SZOFTVERFEJLESZTÉSI TEVÉKENYSÉGEK EGZAKT ÜTEMEZÉSE ERŐS ÉS GYENGE ERŐFORRÁS KORLÁTOK MELLETT[1]

KRUZSLICZ FERENC
PTE Közgazdaságtudományi Kar

Jelen dolgozat során egy olyan új erőforrás ütemezési modellt mutatunk be, mely alkalmas valódi többprojektes környezetben egymástól jelentősen eltérő jellegzetességekkel bíró, korlátos erőforrástípusok kezelésére. Olyan kétfázisú, többkritériumos erőforrás hozzárendelő és kiegyenlítő algoritmuson alapszik, mely kis és közepes feladatok esetén is képes az egzakt megoldások előállítására. A modell életképességét egy szoftverfejlesztési példán keresztül mutatjuk be.

1 Bevezetés

A projektmenedzsmenthez kapcsolódó kutatásokban egyre több olyan alkalmazási terület került előtérbe, ahol párhuzamosan több projekt ütemezését kell megoldani. A projekt egymással kapcsolatban álló tevékenységek halmaza, ahol a tevékenységek közötti kapcsolatokat elsőbbségi feltételek írják le, a tevékenységeket pedig végrehajtásuk időszükségletével és erőforrásigényével jellemezzük. A projektekhez kapcsolódó tervezési és ütemezési feladatok elméleti alapját szolgáltató hálótechnikai módszerek a gráfelmélet egyik legfontosabb és legnehezebb területét jelentik [9]. Az említett problémakör a kombinatorikai optimalizálás "NP-hard" feladatai közé tartozik. Ez a projektmenedzsment szempontjából azt jelenti, hogy még egy kisméretű, könnyen átlátható és könnyűnek tűnő feladat megoldása is rendkívül számításigényes és nehéz lehet.

A projektek tevékenységeinek végrehajtásához különféle erőforrásokra (idő-, emberi-, tárgyi- és pénzügyi stb.) van szükség. Ezek az erőforrások alapvetően két kategóriába sorolhatóak aszerint, hogy azok újrafelhasználhatóake (megújuló erőforrások) vagy sem [3]. Mindeddig főleg olyan területek vizsgálatára került sor, ahol ugyan valóban több —egymástól jól elhatárolható— projektet kell kezelni, ám ezek a projektek mindig egy adott projekt időben eltolt példányai voltak. Egyazon standardizált termék különböző megrendelői igény szerint történő előállításának ütemezési problémáival gyakran találkozhatunk az autóiparban [5], valamint a házgyártásban [1] is. Az így adódó projektek párhuzamosan kerülnek végrehajtásra, és az egyes mun-

[1]Beérkezett: 2003. február 11. A szerző a Pécsi Tudományegyetem Közgazdaságtudományi Kar Gazdálkodástani Doktori Iskolájának hallgatója. Témavezető: Csébfalvi György.

kafázisokhoz rendelt emberi erőforrások esetében a munkaerő tetszőlegesen helyettesíthető. A sorozatgyártás során felmerült problémák szempontjából a tárgyi-, míg az egyedi (főleg szellemi) termékek esetében az emberi erőforrás játszik középponti szerepet. Az eddig használt modellekben tehát a munkaerő, mint korlátos, megújuló erőforrás ugyan megjelent, de mindig mint szabadon helyettesíthető elemekből álló tényező [8]. Az itt ismertetett modell kialakítását sugalmazó szoftverfejlesztési tevékenység is egy tipikus multiprojekt környezet. A szoftverfejlesztésben előforduló feladatok legfőbb jellegzetessége, hogy nemcsak merőben különböző projektekből állnak, hanem a felhasznált emberi erőforrások jellemzői is jelentősen eltérnek, egymást nem helyettesíthető erőforrás osztályokba sorolhatóak.

A következőkben bemutatandó modell és algoritmus a hagyományos projekt modellezési eljárásokhoz képest több szempontból is új elemeket tartalmaz. Egyrészt az egymástól minőségileg eltérő erőforrásokhoz olyan újszerű célfüggvényeket rendel, amelyek az erőforrások hatékony kihasználásának globális jellemzői. Másrészt nem elégszik meg egy hatékony ütemezés előállításával, hanem egzakt módszer lévén az összes hatékony megoldást előállítja. Ez a döntéshozók számára lehetőséget ad a különböző Pareto optimális megoldások további mérlegelésére. Harmadrészt sikeresen ötvözi az erőforrás felhasználással kapcsolatos ütemezési problémák két fő fajtáját, a hozzárendelési és a kiegyenlítési problémák jellegzetességeit azzal, hogy minden erőforrást a rá legjobban jellemző probléma szerint kezel.

2 Modell

A bevezetőben leírt projektek modellezésére és a tevékenységek ütemezésének optimalizálására egy olyan modellt állítottunk fel, mely egyaránt képes kezelni az egymástól lényegesen eltérő jellegzetességű projekteket és korlátos erőforrásokat is.

Jelölje $P = \{P_1, P_2, \ldots P_n\}$ az adott időszak alatt elvégzendő projektek halmazát. Minden $P_i = \{T_{i,j} : j = 1 \ldots n_i\}$ projekt n_i féle tevékenységet tartalmaz, melyek között adott a közvetlen megelőzési relációk $H_i = \cup\{T_{i,j_1} \rightarrow T_{i,j_2}\}$ halmaza. A $T_{i,j_1} \rightarrow T_{i,j_2}$ jelölés azt jelenti, hogy a T_{i,j_1} tevékenységnek még a T_{i,j_2} tevékenység megkezdése előtt be kell fejeződnie.

A rendelkezésünkre álló k darab erősen korlátos erőforrásból álló halmazt jelölje $RH = \{RH^1, RH^2, \ldots RH^k\}$. Az l féle gyengén korlátos erőforrás halmaza pedig legyen $RS = \{RS^1, RS^2, \ldots RS^l\}$. Minden $R \in RH \bigcup RS$ erőforráshoz adott az $R(t)$ függvény, mely a t-edik időpontban az R erőforrásból rendelkezésre álló mennyiséget adja meg.

Definíció: Az R erőforrást *erősen korlátos erőforrásnak* nevezzük, ha egyetlen t időpontban sem használhatunk fel az $R(t)$ értéknél több erőforrásegységet.

Erősen korlátos erőforrások esetében pótlólagos erőforrások bevonása még ideiglenesen sem lehetséges. Erőforrás hiányból származó konfliktusok felol-

dására kizárólag a tevékenységek átütemezése, vagy a projektek legkorábbi befejezési határidejének meghosszabbítása jöhet számításba.

Definíció: Az R erőforrást *gyengén korlátos erőforrásnak* nevezzük, ha bármely t időpontban az $R(t)$ értéknél nagyobb erőforrás igényt is ki lehet elégíteni, pótlólagos erőforrások igénybevétele révén.

A gyengén korlátos erőforrások esetében jelentkező erőforrás hiányok áthidalásához igénybe vett pótlólagos erőforrásokra semmiféle megkötés nincsen azon kívül, hogy az adott tevékenységeket képesek legyenek ellátni. A pótlólagos erőforrás tehát származhat a rendelkézesünkre álló éppen szabad erősen vagy gyengén korlátos más erőforrásokból éppúgy, mint külső erőforrások beszerzéséből.

Az eddig bevezetett jelölések segítségével minden tevékenységet egy $T_{i,j} = (D_{i,j}, RH^1_{i,j}, RH^2_{i,j}, ...RH^k_{i,j}, RS^1_{i,j}, RS^2_{i,j}, ...RS^l_{i,j},)$ vektorral jellemezhetünk, ahol $D_{i,j}$ jelöli az i-edik projekt j-edik tevékenységének hosszát időegységekben, $R_{i,j}$ pedig az adott tevékenység erőforrás igényét az $R \in RH \bigcup RS$ erőforrásra vonatkozóan.

A projektek egy S ütemezését a $T_{i,j}$ tevékenységek kezdési időpontjainak halmazával adhatjuk meg $S = \{S_{i,j} \ : \ i = 1 \ldots n, \ j = 1 \ldots n_i\}$. Az egységes jelölésrendszer érdekében bevezethetünk egy T_0 nyitó, valamint egy T_1 záró, üres (idő- és erőforrás igény nélküli) tevékenységet. A nyitó és záró tevékenységekhez tartozó közvetlen megelőzési relációk halmazát jelölje

$$H = \{T_0 \rightarrow T_{i,j}, \ T_{i,j} \rightarrow T_1 \ : \ i = 1 \ldots n, \ j = 1 \ldots n_i\}.$$

Ennek segítségével a T_0-hoz tartozó S_0 ütemezési érték a projektek végrehajtásának kezdetét, míg a T_1-hez tartozó S_1 érték a legkorábbi befejezési időpontot jelöli. Nem jelent megszorítást, ha feltételezzük, hogy az ütemezések mindig az első időpontban kezdődnek, azaz $S_0 = 0$. Ezen feltételezés mellet az S ütemezés végrehajtásához szükséges idő $E(S) = S_1$. A számos ütemezési lehetőség közül csak a lehetséges ütemezéseket vesszük figyelembe, vagyis azokat, melyek megtartják a közvetlen megelőzési relációkat.

Definíció: S egy *lehetséges* ütemezés, ha minden $T_{i,j_1} \rightarrow T_{i,j_2}$ esetén teljesül az $S_{i,j_1} + D_{i,j_1} \leq S_{i,j_2}$ egyenlőtlenség. A lehetséges ütemezések halmazát jelölje $\Re = \Re(H \bigcup H_1 \bigcup ... \bigcup H_n)$.

A lehetséges ütemezések körét tovább kell szűkítenünk az erőforrás korlátok figyelembevételével. Jelölje $U^R_S(t)$ az R erőforrás t-edik időpontbeli kihasználtságát az S ütemezés mellett. Az $U^R_S(t)$ erőforrás felhasználási hisztogram értékeit az alábbi képlet alapján számíthatjuk ki:

$$U^R_S(t) = \sum_{i=1}^{n} \sum_{j=1}^{n_i} S_{i,j}(t) \cdot R_{i,j} \ , \text{ ahol}$$

$$S_{i,j}(t) = \begin{cases} 1, & \text{ha } S_{i,j} \leq t < S_{i,j} + D_{i,j} \\ 0, & \text{egyébként} \end{cases}$$

Az $S_{i,j}(t)$ függvény segítségével könnyen leírhatjuk a t-edik időpontban folyamatban lévő (aktív) tevékenységek halmazát $A(t) = \{T_{i,j} : S_{i,j}(t) = 1\}$.

Ha a szűkítést az erőforrásoknak egy C halmazára vonatkoztatjuk, akkor csak olyan lehetséges ütemezéseket fogadhatunk el, amelyek minimális idő alatt hajthatók végre úgy, hogy az erőforrás kihasználtság egyetlen $R \in C$ erőforrás esetében sem lépi túl a hozzá tartozó $R(t)$ korlátot.

Definíció: Az S ütemezés a C erőforrás halmazra nézve *elfogadható*, ha $S \in \Re$, továbbá minden $R \in C$ és $0 < t \leq E(S)$ esetén $U_S^R(t) \leq R(t)$, valamint $E(S) = \min\{E(S') : S' \in \Re\}$. A C erőforrás halmazra nézve elfogadható ütemezések halmazát $\Re^C$-vel jelöljük.

Az előbbieknek megfelelő $\Re^{RH}$ halmaz tehát az erősen korlátos erőforrásokra nézve elfogadható ütemezések halmaza, melyet egy hozzárendelési probléma megoldásával állíthatunk elő. Ebben a halmazban kell megkeresnünk a gyengén korlátos RS erőforrásokra vonatkozó hatékony megoldásokat. Az ütemezések hatékonyságának definiálására és mérésére többféle módszer is használható. Modellünkben minden gyengén korlátos erőforráshoz kétféle globális mértéket rendelünk hozzá, és az így kapott mértékek szerinti Pareto optimális megoldások jelölik ki a hatékony megoldásokat.

Az $R \in RS$ gyengén korlátos erőforrás kihasználtsága az S elfogadható ütemezés mellett egyrészt akkor optimális, ha az a lehető legkevesebb erőforrás bővítés mellett valósítható meg. Az R erőforráshoz tartozó célfüggvény ebben az esetben az erőforrás felhasználás maximális szintjét minimalizálja:

$$MU^R = \min_S\{MU^R(S)\}, \text{ ahol } MU^R(S) = \max_t\{U_S^R(t)\}\}$$

Az $R \in RS$ gyengén korlátos erőforrás kihasználtsága az S elfogadható ütemezés mellett másrészt akkor optimális, ha az erőforrás kihasználtság szintje a legkevésbé ingadozó. Másképpen a globális üresjárat (idle time) szintje a legalacsonyabb. Legyen $\overline{U_S^R}(t)$ az R erőforrás U_S^R felhasználási hisztogramjának konkáv burka, melyet az alábbi módon definiálhatunk.

Definíció: A $\overline{h}(t)$ hisztogram a $h(t)$ hisztogram konkáv burka, ha

1. konkáv, azaz minden $t_1 < t_2 < t_3$ esetén $\overline{h}(t_2) \geq \min\{\overline{h}(t_1), \overline{h}(t_3)\}$,
2. és burok, vagyis minden t értékre $\overline{h}(t) \geq h(t)$,
3. valamint minimális abban az értelemben, miszerint a $C(\overline{h}) = \sum_t \overline{h}(t) - h(t)$ érték a lehető legkisebb.

A legkisebb $C(\overline{h})$ értéket a h erőforrás kihasználtsági hisztogram konkávitási mértékének nevezzük [10]. Ez egy olyan globális mérőszám, amely nagyon jól jellemzi a folyamatosan egyenletes terhelést. Ennek segítségével már minden $R \in RS$ erőforrás célfüggvénye könnyen megadható:

$$IT^R = \min_S\{IT^R(S)\},\ ahol\ IT^R(S) = \min_{\overline{U_S^R}}\{C(\overline{U_S^R})\}\}$$

A modell végső és egyben hatékony ütemezéseit az MU^R és az IT^R kritériumok szerinti Pareto optimális megoldások szolgáltatják. Ez a feladat az erőforrás kiegyenlítési problémák közé tartozik. Egy S ütemezést akkor nevezünk több célfüggvény szerint Pareto optimálisnak, ha bármelyik célfüggvény szerinti értéke csak egy másik rovására javítható. A jelen modellre vonatkozó pontos definíció a következőképpen adható meg.

Definíció: Az $S \in \Re^{RH}$ erősen korlátos erőforrásokra elfogadható ütemezés az RS erőforrásokhoz rendelt MU^R és IT^R célfüggvényekre nézve Pareto optimum, ha nem létezik olyan $S' \in \Re^{RH}$ elfogadható ütemezés, amelynél minden $R \in RS$ erőforrásra az $MU^R(S) > MU^R(S')$ és $IT^R(S) > IT^R(S')$ egyenlőtlenségek mindegyike egyszerre teljesül.

3 Algoritmus

A fent ismertetett modellnek egy kétfázisú, implicit leszámláláson alapuló algoritmust lehet megfeleltetni. Az első fázisban az összes ütemezések terét szűkítjük le az RH erőforrásokra nézve elfogadható ütemezések körére. A második fázis során az így meghatározott térben keressük meg az RS erőforrásokra vonatkozó Pareto optimális megoldásokat.

(i) Az első fázisban előállítjuk az $\Re^{RH}$ halmazt. Az RH- beli erőforrásokra nézve elfogadható ütemezések megkeresése egy olyan hozzárendelési probléma megoldását jelenti, amelynél az RS erőforrásokat figyelmen kívül hagyjuk. A lehetséges ütemezések $\Re$ terének erőforrás felhasználási konfliktusait a *minimális összeférhetetlenségi halmazok* segítségével írjuk le, melynek pontos definíciója a [4]-es cikkben olvasható. A minimális összeférhetetlenségi halmazokkal kapcsolatban itt csak annyit jegyzünk meg, hogy egy adott halmazba tartozó tevékenységek egyidejű ütemezése erőforrás felhasználási konfliktussal jár, de a „minimális" jelzőnek megfelelően, a halmaz bármely valódi részhalmazába tartozó tevékenységek egyidejű ütemezését erőforrás kihasználási korlátok már nem akadályozzák. Az $\Re^{RH}$ halmaz elemeit a minimális összeférhetetlenségi halmazok feloldásai szerinti reprezentáns ütemezésekkel írhatjuk le.

Ha a rendelkezésre álló $R \in RH$ erőforrások valamelyikének korlátozottsága miatt a kritikus út módszerével kapott legkorábbi ütemterv nem valósítható meg, akkor az adott erőforrások felhasználási konfliktusainak feloldásával (vagyis bizonyos tevékenységek ütemezésének késleltetésével) olyan ütemtervet kell keresnünk, amely az összes erőforrás korlátot kielégíti, és amely a kritikus út módszerével adódó globális befejezési időpontot minimális mértékben növeli.

(ii) A második fázisban az első fázis eredményeként kapott $\Re^{RH}$-beli ütemezéseket vesszük sorra. Valójában elegendő az első fázisban kapott

keresési fa leveleinek megfelelő reprezentáns ütemezések vizsgálata. Minden ilyen ütemezés esetében az eredeti feladat H illetve H_i közvetlen megelőzési halmazait kibővítjük a keresési fa csomópontjának megfelelő erőforrás felhasználási konfliktusok feloldását leíró relációkkal. Az így módosított feladatokban az erősen korlátos erőforrásokat már figyelmen kívül hagyhatjuk, hiszen a módosított feladat bármely lehetséges megoldása egyben az eredeti feladat RH elemeire nézve elfogadható megoldása is lesz. Ez annak a következménye, hogy az első fázisban az ütemezéseket a *minimális összeférhetetlenségi halmazok* segítségével írtuk le. Az állítás részletes bizonyítása megtalálható a modell alapját képező értekezésben [14]. A kapott kiegyenlítési probléma hatékony megoldásait ugyanazzal az implicit leszámlálási módszerrel állíthatjuk elő, amit az első fázisban alkalmaztunk [12]. Az RS erőforrások gyengén korlátosak, ezért itt nem lesznek erőforrás felhasználási konfliktusok, ami miatt a keresési fák mérete is jelentősen nagyobb lehet. A Pareto optimális megoldások az $R \in RS$ erőforrásokhoz rendelt MU^R és IT^R célfüggvények szerinti minimumok lesznek.

A második fázis végrehajtása után kapott Pareto optimumok egyben az eredeti feladat egzakt megoldásai lesznek.

A modell numerikus kezeléséhez többféle implicit leszámláláson alapuló eljárás közül választhatunk [13,6]. A megvalósítás során ezek közül azt az algoritmust alkalmaztuk, amelynek hatékony metszési szabályai lehetővé teszik nagyobb méretű feladatok megoldását is [7].

4 Alkalmazási példa

A szoftverfejlesztési stratégiákat alapvetően két csoportba lehet sorolni: egyedi megrendeléseknek megfelelő speciális rendszerek készítése, illetve speciális saját termékek gyártása. A legtöbb esetben a két stratégia jól kiegészíti egymást. Például pangó megrendelések mellett önálló termékek fejlesztésével lehet a munkaerő kihasználást egyenletesebbé tenni. Jelen cikk során a szoftverfejlesztés alatt kizárólag azt a folyamatot értjük, amely a technikai specifikáció elkészítésétől a kódoláson keresztül egészen a tesztelésig tart. A hagyományos elméletek szerint a felhasználói igény specifikációjának elkészítését és a rendszertervezést nem lehet szigorúan elválasztani. Most mégis kénytelenek vagyunk ezt megtenni, hiszen a programozói feladatokat és azok időszükségletét csak a technikai specifikáció ismeretében lehet megfelelő pontossággal megbecsülni. Másrészről a technikai specifikáció elkészítése előtti tevékenységeket általában még nem kötik szerződésben rögzített határidők.

A vizsgálat során felmerült projektek legtöbbje három jól elkülöníthető részből állt.

1. Technikai specifikáció készítés. Ez általában egy vezető programozó feladata.

2. Kódolás. A programozók a rendszerterv alapján elkészítik a kódot (valamint az adatbázist, dokumentációt).

3. Ellenőrzés. Az elkészült alkalmazás illetve funkció minőségét és hibamentességét tesztelők vizsgálják meg.

	Erőforrás igény (nap / fő)		
Projekt	D/RH^1	D/RS^1	D/RS^2
Funkció bővítés			
P1	2 / 2	3 / 2	4 / 2
P2	5 / 3	3 / 3	4 / 4
P3	3 / 2	3 / 2	4 / 1
Hibajavítás			
P4	0 / 0	2 / 1	2 / 2
P5	0 / 0	4 / 3	2 / 3
Átfogó ellenőrzés			
P6	0 / 0	0 / 0	2 / 3
P7	0 / 0	0 / 0	3 / 3
Rendelkezésre álló erőforrások			
$RH^1(t) \equiv 5$		$RS^1(t) \equiv 5$	$RS^2(t) \equiv 5$

1. táblázat. Projektek erőforrás igényei

A fenti feladatok a valóságban —egymással folyamatos visszacsatolásban állva— fejlesztési ciklusokat alkotnak. Hogy a szoftverfejlesztési problémára a modellünket mégis alkalmazhassuk, ki kell használni a problémakör két fontos jellegzetességét. A feladat mérete kicsi, kivitelezése rövidtávú. Nem nagy rendszerek fejlesztését kell modellezni, hanem pár lépésből álló, kisebb feladatokat. Ebben az esetben a fejlesztési ciklusokat szükség esetén külön tevékenységek sorozatává fejthetjük ki. Rövidtávú feladatok révén pedig olyan egyszerűsítő feltételezéseket vezethetünk be, miszerint az időszak alatt rendelkezésre álló erőforrások mennyisége konstans.

Az egyes munkafolyamatokat ellátó személyzet a szaktudásának megfelelően az alábbi kategóriákba sorolható:

RH^1 **erőforrás:** A *vezető programozók* —akik általában az egyes projektek gazdái is— a technikai specifikáció kivitelezésében vesznek részt, majd a kritikus kódrészek elkészítésében és a többi programozó munkájának felügyeletében is tevékenykednek. A vezető programozók olyan magas kvalitású lojális alkalmazottak, akik átlátják a teljes rendszert és ellátják a verzió- és release-menedzsmentet is. Ebből következően ez az erőforrás erősen korlátos, hiszen ilyen munkaerőből általában kevesebb van, valamint szak- és rendszerismerete miatt nehezen helyettesíthető és bővíthető. Szükség esetén programozói minőségben is hadrafogható, melyet a projektek tervezése során vehetünk figyelembe. A vezető programozók terhelése ennek következtében erősen ingadozó.

RS^1 **erőforrás:** A *programozók* általában egy-egy adott programnyelv szakértői, akik a specifikáció alapján már önállóan elkészítik és javítják a programkódot. Újabb programozók bevetésének ugyan van egy elég magas induló költsége, révén meg kell ismerkedniük a projekttel illetve a projektben résztvevőkkel, valamint az alkalmazott technológiákkal.

Ezek után azonban a többi programozóval egyenértékű munkát végezhetnek. Ez az erőforrás tehát gyengén korlátos, hiszen magas költségek árán, de bővíthető. Szükség esetén segítik a tesztelők munkáját. A programozói munkaerő csökkentése kevésbé jellemző, hiszen kihasználatlanság esetén a felszabaduló idő kitűnő alkalom a szakmai ismeretek frissítésére és bővítésére.

RS^2 **erőforrás:** A minőségellenőrzés részint a folyamatok szervezése révén, részint az alkalmazott eszközök és technológia révén biztosított, mégis kiemelkedően fontos az utólagos és folyamatos tesztelés. A tesztkörnyezetek és tesztsorozatok ezt a tevékenységet nagy mértékben képesek automatizálni. Kész tesztelési forgatókönyvek esetén ezt a munkát „bárki" el tudja végezni. Ami egyben azt jelenti, hogy ez az erőforrás egyáltalán nem jelent erős korlátot. A *tesztelők* optimális esetben folyamatos és egyenletes terhelés alatt állnak.

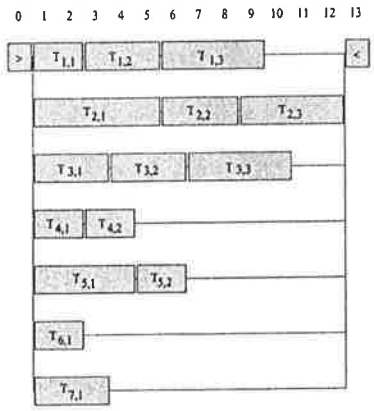
A modellt egy ilyen valós szoftverfejlesztési környezetből származó példán mutatjuk be. A megvizsgált fejlesztési időszak alatt 15 különféle tevékenység ütemezését kell megoldani úgy, hogy mindenfajta erőforrásból egyformán 5-5 fő állt rendelkezésre. A feladat pontos paramétereit az 1. táblázat tartalmazza.

A példa további jellegzetessége, hogy az egyes projektek mind lineáris szerkezetűek, vagyis a tervezési, a kivitelezési és az ellenőrzési fázisok ebben az időrendben követik egymást. Bár a modell egy-egy tevékenységhez több erőforrást is megenged hozzárendelni, az alkalmazási példában ezt a lehetőséget nem használtuk ki. Ez jelentősen leegyszerűsíti a tevékenységek leírását, hiszen az 1. táblázatban az RH^1 értékek a tervezési fázishoz, az RS^1 értékek a kivitelezéshez, az RS^2 értékek pedig az ellenőrzéshez tartoznak. Ebből az is következik, hogy minden egyes tevékenységhez pontosan egyféle erőforrást kell hozzárendelni. A példában szereplő egyes projektek is három fő típusba sorolhatóak, bár általában az adott problémák méretétől és a bonyolultsági fokától függően jelentős átfedések fordulhatnak elő:

1. *Funkció bővítés.* Az ilyen projektekre alapos tervezés a jellemző, hiszen egy meglévő rendszert kell új funkcionalitással ellátni. A programozói munka általában közepes időigényű, ami azzal magyarázható, hogy már meglévő eszközökkel dolgozhatnak tovább. A tesztelési fázis viszonylag hosszabb. A specifikációnak való megfelelésen túl a rendszer többi részének zavartalan működését is ellenőrizni kell.

2. *Hibajavítás.* Itt a tervezési fázis kimarad. A programozói feladatok kevesebb ráfordítást és nagyobb gyorsaságot igényelnek. Az ellenőrzés mindössze a hiba kijavítására terjed ki.

3. *Átfogó ellenőrzés.* Ezt a munkát a tesztelők végzik. Egyik formájában időről időre ellenőrzik az alapfunkciók helyes működését, másik formájában egy-egy területet tüzetes vizsgálat alá vesznek.



(a) Kapcsolatokkal kiegészített Gantt diagram

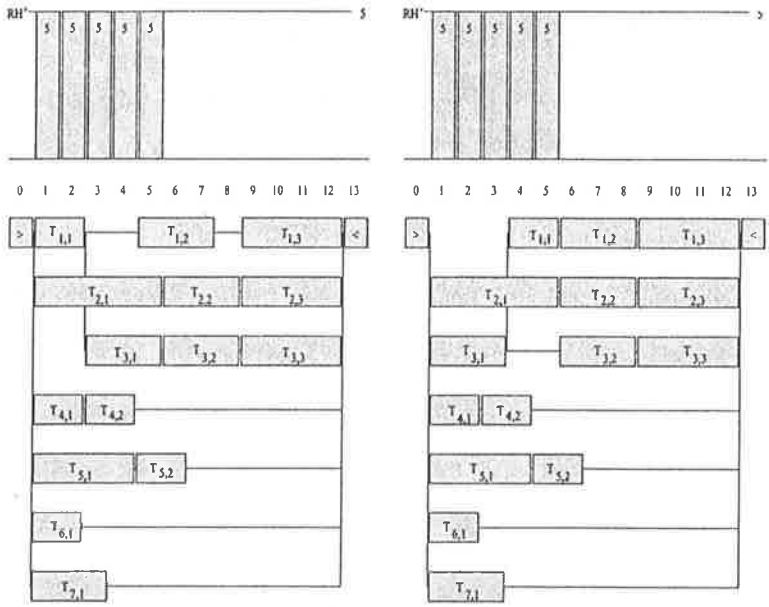
(b) Az RH erőforráshoz tartozó keresési fa

1. ábra. Tevékenységi diagram

Az 1. ábrán a projekteket AON (Activity-On-Node) formában ábrázoltuk [2]. Ebből leolvashatjuk, hogy az erőforrás korlátok figyelembe vétele nélkül legkevesebb 12 nap alatt végezhető el minden feladat. Ilyen határidők mellet az összes lehetséges ütemezések halmazának elemszáma 27 720 000. Amennyiben a határidőt egy nappal meghosszabbítjuk, úgy a lehetséges ütemezések száma 731 808 000-ra nő. Ez a két számadat is jól érzékelteti, hogy a feladat megoldása közel nem triviális.

A feladat speciális szerkezetének köszönhetően az RH^1 erőforrásra nézve elfogadható ütemezések halmazát előállító keresési fa összesen három csomópontot tartalmaz, az eredményül adódó, elfogadható ütemezési osztályok száma kettő. A *2. ábra* ezen két osztály egy-egy reprezentáns ütmezését ábrázolja. A diagramok alatt az osztályokhoz tartozó minimális összeférhetetlenségi halmaz kétféle feloldását tüntettük fel. Ezzel az algoritmus első fázisának végére értünk.

A második fázisban az alapfeladatot kiegészítettük az esethez tartozó H_a illetve H_b közvetlen megelőzési relációval. Az első megoldás esetében a $T_{1,1} \rightarrow T_{3,1}$, míg a második megoldás estén a $T_{3,1} \rightarrow T_{1,1}$ szabályok hozzáadása biztosítja, hogy a módosított feladatok lehetséges megoldásai egyben az eredeti feladat elfogadható megoldásai legyenek. A következő lépés tehát a módosított feladatok RS^1 és RS^2 erőforrásokra vonatkozó hatékony, lehetséges megoldásainak meghatározása. Az (a) esetben a lehetséges ütemezések tere 1 386 000 elemből áll, amelynél a megoldásokat előállító keresési fa 282 csomópontot tartalmaz. A (b) esetben ugyanez az érték 831 600 illetve 150 volt. Az esetekhez tartozó Pareto optimális megoldásokat a *3. ábra* szemlélteti.

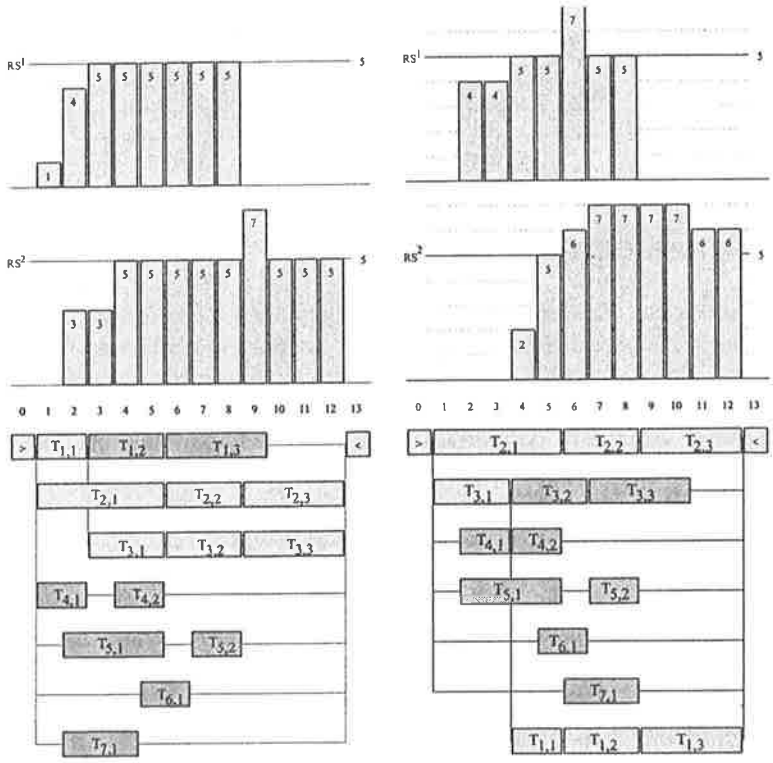


(a) eset $H_a = \{T_{1,1} \to T_{3,1}\}$ (b) eset $H_b = \{T_{3,1} \to T_{1,1}\}$

2. ábra. Elfogadható ütemezések

A minta példában szerencsés módon egyetlen megoldást kaptunk. Más feladatok esetén azonban többféle ütemezés is kielégítheti a Pareto optimum feltételeit. Ezek száma esetleg akkora lehet, hogy már az algoritmus első fázisában szelektálni kell az elfogadható ütemezések között. Ráadásul a valóságban ezen hatékony megoldások közül csak egy ütemezés realizálható. Ilyen döntési helyzetekben általában számos egyéb, nehezen megfogalmazható és modellezhető tényező játszik szerepet. Példánkban a döntéshozatal során az eddig szimmetrikus szerepet betöltő, gyengén korlátos erőforrások között most különbséget tehetnénk. Az RS^1 programozói erőforrás bővítése viszonylag magas indulási költséget és hosszabb betanítási időt igényel. Az RS^2 tesztelői erőforrás esetében az előző problémák egyike sem jelentkezik. Ennek figyelembe vételével azt a hatékony megoldás szerinti optimális ütemezést érdemes választani, ahol a pótlólagos erőforrások költségei minimálisak.

A feladat egészét tekintve azonban a két ütemezéshez tartozó célfüggvény értékek —az MU^1 kivételével— megegyeznek. Mivel az (a) megoldás szerinti ütemezés a domináns, ezért ez az egyedüli Pareto optimum, és mint ilyen egyben az optimum is. Ennek ismeretében az algoritmusunkat jelentősen gyorsíthattuk volna, ha a két esethez tartozó ütemezéseket egy közös keresési fával oldjuk meg.



(a) eset

$MU^1 = 5, IT^1 = 0$
$MU^2 = 7, IT^2 = 0$

(b) eset

$MU^1 = 7, IT^1 = 0$
$MU^2 = 7, IT^2 = 0$

3. ábra. Optimális megoldások

A bemutatott példa mindössze hét kisebb projektet tartalmaz, mégis a megvizsgálandó ütemezések terének elemszáma jelentős. Az algoritmus gyorsaságát a két fázis során használt leszámlálási módszerek határozzák meg. Ez a kritikus faktor további vizsgálat tárgyát képezheti, hiszen nem elegendő egy elfogadható ütemezés előállítása, hanem az összes ilyen ütemezést meg kell keresni.

Az algoritmus számítógépes megvalósítása Visual Basic 6.0 keretrendszerrel történt [14]. A gyorsaság szempontjából lényeges kódrészek C++ nyelven íródtak, és DLL állományként álltak a rendelkezésre. A mérési eredmények egy Pentium 166 MHz processzorral felszerelt számítógépen történt futtatásból származnak.

Algoritmus fázisa	Lehetséges ütemezések száma	Keresési fa csomópontjainak száma	Futási idő (mp)
1. fázis	27 720 000	3	0.150
2. fázis (a) eset	1 386 000	282	1.382
2. fázis (b) eset	831 600	150	0.621

2. táblázat. Futási eredmények

5 Összefoglalás

A dolgozat során ismertetett több projektes modell újszerűsége abban rejlik, hogy képes egymástól eltérő tulajdonságú, korlátos erőforrások ütemezésére. Az erősen illetve gyengén korlátos erőforrások bevezetésével számos eddig nehezen modellezhető probléma vált kezelhetővé. Egy ilyen —a szoftverfejlesztésből származó— mintafeladat egzakt megoldásainak megkeresésén keresztül bemutattuk a 3. részben definiált algoritmust használhatóságát.

A gyakorlati alkalmazások során felmerült annak az igénye, hogy a modellt kiegészítsük az egyes projektek befejezési határidejeivel is. Az i-edik projekt jellemzésére használt vektort ekkor egy újabb E_i értékkel kell kiegészíteni. A dolgozatban vázolt modell erre az esetre is alkalmazható, ha minden határidős projektet kiegészítünk egy olyan erőforrást nem igénylő T_{i,n_i+1} „üres" projekttel, amely időigénye $D_{i,n_i+1} = E(S) - E_i$.

Az illusztrált számadatok is alátámasztják, hogy nagyobb feladatok esetén az egzakt megoldások megkeresése nem kivitelezhető [11], hiszen a leszámlálás időigénye exponenciálisan növekszik. A feladat szerkezetének legjobban megfelelő keresési heurisztikák további vizsgálat tárgyát képezhetik.

Irodalom

1. S. Tsubakitani, R. F. Deckro: A heuristic for multi-project scheduling with limited resources in the housing industry. *Eur. J. of Operational Research* 49 [1990.] p80-91

2. S. E. Elmaghraby: Activity nets: A guided tour through some recent developments *Eur. J. of Operational Research* 82 [1995.] p383-408

3. V. Shankar, R. Nagi: A flexible optimization approach to multi-resource, multi-project planning and scheduling. *5th Industrial Engineering Research Conference*, Minneapolis, MN [1996] p263-267

4. Gy. Csébfalvi, P. Konstantinidis: Egy implicit leszámláláson alapuló új erőforrás kiegyenlítő eljárás. *Szigma* XXIX. [1998.] p43-52

5. Moreno Muffatto: Reorganizing for product development: Evidence from Japanese automobile firms. *Int. J. Production Economics* 56-57 [1998.] p483-493

6. G. Csébfalvi, P. Konstantinidis: A new exact resource balancing procedure for the multiple resource-constrained project scheduling problem *Proc. APMOD '98 Extended Abstracts*, Limasol, Cyprus, 11-13 March, 1998

7. G. Csébfalvi: A fast exact solution procedure for the multiple resource-constrained project scheduling problem. *Proc. APMOD '98 Extended Abstracts*, Limasol, Cyprus, 11-13 March, 1998)

8. B. D. Reyck, W. Herroelen: The multi-mode resource-constrained project scheduling problem with generalized precedence relations *Eur. J. of Operational Research* 119 [1999.] p538-556
9. G. Csébfalvi: Egy optimális erőforrás kiegyenlítő eljárás tevékenységi hálókra. *Új utak a magyar operációkutatásban*, tanulmánykötet, szerkesztők: Komlósi S. és Szántai T., Dialóg Campus, [1999]
10. Gy. Csébfalvi, P. Konstantinidis: A new resource leveling procedure for the multiple resource-constrained project scheduling problem. *Decision Sciences Institute 5th International Conference*, Athens, Greece [1999.] p1723-1725
11. A. Lova, C. Maroto, P. Tormos: A multicriteria heuristic method to improve resource allocation in multiproject scheduling *Eur. J. of Operational Research* 127 [2000.] p408-424
12. G. Csébfalvi: A new multi-criteria resource leveling procedure for the multiple resource-constrained project scheduling problem *Proc. EURO XVII*, [2000] megjelenés alatt
13. M. R. Zamani: A high-performance exact method for the resource-constrained project scheduling problem *Computer & Operations Research* 28 [2001] p1387-1401
14. G. Csébfalvi: *Optimális erőforrás kiegyenlítő modellek tevékenységi hálókra* (Habilitációs értekezés, PTE [2001] Pécs 120p)

EXACT RESOURCE-CONSTRAINED SCHEDULING IN SOFTWARE DEVELOPMENT

In this paper we present a new resource allocation model with hard and soft resource constraints for software development projects. By definition, a hard resource constraint is not resolvable within the given planing horizon, but a soft resource conflict may be managed by a flexible "hiring-firing" strategy. This multi-criteria, multi-project, multi-resource allocation model can be applied to solve small or medium sized real problems even outside of the software development area. According to the definition, an implicit enumeration algorithm is stated that consist of two phases. First a makespan-minimization is applied for the hard resources, than a set of soft resource balancing problem is constructed on the hard resource feasible set of schedules. The practical interpretation of the proposed model is demonstrated in an analysis of a small-scale business software development environment. In this example we exploited the fact that in case of small-scale development the usual iterative "waterfall" structure might be replaced by a serial "designer-programmer-tester" chain.