív pénzmozgás eseményeket nulla időtartamú áltevékenységekkel szemléltetjük, melyek a megelőző és követő tevékenység csoportok kezdő, illetve végpontjához kapcsolódnak elsőbbségi feltételekkel. Ez az új modell alkalmassá teszi eljárásunkat egy harmadlagos szempont szerinti optimalizálásra is, mivel az elsődleges és másodlagos szempont szerint már optimalizált megoldások halmazából kiválaszthatjuk a harmadlagos szempont szerint optimális megoldást, mivel rögzített pénzmozgások és rögzített projekt időtartam mellett a tevékenységeket szabadon mozgathatjuk az eljárás ezen pontján, a mozgatás során nem sértjük meg az erőforráskorlátokat, és nem változik a projekt NPV-je. Az alkalmazott „kiegyenlítő – kisimító" eljárás a projekt időtartamának és a pénzmozgás események pozíciójának rögzítése után az erőforrás egységek indítási - újraindítási eseményeinek számát minimalizálja az erőforráskorlátokat kielégítő tevékenység mozgatások halmazán.

Ismertetjük új modellünket és algoritmusunkat, majd a modellünkben rejlő további lehetőségek szemléltetésére számítási eredményeket közlünk a már ismert J30-1-1 esetre (az első J30-beli eset a PSPLIB teszthalmazból) véletlenszerűen generált pénzmozgás értékekkel.

Az algoritmus a disszertációban eddig ismertetett megoldás, a Sounds of Silence harmónia kereső algoritmus továbbfejlesztett, konfliktusjavító verziójára épül, egy harmónia kereső metaheurisztika és egy MILP formulán alapuló „erőforrás kiegyenlítő-hozzárendelő" eljárás kombinációja. Ez utóbbi Csébfalvi és Konstantinidis [1998] megoldása..

5.3.1. A modell

Az eljárás során a megoldandó modellünk:

$$\boldsymbol{max}\left[NPV = \sum_{i=1}^{E} \sum_{t \in E_i} C_{it} * E_{it} \right] = NPV^* \tag{22}$$

$$X_i + D_i \le X_j,\ i \to j \in \boldsymbol{PS},\tag{23}$$

$$E_i \le X_j,\ i \to j \in \boldsymbol{ES},$$

$$X_i + D_i \le E_j,\ i \to j \in \boldsymbol{PE},$$

$$X_{N+1} = \overline{T} + 1 \tag{24}$$

$$X_i = \sum_{t \in T_i} X_{it} * t,\ T_i = \left\{ \underline{X}_i, \underline{X}_i + 1, \ldots, \overline{X}_i \right\},\ i \in \{1,2,...,N\} \tag{25}$$

$$E_i = \sum_{t \in E_i} E_{it} * t,\ E_i = \left\{ \underline{E}_i, \underline{E}_i + 1, \ldots, \overline{E}_i \right\},\ i \in \{1,2,...,E\}$$

$$\sum_{t \in T_i} X_{it} = 1,\ X_{it} \in \{0,1\},\ i \in \{1,2,...,N\} \tag{26}$$

$$\sum_{t \in T_i} E_{it} = 1,\ E_{it} \in \{0,1\},\ i \in \{1,2,...,E\}$$

$$A_t = \left\{ i \,\middle|\, X_i \le t < X_i + D_i, i \in \{1,2,...,N\} \right\},\ t \in \{1,2,...,T\} \tag{27}$$

$$U_{tr} = \sum_{i \in A_t} R_{ir},\ t \in \{1,2,...,T\},\ r \in \{1,2,...,R\} \tag{28}$$

$$U_{t\,r} \le R_r,\ t \in \{1,2,...,T\},\ r \in \{1,2,...,R\} \tag{29}$$

$$C_{it} = C_i * e^{-\alpha(t-1)},\ i \in \{1,2,...,E\},\ t \in E_i \tag{30}$$

$$X_{it} \in \{0,1\},\ t \in T_i,\ i \in \{1,2,...,N\} \tag{31}$$

$$E_{it} \in \{0,1\},\ t \in E_i,\ i \in \{1,2,...,E\}$$

A modell ismertetésekor csak azt emeljük ki, amiben eltér a (1 - 10) egyenletrendszerrel leírt korábbi modelltől:

- Jelölje E a pénzmozgás tevékenységek számát.
- Jelölje $\boldsymbol{ES}(\boldsymbol{PE})$ az e eseményt ($e \in \{1,2,\ldots,E\}$) közvetlen megelőző (követő) tevékenységek halmazát.
- Jelölje $\underline{E}_i$ ($\overline{E}_i$) az i pénzmozgás esemény legkorábbi (legkésőbbi) kezdési idejét.
- A (31) egyenletben leírt E_{it} bináris változók a pénzmozgás események bekövetkezési idejét határozzák meg. $E_{it} = 1$, ha az i-edik pénzmozgás esemény t időpontban következik be, egyébként $E_{i,t} = 0$.
- Az (22) egyenlet a modell célfüggvényét írja le. A célfüggvény maximalizálja a projekt ideje alatt bekövetkező összes pénzmozgás diszkontált értékének összegét. Megjegyezzük, hogy a legkorábbi ütemezés nem maximalizálja szükségszerűen a pénzmozgások nettó jelenértékét.
- A (23) egyenlet a pénzmozgás tevékenységekkel kiegészített tevékenységhalmaz elemei között fennálló elsőbbségi feltételeket írja le. A tevékenységek között megelőző-követő relációkat feltételezünk. A „normál” tevékenységek időtartama: D_i , míg a pénzmozgás tevékenységek időtartama 0.

- A (25 - 26) egyenletek kiegészültek a pénzmozgás tevékenységekre vonatkozó feltételekkel: A (25) egyenlet biztosítja a korábbiakon felül azt is, hogy minden pénzmozgás tevékenység a legkorábbi és a legkésőbbi kezdési időpontjai közötti időtartamban kezdődjön el. A (26) egyenlet biztosítja, hogy minden pénzmozgás tevékenység is pontosan egyszer kezdődjön el.
- A (30) egyenlet a t időperiódusban kezdődő, i-edik tevékenységhez kapcsolódó pénzmozgást adja meg a projekt kezdetére visszadiszkontálva, a tevékenység befejezési idejének függvényében. A különbség a korábbiakhoz képest az, hogy a pénzmozgásokra önálló (ál)tevékenységként tekintünk.

A korábbi gondolatmenethez hasonlóan itt is megjegyezzük, hogy ha (23) egyenletben ismertetett szokásos formában leírt elsőbbségi feltételeket a teljesen unimoduláris (TU) formulával helyettesítjük, és kihasználjuk hogy a célfüggvény minden komponensében monoton, akkor a nemkorlátos nettó jelenérték probléma (NPVP) polinomiális idő alatt oldható meg, mint egy LP probléma.

5.3.2. Az algoritmus

Ebben a fejezetben ismertetjük az algoritmust, mely a 4.2 pontban bemutatott algoritmus továbbfejlesztése. Csak azokra a pontokra térünk ki, melyekben az új algoritmus eltér a korábbitól.

Az eltérés lényege, hogy a harmónia keresést egy MILP formulán alapuló erőforrás kiegyenlítő-hozzárendelő eljárással kombináltuk. Megkíséreltük minimalizálni a számítási erőfeszítéseket, ezért a karmester csak abban az esetben hívja meg az eljárást, ha a pillanatnyi „legjobb megoldást” cseréljük jobbra. Ekkor a „javított” erőforráskorlátos megoldás halmazon megoldjuk az erőforrás kiegyenlítési problémát, rögzítve a pénzmozgás tevékenységek helyét a pillanatnyi optimális megoldásnak

megfelelően. Az esemény orientált pénzmozgás formulának megfelelően, miután rögzítettük a pénzmozgás tevékenységek helyét, marad valamennyi szabadságunk az erőforráskorlátos tevékenység mozgatások halmazán „kisimítani” az erőforrás profilt.

Az algoritmus kihasználja azt a tényt, hogy az erőforrás kiegyenlítési probléma és az „erős hozzárendelési / dedikált hozzárendelési probléma” elkülönítve kezelhető, mert az adott erőforrás kiegyenlítési mérték invariáns az erőforrás egységek permutációjára, a dedikált erőforrás hozzárendelés nem képes megváltoztatni a fenti mérték értékét.

Az alkalmazott erőforrás kiegyenlítő eljárás a folyamatos munkát preferálja, minimalizálja az erőforrás egységek indítási - újraindítási eseményeinek számát az erőforráskorlátot kielégítő tevékenység mozgatások halmazán, rögzítve a projekt időtartamát és a pénzmozgás események helyét a lokális legjobb megoldásnak megfelelően.

Az erőforrás kiegyenlítési problémát a (32 - 39) egyenletrendszerrel írhatjuk le:

$$\boldsymbol{min}\left[LM = \sum_{r=1}^{R} \sum_{i=1}^{T} CU_{rt}^{+} \right] = LM^{*} \tag{32}$$

$$X_i + D_i \le X_j ,\ i \to j \in \boldsymbol{PS}^{*} \tag{33}$$

$$X_i = \sum_{t \in T_i} X_{it} * t ,\ T_i = \left\{ \underline{X}_i , \underline{X}_i + 1, \ldots , \overline{X}_i \right\},\ i \in \left\{ 1,2,...,N \right\} \tag{34}$$

$$\sum_{t \in T_i} X_{it} = 1 ,\ X_{it} \in \left\{ 0,1 \right\},\ i \in \left\{ 1,2,...,N \right\} \tag{35}$$

$$A_t = \left\{ i \,\middle|\, X_i \le t < X_i + D_i , i \in \left\{ 1,2,...,N \right\} \right\},\ t \in \left\{ 1,2,...,T \right\} \tag{36}$$

$$U_{tr} = \sum_{i \in A_t} R_{ir} ,\ t \in \left\{ 1,2,...,T \right\},\ r \in \left\{ 1,2,...,R \right\} \tag{37}$$

$$U_{t\,\mathrm{r}} - CU_{tr}^{+} + CU_{tr}^{-} = U_{t-1\,\mathrm{r}},\ t \in \{2,3,...,T\},\ r \in \{1,2,...,R\} \tag{38}$$

$$U_{1\,\mathrm{r}} - CU_{1r}^{+} = 0,\ r \in \{1,2,...,R\}$$

$$X_{it} \in \{0,1\},\ t \in T_i,\ i \in \{1,2,...,N\} \tag{39}$$

A modellben $T_i = \{\underline{X}_i, \underline{X}_i + 1, \ldots, \overline{X}_i\}$ jelöli az i tevékenységhez tartozó időablakot a lokális konfliktusjavítás után, ahol $\underline{X}_i$ ($\overline{X}_i$) az i tevékenység legkorábbi/legkésőbbi kezdési ideje rögzített projekt időtartam, és rögzített pozíciójú pénzmozgás események mellett. $\boldsymbol{PS}^{*}$ jelöli a kiterjesztett megelőző-követő relációk halmazát az alkalmazott helyi konfliktus javítás után. CU_{tr}^{+} (CU_{tr}^{-}) jelöli a t időperiódusban az újrainduló (leálló) erőforrás egységek számát az r-edik erőforrásból.

A „dedikált erőforrás hozzárendelési” modell nagyon könnyen fogalmazható meg, mivel az erőforrás kiegyenlítő eljárás pontosan meghatározza a tevékenységek kezdési idejét, így csak azt kell biztosítanunk, hogy minden erőforrás igényt kielégítsünk, és minden egyes erőforrás igényt legfeljebb egy erőforrás egységhez rendeljünk hozzá. A probléma az „egységnyi idejű végrehajtási modell” (UET, unit execution time) variánsa, így polinomiális idő alatt megoldható. Az UET modellben azzal a feltételezéssel élünk, hogy minden tevékenység egységnyi végrehajtási idővel rendelkezik. A megoldási algoritmust, mely polinomiális idő alatt végrehajtható, Cheun és Bulfin [1990] tanulmányában találjuk.

Az algoritmus problémánkra módosított megoldásának ismertetését Csébfalvi és Csébfalvi [2010] tanulmányában találjuk. Az eljárás lényege, hogy egységnyi négyzetekre osztjuk az RCPSP megoldása eredményeképpen kapott erőforrás

felhasználás hisztogram minden elemét. Ezt követően a hisztogram második oszlopától kezdve a következő lépéseket követjük:

1. Tekintjük azokat a tevékenységeket, melyek erőforrás igénye a megelőző oszlopban már szerepel., és a korábbi négyzetekhez „ragasztjuk” az új oszlop egységnyi négyzetét.
2. Az „újonnan induló” erőforrás egységeket véletlenszerűen elosztjuk a fennmaradó szabad helyeken.

Megjegyzendő, hogy a pénzmozgás esemény orientált modellünkben igen sok további lehetőség rejlik. Harmadlagos szempontként a számtalan erőforrás kiegyenlítő-kisimító eljárás közül szemléltetésképpen választottuk a fent leírt eljárást, de a jövőben érdemes megfontolni további eljárások választását is.

5.3.3. Számítási eredmények.

Táblázat 19

Véletlenszerűen generált pénzmozgás események

Esemény	Pénzmozgás esemény	Véletlenszerűen generált pénzmozgás	Közvetlen megelőző események	Közvetlen követő események
1	A	-39	{1,2,3}	
2	B	6		{7,26}
3	C	46		{16,10}
4	D	67		{17,14}
5	E	54		{23,18}
6	F	-49	{22,19,26}	
7	G	73		{7,5}
8	H	-38	{10,21}	
9	I	49		{21,27}
10	J	86	{28,29,30}	{28,29,30}

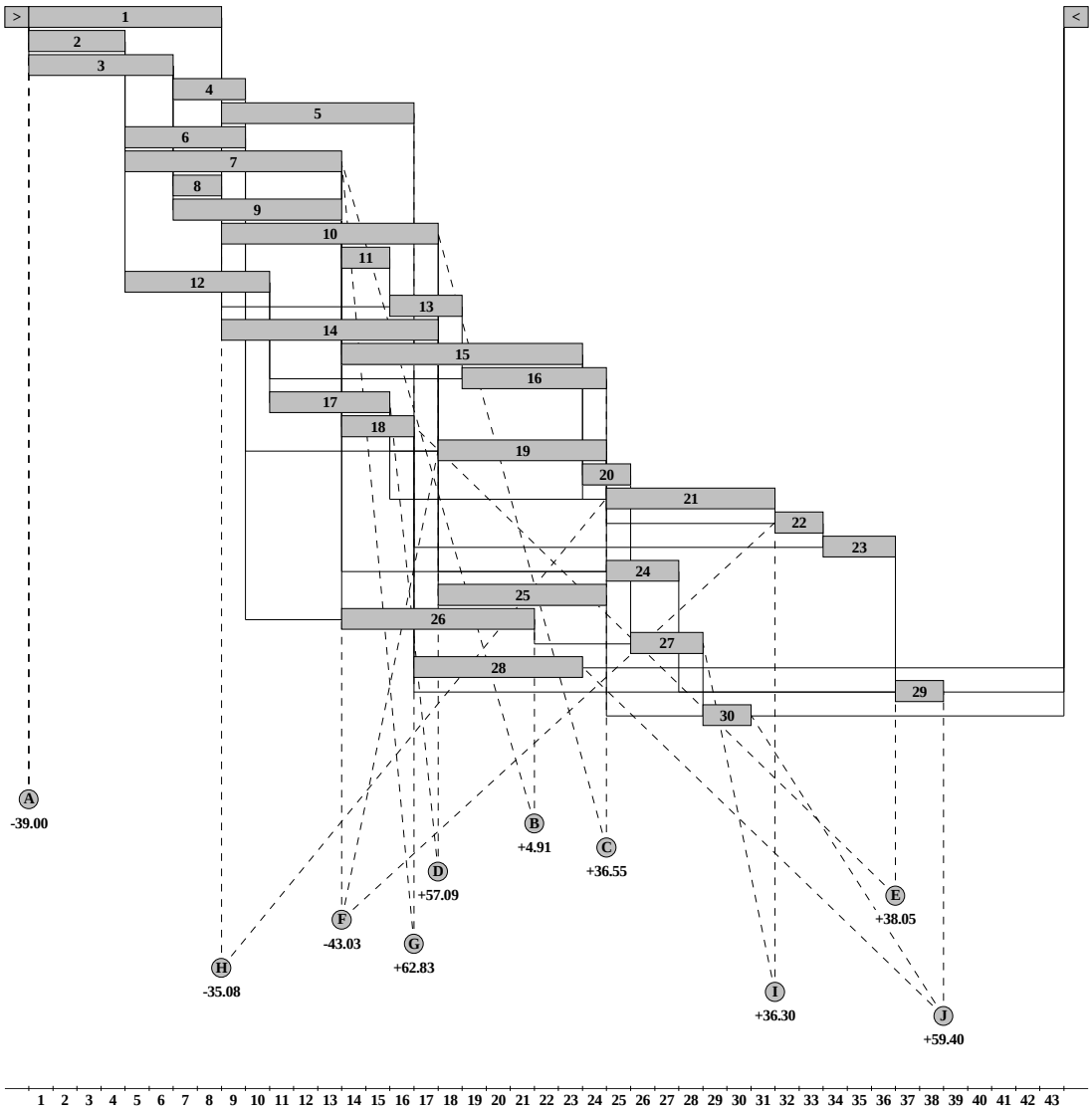
Modellünkben rejlő további lehetőségek szemléltetésére a már korábban említett PSPLIB teszthalmaz-beli J30-1-1 példájára mutatunk be futási eredményeket.

A véletlenszerűen generált pénzmozgás eseményeket a 19. Táblázatban mutatjuk be.

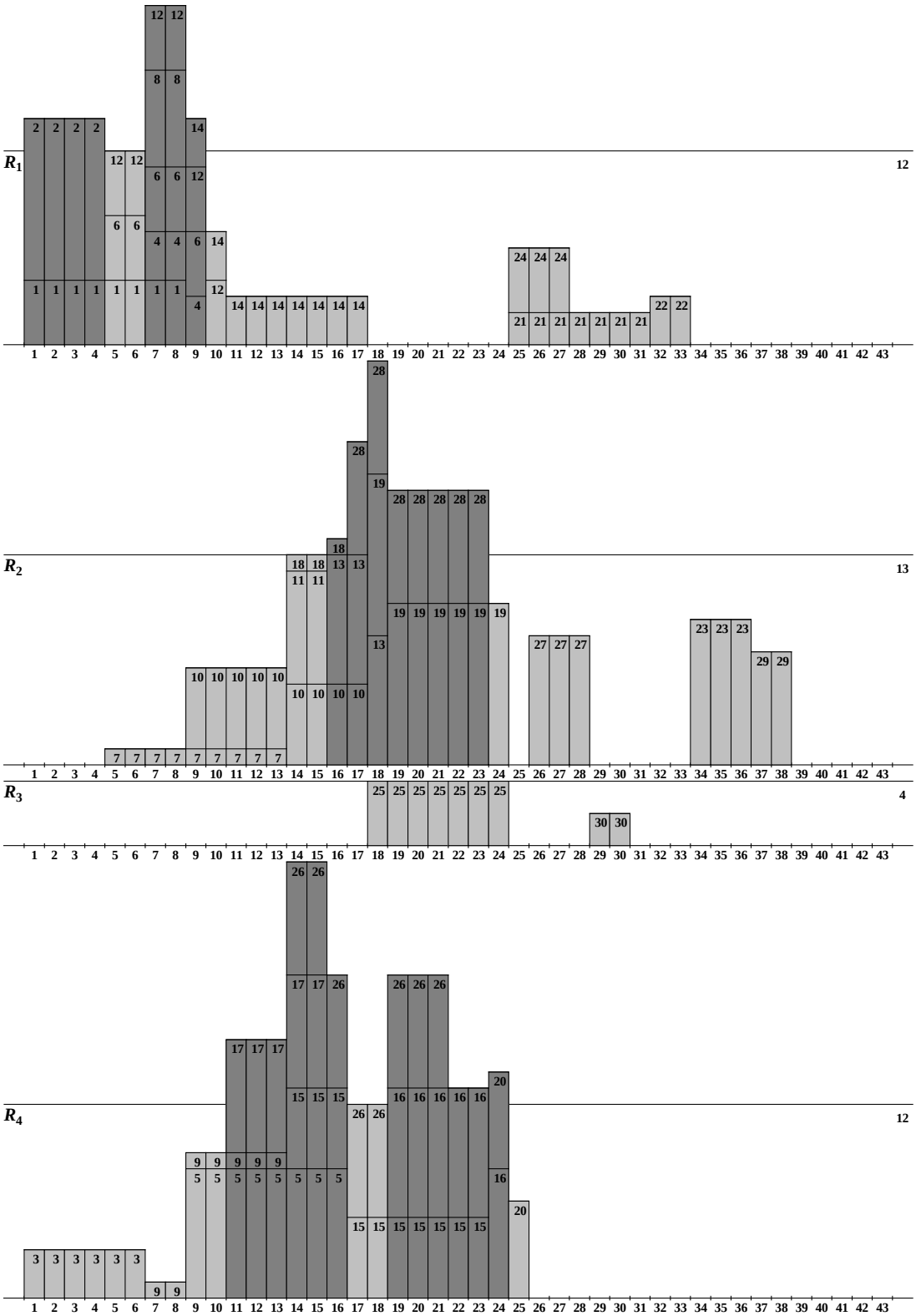
A 16. és 17. ábra a J30-1-1 eset legkorábbi ütemezését ábrázolja, véletlenszerűen generált pénzmozgásokkal. Ebben az esetben a projekt végét a „legjobb” (időtartam minimalizáló) projekt időtartamának megfelelően rögzítettük.

Az ábrákon, a korábban ismertetett ábrázolási módot alkalmaztuk: a tevékenységeket téglalapokkal, a pénzmozgásokat körökkel ábrázoltuk, az elsőbbségi feltételeket egyenesekkel. A „normál” elsőbbségi feltételeket sima vonallal, a „nem triviális”, pénzmozgásokhoz kapcsolódó elsőbbségi feltételeket pontozott vonalakkal szemléltetjük. A valós tevékenységeket számokkal $\{1,2,\ldots,30\}$, a pénzmozgás eseményeket nyomtatott nagybetűkkel $\{A,B,\ldots,J\}$ jelöljük. Az időperiódusok száma: $\{1,2,\ldots,44\}$. A kezdő- és végső áltevékenységeket a $>$ és $<$ szimbólumokkal jelöljük. Az erőforrás felhasználási diagramokon az erőforráskorlátokon felüli túlfogyasztást sötétszürke téglalapokkal jelezzük.

16. ábra A legkorábbi ütemezés a J30-1-1 esetre



17. ábra A legkorábbi ütemezés erőforrás profilja



A legkorábbi ütemezés pénzmozgásainak részletes leírása a 20. Táblázatban található. A legkorábbi ütemezés nem erőforráskorlátos, a pénzmozgása: $NPV = 178.04$.

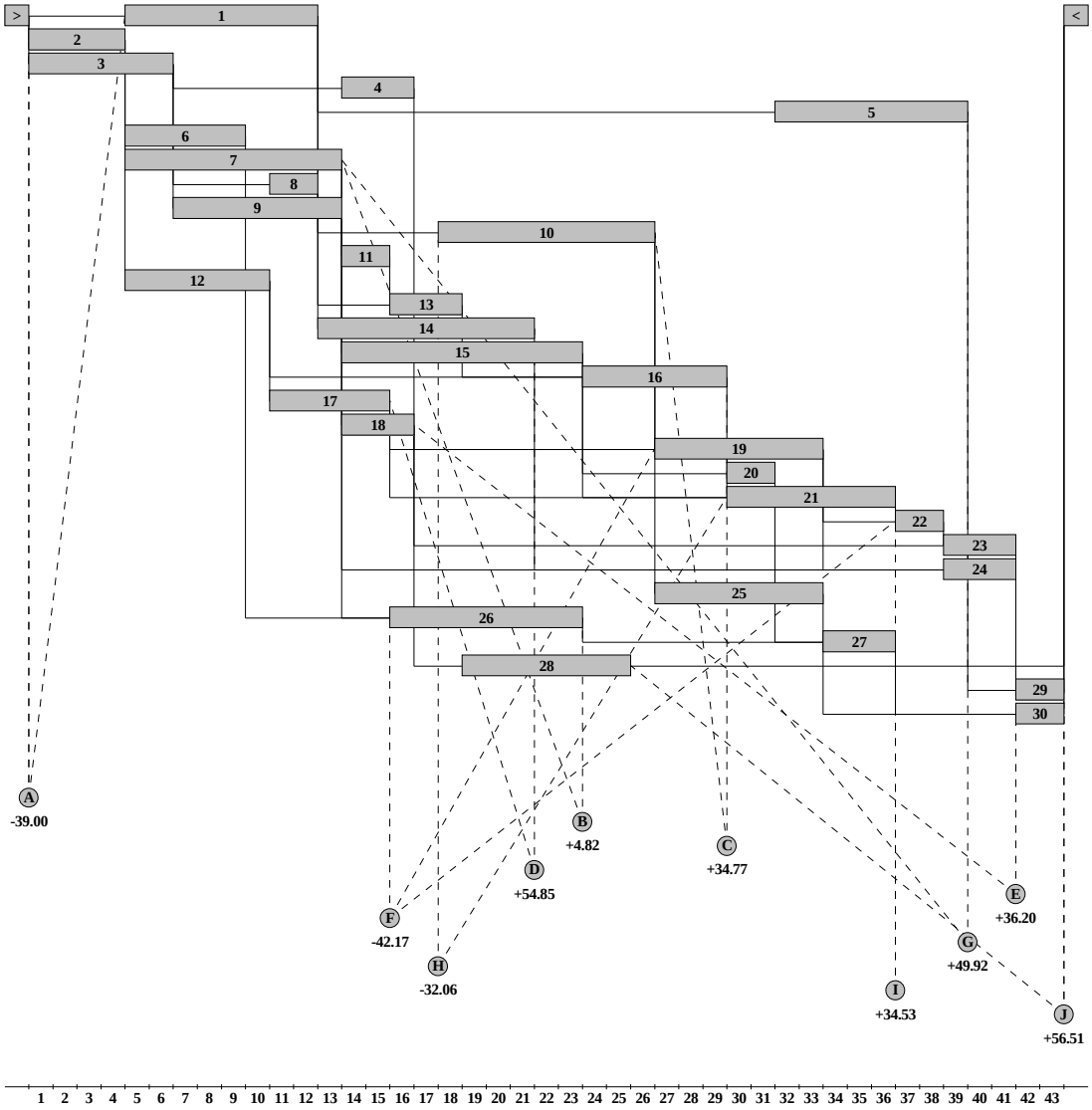
Táblázat 20

Pénzmozgás a legkorábbi ütemezés esetében

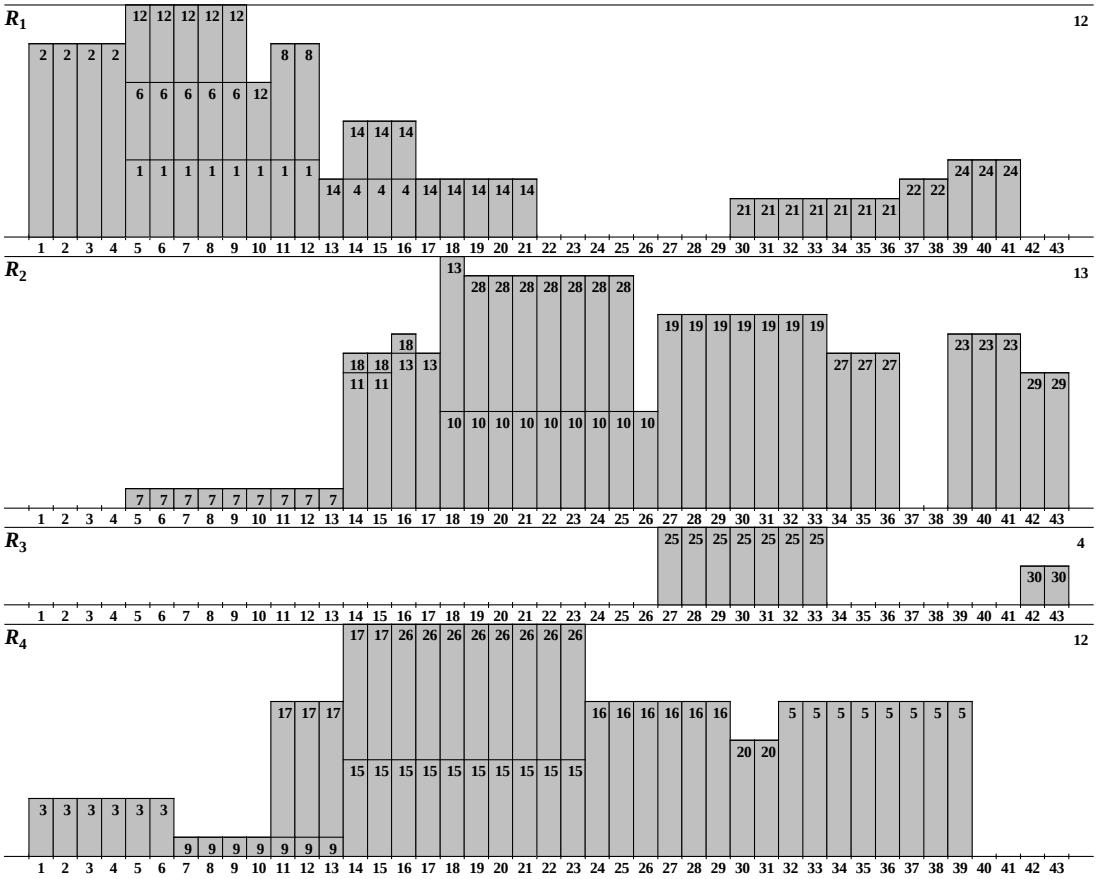
Esemény	Pénzmozgás esemény	Véletlenszerűen generált pénzmozgás	Pillanatnyi pénzmozgás	Kumulált pénzmozgás	Időperiódus száma
1	A	-39	-39.00	-39.00	0
2	B	6	4.91	-34.09	20
3	C	46	36.55	2.46	23
4	D	67	57.09	59.55	16
5	E	54	38.05	97.61	35
6	F	-49	-43.03	54.58	13
7	G	73	62.83	117.41	15
8	H	-38	-35.08	82.33	8
9	I	49	36.30	118.63	30
10	J	86	59.40	**178.04**	37

A 18. és 19. ábrán a „legjobb” ütemezést szemléltetjük, ahol „legjobb” alatt az időtartam minimalizáló ($\overline{T} = 44$) erőforráskorlátos ütemezést értjük, melyre a véletlenszerűen generált pénzmozgás események nettó jelenérték mértéke maximális ($NPV^* = 158.36$).

18. ábra A „legjobb” ütemezés



19. ábra A legjobb ütemezés erőforrás profilja a kiegyenlítés előtt



A „legjobb ütemezésre” vonatkozó pénzmozgás adatok részletes ismertetése a 21. Táblázatban található.

Táblázat 21

Pénzmozgás a „legjobb” ütemezés esetében

Esemény	Pénzmozgás esemény	Véletlenszerűen generált pénzmozgás	Pillanatnyi pénzmozgás	Kumulált pénzmozgás	Időperiódus száma
1	A	-39	-39.00	-39.00	0
2	B	6	4.82	-34.19	22
3	C	46	34.77	0.58	28
4	D	67	54.86	55.44	20
5	E	54	36.20	91.63	40
6	F	-49	-42.18	49.46	15
7	G	73	49.92	99.38	38
8	H	-38	-32.06	67.32	17
9	I	49	34.53	101.85	35
10	J	86	56.51	**158.36**	42

Az erőforrás kiegyenlítés hatását a 22. Táblázatban mutatjuk be. A táblázat az erőforrás egységek újraindítási eseményeinek számát tartalmazza. Kiegyenlítés után ez a szám várakozásainknak megfelelően kisebb.

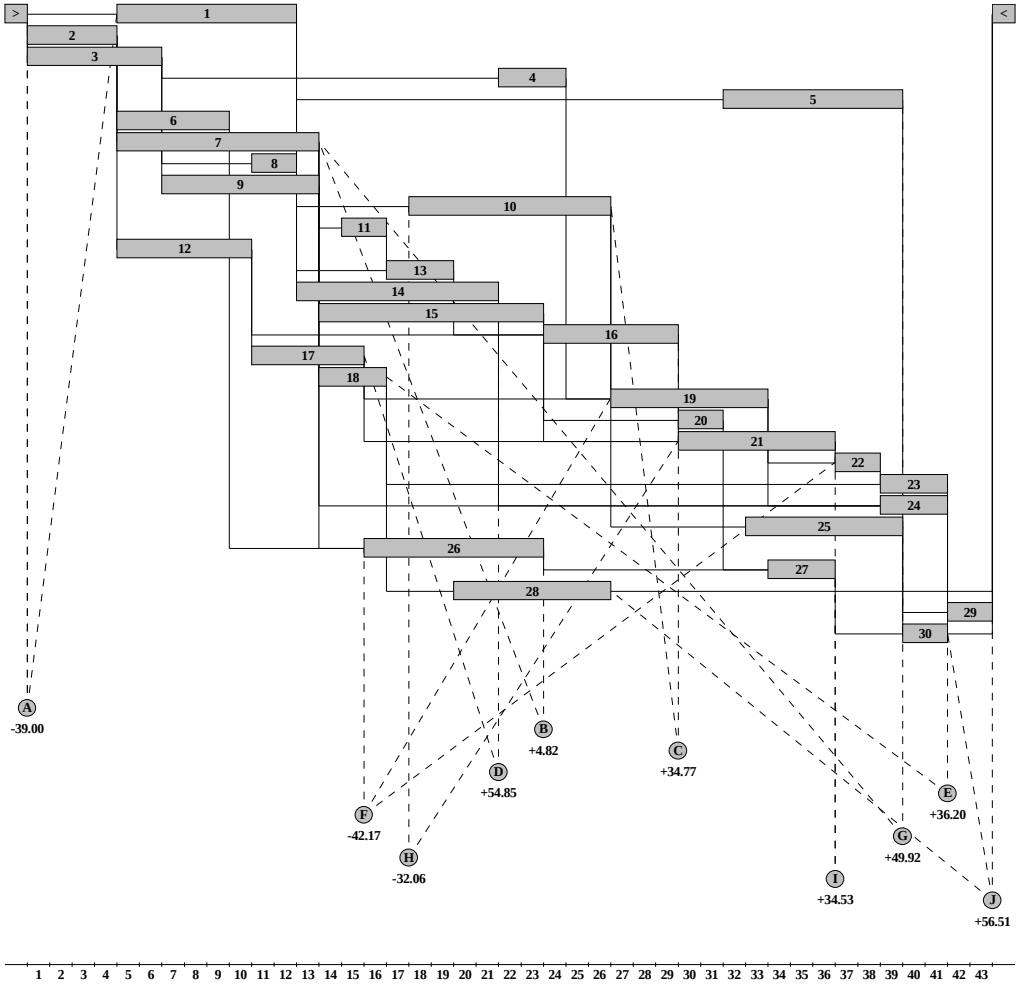
Táblázat 22

Az erőforrás kiegyenlítés hatása

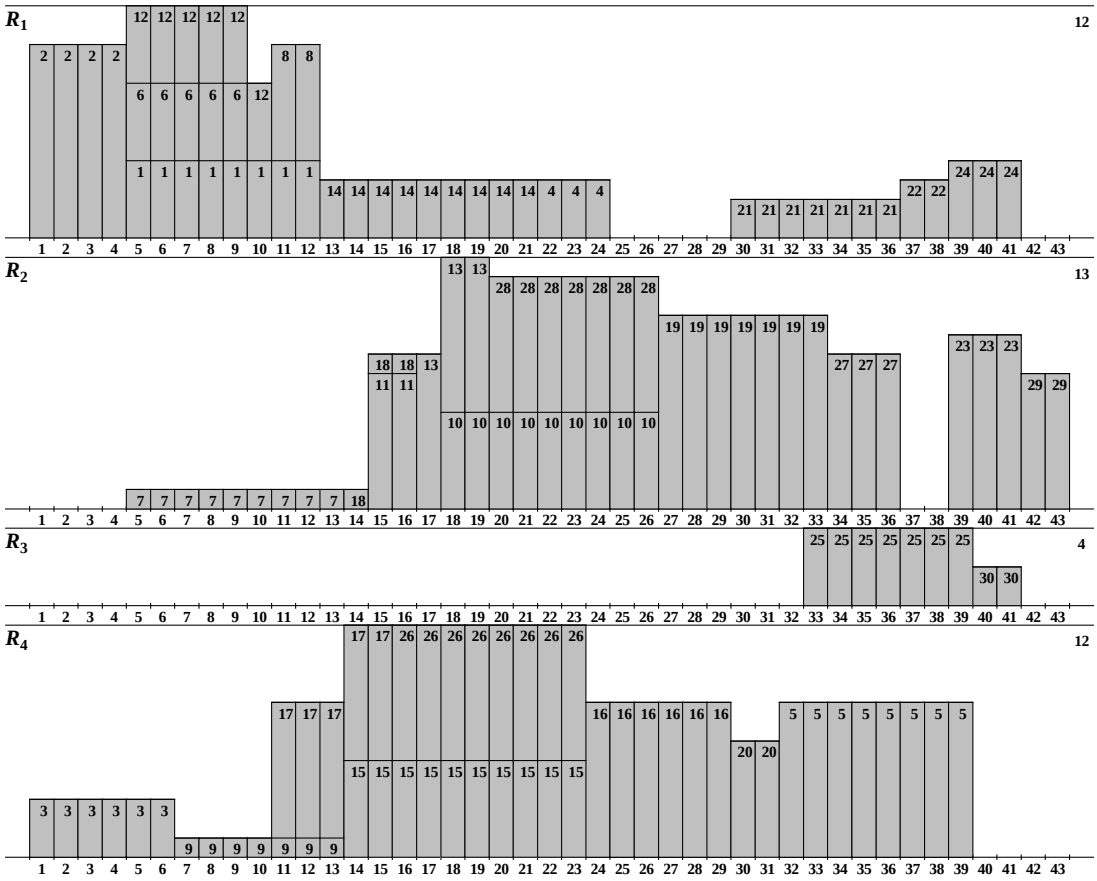
Erőforrás	Kiegyenlítés előtt	Kiegyenlítés után
1	21	18
2	28	22
3	6	4
4	16	16
	71	**60**

A 20. és 21. ábrán legjobb ütemezést szemléltetjük a kiegyenlítés után

20. ábra A legjobb ütemezés a kiegyenlítés után

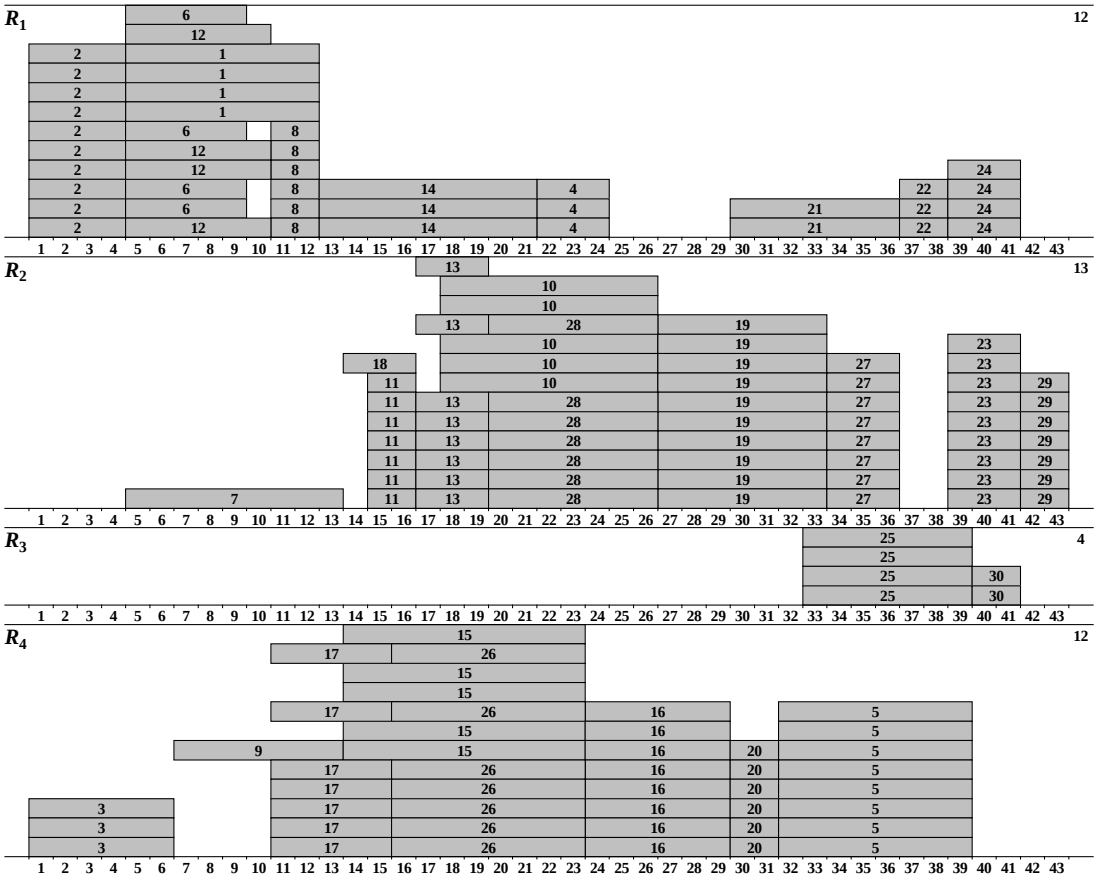


21. ábra A legjobb ütemezés erőforrás profilja a kiegyenlítés után



A 22. ábrán annak a „ legjobb” ütemezésnek az erőforrás felhasználását szemléltetjük, melyet a „dedikált erőforrás hozzárendelési” probléma kezelése után kapunk.

22. ábra Egy „transfer-free” transzformáció a legjobb ütemezés esetén a kiegyenlítés után



6. Összegzés

Disszertációnkban egy új harmóniakereső metaheurisztikát mutattunk be, amely a minimális időtartamú erőforráskorlátos ütemezések halmazán a projekt nettó jelenértékét maximalizálja.

Először a probléma terület legfontosabb fogalmait tekintettük át.

A továbbiakban ismertettük a(z) - elsősorban nemzetközi - szakirodalom korábbi eredményeit, a reguláris, illetve irreguláris modelleket, az erőforráskorlátos, illetve erőforráskorlát nélküli megoldásokat, az egzakt és a heurisztikus megoldásokat. Vizsgáltuk a korábbi időtartam minimalizáló és nettó jelenérték maximalizáló eljárásokat is, mivel disszertáció-béli algoritmusunk két ilyen típusú célfüggvénnyel dolgozik. Kiemelten foglalkoztunk a heurisztikus, és azon belül a metaheurisztikus megoldásokkal, mivel a probléma természetéből adódóan az ilyen megoldások bizonyulnak hatékonynak. Az irodalomban bemutatott eredményekkel támasztottuk alá azt az állításunkat, hogy a disszertációban tárgyalt problémára a jövőben a leghatékonyabb megoldásokat a metaheurisztikák között érdemes keresnünk. Bemutattuk a területen jelenleg egyik leghatékonyabbnak bizonyuló heurisztika típust, a harmónia kereső metaheurisztikát.

A továbbiakban ismertettük új harmóniakereső algoritmusunkat, amely a minimális időtartamú erőforráskorlátos ütemezések halmazán a projekt nettó jelenértékét maximalizálja. A bemutatandó metaheurisztika a Csébfalvi [2007] által a minimális időtartamú erőforráskorlátos ütemezések időtartamának meghatározására és a tevékenységek ennek megfelelő ütemezésére kifejlesztett harmóniakereső

metaheurisztika továbbfejlesztése. A továbbfejlesztés egyik legfontosabb pontja, hogy az új eljárás a rejtett erőforrás felhasználási konfliktusokat konfliktus javító elsőbbségi relációk beépítésével oldja fel, így az alkalmassá válik a másodlagos szempont kezelésére is. A továbbfejlesztés másik lényeges pontja annak a ténynek a felhasználása, hogy a nemkorlátos nettó jelenérték maximalizálási probléma polinomiális idő alatt megoldható, ha a megelőző-követő relációkat teljesen unimoduláris formulával írjuk le. Így algoritmusunk megoldási ideje elfogadható nagyságúvá válik, sőt a legtöbb esetben kifejezetten gyorsnak bizonyul az algoritmus.

Részletesen ismertettük modellünket, az algoritmus főbb lépéseit, végigkövettük egy konkrét projektre alkalmazva az ütemezés lépéseit, majd ismertettük eljárásunk pszeudokódját.

Mivel legjobb tudomásunk szerint ilyen elsődleges - másodlagos szempont szerinti (bi-criteria) optimalizálásos megoldás korábban nem született a pénzmozgásos RCPSP problémára, eljárásunk eredményeit nem tudjuk összehasonlítani korábbi megoldások eredményeivel. Az ajánlott metaheurisztika hatékonyságának és életképességének szemléltetésére számítási eredményeket adunk a jól ismert és népszerű PSPLIB tesztkönyvtár J30 részhalmazán futtatva.

Végül megadtuk a lehetséges továbbfejlesztési irányokat, megemlítve a sikerrel járó, és az eredményt nem adó ötleteket is.

Irodalomjegyzék

Abbasi, B., Shadrokh, S., Arkat, J., (2006). Bi-objective resource-constrained project scheduling with robustness and makespan criteria. *Applied Mathematics and Computation*, 180, 146-152.

Agarwal, R., Tiwari, M., K., Mukherjeee, S., K. (2007). Artificial immune system based approach for solving resource constraint project scheduling problem. *Adv Manuf Technol* 34, 584-593.

Ahuja, H., N. (1976). Construction Performance Control by Networks, Wiley, New York

Al-Fawzan, M., A., Haouari, M. (2005). A bi-objective model for robust resource-constrained project scheduling. *International Journal of Production Economics*, 96(2), 175-187.

Alvarez-Valdés, R., Tamarit, J. M. (1994). Heuristic algorithms for resource-constrained project scheduling: a review and an empirical analysis. In: Slowinski, R., Weglarz, J., editors, Advances in project scheduling, Elsevier Science Publishers. Amsterdam, Springer, 113–134.

Baroum S., M., Patterson, J., H. (1999). An exact solution procedure for maximizing the net present value of cash flows in a network, In: Weglarz J., editor, Project scheduling: recent models, algorithms and applications, Boston, Kluwer Academic Publishers, 107–134**.**

Baroum, S., M., Patterson, J., H. (1996). The development of cash flow weight procedures for maximizing the net present value of a project. *Journal of Operations Management*, 14, 209-227.

Bell, C., E., Park, K. (1990). Solving Resource-Constrained Project Scheduling Problems by A* Search., *Naval Research Logistics*, 37, 1, 61-84.

Bey, R., P., Doersch, R., H., Patterson, J., H. (1981). The net present value criterion: Its impact on project scheduling. *Project Managment Quarterly*, 12 (2), 35-45.

Blazewicz, J., Lenstra, J., Rinnooy Kan, A. (1983). Scheduling subject to resource constraints: Classification and complexity, *Discrete Applied Methematics*, 5, 11-24.

Böttcher, J., Drexl, A., Kolisch, R., Salewski, F., (1999). Project scheduling under partially renewable resource constraints. *Management Science*, 45 (4), 543–559.

Brucker P., Drexl, A., Möhring, R., Neumann, K., Pesch, E. (1999). Resource-constrained project scheduling: notation, classficitaion, models, and methods. *European Journal of Operational Research*, 112 (1), 3-41.

Burgess, A. R. and Killebrew, J. B. (1962). Variation in activity level on a cyclical arrow diagram. *Journal of Industrial Engineering*, 13: 76-83.

Cavalcante, C.C.B., Carvalho de Souza, C, Savelsbergh, M.W.P., Wang, Y., Wolsay, L.A (2001). Scheduling projects with labor constraints, *Discrete Applied Mathematics*, 112, 27-52.

Christofides, N., Alvarez-Valdes, R., and Tamarit, J.M. (1987). Project scheduling with resource constraints: A branch and bound approach, *European Journal of Operational Research*, 29 (3), 262-273.

Csébfalvi, A., Láng, B. An improved hybrid method for the resource-constrained project scheduling problem with discounted cash flows, *Pollack Periodica*, megjelenés alatt

Csébfalvi, G. (2002). A New Exact Implicit Enumeration Procedure for the Resource Constrained Project Scheduling Problem with Discounted Cash Flows, *Proc. 2st Annual McMaster Optimization Conference*: Theory and Applications (MOPTA 02), Hamilton, Ontario, Canada, August 1-3.

Csébfalvi, G. , Konstantinidis, P. (1998). A new exact resource balancing procedure for the multiple resource-constrained project scheduling problem, in Proceedings of APMOD '98, Limasol, Cyprus, 11-13 March.

Csébfalvi, G., (2007). Sounds of Silence: A harmony search metaheuristic for the resource-constrained project scheduling problem. *European Journal of Operational Research*, (under reviewing process).

Csébfalvi, G., Eliezer, O., Láng, B., Levi, R. (2008). A conflict repairing harmony search metaheuristic and its application for bi-objective resource-constrained project scheduling problems, in "Project Management and Scheduling 2008", F. S. Serifoglu, Ü. Bilge (Editors), Istanbul, Turkey, 60-63.

Csébfalvi, A., Csébfalvi G. (2010). A popular but theoretically wrong and misleading resource usage visualization in resource-constrained project scheduling, *Annals of Operations Research*, Special Volume

Davis E.W., (1969). An exact algorithm for the multiple constrained project scheduling problem. PhD thesis, Yale University.

Davis, E., W., (1973). Project scheduling under resource constraints: Historical review and categorization of procedures, *AIIE Transactions*, 5 (4), 297-313.

Davis, E., W., Patterson, J., H. (1975). A comparison of heuristic and optimum solutions in resource constrained project scheduling, *Management Science*, 21 (8), 944-955.

Demeulemeester, E., Herroelen, W. (2002): *Project Scheduling*, A Research Handbook, Kluwer Academic Publishers, Boston Dordecht London

Demeulemeester, E., Herroelen, W., (1992). A Branch-and-Bound Procedure for the Multiple Constrained Resource Project Scheduling Problem, *Management Science*, 38 (12), 1803-1818.

Demeulemeester, E., Herroelen, W., Elmeghraby, S., E. (1996). Optimal procedures for the discrete time/cost trade-off problem in project networks, *European Journal of Operational Research*, 88, 50-68

Doersch, R., H., Patterson, J., H. (1977). Scheduling a project to maximize its present value: a zero-one programming approach. *Management Science*, 23 (8), 882-889.

Dorigo, M., (1992). Optimization, Learning and Natural Algorithms, *Ph.D.Thesis*, Politecnico di Milano, Italy.

Eberhart, R., Kennedy, J., (1995). A new optimizer using particle swarm theory. *Micro Machine and Human Science*, 4-6, 39-43.

Elmeraghby, S., E., Herroelen, W., S. (1990). The scheduling of activities to maximize the net present value of projects. *European Journal of Operational Research*, 49, 35-49.

Erenguc, S., Tufekci, S., S., Zappe, C., J., (1993). Solving time/cost trade-off problems with discounted cash flows using generalized benders decomposition, *Naval Research Logistics*, 40, 25-50.

Erol, O., K., Eksin, I. (2006). A new optimization method: Big Bang–BigCrunch, *Advances in Engineering Software*, 37, 106-11.

Fogel, L., Owens, A. J., Walsh, M. J., Artificial Intelligence through Simulated Evolution, Wiley, New York, USA.

Galperin, Y., Fishman, V., Gibiansky, L. (2006). Method for optimizing net present value of a cross-selling marketing campaign. *US Patent 6993493* - Issued on January 31, 2006. (A dokumentum elérhető a http://www.patentstorm.us/patents/6993493/fulltext.html címen.)

Glover, F., (1986) Future Paths for Integer Programming and Links to Artificial Intelligence, *Computers & Operations Research*, 13 (5), 533-549.

Goldberg, D. E., (1989). Genetic Algorithms in Search, *Optimization and Machine Learning*, Boston , Kluwer Academic Publishers.

Grinold, R., C. (1972). The payment scheduling problem. *Naval Research Logistics Quarterly*, 19 (1), 123-136.

Hajdu, M. (1997). Network Scheduling Techniques for Construction Project Management, Kluwer Academic Publisher, Boston, US.

Hartmann, S., Kolisch, R. (2000). Experimental evaluation of state-of-the art heuristics for the resource constrained project scheduling problem. *European Journal of Operational Research*, 127 (2), 394-407.

Herroelen W., Demeulemeester E., De Reyck B., (1999). A classification scheme for project scheduling, In: Weglarz J., editor, Project scheduling: recent models, algorithms and applications, Boston, Kluwer Academic Publishers, 1–26**.**

Herroelen, W., S., Gallens, E. (1993). Computational experience with an optimal procedure for the scheduling of activities to maximize the net present value of projects, *European Journal of Operational Research*, 65 (3), 274-277.

Holland, J. H.,(1975). *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor.

Icmeli, O., Erenguc, S. (1994). A tabu search procedure for the resource constrained project scheduling problem with discounted cash flows, *Computers and Operations Research*, 21 (8), 841-853.

Icmeli, O., Erenguc, S. (1996 b). A branch and bound procedure for the resource constrained project scheduling problem with discounted cash flows, *Management Sciences*, 42 (10), 1395-1408.

Icmeli, O., Erenguc, S., (1996 a). The resource constrained time/cost tradeoff project scheduling problem with discounted cash flows. *Journal of Operations Management*, 14 (3), 255-275.

Kaveh, A., Talatahari, S., (2010). Optimum design of skeletal structures using imperialist competitive algorithm, *Computers and Structures*, megjelenés alatt

Kelley, J.(1963). *The critical-path method: Resources planning and scheduling*, in: Industrial Scheduling, eds. J.F. Muth and G.L. Thompson, Prentice-Hall, pp. 347–365.

Kennedy, J., Eberhart, R., (1995). Particle swarm optimization, *Proceedings of IEEE International Conference on Neural Networks*, IV, 1942-1948.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., (1983). Optimization by Simulated Annealing, *Science*, 220 (4598), 671-680.

Kolisch, R., Hartmann, S. (1999). Heuristic algorithms for the resource-constrained project scheduling problem, In: Weglarz J., editor, Project scheduling: recent models, algorithms and applications, Boston, Kluwer Academic Publishers, 147–178**.**

Kolisch, R., Hartmann, S., (2000). Experimental investigation of heuristics for Resource-Constrained Project Scheduling. *European Journal of Operational Research*, 127, 394-407.

Kolisch, R., Hartmann, S., (2006). Experimental investigation of heuristics for Resource-Constrained Project Scheduling: An Update. *European Journal of Operational Research*, 174, 23-37.

Kolisch, R., Padman, R., (2001). An integrated survey of deterministic project scheduling, *Omega*, 29 (3), 249-272.

Kolisch, R., Sprecher, A., (1996). *PSPLIB – a project scheduling library*, European Journal of Operational Research, 96, 205-216.

Kolisch, R., Sprecher, A., Drexl, A., (1992). Characterization and generation of a general class of resource-constrained project scheduling problems: easy and hard instances. *Technical Report 301*, Manuskripte aus den Instituten fIur Betriebswirtschaftslehre der UniversitIat Kiel, 2547–2558.

Koza, J., R. (1992) : Genetic Programming: On the Programming of Computers by Means of Natural Selection. *MIT Press*

Láng, B (2009/a): A hybrid method for the resource-constrained project scheduling problem with discounted cash flows, *Conference paper*, CC 2009, The Twelfth International Conference on Civil, B.H.V. Topping, Y. Tsompanakis, (Editors), Civil-Comp Press, Stirlingshire, United Kingdom, paper 3, doi:10.4203/ccp.92.3

Láng, B. (2009/b) : A nettó jelenérték maximalizálása erőforráskorlátos projektekben - egy új harmóniakereső metaheurisztika, *Vezetéstudomány*, 2009, 10, 55-61

Láng, B. (2009/c): Hibrid eljárás a diszkontált pénzáramos erőforráskorlátos projekt ütemezési problémához, *Conference paper*, Országos Gazdaságinformatikai Konferencia, 2009

Láng, B. (2010/b): A robust hybrid method for the resource constrained project scheduling problem with discounted cash flows, *Pollack Periodica,*, megjelenés alatt

Láng, B. (2010/a): A Hybrid Method for the Resource-Constrained Project Scheduling Problem with Discounted Cash Flows and Strip Packing like Resource Constraints, *Conference paper*, CST2010 & ECT2010 Conferences in Valencia, Spain, 14-17 September 2010, megjelenés alatt

Lee, K. S., Geem, Z. W. (2005). A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 194, 3902–3933.

Lee, K. S., Geem, Z. W., Lee, S. H., Bae, K. W. (2005). The harmony search heuristic algorithm for discrete structural optimization. *Engineering Optimization*, 37 (7), 663–684.

Li, K., Y., Willis, R., J. (1992). An iterative scheduling technique for resource constrained project scheduling, *European Journal of Operational Research*, 56, 370-379.

Lova et al. (2000). A multicriteria heuristic method to improve resource allocation in multi-project scheduling. *European Journal of Operational Research,*127: 408-424.

Mahdavi, M., Fesanghary, M., Damangir, E. (2007). An improved harmony search algorithm for solving optimization problems. *Applied Mathematics and Computation*, 188, 1567-1579.

Maravelias, C., T., Grossmann, I., E. (2004). Optimal resource investment and scheduling of tests for new product development. *Computers and Chemical Engineering*, 28, 1021-1038.

Mészáros, Cs. (1996). The Efficient Implementation of Interior Point Methods for Linear Programming and their Applications, *PhD Thesis,* Eötvös Loránd University of Sciences, Hungary.

Mülhenbein, H., Paaß, G., (1996). From recombination of genes to the estimation of distribution I. Binary parameters, *Parallel Problem Solving from Nature*, Lectures Notes in Computer Science 1411, 178-187.

Neumann, K. and Zimmermann, J. (1999). Resource leveling for projects with schedule-dependent time windows. *European Journal of Operational Research*, 117(3): 591-605.

Neumann, K., Schwindt, C., Zimmermann, J. (2002). Project scheduling with Time windows and Scarce Resources, Springer –Verlag Berlin Heiderberg New York

Neumann, K., Schwindt, C., Zimmermann, J.,: (2001.) Project scheduling with Time windows and Scarce Resources, *Omega*, 29, 249–272.

Osman, I. H., Laporte, G., (1996) Metaheuristics: A bibliography. *Annals of Operations Research*, 63 (5), 511-623.

Özdamar, L., Ulusoy, G. (1996): An iterative local constraint based analysis for solving the resource constrained project scheduling problem. *Journal of Operations Management* , 14 (3), 193-208.

Padman, R, Smith-Daniels, D. E., Smith-Daniels, V. L. (1997). Heuristic scheduling of resource-constrained projects with cash flows. *Naval Research Logistics*, 44 (4), 365–381.

Padman, R., Smith-Daniels, D., E. (1993). Early-tardy cost trade-offs in resource constrained projects with cash flows: An optimization-guided heuristic approach, *European Journal of Operational Research*, 64, 295-311.

Padman, R., Smith-Daniels, D., E., Smith-Daniels, V., L. (1990). Heuristic scheduling of resource constrained projects with cash flows, *Working paper*, The Heinz School of Public Policy and Management, Carnegie Mellon University, Pttsburg, 90-96.

Palpant, M., Artigues, C. and Michelon, P. (2004). LSSPER: Solving the resource–constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131, pp. 237-257.

Patterson, J. H, Huber, W. D., (1974). A horizon-varying, zero-one approach to project scheduling. *Management Science*, 20 (6), 990–998.

Patterson, J. H, (1984). A Comparison of Exact Procedures for Solving the Multiple Constrained Resource Project Scheduling Problem. *Management Science* 30, 7, 854-867.

Patterson, J.. H., Talbot, F., B., Slowinski, N., Weglarz, J. (1990). Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling poblems, *European Journal of Operational Research*, 49, 68-79.

Patterson, J.H., Roth, G.W. (1976). Scheduling a project under multiple resource constraints: A 0-1 programming approach. *AIEE Transactions*, 8, 449-455.

Pham D T, Ghanbarzadeh A, Koç E, Otri S, Rahim S, Zaidi M. (2206). The bees algorithm – a novel tool for complex optimisation problems. *Proceedings of 2nd International Virtual Conference on Innovative Production Machines and Systems*, 454–461.

Pinder, J., P., Maruchek, A., S. (1996).Using discounted cash flow heuristics to improve project net present value. *Journal of Operations Management*, 14, 229-240.

Pritsker, A.A., Waters, L.J., Wolfe, P.M. (1969), Multiproject scheduling with limited resources; A 0-1 approach. *Management Science*, 16, 93-108.

Rechenberg, I., (1965). Cybernetic Solution Path of an Experimental Problem. *Royal Aircraft Establishment*, Library Translation, 1122.

Russel, A., H. (1970). Cash flows in networks. *Management Science*, 16 (5), 357-373.

Russell, R. A. (1986). A comparison of heuristics for scheduling projects with cash flows and resource restrictions. *Management Science*, 32(10), 1291–1300.

Schrivjer, A., (1987). Theory of linear and integer programming, New York, John Wiley, 266-267.

Slowinski, R., Weglarz, J., (1978). Solving the general project scheduling problem with multiple constrained resources by mathematical programming. In: Optimization Techniques. Lecture Notes in Control and Information Sciences, 7 (2), Springer-Verlag, Berlin, Heidelberg, New York, 278-288.

Smith-Daniels, D., E. (1986). Summary measures for predicting the net present value, of a project, *Working paper*, Collage of St. Thomas, St. Paul, Minnesota.

Smith-Daniels, D., E., Aquilano, N., J. (1987). Using a late-start resource-constrained project schedule to improve project net present value, *Decision Scinces*, 18 (4), 617-630.

Smith-Daniels, D., E., Smith-Daniels, V., L. (1987). Maximizing the net present value of a project subject to materials and capital constraints, *Journal of Operations Management*, 7 (2), 33-45.

Stinson, J.P., Davis, E.W., Khumawala, B.H. (1978), Multiple resource-constrained scheduling using branch and bound. *IIE Transactions*, 10 (3), 252-259.

Szendrői, E. (2009): A hybrid method for the multi-mode resource-constrained project scheduling problem, *Conference paper*, CC 2009, The Twelfth International Conference on Civil , Structural and Environmental Engineering Computing

Talbot, F. B., Patterson, J. H., (1978). An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource Constrained Scheduling Problems. *Management Science*, 24 (11), 1163-1174.

Talbot, F., B., (1982).: Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science*, 28 (10), 1197-1210.

Tavares, V., L. (1986). Multicriteria scheduling of a railway renewal program. *European Journal of Operational Research*, 25, 395-405.

Tavares, V. L. (1987). Optimal resource profiles for program scheduling. *European Journal of Operational Research*, 29: 83-90.

Tormos, P., Lova, A. (2001). A competitive heuristic solution technique for resource-constrained project scheduling. *Annals of Operations Research* ,102, 65–81.

Valls, V., Ballestin, F., Quintanilla, S. (2005). Justification and RCPSP: A technique that pays. *European Journal of Operational Research*, 165, 375–386.

Vasebi, A., Fesanghary, M., Bathae, S. M. T. (2007). Combined heat and power economic dispatch by harmony search algorithm, *Electrical Power and Energy Systems*, 29 (10), 713-719.

Viana, A., de Sousa, J., P. (2000). Using metaheuristics in multiobjective resource constrained project scheduling, *European Journal of Operational Research*, 120 (2), 359-374.

Vofl, S., Martello, S., Osman, I. H., Roucairol, C. (1999). *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization*. Dordrecht, The Netherlands, Kluwer Academic Publishers.

Yang, X. S. (2008). Nature-Inspired Metaheuristic Algorithms, *Luniver Press*

Yang, B., Geunes, J., O'Brian, W. J. (2001). Resource-constrained Project Scheduling: Past Work and New Directions. *Research Report*, Department of Industrial ans Systems Engineering, University of Florida.

Yang, K., K., Talbot, F., B., Patterson, J., H. (1992). Scheduling a project to maximize its net present value: in integer programming approach. *European Journal of Operational Research*, 64, 188-198.

Younis, M.A. and B. Saad, (1996). Optimal resource leveling of Multi resource projects. *Computers and Industrial Engineering*.

Zhu, P, Padman, R. (1999). A metaheuristic scheduling procedure for resource constrained projects with cash flows. *Naval Research Logistics*, 46, 1-18.

Zimmermann, J. and Engelhart, H. (1998). Lower Bounds and Exact Methods for Resource Leveling Problems. *Report WIOR-517,* University of Karlsruhe.