Eötvös Loránd Tudományegyetem

Természettudományi Kar

Farkas Dóra

Optimalizálás a 3D nyomtatásban

BSc szakdolgozat

Témavezető:

Király Tamás
Operációkutatási Tanszék

Budapest, 2018

Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőmnek, Király Tamásnak a téma kiválasztásában és a szakdolgozat elkészítésében nyújtott segítségéért. A konzultációkon nyújtott hasznos tanácsaival és ötleteivel folyamatosan segítette a munkámat.

Továbbá köszönettel tartozom családomnak, akik mindig támogattak.

Tartalomjegyzék

1. Bevezetés **4**

2. A feladat leírása **5**
2.1. A 3D nyomtatás 5
2.2. Az útvonal meghatározására vonatkozó megkötések 6
2.3. Költségek 7

3. Az utazóügynök feladat és a Lin-Kernighan algoritmus **9**
3.1. Az utazóügynök feladat 9
3.2. k-optimális túrák, δ-utak 9
3.3. A Lin-Kernighan algoritmus 12

4. Egyfejes nyomtatók **17**
4.1. A matematikai modell 17
4.2. A Lin-Kernighan algoritmus módosítása 18

5. Többfejes nyomtatók **20**
5.1. Bevezetés 20
5.2. Útvonalforgatás 21
5.3. Ütközőzónák 22
5.4. Várakoztatás 23
5.4.1. Várakoztatás kétfejes nyomtatók esetén 23
5.4.2. Várakoztatás kettőnél több fej esetén 25
5.4.3. Eredmények 30

Irodalomjegyzék **35**

1. fejezet

Bevezetés

A szakdolgozatban a 3D nyomtatással, ezen belül is elsősorban épületek nyomtatásának optimalizálásával foglalkozom. A nyomtatandó épület alaprajza alapján kell a nyomtatófej útvonalát úgy meghatározni, hogy az egész nyomtatási folyamatot a lehető legolcsóbban, a lehető legrövidebb idő alatt tudjuk megvalósítani.

A 2. fejezetben először rövid ismertetőt adok a 3D nyomtatásról általában, majd rátérek az épületnyomtatásra. Zhang és Khoshnevis [2] cikke szerint bemutatom, hogy milyen szempontok, megszorítások alapján lehet megtervezni az útvonalat, és a feladat költségeit is definiálom, köztük a nyomtatófej sebességéből és az útvonalból kiszámítható nyomtatási időt.

Az útvonal megtervezéséhez az utazóügynök feladatot fogjuk használni, ezt a 3. fejezetben írom le. Ez a feladat NP-teljes, azonban általában a Lin-Kernighan algoritmus a legjobb heurisztika a megoldására. Emiatt Király, Kis, Szegő [3] jegyzete és Applegate, Bixby, Chvátal, Cook [4] cikke alapján ezt is bemutatom.

A 4. fejezetben az egyfejes nyomtatókról lesz szó. [2]-t felhasználva leírom, hogy az épület alaprajzát hogyan tudjuk gráfként reprezentálni, amelyre utána alkalmazni lehet a Lin-Kernighan algoritmust. Mivel így speciális tulajdonságokkal rendelkező gráfot kapunk, ezért a Lin-Kernighan algoritmust módosíthatjuk úgy, hogy ne tegyen meg fölösleges lépéseket, és garantáltan olyan utazóügynök túrát adjon meg, amilyenre a nyomtatási feladatban szükségünk van. Ez utóbbi saját eredmény.

Többfejes nyomtatók útvonalát hasonlóan határozhatjuk meg, mint az egyfejes esetben. Azonban itt további feltétel, hogy a fejek ne ütközhessenek egymással. Ennek megoldására három különböző módszert mutatok be az 5. fejezetben. Az útvonalforgatás és az ütközőzónák használatának ötletét [2]-ből vettem át, a várakoztatási módszer viszont saját munka eredménye. Az ennél leírt, dinamikus programozást használó algoritmusokat le is programoztam, a dolgozat végén ennek futási eredményei találhatók.

2. fejezet

A feladat leírása

A következő fejezet nagyrészt [2]-re épül.

2.1. A 3D nyomtatás

3D nyomtatással többek között háromdimenziós prototípusokat, apró alkatrészeket, de akár egész házakat is elkészíthetünk. A különböző feladatokra különböző méretű nyomtatókat fejlesztettek ki. Az additív gyártási eljárás lényege, hogy a számítógéppel elkészített 3D modellt azonos vastagságú vízszintes rétegekre osztjuk, majd a nyomtató robot segítségével az alkotó anyagot (pl. műanyagot, fémet, betont) a megfelelő alakban egymásra rétegezzük. A hagyományos eljárásokkal szemben – ahol egy nagyobb darabból kiindulva, a felesleges részek eltávolításával kapjuk a kész terméket – a 3D nyomtatás legnagyobb előnyei, hogy segítségével gyorsan, olcsón, geometriai korlátozások nélkül, környezetbarát módon készíthetünk egyedi alkatrészeket. A nagyméretű nyomtatók is könnyen össze-, illetve szétszerelhetők, a működésük - a szükséges felügyeleten kívül - teljesen automatizált. Azonban hátrányai is vannak: pl. a nyomtatott felületek nem mindig érik el a kívánt minőséget, ekkor további utómunkára van szükség.

A nyomtatási folyamat előkészítése több lépésből áll. A számítógépes modellt tervezhetjük közvetlenül erre szolgáló programok segítségével, vagy ha egy tárgyat beszkennelünk, akkor az így kapott pontok koordinátái alapján háromszögráccsal közelíthetjük az eredeti felszínt. A modellt megfelelő irányból párhuzamos síkokkal metsszük el, hogy megkapjuk a kívánt vastagságú rétegeket. A vastagság befolyásolja a gyártmány minőségét, viszont vékonyabb rétegeket használva a nyomtatási idő jelentősen hosszabb. Ezután minden rétegen meghatározzuk, hogy a nyomtatófej milyen utat járjon be a nyomtatás során. A tervezésben ez egy kulcsfontosságú feladat. Az optimalizálás során a felület minőségére, a fej útvonalának összhosszára és a nyomtatási folyamat idejére kell figyelnünk.

Tömör, többnyire összefüggő felületek esetén általában két különböző technikával dolgoznak: a nyomtatófej az adott területen oda-vissza, egy adott tengellyel párhuzamos vonalakon halad végig, vagy az útvonal a körvonallal „párhuzamos". Erről [1]-ben olvashatunk részletesebben.

Épületek nyomtatásakor (contour crafting) a falak felépítése a feladat. Az erre kifejlesztett gépek hasonló tulajdonságokkal rendelkező falakat hoznak létre, mint a széles körben használatos beton téglából épült falak. Az előkészítés során a falrészek nyomtatásának sorrendjét kell úgy meghatároznunk, hogy az építési költségeket minimalizáljuk. Ennek lényegi része a nyomtatófej utazási ideje, vagyis az az idő, amíg eljut egyik faltól a másikhoz, miközben nem nyomtat semmit. Nagyobb épületek esetén többfejes nyomtatót

2.1. ábra. A dán 3DPrinthuset által kifejlesztett nyomtató az első európai 3D-nyomtatott épületen dolgozik. [5]

használva csökkenthetjük az építés idejét és költségét. Ebben az esetben a fejek útvonalát úgy kell meghatároznunk, hogy azok ne ütközhessenek egymással.

Ha a modellben már megvan a megfelelő útvonal, akkor ezt a nyomtató által is értelmezhető utasítások sorozatára kell fordítani, hogy az valóban ki tudja nyomtatni az épületet.

2.2. Az útvonal meghatározására vonatkozó megkötések

A korábban ismertetett tervezési folyamatból mi a nyomtatófej útvonalának meghatározásával, optimalizálásával foglalkozunk. Tehát az épület rétegekre felszelelt modelljét már készen kapjuk. Az útvonal meghatározása alatt azt értjük, hogy a nyomtatási folyamat során minden időpillanatban megadjuk a fej pozícióját, irányát, sebességét, és a beton áramlásának sebességét. A következőket szem előtt kell tartanunk a tervezés során.

Egy rétegen belül a falszegmenseket egyben nyomtatjuk ki, azaz mielőtt egy újabb falat elkezdenénk, az előzőt befejezzük. Így a fej csak ezek végpontjai között mozog. Az ablakok és ajtók miatt szükséges nyílásokat leszámítva a rétegek ugyanolyanok. Azokban a rétegekben, ahol egy falszegmensben ilyen rés van, a fej ugyanúgy halad keresztül, mint a többiben, csak ezekben ilyenkor nem nyomtat.

Mielőtt egy újabb réteghez hozzákezdenénk, az alatta lévő réteget teljesen be kell fejeznünk. Ez igaz több fejes rendszerek esetén is, tehát egy időben az összes fej ugyanazon a rétegen dolgozik. Ez azért szükséges, mert garantálnunk kell, hogy amikor a fej egyik pontból egy másikba utazik (és közben nem nyomtat), akkor ne ütközhessen a már megépített falakkal. A feladat úgy is megoldható lenne, ha nem rétegről rétegre haladnánk. Ekkor viszont az ütközés elkerülése érdekében két pont között a fej nem feltétlen a legrövidebb, egyenes úton haladna. Az ilyen kerülőutak nagyon hosszúak és bonyolultak is

lehetnek, ezért mi feltesszük, hogy a fejnek be kell fejeznie a réteget, mielőtt a következőre rátérne. Ebben az esetben úgy tudjuk elkerülni a fej és a már megépített falak ütközését, hogy mozgás közben a fejet legalább egy réteg magasságával feljebb emeljük.

Egy fejnek minden rétegen ugyanabból a pontból kell indulnia, ahol befejezi a nyomtatást. Továbbá feltesszük azt is, hogy a beton áramlását tökéletesen tudjuk szabályozni: bármikor elindíthatjuk és megállíthatjuk azt.

Ezek alapján elég egy rétegen elvégezni az útvonal megtervezését, mert ugyanez használható lesz az összes rétegen.

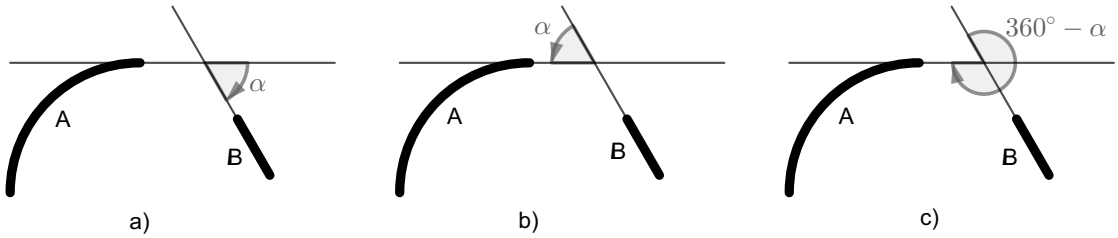
Mielőtt egy újabb réteget elkezdenénk, az alsóbb rétegeknek eléggé meg kell szilárdulni ahhoz, hogy meg tudják tartani a fölötte levőket. Viszont a szomszédos rétegeknek össze kell tapadniuk, ezért túl sok időnek sem szabad eltelnie, míg a fej ugyanoda visszaér a következő rétegen. Ez az idő megegyezik egy réteg nyomtatásának összidejével, amelyet az útvonalak ismeretében ki tudunk számolni. Ha ez egy adott kritikus időnél nagyobb, az arra utal, hogy az építmény túl nagy ahhoz, hogy egyetlen nyomtatófej segítségével építsük fel. Ekkor tehát többfejes nyomtatót kell használnunk. Ha egy réteget hamarabb befejezünk, minthogy a következőt elkezdhetnénk, akkor megpróbálhatjuk csökkenteni a fejek számát, vagy várakozási időt iktatunk be a rétegek nyomtatása közé.

2.3. Költségek

Az építés költsége a munkafolyamat idejéből, a gépek működésére fordított energiából és a felhasznált alapanyag mennyiségéből áll össze.

A falakra vonatkozó költség a beton kibocsátásának sebességétől és a nyomtatófej sebességétől függ. Viszont egy rétegben a fejnek minden falrészen pontosan egyszer kell végigmennie, ezért ha ismerjük a gép paramétereit, akkor a falak nyomtatására vonatkozó költségeket ki tudjuk számolni. Az optimális útvonal tehát nem függ ezektől.

Az utazási költség a fej két pont közötti „tétlen” haladásából és forgásából tevődik össze. Nyomtatás közben a fejnek a fal érintőjére merőleges irányban kell haladnia. Minden falszegmens két végpontjában meghatározzuk a megfelelő irányt, amilyen pozícióban állnia kell a fejnek, amikor ott a fal nyomtatását elkezdi, vagy épp befejezi. Így tehát tudjuk, hogy egyik pontból a másikba történő utazáskor mekkora távolságot kell megtenni, és közben milyen nagyságú szöggel kell elfordulni. Két pont között a mozgási költség a pontok euklideszi távolságából és a gép sebességéből számolható ki. Továbbá ide tartozik az is, amikor a fejet egy szinttel feljebb kell emelnünk, majd vissza leeresztenünk, hogy ne ütközzön egy már megépített fallal. A forgatás költsége kiszámolható a megfelelő falak végpontjába húzott érintő egyenesek szögével. Ha a nyomtató felépítése lehetővé teszi, hogy a fej korlátlanul körbeforduljon, akkor ez a szög ugyanolyan nagyságú, csak ellentétes irányú lesz, ha ellenkező irányban járjuk be a két pont közötti utat (2.2. ábra a) és b) része). Azonban ez nem mindig megengedett, mert így a gépet működtető kábelek és vezetékek összegubancolódhatnak, sérülhetnek. Ennek megelőzése érdekében a fejre egy ütközőt szerelnek, ami megakadályozza, hogy azon keresztülforduljon a fej. Ilyenkor elképzelhető, hogy α helyett $360° - \alpha$ szöggel kell elfordulnunk a másik irányban (2.2. ábra c) része). Ez „elrontja” a feladat szimmetriáját: két pont között különböző költséget eredményezhet, ha ott az ellenkező irányban halad a fej. (Ez aszimmetrikus utazóügynök problémához vezetne, viszont a dolgozatban csak a szimmetrikus változatot mutatom be.) Ebben az esetben a forgásokhoz tartozó költség attól is függ, hogy az ütköző a nyomtatás elején milyen pozícióban áll.



2.2. ábra. Az elfordulás nagysága, amikor a) az A faltól a B-ig megyünk; b) a B-től az A-ig megyünk; c) szintén a B-től az A-ig, de az ütköző miatt nem lehet úgy, mint a b)-ben

Összességében, ha ismert két pont távolsága, és a megfelelő forgási szög, akkor meg tudjuk határozni a mozgáshoz, illetve forgáshoz szükséges időt. Ha a fej mozgás közben is foroghat, akkor a költség az előbbi két idő maximuma lesz. Ha ez nem megengedett, akkor a kettő összegét kell venni.

Ha a költségeket definiáltuk, akkor optimalizálással meghatározhatjuk a legjobb útvonalat. Ehhez a feladatot az utazóügynök problémára vezetjük vissza.

3. fejezet

Az utazóügynök feladat és a Lin-Kernighan algoritmus

A következő fejezetben [3] és [4] alapján ismertetem a Lin-Kernighan algoritmust és a hozzá kapcsolódó fogalmakat.

3.1. Az utazóügynök feladat

Az utazóügynök feladat (traveling salesman problem, röviden TSP) a következő: Adott néhány város, és ezek közül bármely kettő között az utazás költsége. A költségfüggvény szimmetrikus, azaz A-ból B-be ugyanannyiba kerül eljutni, mint B-ből A-ba. A feladatunk, hogy megtaláljuk a legolcsóbb ún. utazóügynök túrát, amelyet követve az utazóügynök minden várost pontosan egyszer látogat meg, és a túra végén a kiinduló városba tér vissza.

A feladat modellezhető egy gráffal, amelyben a csúcsok felelnek meg a városoknak. Adott tehát a $G = (V, E)$ n pontú teljes gráf és egy $c : E \to \mathbb{R}$ költségfüggvény. A feladatunk a legolcsóbb Hamilton-kör megkeresése. Egy Hamilton-kört a csúcsok bejárási sorrendjével írhatunk le: $T = (u_0, u_1, \ldots, u_{n-1})$, ahol $\forall\, i, j = 0, \ldots, n-1, i \neq j : u_i \neq u_j$. Ez a túrának egy irányítást is ad. A továbbiakban jelölje az u csúcsot követő, ill. megelőző csúcsokat u^+, ill. u^-. A túra összköltsége $\tilde{c}(T) = \sum_{u \in V} c(uu^+)$. Mivel a költség szimmetrikus, ezért az ellenkező irányú túrának is ugyanannyi az összköltsége.

Az utazóügynök probléma köztudottan NP-teljes, vagyis az optimális túra meghatározására nem ismert polinomiális algoritmus. Ezért a feladatnak csak közelítő megoldásával foglalkozunk. A gyakorlatban általában legjobban működő algoritmusok Lin és Kernighan heurisztikáján alapszanak.

3.2. k-optimális túrák, δ-utak

Ha már ismerünk egy Hamilton-kört, akkor ezt lokálisan javítva jobb túrához jutunk. Hagyjuk el a T túra k db élét, és ezek helyett vegyünk fel k db másik élet úgy, hogy a kapott T' is Hamilton-kör legyen. Ezt a műveletet *k-cserének* hívjuk.

3.2.1. Definíció. Egy T túra *k-optimális*, ha k-cserével nem javítható, azaz T-ből tetszőleges k-csere során kapott T' túrára teljesül, hogy $\tilde{c}(T) \leq \tilde{c}(T')$.

A *k-opt* algoritmus során egy tetszőleges T túrából k-optimális túrát készítünk úgy, hogy addig hajtunk végre k-cseréket, amíg tudunk vele javítani. Ha k-t növeljük, akkor

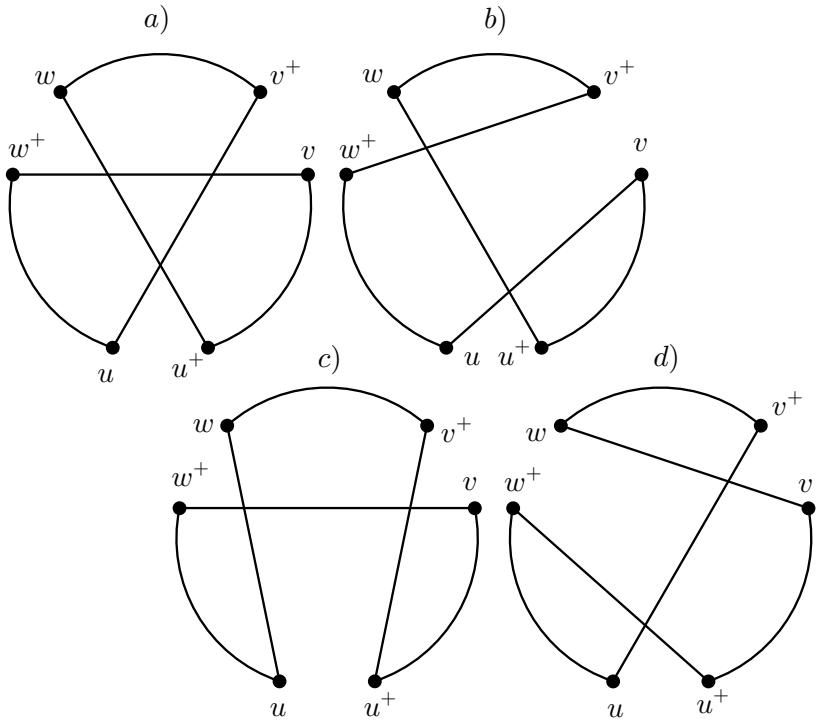
egy k-csere során megvizsgálandó esetek száma, így a számítási idő is nagyon gyorsan nő. Viszont már a 4-opt algoritmus sem javít jelentősen a 3-opt által szolgáltatott túrán, ezért a továbbiakban csak a 2- és 3-optimális túrákkal foglalkozunk részletesebben.

A T túra *2-optimális*, ha tetszőleges u^-u, vv^+ élpárra teljesül, hogy

$$c(u^-u) + c(vv^+) \leq c(u^-v) + c(uv^+)$$

A T túra *3-optimális*, ha 2-optimális és tetszőleges uu^+, vv^+, ww^+ élhármasra teljesül, hogy

$$\begin{aligned} c(uu^+) + c(vv^+) + c(ww^+) \leq \min\{&c(uv^+) + c(vw^+) + c(wu^+),\\ &c(uv) + c(u^+w) + c(w^+v^+),\\ &c(uw) + c(u^+v^+) + c(vw^+),\\ &c(uv^+) + c(wv) + c(u^+w^+)\} \end{aligned} \tag{3.1}$$



3.1. ábra. 3-cserék

A 3.1. ábrán látható, hogy egy 3-csere során milyen lehetőségeink vannak az új élek behúzására, amikor valóban mindhárom élet töröljük, azaz nem 2-cserét hajtunk végre. Annak ellenőrzésekor, hogy egy túra 3-optimális-e, ezt a négy lehetőséget kell megvizsgálnunk minden élhármasra. (3.1)-ben ezen élek költségének vesszük a minimumát.

Nevezzük *fordít*(u_i, u_j)-nek azt az algoritmust, amely az irányított túrában az u_i-től u_j-ig tartó részutat megfordítja, azaz az

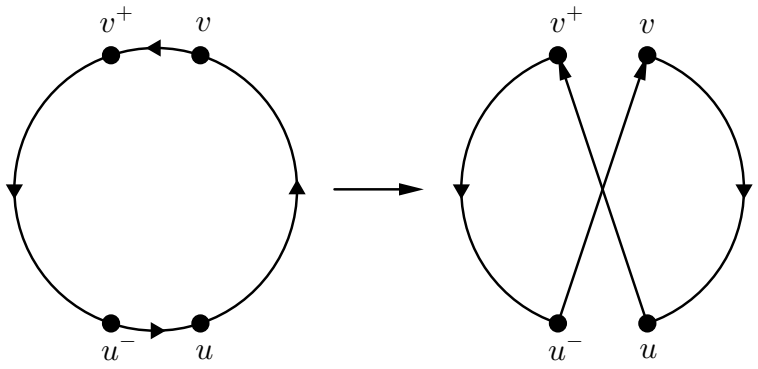
$$(u_0, \ldots, u_i^-, u_i, u_i^+, \ldots, u_j^-, u_j, u_j^+, \ldots, u_{n-1})$$

túrát az

$$(u_0, \ldots, u_i^-, u_j, u_j^-, \ldots, u_i^+, u_i, u_j^+, \ldots, u_{n-1})$$

túrává alakítja át.

3.2.2. Megjegyzés. A *fordít*(u, v) algoritmus megegyezik azzal a 2-cserével, amely az u^-u, vv^+ élek helyett az u^-v, uv^+ éleket veszi be a túrába.



3.2. ábra. *fordít*(u, v)

3.2.3. Állítás. *Bármely k-csere végrehajtható fordítások sorozataként.*

Bizonyítás. A *fordít*(u, u^+) a túrában felcseréli az u és u^+ csúcsokat, a többi csúcs helyén nem változtat. Szomszédos elemek cseréjével a csúcsok bármilyen permutációja előállítható. □

Jobb túrát kaphatunk, ha ahelyett, hogy szimplán csak megfordítanánk a részutat az eredeti helyén, másik két – a túrában eredetileg szomszédos – csúcs közé szúrjuk be valamilyen irányban.

3.2.4. Állítás. *A T túra pontosan akkor optimális az előbb leírt fordításokra és beszúrásokra nézve, ha 3-optimális.*

Bizonyítás. A 3.2.2. megjegyzés (és a 3.2. ábra) alapján a fordítások és a 2-cserék megegyeznek, tehát elég olyan 3-cserékkel foglalkoznunk, amelyek nem 2-cserék. Ennek négy lehetősége a 3.1. ábrán látható. Sőt, ezek közül is elég csak az a)-t és b)-t tekinteni, hiszen a $b), c), d)$ esetek szimmetrikusak, a csúcsok átnevezésével ugyanazt az ábrát kapjuk. Az a) eset megfelel annak a beszúrásnak, amikor a túra w^+-tól u-ig tartó szakaszát eredeti irányítás szerint szúrjuk be a v és v^+ csúcsok közé, a b) esetben pedig ugyanezt fordított irányítással tesszük meg. □

Vezessünk be még egy fogalmat, amelyre a Lin-Kernighan algoritmus során szükségünk lesz.

3.2.5. Definíció. Az $(u_0; u_1, \ldots, u_n)$ csúcssorozatot δ-*út*nak nevezzük, ha $(u_1, \ldots, u_n)$ egy Hamilton-út, és $\exists\ 1 \leq i \leq n : u_0 = u_i$.

3.3. A Lin-Kernighan algoritmus

Lin és Kernighan 1973-ban dolgozta ki az alábbi algoritmust, mely a mai napig az egyik legjobb heurisztika az utazóügynök feladat megoldására.

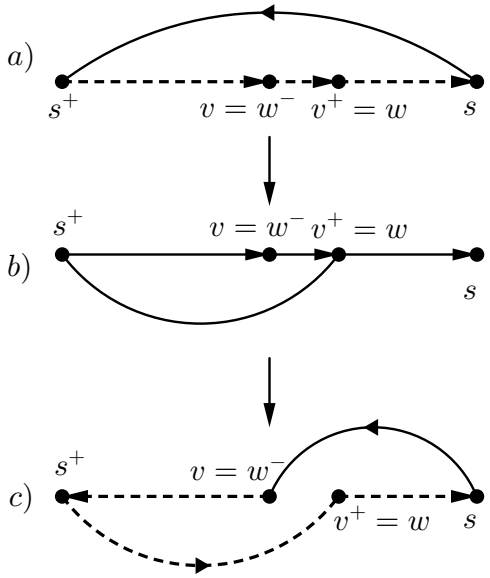
Az algoritmus lényege, hogy fordítások olyan sorozatát próbáljuk megkeresni, amelyeket végrehajtva jobb túrát kapunk. Fontos, hogy a köztes lépésekben kapott túrák nem feltétlenül olcsóbbak az eredeti túránál. Ha a keresés sikeres, akkor végrehajtjuk a fordításokat, és ebből a túrából folytatjuk az algoritmust. Ellenkező esetben új fordításokat keresünk, közben figyelünk arra, hogy a sikertelen próbálkozásokat ne ismételjük meg. Az algoritmus akkor ér véget, ha már egyik kezdőcsúcsból indulva sem tudunk jobb túrát találni.

Legyen T_0 egy túra, és s egy adott csúcs. Általában egy gráfban Hamilton-kört találni LP-nehéz feladat, de most a G teljes gráf, ezért a csúcsok tetszőleges sorrendje utazóügynök túrát ad. Az aktuális T túrát úgy kapjuk meg, hogy T_0-on elvégezzük az F forgatássorozatot. Az s csúcs az algoritmus során nem változik. s^+ és s^- mindig az aktuális T túrában jelentse az s megfelelő irányban vett szomszédjait, ezek tehát változhatnak. Δ-val jelöljük az eddig elért javítások összköltségét, azaz $\Delta = \tilde{c}(T_0) - \tilde{c}(T)$. Kezdetben $\Delta = 0$.

Az algoritmus során *fordít*(s^+, v) alakú fordításokat fogunk tekinteni, ahol v egy s-től, s^--tól és s^+-tól különböző tetszőleges csúcs. Ahogy már a 3.2.2. megjegyzésben is láttuk, ez az ss^+, vv^+ éleket az sv, s^+v^+ élekre cseréli. Egy ilyen fordítás pontosan akkor javít a túrán, ha

$$c(ss^+) + c(vv^+) > c(sv) + c(s^+v^+)$$

teljesül.



3.3. ábra. *fordít*(s^+, v). *a*) a fordítás előtti T túra, benne szaggatottal a fordítás előtti Hamilton-út, *b*) a P δ-út, *c*) a fordítás utáni túra, szaggatottal a fordítás utáni Hamilton-út

Viszont az algoritmus során az ilyen javító fordítások helyett csak azt követeljük meg, hogy a $P = (v^+; s^+, \ldots, v, v^+, \ldots, s)$ δ-út olcsóbb legyen a T_0 túránál, azaz

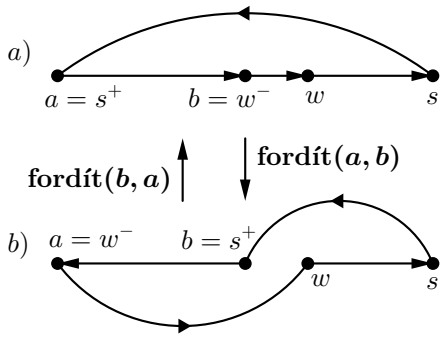
$$0 < \tilde{c}(T_0) - \tilde{c}(P) = \Delta + \tilde{c}(T) - \tilde{c}(P) = \Delta + c(ss^+) - c(s^+v^+)$$

Ha a T túra olcsóbb, mint az eddig megtalált legolcsóbb túra, akkor a fordítások sorozatát eltároljuk, de a keresést tovább folytatjuk, hátha még jobb túrához jutunk.

A továbbiakban legyen $w := v^+$, ekkor persze $v = w^-$.

3.3.1. Definíció. Ha $\Delta + c(ss^+) - c(s^+w) > 0$ akkor w-t *ígéretes csúcs*nak hívjuk.

Az algoritmus során ígéretes w csúcsokat fogunk keresni, majd végrehajtjuk a *fordít*(s^+w^-) fordítást. Természetes feltételnek tűnik, hogy addig keressünk ígéretes csúcsokat, amíg találunk ilyet. Ez azonban biztosan végtelen ciklushoz vezet, ha van legalább egy ígéretes csúcs a T_0 kezdőtúrában, mert az ígéretes csúcs a fordítás után is az marad.



3.4. ábra. Az ígéretesség megmaradása

Tekintsük a 3.4. ábrát. Az a) ábrán $s^+ = a$ és $w^- = b$. Tegyük fel, hogy a w csúcs ígéretes, azaz $\Delta + c(sa) - c(aw) > 0$. A *fordít*$(a, b)$ fordítás után a b) ábrát kapjuk. Mint ahogy a szakasz elején is mondtuk, Δ, s^+, s^- az algoritmus során változik, a b) ábrán $s^+ = b$, és $\Delta_{\text{új}} = \Delta + c(sa) + c(bw) - c(sb) - c(aw)$. A w csúcs a b) ábrán is biztosan ígéretes lesz: $\Delta_{\text{új}} + c(sb) - c(bw) = \Delta + c(sa) - c(aw) > 0$, ui. az előző lépésben w ígéretes volt. A számolás mögött az áll, hogy a két esetben a δ-utak ugyanazokból az élekből állnak, csak a δ-útban lévő kört ellenkező irányban járjuk be. A költségfüggvény szimmetriája miatt a két δ-út költsége megegyezik.

Ha most végrehajtanánk a w csúcshoz tartozó *fordít*(b, a) fordítást a b) ábrán, akkor visszakapnánk az a) ábrán látható túrát.

A fenti problémára megoldást jelent, ha további feltételeket követelünk meg: csak olyan ígéretes w csúcsot választunk, melyre $w^-w \in T_0$, ahol T_0 az eredeti túra, amelyből keresünk. Így csak olyan élet törölhetünk, amely eredetileg is benne volt T_0-ban. Ez nem vonatkozik az ss^+ alakú élekre, hiszen mindig, amikor egy sw^- él bekerül a túrába, az a következő lépésben ss^+ alakú élként kikerül belőle. További kritérium, hogy w-re $s^+w \notin T_0$ teljesüljön. Így csak olyan éleket vehetünk be a túrába, amely eredetileg nem volt benne T_0-ban.

Vegyük észre, hogy w pontosan akkor ígéretes, ha az s^+w él költsége elég kicsi, hiszen a Δ és a $c(ss^+)$ nem függ w-től. Sajnos, ha mohó módon mindig csak azt a w csúcsot nézzük, amelyre az s^+w él költsége minimális, akkor sokszor hosszú fordításssorozatokhoz jutunk, melyek végül nem is javítanak a túrán. Ezért ehelyett tekintsük azt a w csúcsot, amely a fordítás előtti $H = (s^+, \ldots, w^-, w, \ldots, s)$ Hamilton-utat a lehető legjobban javítja. A fordítás utáni Hamilton-út: $H_w = (w^-, \ldots, s^+, w, \ldots, s)$ (ld. 3.3. ábra)

$$w := \text{argmax}_{w \in V \setminus \{s, s^{++}\}}(\tilde{c}(H) - \tilde{c}(H_w)) = \text{argmax}_{w \in V \setminus \{s, s^{++}\}}(c(w^-w) - c(s^+w))$$

Ezt a mennyiséget túl költséges lenne minden w pontra kiszámolni, ezért csak egy előre meghatározott (s-et és s^{++}-t nem tartalmazó) $N(s^+)$ halmaz elemeit tekintjük. Ezeket s^+ *szomszédjai*nak hívjuk[1]. Pl. tipikusan $N(s^+)$ lehet az s^+-hoz legközelebbi k db csúcs valamely rögzített k értékre.

Az eddigieket összefoglalva a T_0 túrából és az s csúcsból induló keresési algoritmus pszeudokódja a következő:

$\Delta := 0$;
$F := \emptyset$; `// fordítások sorozata`
while $\exists\ w \in N(s^+)$ *ígéretes:* $w^-w \in T_0, s^+w \notin T_0$ **do**
  $w := \text{argmax}\{c(w^-w) - c(s^+w) : w \in N(s^+)$ ígéretes, $w^-w \in T_0, s^+w \notin T_0\}$;
  $\Delta := \Delta + c(ss^+) + c(w^-w) - c(sw^-) - c(s^+w)$;
  $Sorba(F, fordít(s^+, w^-))$;
end

Algorithm 1: $keres\_egyszeru(T_0, s)$

Az F fordításssorozatot sor adatszerkezetként implementáljuk. Ennek egyik művelete a $Sorba(F, x)$, amely az x elemet beteszi az F sorba (a végére). A későbbiekben egy másik műveletet is használni fogunk, ez a $Sorból(F)$, amely az első elemet kiveszi az F sorból.

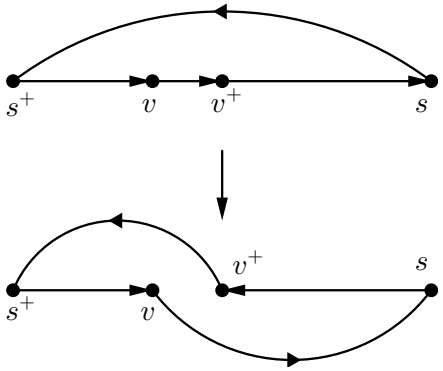
A keresés hatékonysága növelhető, ha egyszerre több fordításssorozatot tekintünk, párhuzamosan több túrát is számon tartunk. A keresési algoritmus első két lépésében nemcsak a $(c(w^-w) - c(s^+w))$-t maximalizáló w csúcsot tekintjük a fordításhoz, hanem ezen érték szerint csökkenő sorrendben az első öt csúcsot vizsgáljuk, és a fordításokat külön-külön adjuk hozzá az eddigi fordításssorozathoz. Így párhuzamosan legfeljebb 25 túrával dolgozunk.

Mak és Morton 1993-ban még egy javítási módszert dolgozott ki. Az eredeti $fordít(s^+, v)$ alakú fordításokkal szimmetrikusan, $fordít(v^+, s)$ alakú fordításokat is engedjünk meg. Ebben az esetben s^+ marad változatlan, az s csúcs megváltozik, ezért ezt minden lépésben át kell állítanunk.

A 3.3.1. definícióhoz hasonlóan most olyan $v \in N(s)$ csúcsokat tekintünk, amelyekre $\Delta + c(ss^+) - c(vs) > 0$. Ezeket nevezzük *M-ígéretes* csúcsoknak. A korábban felvetett problémára, miszerint ígéretes csúcs ígéretes marad, most a következő további feltételek adódnak: $vv^+ \in T_0, vs \notin T_0$. Ezen v-k közül azt választjuk, amelyre a $c(vv^+) - c(vs)$ érték maximális.

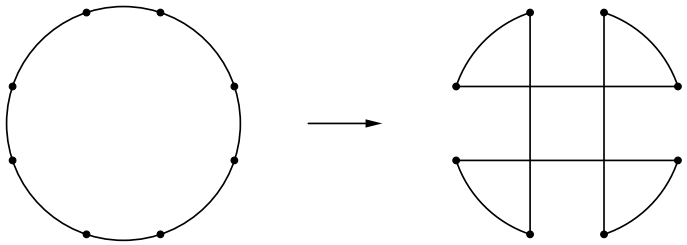
A Mak-Morton-féle fordításokat is figyelembe véve a fejezet végén leírjuk a Lin-Kernighan algoritmus pszeudokódját (ld. 3. algoritmus). Az első két lépésben alkalmazható párhuzamosságot itt nem nézzük.

[1] Általában egy csúcs szomszédjainak azokat a pontokat szoktuk nevezni, amelyekkel éllel össze van kötve, de esetünkben ez az elnevezés nem okoz félreértést, hiszen G teljes gráf.



3.5. ábra. Mak-Morton-féle *fordít*(v^+, s)

Az algoritmus során mindig megjelölt csúcsból indulva keresünk jobb túrát. Kezdetben minden csúcsot megjelölünk. Ha egy v csúcsból sikertelen volt a keresés, akkor jelöletlenné állítjuk, hogy ugyanezt a keresést ne ismételhessük meg. Sikeres keresés után végrehajtjuk a fordításokat, a túra megváltozik. Ekkor Lin és Kernighan eredetileg minden csúcsot újra megjelölt, hiszen az új túrából indulva, a korábban sikertelen kereséshez vezető csúcsokból is kereshetünk sikeresen. Nagyméretű gráfok esetén ez túlságosan nagy futásidőt eredményez. Ezt úgy oldhatjuk meg, ha sikeres keresés után csak olyan csúcsot jelöljük meg újra, amely a fordításssorozat valamelyik fordításának egyik végpontja volt.



3.6. ábra. A láncolt Lin-Kernighan algoritmusban alkalmazott 4-csere

A Lin-Kernighan algoritmust használva a legjobb túrához úgy juthatunk, ha több különböző T_0 kezdőtúrára is lefuttatjuk az algoritmust. Speciálisan a *láncolt Lin-Kernighan algoritmus* során a következő kezdőtúrát a Lin-Kernighan algoritmus által szolgáltatott túra megfelelő módosításával kapjuk. A gyakorlatban jól működik, ha a megtalált túrán végrehajtunk egy speciális 4-cserét (ld. 3.6. ábra). Ez annyira megváltozatja a túra szerkezetét, hogy az új kezdőtúrából indulva jobb túrához juthatunk el. Ha valóban jobb túrát kapunk, akkor ezzel folytatjuk a láncolt Lin-Kernighan algoritmust. Azonban, ha az eddig megtalált legjobb túrán nem tudtunk javítani, akkor a 4-cserét visszafelé hajtjuk végre, hogy visszakapjuk az előző túrát, és egy másik 4-cserével próbálkozunk.

$\Delta := 0$;
$F := \emptyset$;
$\Delta^* := \Delta$; // az eddig megtalált legnagyobb javítás értéke
$F^* := F$; // az eddig megtalált legnagyobb javítás fordítássorozata
while $\exists\ w \in N(s^+)$ *ígéretes:* $w^-w \in T_0, s^+w \notin T_0$ *vagy* $\exists v \in N(s)$ *M-ígéretes:* $vv^+ \in T_0, vs \notin T_0$ **do**
  $mw := \max\{c(w^-w) - c(s^+w) : w \in N(s^+)$ ígéretes$, w^-w \in T_0, s^+w \notin T_0\}$;
  $mv := \max\{c(vv^+) - c(vs) : v \in N(s)$ M-ígéretes$, vv^+ \in T_0, vs \notin T_0\}$;
  if $mw \geq mv$ **then**
    $w := \text{argmax}\{c(w^-w) - c(s^+w) : w \in N(s^+)$ ígéretes$, w^-w \in T_0, s^+w \notin T_0\}$;
    $\Delta := \Delta + c(ss^+) + c(w^-w) - c(sw^-) - c(s^+w)$;
    $Sorba(F, fordít(s^+, w^-))$;
  else
    $v := \text{argmax}\{c(vv^+) - c(vs) : v \in N(s)$ M-ígéretes$, vv^+ \in T_0, vs \notin T_0\}$;
    $\Delta := \Delta + c(ss^+) + c(vv^+) - c(ws^+) - c(vs)$;
    $Sorba(F, fordít(v^+, s))$;
    $s := v^+$;
  end
  if $\Delta > \Delta^*$ **then**
    $\Delta^* := \Delta$;
    $F^* := F$;
  end
end
return F^* ; // sikeretelen keresés: ha $F^* = \emptyset$

Algorithm 2: keres(T_0, s)

$T := T_0$;
for $v \in V$ **do**
  $megjelölt[v] :=\uparrow$;
end
while $\exists v \in V : megjelölt[v] =\uparrow$ **do**
  $F := keres(T, v)$;
  if $F \neq \emptyset$ **then**
    while $F \neq \emptyset$ **do**
      $fordít(x, y) := Sorból(F)$;
      Hajtsuk végre T-n a $fordít(x, y)$-t!;
      $megjelölt[x] :=\uparrow$;
      $megjelölt[y] :=\uparrow$;
    end
  else
    $megjelölt[v] :=\downarrow$;
  end
end
return T;

Algorithm 3: linkernighan(T_0)

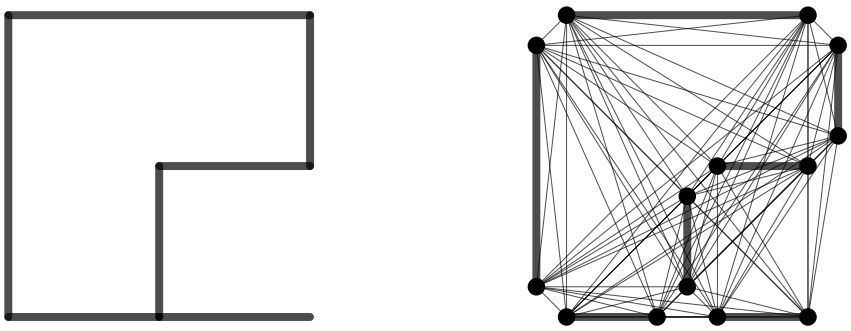
4. fejezet

Egyfejes nyomtatók

4.1. A matematikai modell

Most visszatérünk az eredeti célunkhoz: épületnyomtatók nyomtatófejéhez szeretnénk az optimális útvonalat meghatározni az előbb ismertetett utazóügynök feladat segítségével. Először az egyfejes nyomtatók esetét vizsgáljuk meg [2] alapján.

A felszeletelt épületmodell egy rétegével dolgozunk. Ezt gráffá alakítjuk: a csúcsok a falszegmensek végpontjait, a sarkokat és a falak metszéspontjait reprezentálják, és bármely két csúcs között lévő élt behúzunk, azaz a gráf teljes. A falszegmenseknek megfelelő éleket *fal-éleknek*, a többit *nem fal-éleknek* fogjuk hívni. Az utazóügynök túra minden csúcsot pontosan egyszer érint, a fejnek viszont oda, ahol több fal keresztezi egymást, többször is vissza kell térnie. Emiatt ezeken a helyeken annyi csúcsot veszünk fel, ahány falszegmens találkozik ott. Így minden csúcsból pontosan egy fal-él indul.



4.1. ábra. Az épület alaprajza és gráfos modellje (vastag vonal jelöli a fal-éleket, vékony vonal a nem fal-éleket)

Az építés összideje a nyomtatási időből (fal-éleken töltött időből) és az utazási időből (nem fal-éleken töltött időből) tevődik össze. Mivel a falakon pontosan egyszer kell végigmenni adott sebességgel, ezért a nyomtatás összideje nem függ a fej konkrét útvonalától, ez nem számít az optimalizálás során. A már megépített falakkal való ütközés elkerülése miatt annyiszor kell felemelni, illetve leereszteni a fejet, ahány fal-él van az útvonalban, hiszen csak ezek végpontjaiban van függőleges mozgás. Mivel a fal-élek számát tudjuk az optimalizálás előtt, ezért az ilyen függőleges mozgásokkal sem kell törődnünk az útvonal meghatározásakor.

Ezek alapján egy uv él $c'(uv)$ költségét az u és v pontok távolságával, és a hozzájuk tartozó élek végpontjába húzott érintők szögével tudjuk meghatározni. $c'(uv) \geq 0$ minden

uv élre.

A fenti gráfra a c' költségfüggvénnyel önmagában nem alkalmazható az utazóügynök probléma megoldása, hiszen az egy legolcsóbb Hamilton-kört ad meg, amely bármelyik éleket használhatja. Nekünk viszont bizonyos éleket, amelyek a felépítendő falaknak felelnek meg, mindenképp be kell vennünk a túrába.

A probléma a következőképpen is átfogalmazható: Adott néhány él (a fal-élek), és ezek végpontjainak koordinátái. Meg kell adnunk, hogy ezeket az éleket milyen sorrendben és irányban járjuk be, hogy az utazási idő (a nem fal-éleken töltött összidő) minimális legyen. Az építési időt kiiktathatnánk úgy, hogy a fal-éleket egy-egy ponttá húzzuk össze. Ekkor viszont nem tudnánk megfelelően definiálni két fal között húzódó él költségét. Tekintsük az $u = u_0u_1, v = v_0v_1$ fal-éleket. Az u-ból a v-be négyféleképpen utazhatunk, amelyekhez tartozó költség is 4 különböző érték lehet: $c'(u_0v_0), c'(u_0v_1), c'(u_1v_0), c'(u_1v_1)$. Ebbe a modellbe nem tudnánk bevenni, hogy a fal-éleket pontosan egyszer kell érintenünk az egyik, illetve másik végpontjánál.

Helyette a következőt csinálhatjuk: A fal-élek költségét $-M$-nek definiáljuk, ahol M egy nagy konstans.

$$c(uv) = \begin{cases} -M & \text{ha } uv \text{ fal-él} \\ c'(uv) & \text{ha } uv \text{ nem fal-él} \end{cases}$$

4.1.1. Állítás. *Jelölje n a gráf csúcsainak számát. Ha $M > \frac{n}{2} \max_{e \in E} c'(e)$, akkor a fent definiált c költségfüggvény mellett az optimális utazóügynök túra biztosan tartalmazza a fal-éleket.*

Bizonyítás. Legyen $K = \max_{e \in E} c'(e) \geq \max_{e \in E} c(e)$. Minden csúcsból pontosan egy fal-él indul ki, ezért a fal-élek száma $\frac{n}{2}$.

Ha a T túrában mind az $\frac{n}{2}$ fal-él benne van, akkor $\tilde{c}(T) \leq \frac{n}{2}(-M) + \frac{n}{2}K$.

Ha a T túrában legfeljebb $\frac{n}{2} - 1$ fal-él szerepel, akkor $\tilde{c}(T) \geq (\frac{n}{2} - 1)(-M) + (\frac{n}{2} + 1)0$.

Ezek alapján $\max\{\tilde{c}(T) : T\text{-ben } \frac{n}{2} \text{ db fal-él van}\} \leq \frac{n}{2}(K - M) < (\frac{n}{2} - 1)(-M) \leq$ $\leq \min\{\tilde{c}(T) : T\text{-ben legfeljebb } \frac{n}{2} - 1 \text{ db fal-él van}\}$, ahol a középső egyenlőtlenség $\frac{n}{2}K <$ $< M$ miatt teljesül. □

Az egyfejes nyomtatók esetén a c költségfüggvény mellett alkalmazzuk a láncolt Lin-Kernighan algoritmust.

4.2. A Lin-Kernighan algoritmus módosítása

Az épületnyomtatási feladathoz konstruált gráf rendelkezik pár speciális tulajdonsággal. Ezeket figyelembe véve módosítjuk a Lin-Kernighan algoritmust, hogy az ne tegyen fölösleges lépéseket, illetve mindenképp olyan túrát adjon meg, amely tartalmazza a fal-éleket. Láttuk, hogy bizonyos feltételek mellett az optimális túra biztosan tartalmazza a fal-éleket, azonban a Lin-Kernighan algoritmus általában csak egy közelítő megoldást talál meg. Ezért kell külön garantálnunk, hogy a túra tartalmazza az összes fal-élet. Már a kezdő T_0 túrát is ennek megfelelően választjuk meg, és ugyanezt a köztes lépésekben kapott túrákra is megköveteljük.

A továbbiakban legyen $K = \max_{e \in E} c'(e)$. A gráfnak n csúcsa van, így az utazóügynök túra n élből áll. A 4.1.1. állítás alapján, ha $M > \frac{n}{2}K$, akkor mind az $\frac{n}{2}$ fal-él benne van az optimális túrában. A másik $\frac{n}{2}$ él nem fal-él. És mivel a fal-éleknek nincs közös csúcsuk, így a fal-élek és a nem fal-élek felváltva követik egymást.

A Lin-Kernighan algoritmusban *fordít*(s^+w^-) alakú fordításokat hajtunk végre, ahol w ígéretes csúcs. Emlékeztetőül, ez azt jelentette, hogy $\Delta + c(ss^+) - c(s^+w) > 0$, ahol $\Delta = \tilde{c}(T_0) - \tilde{c}(T)$. Ha a T túra tartalmazza az összes fal-élet, akkor $\frac{n}{2}(-M) \leq \tilde{c}(T) \leq$ $\leq \frac{n}{2}(-M) + \frac{n}{2}K$. Emiatt $|\Delta| \leq \frac{n}{2}(-M) + \frac{n}{2}K - \frac{n}{2}(-M) = \frac{n}{2}K$.

Ha ss^+ fal-él lenne, akkor ez a *fordít*(s^+w^-) során kikerülne a túrából. A fordításnál megköveteltük, hogy $w \neq s$ és $w \neq s^{++}$, így $s^+w \notin F$ miatt s^+w csak nem fal-él lehet. Tetszőleges ennek megfelelő w-re $\Delta + c(ss^+) - c(s^+w) \leq \frac{n}{2}K - M - 0 < 0$. Így ilyenkor nem is létezne ígéretes csúcs. Az, hogy s^-s fal-él, pontosan annak felel meg, hogy az algoritmus során az ss^+ alakú élek mind nem fal-élek. Tehát ha a 3. algoritmusban v választásánál (itt ez felel meg az s-nek) nem csak arra figyelünk, hogy megjelölt legyen, hanem arra is, hogy a v^-v él fal-él legyen, akkor ezzel a csúcsok felét kizárjuk, amelyeket így nem kell fölöslegesen megvizsgálnia az algoritmusnak.

A *fordít*(s^+w^-) során a w^-w él a másik, amely kikerül a túrából. Erre a w ígéretességéből kiindulva nem állíthatunk semmit. Nézzük tehát a másik feltételt, vagyis hogy azt a w-t választjuk az ígéretes csúcsok közül, amelyik maximalizálja a $c(w^-w) - c(s^+w)$ kifejezést. Továbbra is igaz, hogy $s^+w \notin F$, így biztosan nem fal-él.

Ha w^-w fal-él, akkor $-M - K \leq c(w^-w) - c(s^+w) \leq -M - 0$.

Ha pedig w^-w nem fal-él, akkor $0 - K \leq c(w^-w) - c(s^+w) \leq K - 0$.

Mivel $-M < -K$, ezért az algoritmus lehetőleg olyan w ígéretes csúcsot fog választani, amelyre w^-w nem fal-él. Egyedül akkor lehet probléma, ha csak olyan ígéretes csúcs van, amelyre w^-w fal-él, hiszen ekkor a $\max(c(w^-w) - c(s^+w))$ is csak ilyen w-t tud adni. Ezt úgy oldhatjuk meg, ha a keresési algoritmus során további feltételként felvesszük w választásához, hogy a w^-w nem fal-él legyen.

A Mak-Morton-féle *fordít*(v^+s) fordítás során az ss^+-re ugyanazt kapjuk, hogy fölösleges azokat az eseteket nézni, amikor ez fal-él, mert ilyenkor nem létezne ígéretes csúcs. A vv^+ élre pedig az előzőhöz hasonlóan fel kell tennünk, hogy nem fal-él.

Ezekkel a módosításokkal az algoritmus olyan útvonalat fog megadni, amely biztosan tartalmazza a felépítendő falszegmenseket.

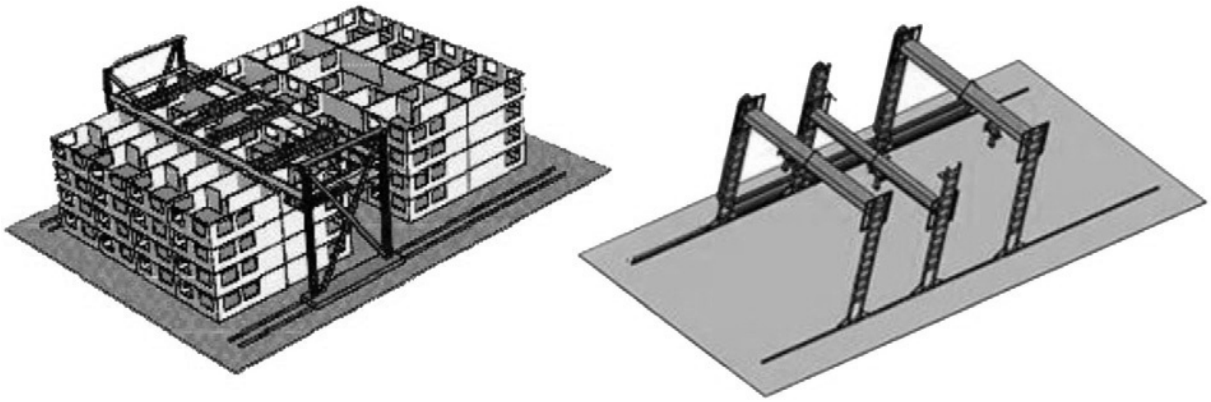
5. fejezet

Többfejes nyomtatók

A következő fejezet elejét [2] alapján dolgoztam ki, az 5.4. szakaszban saját eredmények vannak.

5.1. Bevezetés

Áttérünk a többfejes nyomtatók vizsgálatára. Ezeket több kisebb, vagy egy nagyobb, bonyolultabb szerkezetű épület nyomtatásakor használjuk, amikor egyetlen fej már nem tudna elég gyorsan befejezni egy réteget. Több gép használatával időt takaríthatunk meg, és más költségeket is csökkenthetünk.



5.1. ábra. A többfejes nyomtatók két típusa [2]

A többfejes nyomtatóknak két fajtája van (5.1. ábra). Az egyikben egy felső állványzat tartja a fejeket (overhead multi-nozzle). Függőleges irányban a tartó állvány csak egyszerre tudja mozgatni az összes fejet, ezért ebben az esetben minden fej kénytelen ugyanazon a rétegen dolgozni. Vízszintes irányban a fejek külön-külön tudnak mozogni, azonban keresztezni nem tudják egymást. Továbbá, ha az épület szélesebb, mint a tartó állványzat, akkor egy sín mentén azt is tudjuk mozgatni. A másik nyomtató a több állványzatos (multi-gantry). Ebben minden fejet külön állvány tart, amelyek ugyanazon a sínen mozognak. Ez nagyobb szabadságot ad a rendszernek, hiszen a fejek külön-külön mozoghatnak a térben, viszont a tartóállványok ebben az esetben sem tudják keresztezni egymást. Ennél a nyomtatónál nem kell, hogy a fejek ugyanolyan magasságban helyezkedjenek el. Azonban mi itt is megköveteljük, hogy minden fej ugyanazon a rétegen dolgozzon egyszerre, és csak akkor kezdjenek hozzá a következőhöz, ha az előzőn már minden másik fej befejezte a munkát. A továbbiakban mi az utóbbi, több állványzatos nyomtatóval foglalkozunk.

A nyomtatás megtervezéséhez először is a nyomtatandó falakat fel kell osztanunk a fejek között. Meg kell határoznunk, hogy melyiket melyik fej fogja kinyomtatni. Ezt úgy kell megtennünk, hogy mindegyik fej nagyjából egyszerre végezzen az adott rétegen. Az első lépésben az épület alaprajzát a nyomtatófejeket tartó állvány sínjére merőleges egyenesekkel egyenlő távolságokban osztjuk fel. A több részre vágott falaknál új csúcsokat veszünk fel. Ez a felosztás persze a nyomtatást tekintve nem feltétlenül lesz egyenletes, hiszen egyes fejek területére több falszegmens kerülhet. A felosztás után mindegyik fejre meg kell határoznunk egy-egy útvonalat, amelyet be kell járnia. Ezt hasonlóan tesszük meg, mint az egyfejes nyomtató esetén: definiáljuk a megfelelő gráfot, és a Lin-Kernighan algoritmus segítségével meghatározzuk a legolcsóbb Hamilton-kört. Ha az épület nem lóg túl az első (overhead multi-nozzle) típusú nyomtató állványzatán, azaz az állványzatnak nem kell mozognia a sínen, akkor az optimális útvonal mindkét fajta többfejes nyomtatónál ugyanaz lesz. Kiszámoljuk, hogy melyik fejnek mennyi időre van szüksége, és ez alapján (illetve a korábbi felosztások eredménye alapján) módosítjuk az egyes fejekhez tartozó területek határát. Ezt addig ismételjük, amíg a legnagyobb és legkisebb nyomtatási idő különbsége egy előre megadott kritikus értéknél kisebb nem lesz.

A fejeket úgy kell koordinálnunk, hogy működés közben ne ütközhessenek se egymással, se a korábban megépített falakkal. Ez utóbbit ugyanúgy oldjuk meg, mint az egyfejes nyomtató esetén: rétegről rétegre haladunk, és a fejeket felemeljük, amikor egyik falszegmenstől a másikig utaznak. Így csak a fejek közti ütközés elkerülését kell megoldanunk. Erre a továbbiakban három eltérő módszert, az útvonalforgatást, az ütközőzónák használatát, és a várakoztatást mutatom be. Ezek kombinálhatók is egymással.

5.2. Útvonalforgatás

A nyomtatandó épület alapterületét felosztottuk a fejek között. Mindegyik fej csak a saját területén belül mozog. Emiatt csak úgy fordulhat elő ütközés, hogy két szomszédos fej a területük közös határánál dolgozik egy időben, hiszen a fejek és az őket tartó állványok egy része ilyenkor átlóg a másik fej területére.

Jelölje $x_i(t)$ az i-edik fejet tartó állványnak a sín baloldali végpontjától mért távolságát a t időpillanatban. Legyen n a fejek száma, d az állvány szélessége (illetve az a legkisebb távolság, amilyen közel mehetnek egymáshoz a fejek), és T a réteg kinyomtatásának összideje. Nincs ütközés, ha a következő teljesül:

$$x_{i+1}(t) - x_i(t) \geq d \; \forall t \in [0, T], i = 1, \ldots, n-1$$

Ha az épület csak egyenes falakból áll, és a fejek egyenletes sebességgel haladnak, akkor az x_i függvények grafikonjai egyenes szakaszokból állnak. Ebben az esetben a fenti egyenlőtlenséget elég csak azon t időpillanatokban ellenőrizni, amikor valamelyik fej egy falszegmens végpontjában van.

Az útvonalforgatás lényege, hogy az x_i függvények alapján ütközésmentes útvonalakat hozzunk létre. Az útvonal kezdő- és végpontja egybeesik, ezért megváltoztathatjuk, hogy a fej melyik pontból induljon. Mivel összességében ugyanazt az útvonalat fogja bejárni, a nyomtatás ideje ugyanaz marad. Továbbá a költségek szimmetriája miatt a bejárás irányát is megváltoztathatjuk.

Az útvonalforgatás során az első fej útvonalát rögzítjük, és ehhez képest ellenőrizzük, hogy a második útvonalnak van-e olyan elforgatása, amely során nem ütközik az elsővel. Ha igen, akkor ezt az elforgatást rögzítjük, és továbblépünk a harmadik, negyedik,

stb. útvonalakra. Ha valamelyik útvonalnál azt tapasztaljuk, hogy az bármely pontjából indulva ütközik az előzővel, akkor ezzel a módszerrel nem tudjuk megoldani az ütközés problémáját.

5.3. Ütközőzónák

Az ütközés biztosan elkerülhető, ha ütközőzónákat alakítunk ki a területek határának mindkét oldalán. Ezek mentén a falakat kettévágjuk, és új csúcsokat veszünk fel. Az ütközőzónának szélesebbnek kell lennie az állványzatnál. Ez garantálja, hogy amikor a fej a fő munkaterületén, azaz az ütközőzónán kívül dolgozik, akkor biztosan nem ütközhet a másik fejjel. További kritérium, hogy az ütközőzónában lévő falak felépítésének ideje kevesebb legyen az egész építési idő felénél. Kétfejes nyomtatóknál így megvalósítható, hogy amíg a 'B' fej az ütközőzónában van, addig az 'A' a fő munkaterületen dolgozik. A 'B' fej az összes idő felénél korábban befejezi az ütközőzóna nyomtatását, ekkor 'A' még mindig a fő munkaterületen van. És végül, mikor az 'A' rátér a saját ütközőzónájára, a 'B' már a fő munkaterületen dolgozik. Ezt a sorrendet úgy tudjuk garantálni, hogy mind a négy területre külön útvonalat határozunk meg a Lin-Kernighan algoritmus segítségével. Az egy fejhez tartozó két útvonalat össze kell kötnünk. Ennek a költségét úgy tudjuk minimalizálni, hogy megkeressük a fő munkaterület és az ütközőzóna között húzódó szakaszok közül a minimális költségűt. Ennek a megfelelő zónába eső végpontjából indul a fej. Így amikor az adott zónában lévő túra végén visszaér ide, ezen a szakaszon tud átmenni a másik zónába.

Kettőnél több fejet használó nyomtatóknál – a két szélső fejet leszámítva – a fejek a tőlük balra, illetve jobbra lévő fejjel is ütközhetnek. A kétfejes nyomtatóhoz hasonlóan megoldható, hogy minden határ két oldalán felveszünk egy-egy ütközőzónát. Azaz a fejek nyomtatási területét három részre osztjuk, egy bal- illetve jobboldali ütközőzónára, és a középső fő munkaterületre. Ezen ütközőzónáknak ugyanazokat a feltételeket kell teljesíteniük, mint a kétfejes esetben. A nyomtatás során minden fej balról jobbra halad: először a baloldali ütközőzóna területén nyomtatja ki a falakat, utána a fő munkaterületen, és végül a jobboldali ütközőzónában. Kettőnél több fejes nyomtatóknál az egy fejhez tartozó három útvonal összekötését nem lehet olyan egyszerűen optimálisan megoldani, mint két fej esetén. Minél több ütköző zóna van, annál több részre oszlik egy fej útvonala, ami így már nem lesz optimális. Továbbá ekkor több lesz a kettéosztott fal, így a gráf csúcsainak száma is nő. Ezekből látszik, hogy további ütköző zónák bevezetése növeli a feladat bonyolultságát, és rosszabb megoldást ad.

Többfejes nyomtatóknál sem kell feltétlenül minden fejnek két ütközőzóna. Először minden fejhez egy ütközőzónát veszünk fel a területe bal szélén. Ennek a korábbiakhoz hasonlóan szélesebbnek kell lennie a fejet tartó állvány szélességénél. Meghatározzuk ezekben a fej útvonalát, kiszámoljuk az építési időt. Ha ez több, mint a következő (tőle jobbra lévő) fej ütközőzónában töltött ideje, akkor biztosan nincs ütközés. Ellenkező esetben egy kiegészítő ütköző zónát veszünk fel az eredeti ütközőzóna mellé úgy, hogy ezzel együtt a két ütközőzónában töltött együttes idő már több legyen a következő fej ütközőzónában töltött idejénél. Most is balról jobbra, azaz ütközőzóna, kiegészítő ütköző zóna (ha van), és fő munkaterület sorrendben haladunk. Ezt használva néhány fejnek kettő, a többinek egy ütközőzónája lesz, tehát ez jobb a korábban ismertetett, bal- és jobboldali ütközőzónás megoldásnál.

A kiegészítő ütközőzónákra sincs szükség, ha a módszert kombináljuk az útvonalforgatással. Az előzőhöz hasonlóan minden fejnek egy ütközőzónája lesz a területe bal szélén.

Az így keletkezett $2n$ zóna mindegyikében (az ütközőzónákban és a fő munkaterületeken) meghatározzuk az útvonalat. Ez után az i-edik fej fő munkaterületén lévő útvonalat forgatjuk el úgy, hogy a fej ne ütközhessen az $i+1$-edik fejjel, amikor az az ütközőzónájában lévő útvonalat járja be. Itt az egész nyomtatás ideje helyett elég egy rövidebb időintervallumot tekinteni, ugyanis ütközés csak akkor fordulhat elő, amikor az i-edik fej már befejezte az ütközőzónáját, de az $i+1$-edik még nem kezdett hozzá a fő munkaterületéhez. Emiatt az ütközőzónák bevezetésével nagyobb esélyünk van ütközésmentes útvonalak megtalálására, mint az egyszerű útvonalforgatás esetén.

5.4. Várakoztatás

A várakoztatási módszer lényege, hogy a fejek útvonalába várakozási szakaszokat iktatunk be úgy, hogy ezzel elkerüljük a fejek közti ütközést. A korábbiakban látott módon most is további részekre osztjuk a fejek területét, ütközőzónákat alakítunk ki a terület azon részein, ahol másik fej területével közös határ húzódik. Azonban ebben az esetben a fő munkaterületre és az ütközőzónákra nem alakítunk ki külön útvonalakat, hanem az eredeti, egész területre meghatározott útvonalat fogjuk használni. Az ütközőzónákra vonatkozó egyetlen kritérium, hogy szélesebbek legyenek a fejet tartó állványzatnál. Így ha két szomszédos fejre igaz, hogy minden időpillanatban legfeljebb az egyik tartózkodik az ütközőzónájában, akkor a nyomtatás során sosem tudnak ütközni egymással.

5.4.1. Várakoztatás kétfejes nyomtatók esetén

A fejezet elején, a bevezetésben leírtak alapján a területet több lépésben, újra és újra felosztjuk a két fej között, és mindkét területre külön-külön meghatározzuk a fejek útvonalát, és a bejárásukhoz szükséges időt. A területek határát addig módosítjuk, amíg a két útvonal idejének különbsége egy adott korlát alá nem kerül. A várakoztatás során ezt a két útvonalat fogjuk használni. A következőket mindkét fejre megtesszük. Jelölje T az adott fej útvonalához szükséges összes építési időt. Kiválasztunk egy adott kezdőpontot, ahonnan a fej indulni fog, és kiválasztjuk a körbejárás irányát is. Ezután behúzzuk az ütközőzóna és a fő munkaterület határát, és meghatározzuk az összes pontot, ahol az útvonal keresztülhalad ezen a határon. Majd kiszámoljuk, hogy ezeket a pontokat a fej mikor érinti. Ezeket az időpontokat a 0 és T időpontokkal kiegészítve (ha eredetileg nem szerepeltek köztük) kapjuk a $[0,T]$ intervallum $t_0 < \cdots < t_m$ felosztását. Egy $[t_i, t_{i+1}]$ szakaszon belül a fej vagy végig az ütköző zónában van, vagy végig a fő munkaterületen. Ezek alapján különböztetjük meg a szakaszok *ütköző* vagy *nem ütköző* típusait. Ezek felváltva követik egymást. A szakaszok és azok típusai nincsenek kapcsolatban az eredeti gráf fal-éleivel, illetve nem fal-éleivel, csak az útvonaltól és az ütközőzóna határától függenek.

Az ütközés elkerülése érdekében a fejeken két szakasz közé várakozási időt szúrhatunk be, ezáltal a későbbi szakaszokat eltoljuk. A szakaszokat nem osztjuk további kisebb szakaszokra. A várakozási időkre nincsenek alsó, illetve felső korlátaink, a fejet bármikor, tetszőleges időre megállíthatjuk. A célunk, hogy a várakozásokat úgy határozzuk meg, hogy a két fej ütköző szakaszai diszjunktak legyenek, továbbá, hogy a várakozásokkal együtt az építés összideje minimális legyen.

A [6]-ban található hasonló feladat segítségével írjuk fel formálisan is a feladatot. Jelölje m_i az i-edik fejhez tartozó szakaszok számát, $t_{i,0}, \ldots, t_{i,m_i}$ a felosztási időpontokat, és $\chi_{i,j} \in \{0,1\}$ az i-edik fej j-edik szakaszának típusát. $\chi_{i,j} = 0$, ha nem ütköző, és $\chi_{i,j} = 1$,

ha ütköző szakaszról van szó. Mivel ezek felváltva helyezkednek el, ezért $\chi_{i,j} + \chi_{i,j+1} = 1$ $(i = 1,2, 1 \leq j < m_i)$. Egy adott s megoldás esetén, a várakozásokat is figyelembe véve legyen $S_{i,j}(s)$ vagy röviden $S_{i,j}$ az i-edik fej j-edik szakaszának kezdő időpontja, illetve $C_{i,j}(s)$ vagy $C_{i,j}$ a befejezési időpontja $(i = 1,2, 1 \leq j \leq m_i)$.

A feltételek a következők:

$$C_{i,j} - S_{i,j} = t_{i,j} - t_{i,j-1} \ (i = 1,2, 1 \leq j \leq m_i)$$

Azaz minden szakaszt ugyanannyi idő alatt járunk be, mint az eredeti útvonalban.

$$S_{i,j+1} - C_{i,j} \geq 0 \ (i = 1,2, 1 \leq j < m_i)$$

Azaz a $j+1$-edik szakaszt csak a j-edik befejezése után kezdhetjük el. Ha itt szigorúan pozitív számot kapunk, akkor ezen a helyen ennyi várakozást iktatunk be.

$$S_{i,1} \geq 0 \ (i = 1,2)$$

Azaz a 0 időpontnál korábban nem kezdhetjük el a munkát.

$$\chi_{1,j} = \chi_{2,j'} = 1 \Rightarrow [S_{1,j} - C_{2,j'} \geq 0 \text{ vagy } S_{2,j'} - C_{1,j} \geq 0] \ (1 \leq j \leq m_1, 1 \leq j' \leq m_2)$$

Azaz ütköző szakaszok nem metszhetik egymást. Addig nem kezdhetjük el egyik ütköző szakaszt sem, amíg be nem fejezzük a másikon az aktuális ütköző szakaszt.

Az optimalizálási feladat:

$$\min_s \max(C_{1,m_1}(s), C_{2,m_2}(s))$$

Bármely megoldás esetén az ütköző szakaszok – akármelyik fejhez tartoznak is – nem lóghatnak egymásba, ezért (akár a kezdő, akár a befejező időpontokat tekintve) egy véges sorozatot alkotnak. Ha ezt rögzítjük, akkor a feladatnak létezik egy egyértelmű legjobb megoldása. Ezt nevezzük *lokálisan balra tolt megoldásnak*, amelyet a következő módon kaphatunk meg. Az ütköző szakaszok sorozatát egészítsük ki a nem ütköző szakaszokkal úgy, hogy ha csak az egyik fejhez tartozó részsorozatot nézzük, akkor azok sorrendje egyezzen meg az eredeti sorrenddel. Ezt többféleképpen, de tetszőlegesen megtehetjük, mindegyik ugyanazt a lokálisan balra tolt megoldást fogja adni. Az így kapott sorozatot követve minden szakasz kezdetét úgy határozzuk meg mohó módon, hogy az lokálisan a lehető legkorábbi időpont legyen.

A lokálisan balra tolt megoldások esetén minden szakaszhoz tartozik egy olyan előfeltétel-szakasz, ami miatt azt nem lehetett lokálisan korábban kezdeni. Ez nem ütköző szakasznál csak a saját fejhez tartozó előző ütköző szakasz (vagy a 0 időpont) lehet. Ütköző szakasz esetén pedig vagy a másik fej egyik ütköző szakasza, vagy a saját fejhez tartozó előző nem ütköző szakasz (vagy a 0 időpont) lehet. Ha a szakaszok kezdő időpontját úgy határozzuk meg, mint az előbb, akkor az meg fog egyezni az előfeltétel-szakaszuk befejező időpontjával. Ha a legkésőbb véget érő szakasztól visszafelé haladva meghatározzuk az előfeltétel-szakaszokat, akkor a szakaszoknak egy olyan részsorozatát kapjuk, amelyek a 0 időponttól a végéig szorosan egymás után következnek. Így az ütköző szakaszok sorrendjét fixnek tekintve valóban nem létezik a lokálisan balra tolt megoldásnál jobb megoldás. Ezért az optimális megoldást elég csak ezek közül kiválasztanunk. Azonban az ütköző szakaszoknak exponenciálisan sokféle sorrendje van, ezért nem tudunk minden ilyen megoldást meghatározni, majd ezek közül kiválasztani a legjobbat. Helyette vegyük észre a következőt.

Ha minden ütköző szakasz esetén az előfeltétel-szakasz a saját fejhez tartozó előző nem ütköző szakasz, akkor ez azt jelenti, hogy minden szakaszt közvetlenül az előző után kezdünk, vagyis egyáltalán nincs szükség várakozások beszúrására. Ha viszont van olyan ütköző szakasz (pl. feltehető, hogy az első fejhez tartozó $(1, i)$ szakasz), amelyet a másik fej egyik ütköző szakasza (a $(2, j)$ szakasz) miatt nem tudtunk korábban kezdeni, akkor $C_{2,j} = S_{1,i}$. Ennél az időpontnál a feladatot két kisebb, hasonló feladatra tudjuk szétvágni. A szakaszok sorozatát módosíthatjuk úgy, hogy abban ez a két szakasz közvetlenül egymás után jöjjön: $\ldots, (1, i-1), (2, j), (1, i), (2, j+1), \ldots$. Ez a sorrend is ugyanazt a lokálisan balra tolt megoldást adja. Ha először csak az elejétől a $(2, j)$ szakaszig, és külön az $(1, i)$ szakasztól az utolsóig határozzuk meg a lokálisan balra tolt megoldásokat, majd ezeket egymás után illesztjük, akkor ugyanazt a megoldást kapjuk, mintha az egész sorozatot egyben tekintettük volna. Hiszen a szakaszokat ugyanolyan sorrendben vesszük figyelembe, és a második részben lévő szakaszok egyike sem kezdődhet $C_{2,j}$-nél korábban. A feladat ilyen ütköző szakaszok mentén történő szétvágását használjuk az alábbi dinamikus programozási megoldás során.

Jelölje $f(a, b, c, d)$ $(0 \leq a \leq b \leq m_1, 0 \leq c \leq d \leq m_2 : (a, c) \neq (b, d))$ a legjobb megoldás összidejét abban az esetben, ha az első fej esetén csak a $[t_{1,a}, t_{1,b}]$, a második fej esetén pedig csak a $[t_{2,c}, t_{2,d}]$ intervallumban lévő szakaszokat használjuk, és ezeket úgy tekintjük, hogy a 0 időponttól kezdődnek. Az eredeti feladat optimális megoldása az $f(0, m_1, 0, m_2)$. Általában f-et a következőképpen számolhatjuk ki:

$$f(a,b,c,d) = \begin{cases} t_{2,d} - t_{2,c} & \text{ha } a = b \\ t_{1,b} - t_{1,a} & \text{ha } c = d \\ \max\{t_{1,b} - t_{1,a}, t_{2,d} - t_{2,c}\} & \text{ha } a \neq b, c \neq d, \text{ és nincs ütközés} \\ \min\{f(a, i_1, c, i_2) + f(i_1, b, i_2, d) : a \leq i_1 \leq b, c \leq i_2 \leq d, (i_1, i_2) \neq (a, c), \\ \qquad (i_1, i_2) \neq (b, d)\} & \text{ha } a \neq b, c \neq d, \text{ és van ütközés} \end{cases}$$

Az ütközés azt jelenti, hogy ha a $[t_{1,a}, t_{1,b}]$ és $[t_{2,c}, t_{2,d}]$ intervallumokat közös kezdőpontba toljuk a bennük lévő ütköző és nem ütköző szakaszokkal együtt, akkor lesznek olyan ütköző szakaszok, amelyek egymásba lógnak. Ezt a következőképpen ellenőrizhetjük: egy-egy indexszel végigmegyünk a két intervallumon a szakaszokat elválasztó pontokon ugrálva. Mindig egy-egy ütköző szakasz elején állunk velük, és a korábbi indexszel lépünk előre. Ha így átugorjuk a másik indexet, akkor ott két ütköző szakasz ütközik. Ellenkező esetben még egyet lépünk vele, hogy a következő ütköző szakasz elejére álljunk. Ezt iteráljuk addig, amíg ütközést nem találunk, vagy az egyik indexszel a szakasz végére nem érünk, ebben az esetben ugyanis nincs ütközés.

A kérdés, hogy milyen sorrendben számoljuk ki az f értékeit, hogy amikor fel kell használnunk egy másik f érték meghatározásához, akkor már ki legyen számolva. Ha $a = b$ vagy $c = d$, akkor nincs szükség másik f értékre, ezeket az elején kiszámolhatjuk. Ha $a \neq b$ és $c \neq d$, akkor $f(a, i_1, c, i_2)$ és $f(i_1, b, i_2, d)$ esetén is mindkét fejnél legfeljebb annyi szakaszt tartalmazó intervallumokat tekintünk, mint $f(a, b, c, d)$ esetén, sőt az egyik fejnél biztosan szigorúan kevesebbet. Emiatt $n_1 = 1, \ldots, m_1; n_2 = 1, \ldots, m_2; k_1 = 0, \ldots, m_1 - n_1; k_2 = 0, \ldots, m_2 - n_2$ sorrendben számoljuk ki az $f(k_1, k_1 + n_1, k_2, k_2 + n_2)$ értékeket.

5.4.2. Várakoztatás kettőnél több fej esetén

Ha három vagy még több nyomtatófejünk van, akkor a kétfejes nyomtatókhoz hasonlóan minden fejhez meghatározunk egy útvonalat, rögzítjük a kezdőcsúcsot és a bejárás irányát,

továbbá behúzzuk az ütközőzónák határait. A két szélső fej esetén ugyanazt csináljuk, mint két fej esetén, a középső fejekhez viszont két ütközőzóna is tartozik. Ezeknél a $0 = t_0 < < t_1 < \cdots < t_m = T$ időpontok (esetleg a t_0 és t_m kivételével) olyan időpontokat jelölnek, amikor az útvonalat követve a fej valamelyik ütközőzóna határán áthalad. Bármely $[t_i, t_{i+1}]$ időintervallumban a fej végig az egyik ütközőzónában, a fő munkaterületen vagy a másik ütközőzónában tartózkodik. Most is igaz, hogy az ütköző és nem ütköző szakaszok felváltva követik egymást, az ütköző szakaszoknak azonban most két típusa is van: közben vagy az előző fejjel ütközhet, vagy a következővel. Továbbra is igaz, hogy a szakaszokat nem osztjuk tovább, csak az előre adott szakaszok közé szúrhatunk be várakozásokat, amelyek tetszőlegesen sokáig tarthatnak.

A kétfejes esethez hasonlóan a következő jelöléseket használjuk. N a fejek száma, m_i az i-edik fej szakaszainak száma, $t_{i,0}, \ldots, t_{i,m_i}$ a felosztási időpontok. Az i-edik fej j-edik szakaszának típusa

$$\chi_{i,j} = \begin{cases} -1 & \text{ha az előzővel ütköző szakasz} \\ 0 & \text{ha nem ütköző szakasz} \\ 1 & \text{ha a következővel ütköző szakasz} \end{cases}$$

Az első fejnél nincsenek az előzővel, illetve az utolsónál nincsenek a következővel ütköző szakaszok: $\chi_{1,j} \in \{0,1\}$ $(1 \le j \le m_1)$, és $\chi_{N,j} \in \{-1,0\}$ $(1 \le j \le m_N\}$. Továbbá minden fej minden második szakasza nem ütköző: $\chi_{i,j} + \chi_{i,j+1} = \pm 1$ $(1 \le j \le m_i)$.

Egy adott s megoldás esetén, a várakozásokat is figyelembe véve legyen $S_{i,j}(s)$ vagy röviden $S_{i,j}$ az i-edik fej j-edik szakaszának kezdő időpontja, illetve $C_{i,j}(s)$ vagy $C_{i,j}$ a befejezési időpontja $(1 \le i \le N, 1 \le j \le m_i)$.

A megoldásra vonatkozó feltételek közül az első három ugyanaz, mint két fej esetén:

$$\begin{aligned} &C_{i,j} - S_{i,j} = t_{i,j} - t_{i,j-1} \ (1 \le i \le N, 1 \le j \le m_i) \\ &S_{i,j+1} - C_{i,j} \ge 0 \ (1 \le i \le N, 1 \le j < m_i) \\ &S_{i,1} \ge 0 \ (1 \le i \le N) \end{aligned}$$

Azt pedig, hogy az ütköző szakaszok nem metszhetik egymást, a következőképpen írhatjuk fel:

$$[\chi_{i,j} = 1 \wedge \chi_{i+1,j'} = -1] \Rightarrow [S_{i,j} - C_{i+1,j'} \ge 0 \text{ vagy } S_{i+1,j'} - C_{i,j} \ge 0] \qquad (1 \le i < N, 1 \le j \le m_i, 1 \le j' \le m_{i+1})$$

Az optimalizálási feladat szintén a kétfejes feladat mintájára:

$$\min_s \max\{C_{i,m_i}(s) : 1 \le i \le N\}$$

Sajnos az optimális megoldást már három fej esetén sem tudjuk olyan egyszerűen megadni, mint kétfejes nyomtatóknál. Ott ugyanis kulcsfontosságú volt egy adott időpont mentén a feladat kettévágása. Az továbbra is igaz, hogy ha a háromból csak két, egymás mellett lévő fejet tekintünk, és a szakaszaikat nem lehet várakozás nélkül szorosan egymás után rakni, akkor van olyan időpont, amely mentén a szakaszokat két részre oszthatjuk. Az azonban általában nem igaz, hogy ebben az időpontban a harmadik fej is pont két szakasz határán lenne, így itt nem tudnánk elvágni ezt a szakaszt.

A következőkben megadunk egy egyszerű, és egy bonyolultabb várakoztatási algoritmust. Ezek egyike sem adja meg garantáltan az optimális megoldást.

Három fej esetén az egyszerű várakoztatási algoritmus a következő lépésekből áll:

1. Az első fej szakaszait rögzítjük, ennél a fejnél nem szúrunk be plusz várakozásokat, a fej egyszerűen csak az eredeti útvonal szerint fog végighaladni.

2. A második fej esetén csak olyan várakozásokat iktatunk be, amelyekre az első fej miatt van szükség. Az ennek megfelelő legjobb megoldást keressük. Ezt hasonlóan tesszük meg, mint ahogy két fej esetén is az ütközést vizsgáltuk. Mindkét fej szakaszain végigmegyünk a $t_{i,j}$ pontok mentén az r_1, r_2 indexekkel.

   a) Az indexekkel az első megfelelő ütköző szakasz elejére állunk, azaz $r_1 = \min\{j : 0 \leq j \leq m_i, \chi_{1,j} = 1\}, r_2 = \min\{j : 0 \leq j \leq m_2, \chi_{2,j} = -1\}$. A második fejnél egyelőre csak az olyan ütköző szakaszokkal foglalkozunk, amelyeknek az első fej ütköző szakaszaiba nem szabad belelógniuk.

   b) Ha $t_{1,r_1} \leq t_{2,r_2}$, akkor r_1-et növeljük eggyel, ellenkező esetben r_2-t.

   c) Ha a növelés után továbbra is fennáll az előző egyenlőtlenség, akkor nincs ütközés, a megfelelő indexszel továbblépünk, amíg a következő megfelelő ütköző szakasz elejére nem kerülünk.

   d) Ha r_1 növelése után már $t_{1,r_1} > t_{2,r_2}$, illetve r_2 növelése után $t_{1,r_1} < t_{2,r_2}$ akkor ütközés van. Mivel az első fej útvonalát már rögzítettük, ezt csak úgy tudjuk feloldani, ha a második fejhez várakozást iktatunk be, az ütköző szakaszát eltoljuk az első fej ütköző szakaszának végéhez. A második fejnél a most beszúrt várakozás utáni $t_{2,j}$ értékeket megnöveljük a várakozás hosszával[1].

   e) Ha az előbb r_1-gyel léptünk egyet, akkor továbblépünk vele a következő ütköző szakasz elejéig. Ha azonban r_2-vel léptünk, akkor ez az index most az eltolt ütköző szakasz végén áll. Viszont lehetséges, hogy nem elég ennyivel eltolni, mert még mindig ütközik egy másik ütköző szakasszal, ezért ebben az esetben r_2-vel visszalépünk egyet a most eltolt ütköző szakasz elejére.

   f) A *b)* ponttól folytatjuk az algoritmust. A ciklusból akkor lépünk ki, ha valamelyik indexszel elértük az utolsó szakasz végét, hiszen ekkor már nem lesz szükség több várakozásra. Ekkor a második fej szakaszait a várakozásokkal együtt rögzítjük.

3. A 2. ponthoz hasonlóan járunk el. A második fej rögzített szakaszai mellett keressük meg a második és harmadik fej között lévő ütközéseket, és ezeknek megfelelően vesszük fel a várakozásokat a harmadik fejre.

Az algoritmus egyértelműen általánosítható N fejes nyomtatókhoz is. Az előzőket folytatva, a k-adik lépésben a $k-1$-edik fej már rögzített szakaszait figyeljük, és így határozzuk meg a k-adik fej várakozásait.

Most rátérünk a bonyolultabb várakoztatási algoritmusra. Ennek során egyszerre mindig három fej útvonalával fogunk foglalkozni, ezért először definiáljunk ehhez egy segédfeladatot. Tegyük fel, hogy a $k-1$-edik fej szakaszainak időpontjai már rögzítettek. Itt nem csak a kezdeti, szorosan egymás után következő szakaszokból álló útvonal lehetséges, hanem szerepelhetnek benne rögzített várakoztatások. Emellett a feltétel mellett szeretnénk

[1] A gyakorlatban itt nem megyünk végig az összes $t_{2,j}$ értéken, hanem eltároljuk az eddig beszúrt várakozások összhosszát, és ezt csak akkor adjuk hozzá a megfelelő $t_{2,j}$-hez, amikor r_2-vel elérünk idáig.

a k-adik és $k+1$-edik fej útvonalaiba várakozásokat beszúrni úgy, hogy ezek befejezési ideje közül a nagyobb minimális legyen. A segédfeladat optimális megoldását a következőkben leírtak alapján meg tudjuk határozni. Ez azonban nem egyezik meg e három fejre kitűzött eredeti feladat optimális megoldásával, mert a $k-1$-edik fej útvonalát nem módosíthatjuk.

Először vizsgáljuk csak a $k-1$-edik és a k-adik fej közötti potenciálisan ütköző szakaszokat. Mivel a $k-1$-edik fej rögzített, ezért az előző egyszerű várakoztatási algoritmus $2.a)-f)$ lépései alapján meghatározhatjuk a k-adik fej várakozásait. Az egész rendszerre vonatkozó bármely megoldás esetén a k-adik útvonal legalább olyan hosszú, mint ha csak ezeket, a most kapott várakozásokat vennénk hozzá. Ha az így kapott k-adik útvonal és az eredeti $k+1$-edik útvonal ütköző szakaszai diszjunktak, akkor meg is kaptuk a segédfeladat optimális megoldását.

Ha viszont így a k-adik és $k+1$-edik fej között ütközés van, akkor a segédfeladat optimális megoldásában biztosan van olyan várakozás, amire e fejek ütközésmentesítése érdekében van szükség. (Hangsúlyozzuk, hogy a k-adik és $k+1$-edik fejek közti ütközést nem az eredeti útvonalakkal vizsgáljuk, hanem a k-adik útvonalat már előre módosítjuk a $k-1$-edik fej alapján.) Ebben az esetben a k-adik és $k+1$-edik fej szakaszait a kétfejes feladat megoldásához hasonlóan két részre tudjuk osztani, a feladatot kisebb feladatokra vágjuk. Ugyan elképzelhető, hogy a $k-1$-edik fej az elvágó időpontban valamelyik szakasz belsejében van, ez azonban most nem okoz gondot, mert a $k-1$-edik útvonal már rögzített, a megfelelő szakaszt kettévághatjuk. A következőképpen járunk el. Először az egész $k-$ -1-edik útvonalat és a másik két fej esetén a szakaszok első felét figyelembe véve oldjuk meg a segédfeladatot. Az így kapott megoldás esetén a k-adik és $k+1$-edik útvonalak közül a hosszabbik ideje lesz az elvágó időpont. E mentén a $k-1$-edik fej szakaszait is kettéosztjuk. Ha az elvágó időpont valamelyik szakasz belsejébe esik, akkor azt a szakaszt kettévágjuk. Ezután a $k-1$-edik útvonalon csak az elvágó időpont utáni szakaszokat, a k-adik és $k+1$-edik útvonalak esetén pedig a korábbi kettéosztás szerint a szakaszok második felét vesszük figyelme a segédfeladat megoldása során. Végül az így kapott megoldásokat egymás mögé illesztjük, így megkapjuk az eredeti segédfeladat megoldását.

Tehát most is a kétfejes esethez hasonló szétvágásos technikát alkalmazzuk a segédfeladat dinamikus programozási megoldása során. Jelölje $f_k(a,b,c,d)$ $(0 \le a \le b \le$ $\le m_k, 0 \le c \le d \le m_{k+1})$ a k-adik és $k+1$-edik fej befejezési időpontjai közül a nagyobbikat a következő segédfeladat optimális megoldása esetén. A $k-1$-edik fejnél az $[f_k(0,a,0,c), t_{k-1,m_{k-1}}]$, a k-adik fejnél a $[t_{k,a}, t_{k,b}]$, a $k+1$-edik fejnél pedig a $[t_{k+1,c}, t_{k+1,d}]$ intervallumokba eső szakaszokat tekintjük, a $k-1$-edik fej útvonalában lehetnek már előre beszúrt várakozások, de ez minden esetben rögzített, és a segédfeladat optimális megoldását úgy határozzuk meg ezekre az intervallumokra, hogy mindegyik fej a 0 időpontból induljon. Legyen továbbá $v(a,b,c)$ $(0 \le a \le b \le m_k, 0 \le c \le m_{k+1})$ a k-adik fej szükséges várakozásainak összhossza abban az esetben, amikor csak az $[f_k(0,a,0,c), t_{k-1,m_{k-1}}], [t_{k,a}, t_{k,b}]$ intervallumokat tekintjük, és csak a k-adik és a rögzített $k-1$-edik fej közti ütközéseket akarjuk elkerülni. Ezt az egyszerű várakoztatási algoritmus alapján tudjuk meghatározni.

$$f_k(a,b,c,d) = \begin{cases} t_{2,d} - t_{2,c} & \text{ha } a = b \\ \max\{t_{k,b} - t_{k,a} + v(a,b,c), t_{k+1,d} - t_{k+1,c}\} & \text{ha } a \neq b, \text{ és nincs ütközés} \\ \min\{f_k(a,i_1,c,i_2) + f_k(i_1,b,i_2,d) : a \le i_1 \le b, c \le i_2 \le d, (i_1,i_2) \neq (a,c), \\ \qquad (i_1,i_2) \neq (b,d)\} & \text{ha } a \neq b, \text{ és van ütközés} \end{cases}$$

Ütközés alatt azt értjük, hogy ha csak a megfelelő intervallumokat tekintjük, és a k-

adik útvonalba már beillesztettük a $k-1$-edik fej miatt szükséges várakoztatásokat, akkor a k-adik és $k+1$-edik fej között van-e ütközés. Ezt az egyszerű algoritmus alapján tudjuk ellenőrizni. Ott a ciklusból akkor is kilépünk, ha már találtunk egy ütközést. Viszont ha amiatt állunk meg, mert valamelyik index elérte az intervallum végét, akkor nincs ütközés.

Már csak az f_k és v értékek kiszámolásának sorrendjét kell ügyesen megválasztanunk. A $v(a,b,c)$ egyetlen előfeltétele minden b-re, hogy az $f_k(0,a,0,c)$-t már tudjuk. Emiatt mindig, amikor egy ilyen alakú f értéket meghatározunk, utána rögtön kiszámoljuk a $v(a,b,c)$-t is minden b-re. Itt valójában elég egyszer végigmenni az egyszerű várakoztatási algoritmus 2. lépésén $b = m_k$ választással, mert a többi b-re ugyanezen várakozások közül az első néhányat fogjuk megkapni (a b időpontban vagy utána kezdődő várakozásokat kell elhagyni).

Ha $a = b$, akkor $f_k(a,a,c,d)$-t közvetlenül ki tudjuk számolni. Egyébként az $f_k(a,b,c,d)$ előfeltétele a $v(a,b,c)$, azaz az $f_k(0,a,0,c)$, és azon $f_k(a',b',c',d')$ értékek, ahol $[a',b'] \subseteq$ $\subseteq [a,b], [c',d'] \subseteq [c,d]$, és legalább az egyik esetben valódi részintervallumról van szó. Emiatt az $f_k(a,b,c,d)$-nél korábban számoljuk ki az összes olyan $f_k(a',b',c',d')$-t, ahol $b' \leq b, d' \leq d, (b',d') \neq (b,d)$. Ezek között ott van $f_k(0,a,0,c)$ is, ui. $a < b, c \leq d$. Továbbá az $f_k(a',b,c',d)$ értékek közül azokat kell hamarabb kiszámolni, amelyekre $a' \geq a, c' \geq$ $\geq c, (a',c') \neq (a,c)$.

Ezek alapján a következő sorrendet alkalmazzuk: először az előfeltétel nélküli $f_k(a,a,c,d)$ értékeket számoljuk ki (tetszőleges sorrendben), majd a $b = 1,\ldots,m_k; d = 0,\ldots,m_{k+1}; c =$ $= d,\ldots,0; a = b-1,\ldots,0$ sorrendet követjük.

Mivel a $k-1$-edik útvonal rögzített, ennek hosszát előre tudjuk, így a segédfeladat megoldása alatt a másik két fej optimális várakoztatása esetén a hosszabbiknak a hosszát értjük, azaz az $f_k(0,m_k,0,m_{k+1})$ értéket.

A segédfeladat megoldása után most valóban rátérünk a bonyolultabb várakoztatási algoritmusra. Első lépésként az első két fejre végrehajtjuk a kétfejes algoritmust, majd az első fejet ezekkel a várakozásokkal rögzítjük. Ezt a rögzített útvonalat használva megoldjuk a segédfeladatot a 2. és 3. fejre, és rögzítjük a 2. fejet is, stb. Vegyük észre, hogy az első két fejre alkalmazott kétfejes algoritmus megegyezik annak a segédfeladatnak a megoldásával, ahol a 0. fejnél nincsenek további korlátozások az 1. fej útvonalára, azaz pl. a 0. fej útvonala egyetlen pontból áll. Ha felveszünk egy ilyen segéd 0. fejet, akkor az algoritmus során egymás után $k = 1,\ldots,N-1$-re megoldjuk a segédfeladatot, és így kapjuk az eredeti feladat egy megoldását. Ez nem az optimális megoldást adja, azonban az egyszerű várakoztatási algoritmusnál jobb megoldást szolgáltat.

5.4.1. Állítás. *A bonyolultabb várakoztatási algoritmus során kapott útvonalak hosszainak maximuma:* $f_{\max} = \max\{f_k(0,m_k,0,m_{k+1}) : 1 \leq k < N\}$.

Bizonyítás. Legyen először $1 \leq k \leq N-2$.

Ha a k-adik lépésben a k-adik útvonal hosszabb volt a k+1-ediknél, akkor $f_k(0,m_k,0,m_{k+1})$ megegyezik a k-adik fej hosszával, amelyet ekkor rögzítünk is, tehát ez az útvonal kerül be a megoldásunkba. Emiatt $f_{\max} \geq f_k(0,m_k,0,m_{k+1})$.

Ha a k-adik lépésben a $k+1$-edik útvonal volt a hosszabb, akkor itt az igaz, hogy a $k+1$-edik fejjel biztosan nem tudunk $f_k(0,m_k,0,m_{k+1})$-nél hamarabb végezni ütközés nélkül, így a következő, $k+1$-edik lépésben erre a fejre kapott, és ekkor rögzített útvonal hossza legalább $f_k(0,m_k,0,m_{k+1})$. Tehát ebben az esetben is $f_{\max} \geq f_k(0,m_k,0,m_{k+1})$.

Az utolsó $k = N-1$-edik lépésben megkapott $N-1$-edik és N-edik útvonal is bekerül a megoldásunkba, így ezek hosszának maximumára is igaz, hogy $f_{N-1}(0,m_{N-1},0,m_N) \leq$ $\leq f_{\max}$. □

A bonyolultabb várakoztatási algoritmust nem csak az első fejtől az N-edikig haladva hajthatjuk végre, hanem fordított irányban is. Ekkor a segédfeladat során a $k+1$-edik fej útvonala rögzített, és ehhez határozzuk meg a k-adik és $k-1$-edik útvonalak legjobb várakoztatását. A kezdő lépésben az N-edik és $N-1$-edik útvonalak várakoztatásához egy 1 pontból álló segéd $N+1$-edik fejet veszünk fel, majd visszafelé haladva oldjuk meg egymás után a segédfeladatokat. Mivel ez szimmetrikus a korábban részletezett, elölről haladó bonyolultabb várakoztatási algoritmusra, ezért a jelölések és az algoritmus lépései egyértelműen módosíthatók erre az esetre is. A visszafelé haladó algoritmus által szolgáltatott útvonalak hosszának maximumát most is a közbülső lépésekben kiszámolt f_k értékek maximumaként kapjuk.

Ha $N \geq 5$, akkor egy harmadik lehetőség, hogy egy K-adik belső fejtől indulunk ($3 \leq K \leq N-2$). Ekkor először a K-adik és $K+1$-edik fejre meghatározzuk a két fejre vonatkozó optimumot, f_K-t. Ezután $k = K+1, \ldots, N-1$ estén előrefelé haladva a bonyolultabb várakoztatási algoritmussal meghatározzuk ezen k-kra f_k-t. Végül K-tól visszafelé meghatározzuk a többi útvonalat is. A K-adik útvonal már megvan, itt tehát először a K-adik rögzített fej mellet alkalmazzuk a segédfeladatot a $K-1$-edik és $K-2$-edik fejre. Azaz $k = K-1, \ldots$,2 esetén számoljuk ki f_k-t.

5.4.2. Állítás. *Ha a bonyolultabb várakoztatási algoritmus során a K-adik belső fejtől indulunk, akkor $f_{\max} = \max\{f_k : 1 < k < N\}$.*

Bizonyítás. f_K-ra tekinthetünk úgy, hogy egy $(K-1)'$ ideiglenes, egy pontból álló segédfej mellett alkalmaztuk a segédfeladatot a $(K-1)', K, K+1$ fejekre. Emiatt ha csak a $K, \ldots, N$-edik fejeket tekintjük, akkor ezekre csak az előre haladó algoritmust alkalmaztuk, ezek útvonalainak maximuma $\max\{f_k : K \leq k < N\}$.

f_K-t viszont úgy is megkaphatjuk, ha egy $(K+2)'$ egypontos segédfejet veszünk fel, és így alkalmazzuk a fordított irányú segédfeladatot. Így, ha csak az $1, \ldots, K+1$ fejeket tekintjük, akkor az ezekre kapott f_k-k megegyeznek azokkal az értékekkel, amelyeket a visszafelé haladó algoritmus segítségével kapnánk. Emiatt ezen útvonalak maximuma $\max\{f_k : 1 < k \leq K\}$. A kettőt összerakva kapjuk az állítást. □

A továbbiakban az elölről és hátulról induló algoritmusok mellett csak a $K = [\frac{N}{2}]+1$, középről induló esettel foglalkozunk.

5.4.3. Eredmények

Az előző részben bemutatott algoritmusokról programokat is készítettem C++-ban. Az alábbiakban ezek futási eredményét fogom elemezni.

Az inputokat minden esetben véletlenül generáltam. Minden fejhez az m_i értéket, azaz a szakaszok számát egyenletesen választottam az $M = \{m_{\min}, m_{\min}+1 \ldots, m_{\max}\}$ halmazból. Majd minden fejre, egymástól függetlenül $\frac{1}{2}-\frac{1}{2}$ valószínűséggel kiválasztottam, hogy ütköző vagy nem ütköző szakasszal kezdődjön-e. Ezek a típusok ezután az adott fej esetén felváltva követik egymást. Az első és utolsó fej kivételével az ütköző szakaszok esetén is $\frac{1}{2}-\frac{1}{2}$ valószínűséggel választottam ki, hogy az előzővel vagy a következővel ütközzön. A szakaszok hosszait egyenletes eloszlás szerint határoztam meg: ütköző szakasz esetén a $[p_{\min}, p_{\max}]$ intervallumból, nem ütköző szakasz esetén pedig a $[q_{\min}, q_{\max}]$ intervallumból.

Három fej esetén az $M = \{8,9,10,11,12\}$ halmazt, és különböző n értékekre a $[p_{\min}, p_{\max}] = [15-n,25-n]$ és $[q_{\min}, q_{\max}] = [15+n,25+n]$ intervallumokat használtam. Minél kisebb az n, annál hosszabbak az ütköző szakaszok, negatív n esetén pedig várható értékben hosszabbak a nem ütköző szakaszoknál. Az intervallumokat azért választottam így, mert

egy fejhez átlagosan 10 szakasz, azaz 5 ütköző és 5 nem ütköző szakasz kerül. Egy ütköző szakasz hosszának várható értéke $20-n$, egy nem ütközőé pedig $20+n$, így a várható érték linearitása miatt egy útvonal összhosszának várható értéke $5(20-n)+5(20+n)=200$. Az így kapott megoldások hosszai összehasonlíthatók.

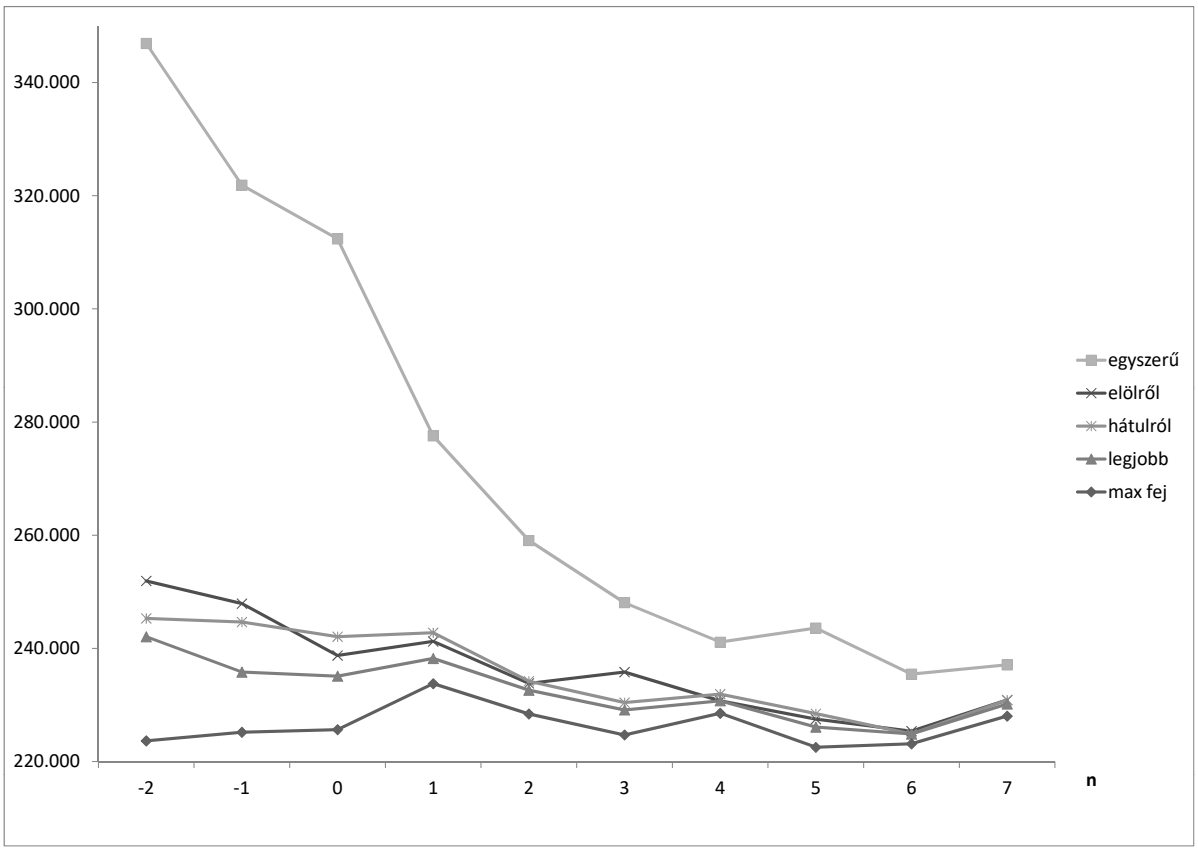
Az $n=-2,\ldots,7$ értékek mindegyikére 30-30 inputot generáltam, és meghatároztam az egyszerű, illetve az elölről és hátulról induló bonyolultabb algoritmus által szolgáltatott megoldásokat, majd kiválasztottam ezek közül a legjobbat. Az optimális megoldás hosszára egy egyszerű alsó korlát az eredeti útvonalak közül a leghosszabbnak a hossza, hiszen ha ezen a fejen nem is iktatunk be várakozásokat, akkor sem lehet ennél hamarabb végezni vele. A grafikonokon ezt jelöli a „max fej". Az 5.2. ábrán az egyes n értékekre vonatkozó 30 adat átlagát láthatjuk.

Az 5.1. táblázatban ugyanezen inputokra vonatkozó további eredmények találhatók. Az „elölről", illetve „hátulról" oszlopokban azt láthatjuk, hogy a 30-ból hány esetben adta a legjobb megoldást az elölről, ill. hátulról induló algoritmus. (Az összegük azért nem 30, mert sok esetben mindkettő ugyanazt a megoldást adta.) Az utolsó oszlop azon esetek számát mutatja, amikor a megtalált legjobb megoldás megegyezett az optimum alsó korlátjával, vagyis biztosan optimális.

Ugyanilyen paraméterekkel az 5 fejes esetet is teszteltem, itt viszont már volt értelme a középről induló algoritmusnak is, ezért azt is lefuttattam, és a legjobb megoldás kiválasztásánál ezt az eredményt is figyelembe vettem. Az erre vonatkozó eredményeket az 5.3. ábra és az 5.2. táblázat mutatja.

A grafikonokból és táblázatokból látszik, hogy a megtalált megoldások valóban mind legalább akkorák, mint az eredeti leghosszabb útvonal (max fej). Az elölről, középről, illetve hátulról induló bonyolultabb algoritmusok között nem látható szignifikáns eltérés, a konkrét inputtól függ, hogy ezek közül melyik adja a legjobb megoldást. Persze az elölről és hátulról induló algoritmusok átlagos eredményei között nem is várunk különbséget, hiszen ezek szimmetrikus módszerek, és a szakaszokat minden fejre függetlenül és azonos eloszlás szerint választottuk. A középről induló algoritmus akár lehetne jobb is, de az eredmények alapján úgy látszik, ez általában rosszabb megoldásokat ad, mint a másik kettő. Viszont ezt sem fölösleges megvizsgálni, mert voltak olyan inputok, amelyekre a középről induló algoritmus adta az egyedüli legjobb megoldást. Ha az ütköző szakaszok arányaiban hosszabbak, mint a nem ütköző szakaszok (kisebb n-ek), akkor az egyszerű algoritmus nagyon rossz megoldásokat ad a bonyolultabb algoritmusokhoz képest. Ha viszont az ütköző szakaszok rövidebbek (nagyobb n-ek), akkor még az egyszerű algoritmus sem ad olyan rossz megoldást. Viszont a bonyolultabb algoritmusok legjobb megoldása még itt is közelebb van az alsó korláthoz, mint az egyszerű megoldás eredményéhez. Sőt, sok esetben el is éri az alsó korlátot, azaz biztosan optimális.

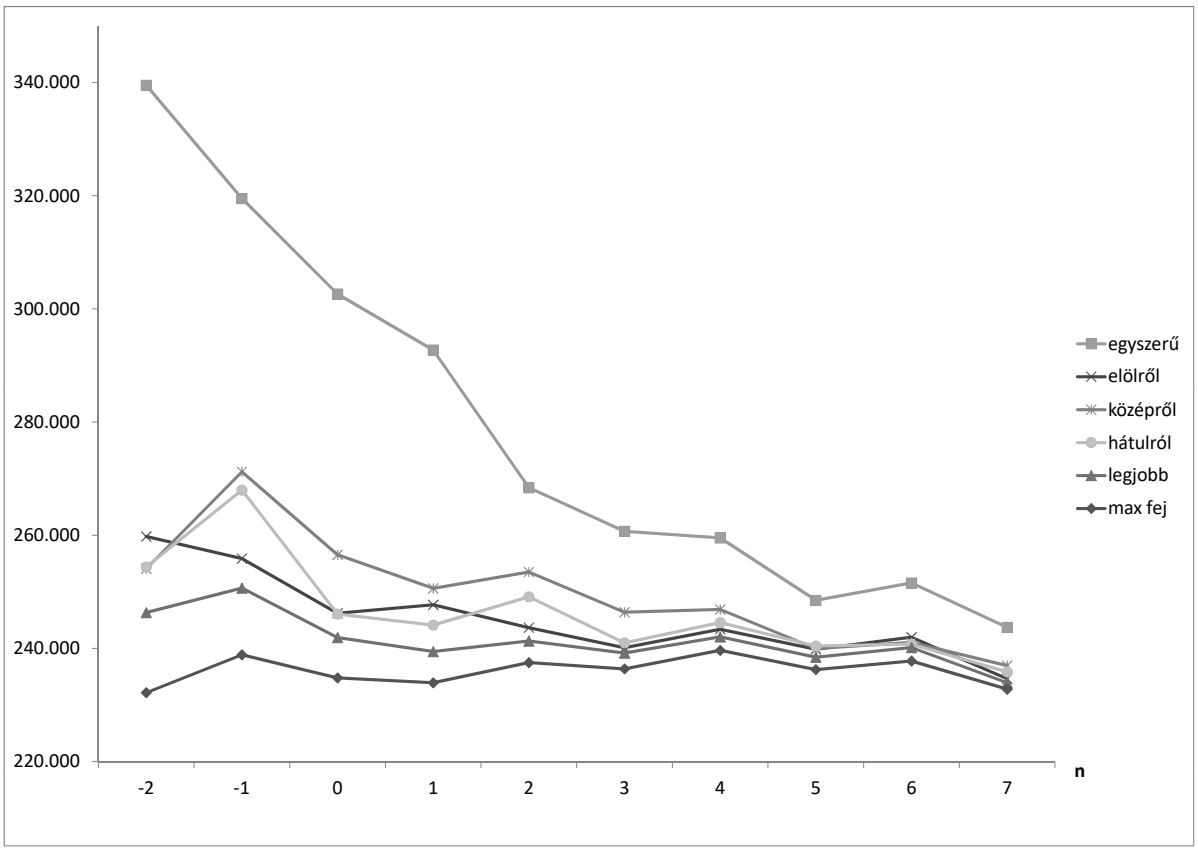
A kétfejes nyomtatókhoz csak $n=2$, azaz a $[13{,}23],[17{,}27]$ intervallumok esetén készítettem grafikont, viszont egy fejhez több szakaszt választottam: $M=\{45{,}46,\ldots,55\}$. A korábbi 3, ill. 5 fejre vonatkozó inputokkal ellentétben, amelyeknél minden n esetén néhány másodperc alatt megkaptam a megoldást, itt a sok szakasz miatt a program kb. 25 percig futott, mire megadta mind a 30 inputra a kezdeti maximális útvonal hosszát, az optimális megoldást és az egyszerű algoritmus által adott megoldást. Az 5.4. ábrán a 30 teszteset eredményét láthatjuk, ezek sorrendje a vízszintes tengely mentén nem mond semmit, hiszen az inputokat egymástól függetlenül, azonos paraméterekkel generáltam.



5.2. ábra. A különböző algoritmusok eredményeinek átlaga 3 fejes nyomtató esetén.

n	elölről	hátulról	biztosan optimális
-2	22	24	0
-1	25	20	7
0	24	25	4
1	21	23	12
2	26	25	20
3	22	26	13
4	30	26	22
5	26	27	15
6	29	30	21
7	27	26	18

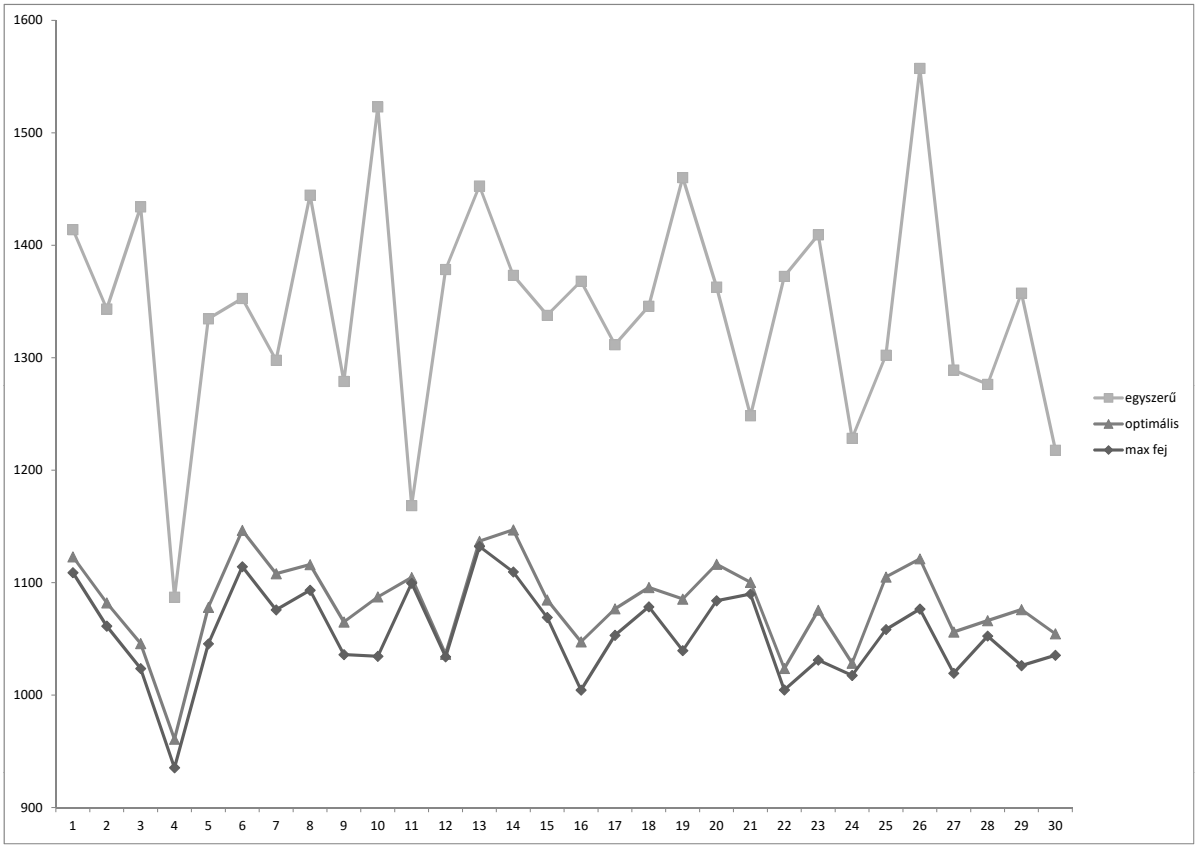
5.1. táblázat. A 3 fejes nyomtató esetén melyik módszer hány esetben adta a legjobb megoldást.



5.3. ábra. A különböző algoritmusok eredményeinek átlaga 5 fejes nyomtató esetén.

n	elölről	középről	hátulról	biztosan optimális
-2	17	18	18	1
-1	18	14	20	4
0	23	12	20	10
1	22	19	21	9
2	24	17	20	12
3	28	18	26	15
4	25	24	26	21
5	27	25	27	24
6	27	28	27	20
7	28	24	26	26

5.2. táblázat. Az 5 fejes nyomtató esetén melyik módszer hány esetben adta a legjobb megoldást.



5.4. ábra. A kétfejes nyomtatóra vonatkozó 30 eredmény

Irodalomjegyzék

[1] Yuan Jin, Jianke Du, Zhiyong Ma, Anbang Liu, Yong He, *An optimization approach for path planning of high-quality and uniform additive manufacturing*, The International Journal of Advanced Manufacturing Technology (2017) 92:651-662

[2] Jing Zhang, Behrokh Khoshnevis, *Optimal machine operation planning for construction by Contour Crafting*, Automation in Construction 29 (2013) 50-67

[3] Király Tamás, Kis Tamás, Szegő László, *Online jegyzet az Egészértékű Programozás I. és II. tárgyhoz* (2017)

[4] David Applegate, Robert Bixby, Vašek Chvátal, William Cook, *Finding tours in the TSP*, Research Report, Universität Bonn, Institut für Ökonometrie und Operations Research (1999)

[5] https://3dprinthuset.dk/europes-first-3d-printed-building/

[6] Chengbin Chu, Jean-Marie Proth. *Single machine scheduling with chain structured precedence constraints and minimal and maximal separation times.* [Research Report] RR-2268, INRIA. 1994, pp.16. <inria-00074403>