# Ipari ütemezési feladat megoldása egzakt és metaheurisztikus optimalizálási módszerek kombinálásával

Önálló projekt, szakmai gyakorlat I.
2021/2022 1. félév

*Hatala Imre*

## A projekt témája

A projektben vizsgált ütemezési feladat egy komplex gyártási folyamat ütemezése. A feladat a flexibilis flow shop (Flexible Flow Shop Problem, FFSP) ütemezési problémák körébe tartozik. Általánosan a shop modellekben $n$ munkát kell $m$ gépen elvégezni úgy, hogy minden egyes munkára adott, hogy mely gépeken és hányszor kell megmunkálni. Ha a munkáknak ugyanabban a sorrendben kell végiglátogatniuk a gépeket, akkor flow shop feladatról beszélünk. Ennek általánosítása a flexibilis flow shop feladat, ahol az egyes részmunkák több gépen is elvégezhetők. Itt a munkafolyamat szakaszokra osztott a részmunkák szerint, ahol az egyes szakaszok több gépet is tartalmazhatnak, és ezek közül minden munkának csak egyet kell érintenie.

A gyakorlatban előforduló ütemezési problémák jellemzően NP-nehéz feladatnak bizonyulnak. A flow shop feladatnak például már az a változata is NP-nehéz, amikor a gépek száma három és célunk a teljes átfutási idő minimalizálása. (A következő tételben az ütemezési problémák jelölésére elterjedt három mezős jelölést használjuk.)

**Tétel.** Az $F_3\|C_{max}$ feladat NP-nehéz.

**Bizonyítás.** Megmutatjuk, hogy a partíció feladat, amely NP-teljes, visszavezethető az $F_3\|C_{max}$ feladatra. Adott egy partíció feladat, ahol azt kell eldönteni, hogy az $a_1, a_2, \ldots, a_n$ számoknak, ahol $\sum_i a_i = 2b$, létezik-e olyan $P_1, P_2$ partíciója, amelyre $\sum_{a_i \in P_1} a_i = b$. Tekintsük azt a három gépes flow shop feladatot, amelyben $n+1$ munka van és az egyes részfeladatok ideje: $p_{1,i} = p_{3,i} = 0$ és $p_{2,i} = a_i$ $(i = 1, 2, \ldots, n)$, valamint $p_{1,n+1} = p_{2,n+1} = p_{3,n+1} = b$.

Ennek az ütemezési feladatnak pontosan akkor van legfeljebb $3b$ teljes átfutási idejű megoldása, ha a partíció feladat megoldható. Ha ugyanis van ilyen ütemezés, akkor az csak úgy nézhet ki, hogy az $n+1$. munka állásidő nélkül végigfut a három gépen, a többi munkából pedig néhány előtte fut a második gépen, $b$ összidővel, a többi pedig utána, szintén $b$ összidővel. Ekkor tehát a munkák sorrendje a második gépen a partíció feladat egy megoldását adja. Megfordítva, ha a partíció feladatnak létezik megoldása, akkor a munkákat tudjuk az előbbi módon ütemezni, így egy $3b$ teljes átfutási idejű ütemezést kapunk. □

**A vizsgált probléma**

A projektben vizsgált gyártási folyamat a következő. Egy gyárban alapanyagból 5 lépésen keresztül készül késztermék. Minden lépésnél 5-10 gép áll rendelkezésre, amelyek bármelyikén elvégezhető az adott művelet. A megrendelések a késztermékekre érkeznek, egy tervezési ciklusban jellemzően 100-200 késztermék megrendeléseit kell kielégíteni. A célfüggvényt a következő üzleti szempontok határozzák meg:

- a megrendelések kielégítése határidőre,
- a teljes átfutási idő minimalizálása,
- a gépek állásidejének minimalizálása.

Az ütemezésnek a következő korlátokat kell figyelembe vennie (a következőkben termék alatt egyaránt értünk köztes és késztermeket is):

- minden termékre adott, hogy a gyártása esetén mennyi az egyszerre gyártható minimális és maximális mennyiség,
- minden gépre adott, hogy az egyes termékekek között mennyi az átállási idő,
- minden termékre adott, hogy egy egysége melyik termékből, hány egység felhasználásával áll elő,
- minden lépésre adott, hogy a lépés végén létrejövő termékekből legfeljebb mekkora mennyiség lehet egyszerre jelen - termékenként és összesítve is.

Inputként adott továbbá természetesen a késztermékre vonatkozó megrendelések mennyisége határidővel, valamint a gyár aktuális állapota, azaz

- az egyes gépek mely időponttól használhatók,
- az egyes gépeken melyik terméket gyártották utoljára,
- az egyes lépések végén mennyi termék várakozik aktuálisan.

A projektben vizsgált probléma láthatóan jóval általánosabb mint az $F_3||C_{max}$ feladat (azt speciális esetként tartalmazza), tehát NP-nehéz. Ez nem meglepő, valós ütemezési problémák megoldása tipikusan az alapfeladatoknál sokkal komplexebb (általánosabb) modellt követel meg. Ilyen feladatoknál, ha a probléma nagy méretű, az optimális (vagy közel optimális) megoldás megtalálásához csak heurisztikus módszerek állnak rendelkezésünkre.

A problémát megoldó jelenlegi algoritmus egy MILP modellen alapszik, valamint probléma specifikus heurisztikákat és egyszerűsítéseket alkalmaz. Ez az algoritmus is elegendően jól megoldja a feladatot, azonban szeretnénk még hatékonyabbá tenni metaheurisztikus módszerek segítségével. A metaheurisztikák használatától két téren várunk javulást: egyrészt a teljes algoritmus futását szeretnénk gyorsabbá tenni, másrészt a megtalált megoldás minőségét javítani.

### Mire jó egy metaheurisztika?

A gyakorlatban előforduló, NP-nehéz feladatok esetében az optimális megoldás megtalálása egzakt módszerekkel sokszor reménytelen, mivel a probléma méretének növekedésével a futásidő olyan gyorsan nő, hogy hamar átlépi az elfogadható szintet. Ehelyett, a gyakorlatban, megelégszünk "elég jó" megoldásokkal, amit viszont gyorsan elő tudunk állítani. Ilyen megoldások előállítására használhatunk metaheurisztikákat, amelyek – szemben a probléma specifikus heurisztikákkal – általános megoldási módszerek, algoritmus sémák, amelyeket különböző probléma típusokra is alkalmazhatunk. Metaheurisztikák segítségével nagyméretű optimalizálási problémákra is képesek vagyunk "elég jó" megoldást adni "elég rövid" idő alatt. A metaheurisztikák közelítő megoldást adnak abban az értelemben, hogy a globális optimum elérése nem garantált, ráadásul – a hagyományos approximációs algoritmusokkal ellentétben – a globális optimumtól való eltérés mértékére sem adnak becslést. Nagy előnyük viszont nagy méretű problémák esetén az említett gyorsaság, valamint az egyszerűség, ami gyors tervezést és implementálást tesz lehetővé.

### A lokális keresés és alkalmazása a vizsgált problémára

A lokális keresés (local search, LS) egy széles körben alkalmazott metaheurisztika, amely számos optimalizálási probléma esetén alkalmazható hatékonyan, köztük ütemezi problémák megoldására is. Alapelve, hogy egy adott megoldás lokális környezetében keres javítást. A keresés egy adott kezdeti megoldásból indul, majd iterációkat hajt végre úgy, hogy minden iterációban az aktuális megoldást megpróbálja egy jobbra cserélni. Ehhez azonban a keresési térnek csak egy szűk részhalmazát tekinti át, amelyet egy meghatározott szomszédsági feltétel határoz meg. A legegyszerűbb lokális keresés megáll, amikor olyan pontba jut, ahol a lehetséges alternatív megoldások egyike sem javít a célfüggvényen. Ekkor az algoritmus egy lokális optimumot talál.

Egy konkrét problémát megoldó lokális keresés algoritmus megalkotásához a következőkre van szükség.

1. Kezdeti megoldást kell számítunk.
2. Meg kell határoznunk a szomszédsági relációt.
3. Meg kell határoznunk a stratégiát, ami alapján alternatív megoldást választunk az egyes iterációkban.

Utóbbira tipikus példák a legjobb javítást felmutató alternatív megoldás választása (legjobb), a legelső megtalált javító alternatíva választása (leggyorsabb), vagy véletlenszerű alternatív megoldás választása (randomizált).

Az egyszerű lokális keresés algoritmus előnye, hogy egyszerű megtervezni és implementálni, valamint gyorsan ad használható eredményt. Azonban alapvető jellegzetessége, hogy csupán lokális optimumot talál. Ha ez a megoldás "túl

messze" van a valódi optimumtól, akkor nem lehetünk elégedettek az eredménnyel. Ennek a problémának a megoldására, azaz hogy a keresés kilépjen a lokális optimumból, különböző technikák léteznek, mint például keresés több különböző kezdeti megoldásból, a szomszédsági reláció (ideiglenes) megváltoztatása vagy nem javító cserék megengedése. Utóbbi két esetben természetesen leállási feltétel bevezetése is szükséges. (A nem javító cserék megengedése esetén további kérdéseket is tisztázni kell. Az ilyen típusú algoritmusok a lokális keresések egy külön családját alkotják. Ilyenek például a tabu search és a simulated annealing).

A projektben vizsgált ütemezési probléma esetén ötvözhetjük a lokális keresést és az MILP modellt. A következő konfigurációt alkalmazzuk.

1. Az MILP modell segítségével számoljuk ki a kezdeti megoldást, ahonnan a lokális keresést indítjuk.
2. A szomszédsági relációt a döntési változók lehetséges értékein értelmezett Hamming-távolság függvényében határozzuk meg: két megoldást szomszédosnak tekintünk, ha elég közel vannak egymáshoz. (Ez egy szabályozható paraméter.)
3. A javító megoldásokat az MILP modellt megszorításával keressük, ahol csak azokat a megoldásokat engedjük meg, amelyek az aktuális megoldástól legfeljebb a megengedett mértékben térnek el.

Javító megoldások választásánál a következő stratégiákat tekintjük: az elérhető legjobb alternatíva választása, az első megtalált javító alternatíva választása, az első alternatíva választása, amely legalább $\Delta$ mértékű javítást eredményez a célfüggvényben (itt $\Delta$ szabályozható paraméter).

Ha lokális optimumba ér az algoritmus, akkor egy iteráció erejéig módosítjuk a szomszédsági reláció feltételét: nagyobb távolságra lévő alternatív megoldásokat is szomszédnak tekintünk, azaz a lehetséges megoldásoknak egy nagyobb sugarú gömbjét tekintjük a Hamming-távolság szerint. Ha ezen a bővebb halmazon sem találunk javító megoldást, akkor egy még bővebb halmazt tekintünk. Természetesen elég a gömbnek csak azokat a pontjait vizsgálni, amelyek a megelőző iterációkban még nem szerepeltek a lehetséges alternatívák halmazában. Ez a feltétel is megfogalmazható az MILP modell alkalmas megszorításával.

Az algoritmus leállási feltétele, hogy n iteráción keresztül nem találunk javító megoldást. Az algoritmus további finomításaként kibővíthetjük a leállási feltételt időkorláttal, valamint hibahatárral is. Utóbbit az MILP modellből kapjuk.

További vizsgálat tárgyát képezi, hogy milyen feltétellel érdemes definiálni a szomszédsági relációt a konkrét ütemezési probléma esetén. Kézenfekvő lehetőség, hogy két ütemezést akkor tekintsünk szomszédosnak, ha egyik megkapható a másikból úgy, hogy egy lépésben két különböző (akár különböző gépeken található) munka helyét felcseréljük. Általánosabb értelemben az a kérdés, hogy hogyan érdemes definiálni két ütemezés távolságát, hogy az a lokális keresés során valóban előnyös szomszédsági relációt eredményezzen. Ez a probléma és a jelenlegi MILP modell további vizsgálatát követeli meg.

## Összegzés

Ebben a félévben egy ipari ütemezési problémát vizsgáltam. Ezen belül egyrészt a konkrét üzleti probléma és az optimalizálási feladat megismerésével, megértésével foglalkoztam, másrészt megismerkedtem a metaheurisztikák, azon belül is a lokális keresés alapjaival. Megvizsgáltam, hogy hogyan alkalmazható a lokális keresés az ütemezési probléma megoldására, illetve hogyan építhető be a feladat megoldására készült jelenlegi algoritmusba.

A projekt folytatásaként a felvázolt konfigurációk implementálásával és az általuk szolgáltatott eredmények vizsgálatával tervezek foglalkozni. További lehetséges kutatási irány egy másik metaheurisztika, a hangyakolónia optimalizálási módszer (Ant Colony Optimization, ACO) alkalmazása a feladatra.

## Irodalom

[1] El-Ghazali Talbi: *Metaheuristics: From Design to Implementation.* John Wiley and Sons, (2009)

[2] Michael L. Pinedo: *Planning and Scheduling in Manufacturing and Services.* Springer Science+Business Media, (2005)

[3] Jordán Tibor: *Ütemezés: Elmélet és algoritmusok.* Egyetemi jegyzet, ELTE, (2020)