LÁNG Blanka

# A NETTÓ JELENÉRTÉK MAXIMALIZÁLÁSA ERŐFORRÁS-KORLÁTOS PROJEKTEKBEN

## – EGY ÚJ HARMÓNIAKERESŐ METAHEURISZTIKA

**Ebben a tanulmányban a szerző egy új harmóniakereső metaheurisztikát mutat be, amely a minimális időtartamú erőforrás-korlátos ütemezések halmazán a projekt nettó jelenértékét maximalizálja. Az optimális ütemezés elméletileg két egész értékű (nulla-egy típusú) programozási feladat megoldását jelenti, ahol az első lépésben meghatározzuk a minimális időtartamú erőforrás-korlátos ütemezések időtartamát, majd a második lépésben az optimális időtartamot feltételként kezelve megoldjuk a nettó jelenérték maximalizálási problémát minimális időtartamú erőforrás-korlátos ütemezések halmazán. A probléma NP-hard jellege miatt az egzakt megoldás elfogadható idő alatt csak kisméretű projektek esetében képzelhető el. A bemutatandó metaheurisztika a Csébfalvi (2007) által a minimális időtartamú erőforrás-korlátos ütemezések időtartamának meghatározására és a tevékenységek ennek megfelelő ütemezésére kifejlesztett harmóniakereső metaheurisztika továbbfejlesztése, amely az erőforrás-felhasználási konfliktusokat elsőbbségi kapcsolatok beépítésével oldja fel. Az ajánlott metaheurisztika hatékonyságának és életképességének szemléltetésére számítási eredményeket adunk a jól ismert és népszerű PSPLIB tesztkönyvtár J30 részhalmazán futtatva. Az egzakt megoldás generálásához egy korszerű MILP-szoftvert (CPLEX) alkalmaztunk.[1]**

*Kulcsszavak:* **metaheurisztika, harmóniakeresés, jelenérték**

### Szimbólumlista

| | |
|---|---|
| $A_t$ | A $t$ időperiódusban aktív tevékenységek halmaza |
| $D_i$ | Az $i$-edik tevékenység nem megszakítható hossza |
| $E$ | 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 erőforrásigény az $r$-edik erőforrásból, $r \in \{1,...R\}$ |
| $R$ | Az erőforrások száma |
| $R_r$ | Az $r$-edik erőforrás felső korlátja |
| $T$ | A tevékenységek hosszának 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 pénzárama |
| $C_{i,t}$ | A $t$ időperiódusban kezdődő, $i$-edik tevékenység pénzárama a projekt kezdetére visszadiszkontálva, $i = \{1,2,...N\}$, $t \in Ti$ |
| $NPV$ | A projekt nettó jelenértéke |
| $U_{t,r}$ | A $t$ időpontban, az $r$-edik erőforrásból felhasznált mennyiség |
| $X_i$ | Az $i$-edik tevékenység kezdési időpontja |
| $\underline{S_i}$ | Az $i$-edik tevékenység legkorábbi kezdési időpontja a nem korlátos, csak elsőbbségi korlátokat kielégítő esetben |
| $\overline{S_i}$ | Az $i$-edik tevékenység legkorábbi kezdési időpontja a nem korlátos, csak elsőbbségi korlátokat kielégítő esetben |
| $T_i$ | Az $i$-edik tevékenység legkorábbi és legkésőbbi kezdési időpontja közé eső értékek |
| $T$ | A projekt időperiódusainak száma, $t \in \{1,2,...,T\}$ |
| $\underline{T}$ | Az elsőbbségi korlátokat kielégítő projekt minimális hossza |

## A modell

A modell megoldása során az erőforrás-korlátos 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 hosszának minimalizálása mellett a project nettó jelenértékét maximalizáljuk.*

Aki a feladatütemezés alapvető fogalmairól szeretne egy áttekintést, annak megfelelő összefoglalás például Kolisch-Padman (2001) cikke, aki hosszabb összefoglalást szeretne, Neumann et al. (2002), vagy Demeulemeester et al. (2002) könyve.

A modellben a következő feltevésekkel élünk: A projekt egy megvalósítási módú, $N$ darab valós tevékenységekből áll, $i \in i = \{1,2,...N\}$. A tevékenységek nem megszakíthatóak végrehajtásuk során, a hosszuk $D_i$. $X_0$ és $X_{N+1}$ 0 hosszúságú általvékenységek, mellyel a projekt kezdetét és végét jelezzük. A tevékenységek között megelőző-követő relációkat feltételezünk. Az erőforrásaink korlátosak és megújulóak, tehát a felső korlátot a projekt során állandónak feltételezzük. Minden tevékenységet erőforrásigényével is jellemzünk, mely a tevékenység végrehajtása során állandó. Jelöljük a tevékenységek hosszának összegét $T$-al, ami egy extrém gyenge felső korlát az elsőbbségi feltételeket és az erőforráskorlátokat kielégítő projekt hosszára, és rögzítsük a $T$+1 időperiódusba a projekt végét jelképező $N$+1-ik általvékenységet. Minden tevékenységhez pénzáramot társítunk, mely a tevékenység befejeződésekor lép fel, és a projekt kezdetére diszkontáljuk vissza.

A megoldandó modell:

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

$$X_i + D_i \le X_j,\ i \to j \in E \quad (2)$$

$$X_{N+1} = \overline{T} + 1 \quad (3)$$

$$X_i = \sum_{t\in T_i} X_{it} * t,\ T_i = \left\{t \middle| \underline{S}_i \le t \le \overline{S}_i\right\},\ i \in \{1,2,...,N\} \quad (4)$$

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

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

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

$$U_{tr} \le R_r,\ t \in \{1,2,...,T\},\ r \in \{1,2,...,R\} \quad (8)$$

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

Az (1) egyenlet a modell célfüggvényét írja le. Cél a projekt nettó jelenértékének maximalizálása. A projekt nettó jelenértékét úgy kapjuk, hogy a projekt összes tevékenységének pénzáramát a projekt kezdetére diszkontáljuk, és az így kapott értékeket összegezzük.

Az (2) egyenlet a tevékenységek között fennálló elsőbbségi feltételt írja le. Ha a $j$ tevékenységnek az $i$ tevékenység közvetlen előzménye, akkor a $j$ tevékenység nem kezdődhet el mindaddig, amíg az $i$ tevékenység be nem fejeződik.

Az $i = 0$ és az $i = N$+1 esetben tekintsük $X_0$ és $X_{N+1}$ *0* hosszúságú általvékenységeket, mellyel a projekt kezdetét és végét jelezzük. Az (3) egyenlet szerint projekt végét jelző $X_{N+1}$ általvékenységet a $\overline{T}$+1 időpontba rögzítjük.

Az (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.

Az (6) egyenlet megadja a $t$ időperiódusban aktív tevékenységek halmazát.

Az (7) egyenlet a $t$ időperiódusban az $r$-edik erőforrásból fogyasztott mennyiséget adja meg.

Az (8) egyenlet az erőforráskorlát kielégítését követeli meg. Az erőforráskorlátokat állandónak feltételezzük a projekt időtartama alatt.

Az (9) egyenlet a $t$ időperiódusban kezdődő, $i$-edik tevékenység pénzáramát adja meg a projekt kezdetére visszadiszkontálva. A pénzáram lehet pozitív, negatív, és 0, és feltevésünk szerint tevékenység befejezési időpontjában következik be.

Az 1. és 2. ábrán ugyanazon projekt két különböző ütemezését láthatjuk. Az első ütemezés nem erőforráskorlátos, a projekt nettó jelenértéke 2714,55. A 10. tevékenységre plusz elsőbbségi feltétel bevezetésével elérjük, hogy a második ütemezés már erőforráskorlátos, és a nettó jelenértéke még nőtt is: 2819,62, köszönhetően annak, hogy a nagy negatív pénzáramos tevékenységeket igyekeztünk késleltetni, a pozitív pénzárammal rendelkezőket pedig minél korábbi időpontra ütemezni.. A projekt hossza közben nem változik.

### *A harmóniakereső algoritmusok*

A korábbi évtizedekben számos egzakt és heurisztikus megoldás született a különböző feladatütemezési problémákra, azonban ezek kevéssé bizonyultak hatékonynak, újabb, erőteljesebb algoritmusokra lett szükség.

Az erőforráskorlátos NPV maximalizálás problémája egzakt megoldása már kevés tevékenységből álló projektek esetén is igen időigényes. Mivel egy projektütemezési probléma az életben gyakran több száz, vagy több ezer tevékenységből áll, a közepes és nagyméretű projektek megoldásához szükséges a heurisztikus megoldások igénybevétele. Megfelelő heurisztikák megtalálásával elfogadható futási idejű „jó minőségű" megoldáshoz juthatunk. Az utóbbi években a hatékony heurisztikákat a metaheurisztikák között keresték, azonban a „régi" metaheurisztikák nem bizonyultak elég hatékonynak, gyökeresen új eljárásra volt szükség.

Ilyen hatékony heurisztikának bizonyult az alábbi.

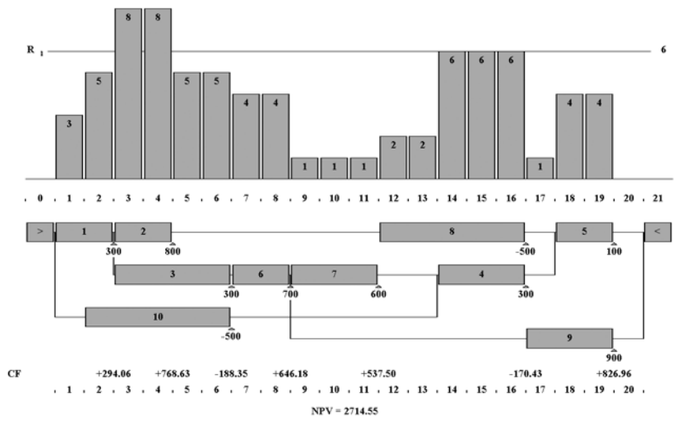
Az utóbbi években Lee et al. (2005) kifejlesztettek egy új harmóniakereső (harmony search, HS) igen hatékony metaheurisztikus algoritmust. Mivel a cikkbeli algoritmus is egyfajta HS, ezért az alábbiakban ennek a metaheurisztikának az ismertetése következik:





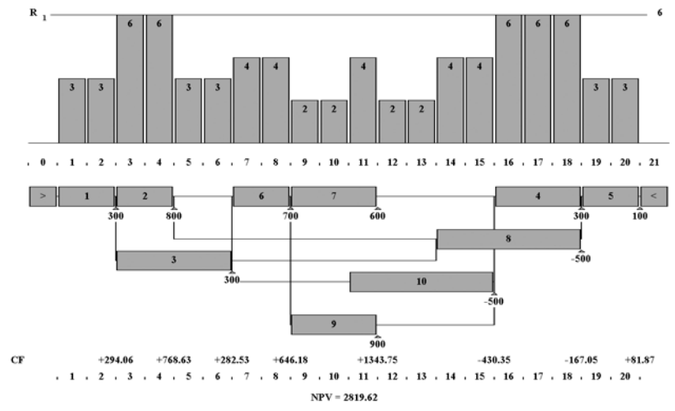
*1. ábra*

**Projekt diszkontált pénzáramokkal**




*2. ábra*

**Egy lehetséges optimális ütemezés**




A harmóniakeresés egyesíti az ismert metaheurisztikák struktúráját. A tabukeresésből átveszi a „tabulistát" (a HS-ben harmony memory, HM), a szimulált hűtésből a hőmérséklet/ráta változtatását (a HS-ben harmony memory consideration, HMCR). A genetikus algoritmushoz hasonlóan szimultán kezel egyszerre több vektort. Ez a vektorkezelési mód tág keresési teret eredményez, megkönnyíti a globális optimumhoz való konvergálást, így elkerüljük a 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 HS algoritmust a tökéletes harmóniaállapot keresésének a zenei eljárását használva alakították ki, a zenéből vett párhuzamokra épít. A zenei harmónia analóg az optimalizációs megoldásvektorral, a zenei rögtönzések analógok az optimalizációs technikák keresési sémáival.

Miben rejlik a harmóniakeresés hatékonyságának titka? A HS algoritmus előnyös tulajdonsága, hogy nem igényel kezdeti értékeket a döntési változókhoz. Továbbá, a gradiens keresés helyett a HS 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 hangmagasságszabályozó rátán (PAR, Pitch Adjusting Random Choosing, vagy más néven SAR, Sound Adjusting Rate) alapul. Korábbi metaheurisztikus optimalizációs algoritmusokkal összehasonlítva a HS algoritmus kisebb matematikai igényű, és jobban alkalmazkodik, könnyebben adoptálható az optimalizációs problémák különböző típusai számára. A fenti tulajdonságainak köszönhetően a HS jobb megoldásokat ad, mint az eddig ismert algoritmusok. Ezt a tényt számos kutató számítási próbákkal, esettanulmányokkal is alátámasztja (Lee et al., 2005; Mahdavi et al., 2006; Vasebi et al., 2006), hogy csak néhány példát

említsünk). A HS nagyon jól azonosítja a keresési tér magas teljesítményű részterületét, és elfogadható futási idővel rendelkezik.

A HS 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. 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.

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

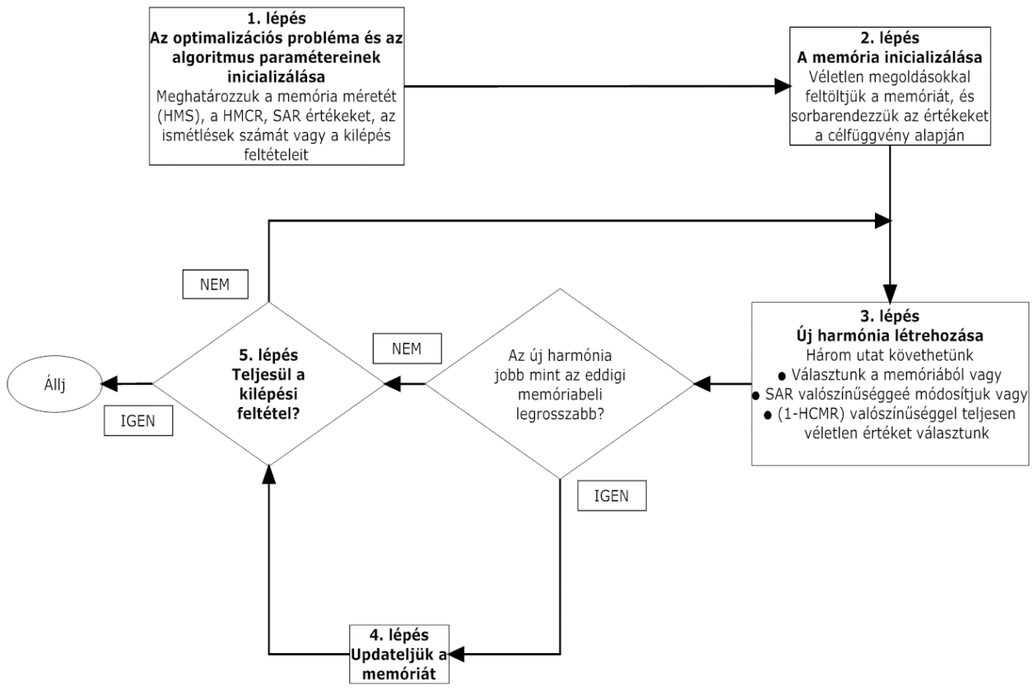
- a memóriában tárolt dallamok közül választ, vagy
- a memóriában tárolt dallamok közül választ, és azt módosítja kissé, vagy
- egy teljesen véletlen hangzást választ.

A HS követi ezt az eljárást, amikor döntési változókat hoz létre:

- HMCR valószínűséggel választunk a HS memóriából (*memory consideration*)
- melyet SAR valószínűséggel módosítunk, (1-SAR) valószínűséggel változatlanul hagyjuk (*pitch adjustment*).

*3. ábra*

**A harmony search algoritmus**




A 3. ábrán szemléltetjük a HS algoritmus lényegét:

(1) Meghatározzuk a memória méretét (HMS), a HMCR, SAR értékeket, és az ismétlések számát, vagy a kilépési feltételt.

(2) Ezután véletlenszerű megoldásokkal felöltjük a memóriát, és sorba rendezzük az értékeket a célfüggvény alapján.

A módosítás egy $bw \times \underline{U}\ (-1,1)$ érték hozzáadásával történik, ahol *bw* sávszélességnek az algoritmus felhasználója választ egy megfelelő értéket $\underline{U}\ (-1,1)$, egy egyenletes eloszlás –1 és 1 között.

- vagy egy teljesen véletlen értéket választunk (1-HMCR) valószínűséggel (*randomization*).





(4) Update-eljük a memóriát: Ha az így kapott harmónia jobb, mint az eddigi memóriabeli 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 futásiidőkorlátot.

A HS algoritmus egy továbbfejlesztése a Mahdavi et al. (2006) nevéhez *fűződő improved harmony search algorithm* (IHS). Az IHS a HS finomhangolása, jobb teljesítményt nyújt annál. Az új elem ebben az algoritmusban csupán annyi, hogy a SAR 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 próbákkal igazolnak a tanulmányban.

## Az algoritmus

Akárcsak a HS, a cikkbeli algoritmus is zenei analógiával dolgozik. Kiindulási alapját Csébfalvi erőforrás-korlátos projektütemezési problémák megoldására írt „Sound of Silence" algoritmusa adta, annak egy konfliktusjavító változata. Azonban jelentős különbségek is vannak a két heurisztika között. A módosításokra elsősorban azért volt szükség, mert a mi megoldandó problémánk kétmenetes, az RCPSP pedig egymenetes, egy célfüggvényes probléma. Az egyik leglényegesebb jellemzője a problémámnak a másodlagos kritérium – a nettó jelenérték maximalizálás – megléte. A megközelítés újdonsága, hogy az NPV maximalizálásra másodlagos célként tekintünk, tehát az erőforrás-korlátos hosszminimalizáló ütemezések halmazán keressük azt az ütemezést, melynek nettó jelenértéke maximális. Olyan típusú hierarchikus projektütemezés, ahol elsődleges cél (primary criterion) a projekthossz-minimalizálás, másodlagos cél (secondary criterion) a nettó jelenérték maximalizálása, legjobb tudomásunk szerint eddig ezenkívül még nem készült. Ez a kétmenetes (bi-criteria) megközelítés reálisabb, mint a hagyományos egymenetes. Az utóbbiban ugyanis elméletileg előfordulhat olyan eset, hogy egy negatív pénzáram 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 *időalapú modellek*ben (lásd például Pritsker et al. [1969] az erőforrás-korlátokat kielégítő ütemezéseket a tevékenységek kezdési idejével reprezentáljuk. Ebben a modellben az explicit erőforráskorlát-kezelésnek megfelelően a tevékenységek mozgatása sértheti az erőforrás korlátokat, azaz előfordulhat olyan eset, amikor egy tevékenység eltolásának eredményeként az ütemezés már nem elégíti ki az erőforrás korlátokat. A másodlagos kritérium megléte miatt a konfliktus javító modell alkalmazására volt szükség.

A *tiltott halmazok elvén* alapuló konfliktusjavító modellben az erőforrás-korlátokat kielégítő ütemezést a beillesztett konfliktus javító relációk halmazával reprezentáljuk. Az implicit erőforráskorlát-kezelésnek megfelelően ebben a modellben az erőforrás-korlátok kielégítésére nem hat a *feasible* (azaz elsőbbségi és erőforrás korlátokat kielégítő) tevékenységek eltolása, mozgatása.

*Tiltott halmaz*ról beszélünk, ha (1) minden halmazbeli tevékenységet végrehajthatunk párhuzamosan, párhuzamos végrehajtásuk nem sérti a tevékenységek között fennálló elsőbbségi feltételeket; (2) van azonban olyan erőforrás, melyből kevés van ahhoz, hogy minden tevékenységet végrehajtsunk a halmazból párhuzamosan; (3) a halmaz nem tartalmaz valódi tiltott halmaz részhalmazt.

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

Mivel a konfliktusjavító változatban az elsődleges változók a konfliktusjavító relációk, a megoldás egy projekt hossz-minimalizáló erőforráskorlátokat kielégítő halmaz lesz, melyben minden mozgatható tevékenység eltolható anélkül, hogy az erőforráskorlátokat megsértenénk.

A konfliktusjavító modell hosszminimalizáló változata védett a tevékenységek mozgatásával szemben, így bevezethetünk egy másodlagos teljesítménymértéket is – az NPV-t –, mellyel a generált megoldáshalmazból kiválaszthatjuk a számunkra „legjobb" ütemezést.

A HS (IHS) algoritmus explicit, mivel direkt dolgozik a hangokkal. A cikkbeli algoritmus – mint „elődje", a Csébfalvi-féle Sounds of Silence 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 cikkbeli algoritmusban is, az improvizáció egy – a karmester

által választott – többé-kevésbé harmonikus dallam módosítása. Ezenfelül mindkét algoritmusban minden döntésért a karmester a felelős, a zenészek ilyenformán csupán egy „döntéstámogató rendszert” alkotnak.

Az SoS algoritmusban a központi elem egy egyszerű javítás nélküli *forward-backward lista ütemezési eljárás*. Ebben az eljárásban az egyetlen, de igen jelentős újdonság a gyors és hatékony tevékenység lista generátor, amely független az alkalmazott metaheurisztikus kerettől, és amely igen jó minőségű, közel optimális megoldást ad viszonylag nagy méretű problémákra is, elfogadható idő alatt.

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

(1) a zenekar $N$ zenészből áll; (2) a polifonikus hangzás $N$ hangból (sound) áll és $R$ szólamból (phrase); (3) minden $i = 1,2,...,N$ zenész pontosan egy polifonikus hangért felel; (4) minden $i = 1,2,...,N$ polifonikus hangot a következő halmazzal tudunk jellemezni: $(X_i,D_i)$, $\{R_{i,r} \mid r = 1,2,...R\}$; (5) minden $r = 1,2,...,R$ szólamban a hangok a párhuzamos hangokhoz adódnak hozzá (6) minden $r = 1,2,...,R$ szólamban csak a „magas” hangok hallatszanak, melyekre $\{U_{t,r} \mid U_{t,r} > R_r$, t = 1,2,...,$T\}$; (7) Az improvizáció során minden $i = 1,2,...,N$ zenész egy $I_i \in [-1,1]$ értéket ad meg „elképzelésként”, mellyel annak a szándéknak az erősségét jelzi, mellyel szeretne belépni a melódiába. Ha az érték 1, akkor minél hamarabb, ha 0, akkor minél később szeretne belépni, ha 0 –1, akkor tanácstalan a belépési szándékát illetően. (8) Az improvizációs eljárás során az összes döntés a karmester felelőssége, ideértve a dallam választását és a zenészek elképzeléseinek összehangolását. (9) Az eljárás során a legrövidebb, a fentieknek megfelelő „magas” hangú dallamot, máshogyan megfogalmazva az analógiát, a „leghosszabb csendet” keressük.

Az eljárás során a karmester egy egyszerű, de szokatlan egész lineáris programozási problémát old meg, amikor összehangolja a zenészek elképzeléseit abból a célból, hogy minél jobb harmóniát találjunk:

$$\min\left\{\sum_{i=1}^{N} I_i X_i \mid X_i + D_i \le X_j, \forall i \to j \in E, \underline{S_i} \le X_i \le \overline{S_i}, X_i egész \forall i \in \{1,2,...,N\}\right\}$$

Tekintsünk egy konkrét esetet, amikor a zenészek elképzelései a következők:

$\{0.3, –0.1, 0.5, –0.3, 0.4, –0.2, 0.3, –0.6, 0.7, –0.6\}$.

Ekkor a fenti minimalizálandó képlet a következő lesz:

$$0.3X_1 – 0.1X_2 + 0.5X_3 – 0.3X_4 + 0.4X_5 – 0.2X_6 + 0.3X_7 – 0.6X_8 + 0.7X_9 – 0.6X_{10}$$

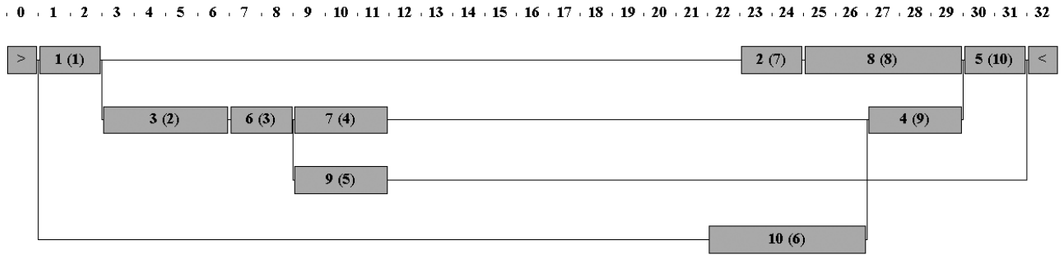
Az eljárás kezdetekor a karmester összegyűjti az elképzeléseket. Teljesen véletlenszerű megoldásokkal tölti fel az $I$ halmazt, a repertoárt. Az **improvizáció** során előállítja az új $I^{new}$ értékeket a régi $I^{old}$ értékekből. A HS eljárásnak megfelelően először egy bizonyos valószínűséggel választ 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 a véletlenszerű hanggenerálás.

Ezen a ponton a karmester előállította az új „elképzeléseket”.

Aztán megoldja a fenti ILP problémát LP problémaként, alkalmazva egy gyors belsőpont eljárást a **tevékenységlista-ütemezési eljárás**ban. Rendszerint ennek az optimalizációnak az eredménye egy igen érdekes ütemezés. A fenti példa esetében például a következőképpen néz ki: (A téglalapokon szereplő első szám a tevékenység száma, zárójelben az eredményképpen kapott fontossági sorrendben betöltött helye.)

***4. ábra***

**A tevékenység lista ütemezési eljárás egy lehetséges eredménye**




A karmester ezt az ütemezést csak arra használja, hogy a zenészek, azaz a hangok végső sorrendjét határozza meg. Az ábrán szereplő esetben ez a sorrend: $\{1,3,6,7,9,10,2,8,4,5\}$. Abban az esetben, amikor két tevékenységnek ugyanaz a kezdőideje, véletlenszerűen állapítja meg a sorrendet.





Ezután a tevékenységsorrendből egy **forward-backward ütemezés** segítségével egy ütemezést hozunk létre. Ez az eljárás az $L$ tevékenységlistát $S(L)$ ütemezéssé transzformálja, egyenként véve a tevékenységeket a tevékenységlistáról, és ütemezve őket a legkorábbi kezdési időpontra, amit az elsőbbségi és erőforrás korlátok megengednek. A konfliktusjavító modell döntő jelentőségű pontja a tiltott halmazok számítása. Az eljárásunk e pontján még nem határozzuk meg a tiltott halmazokat az időigény csökkentése miatt, helyettük a karmester egy egyszerű, bár igen hatékony és gyors hüvelykujjszabályt alkalmaz, hogy a tiltott halmazok számításának az időigényét csökkentse. A karmester – explicit tiltott halmaz számítása 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. Ezt akkor teszi, amikor az erőforráskorlátokat sértené $j$ tevékenység korábbi ütemezése. Az eljárás eredménye egy aktív ütemezés lesz, látható konfliktus nélkül.

Ezután a karmester pontosan egy lépésben javítja az összes rejtett konfliktust, beillesztve mindig a legjobb konfliktusjavító relációt minden tiltott halmaz esetén. Ebben a helyzetben a „legjobb” egy olyan $i \rightarrow j$ relációt jelent két tiltott halmazbeli tag között, melyekre a tartalékidő maximális. Azért a maximális tartalékidejűt választja, hogy olyan ütemezést kapjon, amelyben minél több „lag” van, így minél jobban mozgathatjuk a tevékenységeket, azaz nagyobb valószínűséggel kapok a másodlagos kritériumnak minél inkább megfelelő ütemezést.

A populáció elemei olyan ütemezések, melyekben minden látható és rejtett konfliktust kijavítottunk explicit vagy implicit relációk beépítésével. Mindig azt az ütemezést cseréljük az eljárás során, melynél van jobb, azaz rövidebb ütemezés, vagy ha a populációbeli legrosszabb vele egyező hosszú, 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 erőforrás-korlátos ütemezés. Minél több ilyen van, annál nagyobb eséllyel kapunk a másodlagos feltételnek megfelelő jó megoldást.

A megoldás igen lényeges eleme, hogy amennyiben a hagyományos megelőző-rákövetkező formulát egy totálisan unimoduláris (TU) megelőző-rákövetkező formulával helyettesítjük, lineáris programozási feladathoz jutunk, így polinomiális idő alatt megoldhatjuk a problémát. Bár az erőforráskorlátok széttörik a TU formát, esetünkben a tiltott halmazok alkalmazásával mégis alkalmazhatjuk a TU-t.

Az eljárás eredményeképpen a repertoár a végén közel optimális ütemezésekből fog állni.

### *Számítási eredmények*

A fenti megoldást a jól ismert és népszerű PSPLIB teszt könyvtár (http://129.187.106.231/psplib/) J30 alkönyvtárának első 40 projektjén teszteltük. Ebben a J30 részhalmazban a valós tevékenységek száma 30. Az egzakt megoldásokat egy korszerű MILP solverrel (CPLEX) generáltuk. A pénzáramokat a [–50, 100] tartományon egyenletes eloszlással generáltuk. Az eredményeket az alábbi táblázaton mutatjuk be. Az eredmények is mutatják, hogy az algoritmus hatékony és életképes.

| Ismétlésszám | 100 | 500 | 1000 |
|---|---|---|---|
| Átlagos eltérés (%) | 5.10 | 3.19 | 1.45 |
| Standard Deviation | 8.45 | 5.17 | 2.81 |
| Átlagos idő (mp) | 2.13 | 3.95 | 8.85 |

## Lábjegyzet

[1] A cikk egy készülőfélben lévő phd-dolgozat alapján született, a cikk szerzőjének phd-konzulense dr. Csébfalvi György (Pécsi Tudományegyetem).

## Felhasznált irodalom

*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)

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

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

*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.

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

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

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

*Vasebi, A. – Fesanghary, M. – Bathaee, S.M.T.* (2007): Combined heat and power economic dispatch by harmony search algorithm, Electrical Power and Energy Systems

Cikk beérkezett: 2009. 3. hó
Lektori vélemény alapján véglegesítve: 2009. 5. hó