Általános algoritmustervezési módszerek

Ebben a részben arra mutatunk példát, hogy miként használhatóak olyan általános algoritmustervezési módszerek mint a dinamikus programozás és a korlátozás és szétválasztás elve ütemezési feladatok megoldására.

Dinamikus programozás

A dinamikus programozás lényege, hogy az optimális megoldás célfüggvényértékét kisebb részfeladatok célfüggvényértékei alapján határozzuk meg, és az aktuális részfeladatok célfüggvényértékeit is rendre rekurzív módon mindig kisebb részfeladatok alapján kapjuk meg. Az eljárás lényegében felfogható mint egy táblázat soronkénti kitöltése. Az algoritmus általában egy rekurzív reláción alapul, amely megadja, hogy miként kapjuk meg a bonyolultabb részfeladatok megoldásait az egyszerűbb részfeladatok megoldásai alapján. Az eljáráshoz szükséges még a kezdeti feltételek megadása, amelyek a legegyszerűbb részfeladatok (általában triviális) megoldásait adják meg. Nem részletezzük a dinamikus programozási eljárás ismertetését, a részletek megtalálhatóak a legtöbb bevezető algoritmuselméleti könyvben (pl [1]).

Két dinamikus programozási algoritmust ismertetünk mindkettőt az $1||\sum h_j(C_j)$ problémára (egy gépünk van, a célfüggvény a $\sum h_j(C_j)$ függvény minimalizálása, ahol h_j egy monoton függvény minden j-re). A célfüggvény reguláris (monoton a befejezési időkben), így csak olyan ütemezéseket kell vizsgálnunk, amelyekben a gép folyamatosan dolgozik.

Előre építő algoritmus $1||\sum h_j(C_j)$

Legyen J egy részhalmaza a munkáknak és tegyük fel, hogy elsőként a J-beli munkákat hajtjuk végre! Legyen $V(J)$ az optimális $\sum_{j\in J} h_j(C_j)$ érték, amely a J halmaz ütemezésével elérhető! Ekkor a dinamikus programozási eljárás a következő kezdeti feltételeken és a következő rekurzív reláción alapul.

Kezdeti feltételek:

$V(\{j\}) = h_j(p_j), \quad j = 1, \ldots, n$

Rekurzív reláció:

$$V(J) = \min_{j \in J}\{V(J \setminus \{j\}) + h_j(\sum_{k \in J} p_k)\}.$$

Az eljárás minden iterációs részben meghatározza egy adott halmazára a munkáknak az optimális ütemezési sorrendnek a költségét. Kezdetben az 1-elemű halmazokra ezt az értéket a kezdeti feltételek adják meg. Ezt követően amennyiben az összes l-elemű halmazra ismert az optimális érték a rekurzív reláció alapján meghatározzuk az $l+1$-elemű munkahalmazokra is. Az optimális ütemezés költségét a teljes halmazra kapott érték adja meg. Amennyiben az eljárás során minden halmazra megjegyezzük, hogy a rekurzív reláció során mely j munka esetén kaptuk a minimális értéket, (azaz mely munkát kell utoljára ütemezni a részhalmaz optimális ütemezéséhez) akkor, ezen információ alapján egyből megkapjuk az optimális ütemezést is.

Vizsgáljuk most meg az eljárás időigényét! A $V(J)$ értéket ki kell számítanunk minden J részhalmazára a munkáknak, a függvényérték kiszámítható lineáris időben. Mivel egy n elemű halmaz részhalmazainak száma 2^n, ezért az eljárás időigénye $O(n2^n)$.

Hátra építő algoritmus $1||\sum h_j(C_j)$

A hátrafele építő algoritmus az ütemezést az utolsó munkákból kiindulva építi fel. Ismét J a munkák egy részhalmaza és feltesszük, hogy utolsóként a J-beli munkákat hajtjuk végre. A J^C halmaz a J halmaz komplementere, azaz azon munkák amelyeket az ütemezés elején hajtunk majd végre. A dinamikus programozással kiszámított függvény $V(J)$ a minimális költsége a J halmazbeli munkáknak, azaz a minimális $\sum_{j \in J} h_j(C_j)$ érték azon feltételek mellett, hogy a J halmaz munkáit ütemezzük az ütemezés végén. Ekkor a dinamikus programozási eljárás a következő kezdeti feltételeken és a következő rekurzív reláción alapul.

Kezdeti feltételek:

$V(\{1, \ldots, j-1, j+1, \ldots, n\}) = h_j(\sum_{i=1}^{n} p_i), \quad j = 1, \ldots, n$

Rekurzív reláció:

$$V(J) = \min_{j \in J}\{V(J \setminus \{j\}) + h_j(\sum_{k \in J^C \cup \{j\}} p_k)\}.$$

Az eljárás hasonló az előző részben megadott eljáráshoz. Az eljárás minden iterációs részben meghatározza egy adott halmazára a munkáknak

az optimális sorrendben a költséget feltéve, hogy ez a halmaz helyezkedik el az ütemezésben a munkák sorrendjének a legvégén. Kezdetben az 1-elemű halmazokra ezt az értéket a kezdeti feltételek adják meg, az értékek azon észrevétel alapján adódnak, hogy az utolsó munka befejezési ideje a sorrendtől függetlenül $\sum_{i=1}^{n} p_i$. Ezt követően amennyiben az összes l-elemű halmazra ismert az optimális érték, akkor a rekurzív reláció alapján meghatározzuk az $l+1$-elemű munkahalmazokra is. Az optimális ütemezés költségét a teljes halmazra kapott érték adja meg. Amennyiben az eljárás során minden halmazra megjegyezzük, hogy a rekurzív reláció során, mely j munka esetén kaptuk a minimális értéket, (azaz mely munkát kell elsőként ütemezni a részhalmaz optimális ütemezéséhez) akkor ezen információ alapján egyből megkapjuk az optimális ütemezést is.

Most bemutatunk egy dinamikus programozsi algoritmust a $Pm||C_{\max}$ problmra, (m egyforma prhuzamos gp, a cl a maximlis befejezsi id minimalizlsa), amely algoritmusban felttelezzk, hogy egeszek a vgrehajtsi idk. A probléma egy gép esetén triviális bármely olyan ütemezésre, amelyben a gép folyamatosan dolgozik, (nem várunk a munkák elvégzése közben) a $\max\{C_j : 1 \leq j \leq m\}$ érték megegyezik a végrehajtási idők összegével. Továbbá az is nyilvánvaló, hogy amennyiben minden munkát elvégzünk, akkor nem fejezhetjük be a munkákat ezen időpont előtt. Ha a gépek száma több, mint 1, akkor a feladat lényegesen nehezebb. Igazolást nyert, hogy $n \geq 2$ esetén a probléma NP-nehéz. Másrészt ebben az esetben egy lehetséges ütemezést meghatározhatunk azáltal, hogy az egyes munkákat mely gépekhez rendeljük hozzá. Amennyiben minden gépre megkapjuk, hogy mi a géphez rendelt munkáknak a halmaza, akkor minden egyes gépre, az ott levő munkákat optimálisan úgy ütemezhetjük, hogy a a gép folyamatosan dolgozzon, és minden ilyen ütemezésre ugyanannyi lesz a gépen a maximális befejezési idő, a munkáknak a végrehajtási idejeinek az összege. Ezt az értéket a gép *töltésének* nevezzük. A fogalmat használjuk általánosabb értelemben is, gépek egy halmazának a töltésén a gépekhez rendelt munkák végrehajtási idejeinek az összegének és a gépek számának a hányadosát értjük.

Legyen $P = \sum_{j=1}^{n} p_j$. Definiáljuk az $S_i(K_1, \ldots, K_{m-1})$, értékeket $i = 0, 1, \ldots, n$ a következőképpen. Legyen $S_i(K_1, \ldots, K_{m-1})$ az a minimális töltés, amely elérhető az M_m gépen az első i munka ütemezése során olyan ütemezéssel, amelyben az M_j gépen a töltés legfeljebb K_j minden $j = 1, \ldots, m-1$-re. A függvény definíciója alapján $S_i(K_1, \ldots, K_{m-1}) = \infty$, ha $K_j < 0$ valamely j-re, hiszen ekkor nincs a j-edik gép töltésére vonatkozó

feltételt kielégítő ütemezés. Továbbá $S_0(K_1, \ldots, K_{m-1}) = 0$ minden $0 \leq K_j \leq P$ értékre.

Egyszerű esetszétválasztás alapján (attl fggen, hol temezzk az utols munkt) adódik, hogy az S_i függvényekre teljesül a következő rekurzió:

$$S_{i+1}(K_1, \ldots, K_{m-1}) = \min\{p_{i+1} + S_i(K_1, \ldots, K_{m-1}), \mathrm{MIN}\},$$

ahol

$$\mathrm{MIN} = \min_j\{S_i(K_1, \ldots, K_{j-1}, K_j - p_{i+1}, K_{j+1}, \ldots, K_{m-1}\}.$$

A fenti észrevételek alapján könnyű felépíteni egy dinamikus programozási algoritmust, amely megoldja a $Pm||C_{max}$ problémát egész végrehajtási idők mellett. A kezdeti feltételek és a rekurzió alapján kiszámoljuk az $S_n(K_1, \ldots, K_{m-1})$ értékeket, minden egész $0 \leq K_j \leq P$, $j = 1, \ldots, m-1$ értékre. Ezt követően az optimális célfüggvényérték a minimális

$$\max\{S_n(K_1, \ldots, K_{m-1}), \max_{1 \leq j \leq m-1} K_j\}$$

érték lesz. Másrészt ha az eljárás során (mint az dinamikus programozási algoritmusoknál szokásos) mindig megjegyezzük, hogy az aktuális S_{i+1} függvényértéket melyik S_i érték alapján kapjuk, akkor az optimális célfüggvényértéket adó S_n érték alapján visszafejthető az optimális ütemezés.

Tehát a fentiekben bemutatott algoritmus megoldja a problémát. Vizsgáljuk a futási időt. Az eljárás során lényegében ki kell töltenünk n darab P^{n-1} méretű táblázatot (a lehetséges $K_1, \ldots, K_{n-1}$ értékek), minden érték kiszámítása m műveletet igényel, tehát a futási idő $O(mnP^{m-1})$, azaz rögzített m mellett polinomiális n-ben és $P = \sum_{j=1}^{n} p_j$-ben. A probléma az, hogy az input mérete nem P, hanem $\log P$, mivel binárisan reprezentáljuk a számokat. Unáris reprezentálás mellett az algoritmus polinomiális lenne. (Érdemes megjegyeznünk, hogy az ilyen algoritmusokat *pszeudopolinomiális* algoritmusoknak nevezzük.)

Korlátozás és szétválasztás elve

A korlátozás és szétválasztás elvén alapuló eljárások rendkívül széles körben használtak optimalizálási problémák megoldására. Tekintsünk egy tetszőleges minimalizálási feladatot, ahol a z függvény minimumát keressük egy L halmazon. Az eljáráshoz két függvény szükséges.

Az egyik a szétválasztási függvény, amely a lehetséges megoldások egy tetszőleges részhalmazához a halmaz egy valódi osztályozását rendeli

hozzá (bizonyos változatokban egy a lehetséges megoldásokat tartalmazó, bővebb, de jobb struktúrával rendelkező halmazra definálják a szétválasztási függvényt). A másik egy korlátozó függvény, amely egy tetszőleges $L' \subseteq L$ halmazhoz az L' halmazon felvett függvényértékek egy alsó korlátját rendeli hozzá.

A korlátozás és szétválasztás elvének alapgondolata, hogy a lehetséges megoldásokat egy fa struktúra leveleiben reprezentáljuk, és utána ezt a struktúrát járjuk be. A fa struktúrát a szétválasztási függvény által kapjuk meg. A korlátozó függvény szerepe ott jön elő, hogy a korlátozó függvény segítségével levághatjuk bizonyos részfáit a lehetséges megoldásokat ábrázoló struktúrának. Amennyiben egy részfára tudjuk, hogy a benne szereplő lehetséges megoldások mindegyikére a célfüggvény értéke legalább C és van egy lehetséges megoldásunk, amelyre a célfüggvényérték C, akkor a részfát elhagyhatjuk, hisz nem találhatunk benne jobb megoldást a már ismertnél.

Nem mutatjuk be a korlátozás és szétválasztás elvén alapuló eljárás formális és részletes leírását, nem ismertetjük az eljárás helyességének igazolását, a különböző változatokat. Mindezek a részletek megtalálhatóak a [2] egyetemi jegyzetben.

$1|r_j|L_{max}$ probléma

Példaként az $1|r_j|L_{max}$ problémát tekintjük (egy gép, r_j érkezési és d_j határidők, cél az $L_j = C_j - d_j$ értékek maximumának minimalizálása). A szétválasztás elve itt azon alapul, hogy az ütemezést az időrendben első munkától kezdve építjük fel. A lehetséges munkasorrendeket tartalmazó fát a következőképpen konstruálhatjuk meg. A 0-adik szintben a gyökérben még egyetlen munka helyét sem fixáltuk az ütemezésben. Ezt követően az első szinten n csúcs van, minden egyes munkához a megfelelő részfa azokat a sorrendeket fogja tartalmazni, amelyben az adott munkát hajtjuk végre elsőként. Ezt követően a második szinten a sorrendben második munkát fixáljuk, tehát $n(n-1)$ csúcs van a második szinten. Így folytatva az n-edik szinten a levelekben megkapjuk az összes lehetséges munkasorrendet. Az eljárásban nem kell minden esetben az összes lehetséges kezdeti sorrendet figyelembe vennünk. Ha a $k-1$-edik szinten a $j_1, j_2, \ldots, j_{k-1}$ munkákat már fixáltuk a sorrend elején, akkor csak azokat a j_k munkákat kell figyelembe vennünk következő munkaként, amelyekre

$$r_{j_k} < \min_{l \in J}\{\max(t, r_l) + p_l\},$$

ahol J a még nem ütemezett munkák halmaza, és t a j_{k-1} munka befejezési

ideje. Azért elegendő a fenti feltételt kielégítő munkákat figyelembe venni, mert a fenti feltételt nem kielégítő munkák előtt még ütemezhetünk más munkát anélkül, hogy a tekintett munka kezdési idejét csökkentenénk.

A következő korlátozó függvényt alkalmazzuk. A $k-1$-edik szinten a hátralevő munkákat ütemezzük megszakítást is megengedve. Az egyszerű elveknél használt bizonyításhoz hasonlóan igazolható, hogy ezt az ütemezési problémát az EDD szabály optimálisan megoldja. Amennyiben a korlátozó függvény kiszámítása során olyan lehetséges megoldást kapunk, amelyben nincsenek megszakítások, akkor egy újabb lehetséges megoldáshoz jutunk, és mindazon részfákat, amelyekhez nagyobb korlát tartozik, mint a kapott megoldáshoz elhagyhatjuk. □

References

[1] T. H. Cormen, C. E. Leiserson, R. l. Rivest, *Introduction to algorithms*, MIT Press, 1990.

[2] Imreh B., *Kombinatorikus Optimalizálás*, NOVODAT, 1999.