Approximációs sémák

Az approximációs sémák tulajdonképpen approximációs algoritmusokból álló algoritmusosztályok. Approximációs algoritmusoknak egy olyan sorozatát keressük, amely tetszőlegesen kicsi ε esetén tartalmaz $1+\varepsilon$ approximációs algoritmust. Minden algoritmusnak polinomiális idejűnek kell lennie, de az időigény kiszámítása során a ε értéket konstansként kezeljük. Az időigény és a ε érték kapcsolatától függően az alábbi sémákat definiálhatjuk.

Definíció: Algoritmusok egy H_ε osztályát *polinomiális idejű approximációs sémának* nevezzük (polynomial time approximation scheme, PTAS rövidítve), amennyiben minden $\varepsilon > 0$ esetén H_ε egy $1+\varepsilon$ approximációs algoritmus, amely polinomiális időigényű, (az időigény tetszőlegesen függhet a ε konstanstól).

Definíció: Amennyiben egy H_ε polinomiális idejű approximációs sémára, a H_ε algoritmusok időigénye nem csak az input méretében, hanem $1/\varepsilon$-ban is polinomiális az algoritmusosztályt *teljesen polinomiális approximációs sémának* nevezzük (fully polynomial time approximation scheme, FPTAS rövidítve).

Az approximációs sémák területe igen széles körben vizsgált. Ebben a jegyzetben néhány példán keresztül bemutatjuk az alapvető technikákat, amelyeket ütemezési problémák esetén approximációs sémák kidolgozására alkalmazni szoktak, az érdeklődő olvasó az approximációs sémák területének részletes bemutatását megtalálhatja a [2], [3] munkákban.

Approximációs sémák esetén polinomiális idejű algoritmussal szeretnénk kapni egy megoldást. Lényegében az eredeti problémát egyszerűsítjük le úgy, hogy egyrészt az egyszerűsített változat már megoldható legyen, másrészt ne veszítsünk sokat a megoldás pontosságából. Az approximációs algoritmusok általában három osztályba sorolhatóak aszerint, hogy a probléma egyszerűsítését hol hajtjuk végre. Egyszerűsíthetünk, az inputon, az outputon és a megoldást adó algoritmus egyes fázisai között. Elsőként az egyik legegyszerűbb NP-nehéz ütemezési problémára (amelyben két azonos gép van és a maximális befejezési időt minimalizáljuk) mutatunk be három approximációs sémát, mindhárom, a fentiekben említett megközelítésre egyet - egyet, majd egy bonyolultabb eljárást ismertetünk, amely segítségével dinamikus programozási algoritmusokat transzformálhatunk approximációs sémává.

Az alábbi három példában jelölje $J = \{j_1, \ldots, j_n\}$ a munkák halmazát, rendre $p_1, \ldots, p_n$ a végrehajtási időket, és legyen $\varepsilon > 0$ tetszőleges kicsi szám. Legyen továbbá $L = \max\{\max p_i, (\sum_{i=1}^{n} p_i)/2\}$. Ekkor miként azt az approximációs algoritmusokat tárgyaló fejezetben igazoltuk, két gép esetén $L \leq OPT(J) \leq 2L$.

Inputban egyszerűsítő PTAS a $P2||C_{max}$ problémára.

Az első séma alapgondolata, hogy az apró munkákat nem kezeljük egyenként, hanem olyan tölteléknek tekintjük őket, amelyeknek csak az össztöltése számít.

Approximációs séma I. ($A1_\varepsilon$)

1. Lépés: Osszuk a munkákat két halmazba! Legyenek az $\varepsilon L/2$-nél nem kisebb munkák a nagy munkák, a kisebbek a kicsi munkák. Konstruáljuk a következő J' inputot. A nagy munkákat változatlanul hagyjuk, a kicsi munkákat pedig $\varepsilon L/2$ nagyságú munkákkal helyettesítjük a következőképpen. Ragasszuk össze az összes kicsi munkát egyetlen óriásmunkába, és az így kapott óriásmunkát szeleteljük fel $\varepsilon L/2$ nagyságú darabokra, a maradék $\varepsilon L/2$-nél kisebb darabot dobjuk el! Nevezzük ezeket a munkákat gyűjtőmunkáknak!

2. Lépés: Oldjuk meg az I' inputra a problémát, az összes lehetséges megoldást megvizsgálva! Mivel az egyes gépeken nem számít a munkák sorrendje, ezért feltehetjük, hogy a kapott optimális megoldásban, mindkét gépen először a régi nagy munkákat hajtjuk végre, utána a gyűjtőmunkákat. Továbbá a gyűjtőmunkák mind $\varepsilon L/2$ nagyságú munkák, így feltehetjük, hogy az első gépen a gyűjtőmunkák a felszeletelt óriásmunka kezdőszeletét alkotják, a második gépen a végszeletét. Jelölje ezt az ütemezést S'!

3. Lépés: Az S' ütemzésből konstruáljuk meg J egy ütemezését! A nagy munkákat ütemezzük ugyanazon a gépen, mint S', a gyűjtőmunkákat pedig bontsuk vissza az eredeti kicsi munkákra, továbbá az óriásmunka konstruálásánál eldobott részt még illesszük a második gép végére! Mivel a gépeken az óriásmunka kezdő illetve végszelete van, ezért a visszabontás során csak a első gép utolsó, és a második gép első gyűjtőmunkájánál keletkezik nem visszabontható, szétvágott kicsi munka. Ezt ütemezzük az első gépen!

Tétel *$A1_\varepsilon$ egy PTAS.*

Bizonyítás: Vizsgáljuk elsőként a futási időt! Az első és a harmadik lépés nyilvánvalóan polinomiális. A második lépésben az I' inputra megvizsgáljuk

az összes lehetséges megoldást, ez $O(2^{n'})$ lépés, ahol n' a munkák száma. Másrészt minden munka legalább $\varepsilon L/2$ méretű, a munkák össztöltése az átdarabolás során nem növekedhetett, így a munkák száma legfeljebb $4/\varepsilon$. Következésképp rögzített konstans ε mellett ezen rész futási ideje konstans. (Érdemes megjegyeznünk, hogy ez a konstans exponenciális $1/\varepsilon$-ban, ezért nem FPTAS-t kapunk.) Következésképp a futási idő valóban polinomiális az input méretében minden ε-ra.

Most vizsgáljuk az approximációs hányadost! Elsőként vegyük észre, hogy $OPT(J') \leq OPT(J) + \varepsilon L/2$. Valóban J egy optimális ütemezésében, a nagy munkákat helyben hagyva, a kicsi munkákat kicserélve annyi εL nagyságú munkával, hogy az össztöltésük ne csökkenjen, egy olyan lehetséges megoldását kapjuk J'-nek, amelyre a maximális befejezési idő legfeljebb $\varepsilon L/2$-vel növekedett. Másrészt a gyűjtőmunkák szétdarabolása során is legfeljebb $\varepsilon L/2$-vel növekszik a maximális befejezési idő, hiszen az első gépen az utolsó kicsi munka másik gyűjtőmunkába eső darabja, a második gépen az óriásmunkából eldobott rész növelheti meg a maximális befejezési időt.

Következésképpen

$$A1_\varepsilon(J) \leq OPT(J') + \varepsilon L \leq OPT(J) + 2\varepsilon L/2 \leq (1+\varepsilon)OPT(J),$$

amivel igazoltuk, hogy a definiált eljárásosztály valóban egy approximációs sémát ad. □

Outputban egyszerűsítő PTAS a $P2||C_{max}$ problémára.

Az outputban egyszerűsítő sémák alapgondolata általában az, hogy a lehetséges megoldásokat polinomiális számú osztályba osztjuk fel, és minden osztályra alkalmazunk egy gyors algoritmust, amely kijelöl az osztályból egy reprezentánst. Amennyiben minden reprezentáns közel van az osztályba tartozó legjobb lehetséges megoldáshoz, akkor a reprezentánsok közül a legjobb közel lesz az optimális megoldáshoz, így egy jó approximációs algoritmust kapunk.

Approximációs séma II. ($A2_\varepsilon$)

1. Lépés: Osszuk a munkákat két halmazba! Legyenek az εL-nél nem kisebb munkák a nagy munkák, a kisebbek a kicsi munkák! Vegyük a nagy munkák összes lehetséges ütemezését, ezeket nevezzük parciális ütemezéseknek! Minden parciális ütemezésre definiáljuk a lehetséges

ütemezéseknek egy osztályát, amely azokat a lehetséges megoldásokat tartalmazza, amelyekben a nagy munkák az adott parciális ütemezésnek megfelelően vannak a gépekhez hozzárendelve!

2. Lépés: Minden osztályra számoljunk ki egy reprezentánst! A reprezentánst úgy kapjuk, hogy a parciális ütemezést kiterjesztjük a kicsi munkákkal, azokat a LISTA algoritmus szerint ütemezve (az aktuális munka arra a gépre kerül, ahol kisebb az aktuális töltés).

3. Lépés: Vegyük a reprezentánsok közül a legkisebb célfügvényértékkel rendelkező megoldást!

Tétel *$A2_\varepsilon$ egy PTAS.*

Bizonyítás: Vizsgáljuk elsőként a futási időt! Mivel a nagy munkák mérete legalább εL és a munkák össztöltése legfeljebb $2L$, ezért a nagy munkák száma legfeljebb $2/\varepsilon$. Következésképpen a parciális ütemezések, és így az osztályok száma legfeljebb $2^{2/\varepsilon}$. Minden osztályra a reprezentáns kijelölése lineáris időben fut, így rögzített ε mellett az algoritmus futási ideje valóban polinomiális. (Fontos megjegyeznünk, hogy $1/\varepsilon$-ban exponenciális, így ismét csak PTAS-t kapunk.)

Vizsgáljuk most az approximációs hányadost! Különböztessünk meg két esetet attól függően, hogy milyen reprezentánsokat kaptunk!

Elsőként tegyük fel, hogy van olyan reprezentáns, amelyben egy kicsi munka éri el a maximális befejezési időt. Ez akkor fordul elő ha mindkét gépre kerül kicsi munka vagy ha csak a parciális ütemezésben kisebb töltésűre, de az utolsó kicsi munkával a töltés túlnő a parciális ütemezésben nagyobb töltésű gép töltésén. Mindkét esetben ezen reprezentánsra a gépeken a befejezési idők (esetünkben megegyeznek a töltéssel) különbsége legfeljebb εL. Másrészt az össztöltés legfeljebb $2L$, így azt kaptuk, hogy

$$A2_\varepsilon(J) \leq L + \frac{\varepsilon}{2}L \leq (1 + \varepsilon/2)OPT(J).$$

A második esetben nincs olyan reprezentáns, amelyben kicsi munka éri el a maximális befejezési időt. Ez azt jelenti, hogy minden osztályra a reprezentáns optimális megoldást ad, hiszen a költség megegyezik a parciális ütemezés költségével. Másrészt ebből következik, hogy a reprezentánsok között szerepel egy optimális megoldás is. □

Algoritmusban egyszerűsítő FPTAS a $P2||C_{max}$ problémára.

Ezt a megközelítést akkor szokás használni, ha van a problémára egy exponenciális megoldó algoritmusunk, amely több fázisból áll. Ekkor bizonyos esetekben az egyes fázisok közé valamilyen egyszerűsítő lépést beiktatva egy approximációs sémát kaphatunk.

Elsőként tekintsük a következő egyszerű algoritmust, amely meghatározza az összes (a, b) párt, amelyek előállhatnak a töltésként a két gépen a J munkahalmaz ütemezése után!

Egy exponenciális idejű algoritmus

1. Lépés: Legyen $i = 0$, $S_0 = \emptyset$!

2. Lépés: Amennyiben $i = n$ véget ért az eljárás, egyébként minden $(a, b) \in S_i$ párra képezzük az $(a + p_{i+1}, b)$ és $(a, b + p_{i+1})$ párokat, és ezen párok alkossák az S_{i+1} halmazt!

Egyszerűen igazolható teljes indukcióval, hogy az S_i halmaz minden i-re azokat az (a, b) párokat tartalmazza, amelyek előállhatnak töltésként a két gépen a J munkahalmaz első i elemének az ütemezése után. Következésképp kiválasztva az S_n halmazból azt a párt, amelyre az elemek maximuma a minimális megkapjuk az optimális célfüggvényértéket. Amennyiben a párokat kiegészítjük azzal (a dinamikus programozási algoritmusok esetén szokásos) információval, hogy miként kaptuk meg a párt a megelőzőből (ez egy extra bit, amely megadja, hogy az utolsó munkát melyik géphez rendeltük), akkor megkapjuk az optimális megoldást is.

Tehát a fenti algoritmus alapján megkaphatjuk az optimális ütemezést. Az egyetlen probléma az, hogy az S_i halmazok elemszáma rendkívül gyorsan növekedhet, elérheti a 2^i értéket. Amennyiben ezt az elemszámot korlátozni tudjuk, akkor egy polinomiális algoritmust kapunk. Az alábbiakban megvizsgáljuk miként tehetjük ezt meg oly módon, hogy ne veszítsünk sokat a pontosságból. A továbbiakban feltesszük, hogy a munkák végrehajtási idejei egészek, így minden végrehajtási idő legalább 1.

Legyen $\Delta = 1+\varepsilon/(2n)$, $P = \sum_{i=1}^{n} p_i$! Osszuk fel a síkban a $P \times P$ nagyságú négyzetet tartományokra! A P_{jk}, $j \geq 1$, $k \geq 1$ tartomány tartalmazza azokat az (a, b) pontokat, amelyekre $\Delta^j \leq a \leq \Delta^{j+1}$, $\Delta^k \leq b \leq \Delta^{k+1}$ teljesül! A P_{0k}, $k \geq 1$ tartomány tartalmazza azokat az (a, b) pontokat, amelyekre $0 \leq a \leq \Delta$, $\Delta^k \leq b \leq \Delta^{k+1}$ teljesül! A P_{j0}, $j \geq 1$ tartomány tartalmazza azokat az (a, b) pontokat, amelyekre $\Delta^j \leq a \leq \Delta^{j+1}$, $0 \leq b \leq \Delta$ teljesül! A P_{00} tartomány tartalmazza azokat az (a, b) pontokat, amelyekre

$0 \leq a \leq \Delta$, $0 \leq b \leq \Delta$ teljesül! Az approximációs séma alapgondolata, hogy az ugyanazon tartományba eső pontpárok közel vannak egymáshoz, így elég mindegyik tartományból egy pontot megőrizni az S_i halmazokban.

Approximációs séma III. ($A3_\varepsilon$**)**

1. Lépés: Legyen $i = 0$, $S_0 = \emptyset$!

2. Lépés: Amennyiben $i = n$ véget ért az eljárás, egyébként minden $(a, b) \in S_i$ párra képezzük az $(a + p_{i+1}, b)$ és $(a, b + p_{i+1})$ párokat, és ezen párok alkossák az S'_{i+1} halmazt!

3. Lépés (ritkítás) Az S'_{i+1} azon elemeinek halmazát, amelyben egyik érték sem 0 ritkítsuk úgy, hogy minden P_{jk} tartományból csak egy elemet tartsunk meg!

Tétel $A3_\varepsilon$ *egy FPTAS-t ad az optimális célfüggvényérték meghatározására.*

Bizonyítás: Vizsgáljuk elsőként a futási időt! Ehhez össze kell számolnunk a tartományok számát. A defincióból egyből adódik, hogy a tartományok száma legfeljebb $(\log_\Delta P + 1)^2 = O((\log_\Delta P)^2)$. Másrészt

$$\log_\Delta P = \frac{\ln P}{\ln \Delta}.$$

Továbbá $\ln\Delta = \ln(1 + \varepsilon/(2n)) = \Omega(\varepsilon/n)$, amiből adódik, hogy a tartományok száma polinomiális n-ben $1/\varepsilon$-ban és $\ln P$-ben, azaz az input méretében és $1/\varepsilon$-ban. Ebből következik, hogy az algoritmus is polinomiális ezekben az értékekben, azaz futási idő szempontjából valóban egy FPTAS-t adtunk meg.

Vizsgáljuk az approximációs hányadost! Minden iterációban van egy ritkító fázis. A ritkító részben minden párhoz egy olyan párt őrzünk meg, amely ugyanabban a tartományban van, így mindkét komponensben a megfelelő érték legfeljebb egy multiplikatív Δ hibában tér el. Következésképp minden lehetséges ütemezést leíró párhoz van az S_n halmazban egy olyan pár, amely legfeljebb egy Δ^n multiplikatív hibában tér el. Tehát az algoritmus Δ^n approximációs. Másrészt egyszerűen adódik az $(1+1/n)^n \leq e$ egyenlőtlenség alapján, hogy

$$\Delta^n = (1 + \varepsilon/2n)^n \leq 1 + \varepsilon,$$

amivel az állítást igazoltuk. □

A fenti eljárás könnyen módosítható úgy, hogy az optimális ütemezést is megkapjuk. Számhármasokat kell használnunk, ahol a harmadik komponens attól függ, hogy az utolsó munkát melyik gépen ütemeztük a töltéseket eredményező ütemezésben. Ebben az esetben az optimális (approximációs) számhármasból visszafejthető egy optimális (approximációs) ütemezés.

Dinamikus programozáson alapuló FPTAS a $Pm||C_{max}$ problémára.

Ebben a részben arra mutatunk példát miként transzformálhatóak bizonyos dinamikus programozási algoritmusok FPTAS-á. Az FPTAS alapötlete az, hogy az inputot kerekítjük, az így kapott új inputra végrehajtjuk a dinamikus programozási eljárást, amely a kerekített inputra polinomiális idejű lesz, és a kapott optimális megoldást visszatranszformáljuk az eredeti feladat egy lehetséges megoldásává. Tehát az alábbi algoritmus is az inputban egyszerűsítő algoritmusok családjába tartozik. A bemutatott algoritmus lényegében megegyezik a [1] cikkben ismertetett eljárással. Használjuk a dinamikus programozási részben ismertetett dinamikus programozási algoritmust, a követhetőség érdekében iott ismét ismertetjük az eljárást.

Dinamikus programozási eljárás a $Pm||C_{max}$ problémára egész ütemezési dők mellett

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_i minden $j = 1, \ldots, m-1$-re. A függvény definíciója alapján adódik, hogy $S_i(K_1, \ldots, K_{m-1}) = \infty$, 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 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 ütemezési

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^{m-1} méretű táblázatot (a lehetséges $K_1, \ldots, K_{m-1}$ értékek), minden érték kiszámítása m műveletet igényel, tehát a futási idő $O(nmP^{m-1})$, 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.) Az alábbiakban bemutatjuk miként konstruálható egy FPTAS a fentiekben bemutatott dinamikus programozási eljárás alapján.

FPTAS a dinamikus programozási eljárás alapján

Az FPTAS előkészítő részeként hajtsunk végre egy konstans approximációs hányadossal rendelkező approximációs eljárást az inputon. A kapott célfüggvényértéket jelölje L. Esetünkben használhatjuk a LISTA algoritmust, amely lineáris időigényű. Ekkor az L számra teljesül az $OPT(J) \leq L \leq 2OPT(J)$ egyenlőtlenség.

Approximációs séma IV. ($A4_\varepsilon$)

1. lépés: Konstruáljuk meg a J inputból J' inputot a következőképpen! Minden j-re a p_j végrehajtási idővel rendelkező j-edik munkát helyettesítsük egy $p'_j = \lfloor p_j/Z \rfloor$ végrehajtási idővel rendelkező munkával, ahol $Z = \varepsilon L/2n$!

2. lépés: Oldjuk meg az 1. lépés során kapott J' inputot a fentiekben ismertetett dinamikus programozási eljárással!

3. lépés: A J' input optimális ütemezéséből J egy közelítő megoldását kapjuk, ha minden munkát ugyanazon a gépen ütemezünk, amelyen a módosított input optimális megoldása a megfelelő módosított munkát ütemezi.

Tétel *$A4_\varepsilon$ egy FPTAS.*

Bizonyítás: Elsőként vizsgáljuk a futási időt! A dinamikus programozási algoritmus időigénye $O(nmP^{m-1})$. A módosított J' inputra

$$P' \leq \sum_{j=1}^{n} \lfloor p_j/Z \rfloor \leq \sum_{j=1}^{n} p_j/Z \leq mOPT(J)/Z = \frac{mOPT(J)}{\varepsilon L/2n} \leq 4mn/\varepsilon.$$

Következésképp az eljárás időigénye rögzített m mellett valóban polinomiális az input méretében és $1/\varepsilon$-ban.

Vizsgáljuk meg az approximációs hányadost!

Mivel $Z \cdot OPT(J') \leq OPT(J)$ és $A4_\varepsilon(J) \leq Z \cdot OPT(J) + nZ$, ezért $|A4_\varepsilon(J) - OPT(J)| \leq nZ$. Következésképpen:

$$\frac{|A4_\varepsilon(J) - OPT(J)|}{OPT(J)} \leq \frac{nZ}{OPT(J} \leq \frac{\varepsilon L}{2OPT(I)} \leq \varepsilon.$$

□

References

[1] E. Horowitz, S. Sahni: Exact and approximate algorithms for scheduling non identical processors, *Journal of the ACM*, **23**, 317–327, 1976.

[2] P. Schuurman, G. J. Woeginger, Approximation Schemes - A Tutorial, megjelenés alatt (elérhető Woeginger weboldaláról)

[3] V. Vazirani *Approximation Algorithms*, Springer-Verlag, Berlin, 2001