Eötvös Loránd Tudományegyetem
Természettudományi Kar

Kétdimenziós téglalappakolások

Szakdolgozat

Karaffa Csilla
Matematika BSc., Alkalmazott matematikus szakirány

Témavezető: Király Tamás, egyetemi adjunktus
Operációkutatási Tanszék

Budapest, 2011

Nyilatkozat

Név: Karaffa Csilla

ELTE Természettudományi Kar, matematika BSc szak

ETR azonosító: KACPAAT.ELTE

Szakdolgozat címe: Kétdimenziós téglalappakolások

A szakdolgozat szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló munkám eredménye, saját szellemi termékem, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Aláírás

_____

Dátum

_____

Köszönetnyilvánítás

Ezúton szeretném megköszönni témavezetőmnek, Király Tamásnak a rengeteg, konzultációra fordított időt, a hasznos tanácsokat és ötleteket.

Köszönet illeti továbbá barátomat, amiért mindig segítséget nyújtott, ha az illusztrálásra készült program írása közben elakadtam, valamint öcsémet a fogalmazási és helyesírási hibák korrigálásáért. Külön köszönet jár nagynénémnek a szakdolgozatíráshoz biztosított ideális körülményekért.

Ez a munka az ő támogatásuk nélkül nem jöhetett volna létre.

Tartalomjegyzék

1. fejezet

Bevezetés

Sok, a gyakorlati életben előforduló problémát át lehet fogalmazni különböző alakú modulok valamilyen szempont szerinti pakolási feladatára két vagy három dimenzióban. Mindkét dimenzióban lehet a feladatunk ládapakolás, ahol a célunk egy adott méretű konténerbe minél jobb helykihasználással bepakolni a modulokat, illetve az, hogy az összes modult a lehető legkevesebb ilyenbe pakoljuk bele. A sávpakolásnál a konténer mérete az egyik koordinátatengely mentén nincs korlátozva, ilyenkor a feladatunk az összes modult elhelyezni benne úgy, hogy ezen tengely mentén minimalizálunk. Ha a konténer mérete semelyik irányban sem korlátozott, akkor a feladat a terület/térfogat-minimalizálás. A modulok lehetnek téglalapok/téglatestek, sokszögek/poliéderek vagy teljesen általános alakúak is. Például valamilyen dobozolt szállítmány optimális bepakolása vagonokba a háromdimenziós ládapakolásnak, egy szabásminta részeinek kivágása az anyagból (ipari körülmények között) a kétdimenziós sávpakolásnak felel meg általános alakú modulokra.
Ezek közül a továbbiakban területminimalizálási feladatról lesz szó téglalapokra, melynek egyik legfontosabb alkalmazási területe a VLSI-tervezés. Itt azonban nem csak a területminimalizálás a cél, az optimális huzalozás kialakítása is. A feladatot tovább nehezíti az, hogy egyes alkatrészek csak az áramköri lap bizonyos részén helyezkedhetnek el, vagy szükség szerint egyesek elforgathatók 90°-kal, illetve előfordulhat, hogy az oldalaik hossza nem kötött, azaz a terület megtartásával bizonyos keretek között változtatható. Ennek a dolgozatnak a keretei nem teszik lehetővé egy ilyen összetett feladat tárgyalását. Azonban létezik egy, a gyakorlatban is használt internetes technológia, melyben megjelenik a területminimalizálási feladatnak egy speciális változata.

CSS Sprite

Egy weboldal - a legegyszerűbbeket kivéve - mindig több fájlból áll. Ezek között van az oldal szerkezetét leíró HTML, a stílusért felelős CSS, a különböző szkriptfájlok, valamint az oldalon megjelenítendő képek is. Ezek mind az adott weboldalt üzemeltető szolgáltató szerverén helyezkednek el. Ahhoz, hogy az oldal a böngészőben tökéletesen megjelenjen és funkcionáljon, az őt alkotó összes fájlnak le kell töltődnie a számítógép memóriájába. Ennek érdekében a böngésző HTTP-kéréseket küld a szerver felé, először a HTML-fájlért, majd az ebben található hivatkozások sorrendjében a többiért. Mivel a böngésző csak néhány szálon kommunikál a szerverrel, az egy időben letölthető fájlok száma korlátozva van. Másrészt nagyon kis méretű fájlok esetén előfordulhat, hogy a letöltésükre irányuló kérés mérete nagyobb, mint maga a fájl. Emiatt ha egy weboldal sok kisebb képet tartalmaz, célszerű ezekből egyetlen, nagyobb képet - úgynevezett CSS Sprite-ot - létrehozni, és az oldalon a képek helyén a Sprite képet jeleníteni meg úgy, hogy csak a megfelelő részlete látszódjon. Ez a CSS technológiával egyszerűen megvalósítható. Így a HTTP-kérések számát jelentősen csökkenteni tudjuk, ezzel pedig az oldal betöltődését gyorsítani.
A Sprite kép készítése során törekszünk arra, hogy mérete a lehető legkisebb legyen. Ez függ a kép formátumától, de a pixelekben számított méretétől is. Ez utóbbinak a lehető legkisebbé tétele a kétdimenziós területminimalizálási feladatnak az a speciális esete, amikor a modulok téglalap alakúak, fix méretűek és nem elforgathatóak. Dolgozatomban megkísérlem bemutatni ennek a feladatnak a különböző megoldási lehetőségeit, a legelterjedtebb reprezentációknak a feladatra specifikált leírásán keresztül az ezekre alkalmazott alapvető heurisztikus algoritmusok bemutatásáig. Szerepelni fog a különböző módszerek elméleti szempontok (pl. futásidő) szerinti összehasonlítása, valamint gyakorlati eredmények is, melyeket a mellékelt DVD-n található program segítségével állítottam elő.

2. fejezet

Nagyságrendek

Minden algoritmusnál, melyet számítógépen szeretnénk futtatni, nagy jelentősége van a nagyságrendeknek. Egyrészt lényeges, hogy a struktúrák, melyeket az algoritmus használ, mekkora memóriát foglalnak, valamint az is, hogy maga az algoritmus mennyi idő alatt fut le. Mint ismeretes, a nem polinomiális algoritmusok már kis számú bemenő adat esetében sem érnek véget olyan időben, hogy a gyakorlatban használhatóak legyenek. Például egy $(n!)^2 n^2$ nagyságrendű algoritmus (a későbbiekben találkozni fogunk ilyennel) már $n = 7$ esetén is minimum órákig futna. A polinomiális algoritmusoknál is a minél kisebb kitevőket részesítjük előnyben, és általánosságban elmondható, hogy az n^3-nél nagyobb nagyságrend nem megfelelő a bemenő adatok nagy száma esetén. Ha a bevezetőben ismertetett gyakorlati problémánál a technológiából adódóan $n = 100$-nál több bemenő adatra nem kell számítani, a későbbiekben ismertetett algoritmusok által használt memória nagyságrendjét és futási idejüket ettől függetlenül általánosságban határozzuk meg. A gyakran előfordulóakat ebben a fejezetben tárgyaljuk.

2.1. Nevezetes számok

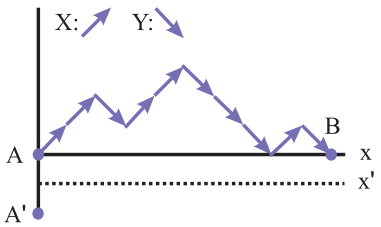
1. Tétel. *Legyen S $2n$ hosszú XY sorozat, melyben az X-ek és Y-ok száma megegyezik, és S semelyik prefixe sem tartalmaz több Y-t, mint X-et. A különböző ilyen S sorozatok száma megegyezik az n. Catalan-számmal:*

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Nagyságrendben:

$$C_n \sim O(2^{2n}/n^{1.5})$$

Bizonyítás. Ábrázoljuk a lehetséges XY sorozatokat koordináta-rendszerben az ábrán látható módon.



Legyen A és B távolsága $2n$. Az A-ból B-be vezető olyan utak száma, melyeknek nincs pontja az x-tengely alatt, megegyezik a megengedett XY sorozatok számával. Az $A-B$ utak számát megkaphatjuk úgy, hogy az összes $A-B$ út számából levonjuk azokat, melyeknek van közös pontja az x'-tengellyel. Az utóbbiból viszont a tükrözési elv alapján pontosan annyi van, mint amennyi az összes $A'-B$ út száma.
Így a keresett szám:

$$K = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1}\frac{(2n)!}{(n!)^2}$$

A Stirling-formulával aszimptotikusan becsülhetjük $n!$-t:

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Ezt felhasználva K becslésére:

$$\begin{aligned} K &\sim \frac{1}{n+1}\frac{\sqrt{2\pi 2n}\left(\frac{2n}{e}\right)^{2n}}{\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)^2} = \frac{1}{n+1}\frac{2\sqrt{\pi n}\left(\frac{2n}{e}\right)^{2n}}{2\pi n\left(\frac{n}{e}\right)^{2n}} = \\ &= \frac{2^{2n}}{\sqrt{\pi n}(n+1)} = O(2^{2n}/n^{1.5}) \end{aligned}$$

□

1. Állítás. *A különböző n csúcsú bináris fáknak a száma megegyezik C_n-nel.*

Bizonyítás. Indukcióval bizonyítunk. A különböző 1 csúcsú ($n=1$) bináris fák száma $B_1 = 1 = C_1$.
Egy $n \geq 1$ csúcsú bináris fának a gyökértől különböző $n-1$ csúcsából k darab van a bal részfában, $n-k-1$ a jobb részfában. Az összes különböző $n+1$ csúcsú bináris fa számát úgy kapjuk, hogy összegzünk az összes lehetséges k-ra.

$$B_n = \sum_{k=0}^{n-1} B_k B_{n-k}$$

Az indukciós feltevés szerint a részfákra igaz az állítás, így

$$B_n = \sum_{k=0}^{n-1} C_k C_{n-k} = C_n$$

□

Megjegyzés. Ha az előző állításban tárgyalt általános alakú bináris fákat kiegészítjük úgy, hogy minden olyan csúcshoz, melynek 2-nél kevesebb gyereke van, hozzáadunk annyi gyereket, amennyit maximálisan lehet, akkor pontosan azokat a bináris fákat kapjuk, melyeknek n belső csúcsuk és $n+1$ levelük van, és igaz rájuk, hogy az összes belső csúcsnak 2 gyereke van. A fenti állítás alapján az ilyen, különböző bináris fák száma is C_n.

1. Definíció. Nevezzük megengedettnek egy sorozat olyan zárójelezéseit, amelyre igazak a következők:

(i) a zárójelezés helyes zárójelezés,
(ii) egy zárójel minimálisan két tagot tartalmaz,
(iii) nincs dupla zárójel semelyik tagcsoport körül,
(iv) van egy zárójelpár, mely az egész sorozatot magába foglalja.

Példa. Tekintsük az $ABCD$ sorozatot. Ekkor nem megengedett zárójelezések:

(ii) szerint $(AB(C)D)$
(iii) szerint $(AB((CD)))$
(iv) szerint $(AB)(CD)$

Megengedett például $(A(BCD))$, $((AB)(CD))$.

A csupán egy elemből álló sorozattal baj van, hiszen ekkor (ii) szerint - mivel egy tagból áll - nem lehetne bezárójelezni, míg (iv) megköveteli a befoglaló zárójelet. Ebben az egy esetben megengedjük az (A) alakú zárójelezést.

2. Tétel. *Egy n elemű sorozat esetén a fent definiált megengedett zárójelezések száma A_n, ahol A_n az n. szuper Catalan-szám, vagy kis Schröder-szám,*

$$\begin{aligned} A_n &= \frac{(3(2n-3))A_{n-1} - (n-3)A_{n-2}}{n} \\ A_1 &= A_2 = 1 \end{aligned}$$

2. Definíció. A következő formulával meghatározott B_n számokat Baxter-számoknak nevezzük:

$$B_n = \binom{n+1}{1}^{-1}\binom{n+1}{2}^{-1}\sum_{k=1}^{n}\binom{n+1}{k-1}\binom{n+1}{k}\binom{n+1}{k+1}$$

2. Állítás. *[7]*

$$B_{n+1} = \sum_{i,j \geq 0} T_n(i,j), \quad n \geq 0$$

Ahol

$$\begin{aligned} T_{n+1}(i+1,j+1) &= \sum_{k=1}^{\infty} T_n(i+k,j) + T_n(i,j+k) \\ T_0(i,j) &= \begin{cases} 1 & ha\ i,j = 0 \\ 0 & egyébként \end{cases} \\ T_n(i,j) &= 0 \quad ha\ i < 0\ vagy\ j < 0\ vagy\ i+j > n \end{aligned}$$

2.2. Ábrázolás számítógépen

A későbbiekben bemutatásra kerülő, a pakolásokat leíró reprezentációk a következő három struktúrával dolgozhatnak:

(i) Sorozatok

(ii) Kijelölt gyökérponttal rendelkező bináris fák

(iii) Kijelölt gyökérponttal rendelkező általános fák, ahol egy csúcs gyerekeinek sorrendje lényeges.

Mindhárom adatstruktúra elemei lehetnek a $0 \ldots n$ egész számok, és esetleg még m-féle speciális karakter úgy, hogy az egész számok mindegyike legfeljebb egyszer szerepelhet a struktúrában, míg a speciális karakterekre nincs ilyen megkötés. Ezen kívül a sorozatok lehetnek csak a 0 és az 1 számjegyet tartalmazó úgynevezett bitsorozatok is.
Jó választás a speciális karaktereket az $n+1, n+2, \ldots, n+m$ egész számokként ábrázolni, hiszen ezek a számok nem szerepelnek a struktúra eredeti számhalmazában, így nem okozhatnak kavarodást, de ekkor a struktúra azonos típusú elemeket fog tartalmazni. Elméletben egy a egész szám tárolásához $\lceil \log a \rceil$ bitre van szükség, így az első adatstruktúra esetén - ha az előbb javasolt ábrázolást választjuk - az adatok tárolásához szükséges bitek számát felülről becsülhetjük $(n+m)\log\lceil n+m \rceil$-vel. Az $m = 0$ speciális esetben ez az $n\lceil \log n \rceil$ alakot ölti.
A gyakorlatban a programozási nyelveknek nem létezik l bites beépített egésztípusuk minden l-re. Általában az 1, 8, 16, 32 és 64 bites típusok közül választhatunk. Így ha például a legáltalánosabban használt 32 bites típussal dolgozunk, akkor az összes egészünk 32 bitet fog foglalni a memóriában,

hiába kisebb az általunk használt legnagyobb szám is 2^{32}-nél. A $0-1$ sorozatoknál azonban létezik a pontosan illeszkedő 1 bites méret, az úgynevezett logikai típus.
Ezenkívül nem elég csak az elemeket ábrázolnunk, a köztük levő viszonyokat – például sorozatnál a sorrendet – is tárolnunk kell. Ezt a sorozatok és a fák esetén is megtehetjük tömbbel vagy valamilyen mutatókat használó adatszerkezettel. Mindkét adatszerkezetnek megvannak az előnyei és a hátrányai.
A tömb méretét előre meg kell adni. Egy n méretű tömb esetén a memóriában n-szer a tömbben tárolt típus méretének megfelelő hely lesz lefoglalva, és mivel ilyenkor a tömb elemeinek lefoglalt rekeszek egymás után következnek a memóriában, ennek a megvalósításnak az előnye, hogy az elemek közvetlenül címezhetőek. Az előzőekből adódik a hátránya is, miszerint a tömbök mérete nem dinamikus, így beszúrás és törlés esetén az egész tömböt át kell másolnunk, így ezeknek a műveleteknek a végrehajtása a tömb méretével arányos időt vesz igénybe.
Mutatókat használó adatszerkezetek esetén a foglalt rekeszek nem egymás után következnek a memóriában, így a méret dinamikus, a törlés és beszúrás konstans időt vesz igénybe. Másrészről ekkor nem csak magukat az elemeket, hanem a következő/előző (fák esetén a gyerekek/szülő) elemre mutató mutatót is tárolnunk kell. Mivel a kezdő elemen kívül (ilyen például a sorozat első eleme vagy a fa gyökere) a többi elem, csak a szomszédain keresztül érhető el, ezért a tömbtől eltérően ebben az esetben egy elem megtalálása az adatszerkezet elemszámának valamilyen függvényével arányos, nem konstans időben tehető meg.

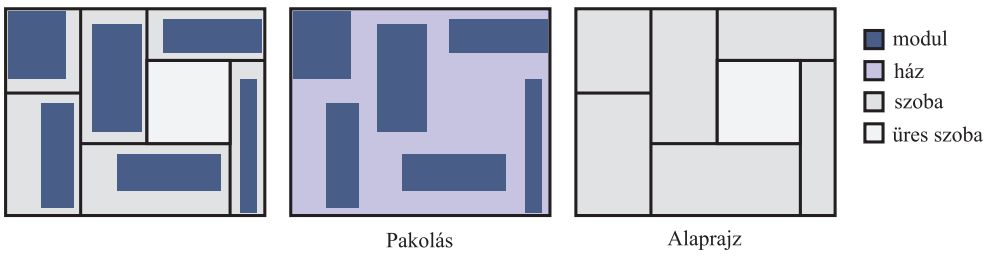
3. fejezet

Alapfogalmak

Legyen $\mathcal{M}$ téglalap alakú modulok m elemű halmaza. Minden modul szélessége és magassága adott valós szám, orientációja fix.
Egy a modul szélessét w_a, magasságát h_a jelöli. A modul koordinátáin a bal alsó sarok koordinátáit értjük, melyeket x_a-val illetve y_a-val jelölünk. Egy $x_a = 0, y_a = 0, h_a = 0, w_a = 0$ tulajdonságokkal rendelkező a modult ***üres modul***nak nevezünk.

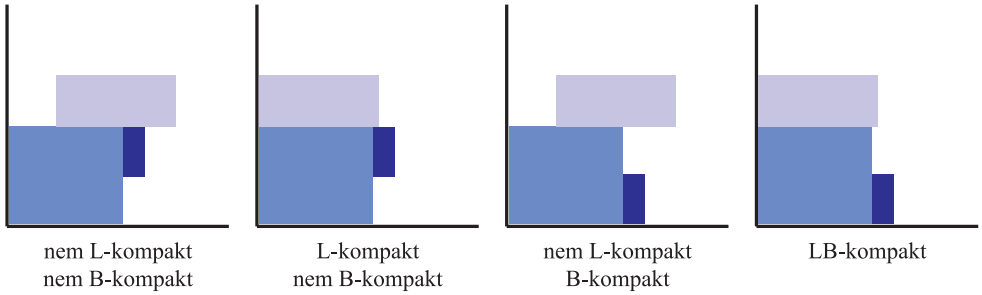
3.1. Pakolás

$\mathcal{M}$ elemeinek egy átfedés nélküli elhelyezését a síkon ***pakolás***nak (*packing*) nevezzük. Egy pakolást úgy adhatunk meg, hogy megadjuk a benne szereplő összes modul koordinátáit. A pakolást határoló minimális területű téglalap a ***ház*** (*chip*).



3.1. ábra. Pakolás és alaprajz

Egy pakolást ***B-kompakt***nak (***L-kompakt***nak) mondunk, ha semelyik modul nem mozdítható el az y-tengellyel (x-tengellyel) párhuzamosan lefelé (balra), ha a többi modul fixen marad. Egy pakolást ***LB-kompakt***nak mondunk, ha B-kompakt és L-kompakt.



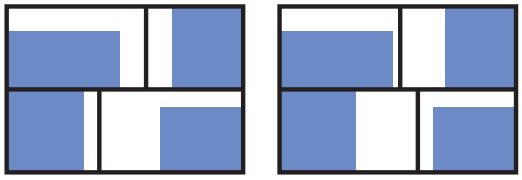
3.2. ábra. LB-kompakt pakolás

3.2. Alaprajz

Az ***alaprajz*** (*floorplan*) a ház felosztása téglalapokra, melyeket ***szobá***knak (*room*) nevezünk. Minden szoba legfeljebb egy modult tartalmazhat. Az egyetlen modult sem tartalmazó szobát ***üres***nek (*empty*) hívjuk. A szobák határait - beleértve a ház négy oldalát - ***fal***nak (*cutting-seg*) nevezzük. A ház oldalainak kivételével minden fal egy rá merőleges falban végződik, ***T-érintkezés***t (*T-intersection*) formálva.
Egy alaprajzot ***mozaik***nak (*mosaic*) nevezünk, ha a következők teljesülnek: [3]

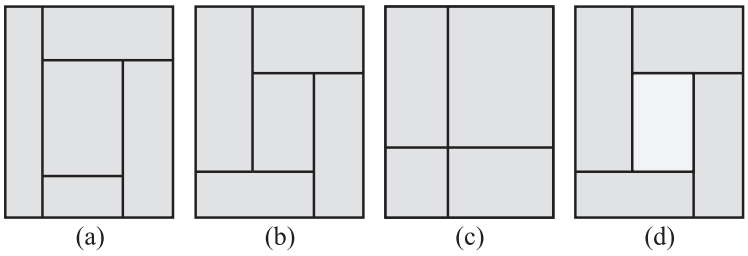
(i) Minden szoba pontosan egy modult tartalmaz, azaz nincs üres szoba.
(ii) Az alaprajz nemelfajult, azaz nincs + alakú T-érintkezés.

Ugyanahhoz a pakoláshoz tartozó két mozaik alaprajzot ekvivalensnek tekintünk, ha az egyik a másikba transzformálható az egy egyenes mentén az arra merőleges falak eltolásával. A 3.3. ábrán két ekvivalens alaprajz látható.



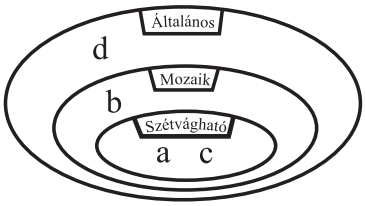
3.3. ábra. Ekvivalens alaprajzok

Egy alaprajzot ***szétvágható***nak (*slicing*) nevezünk, ha horizontális és vertikális kettévágásokkal rekurzívan előállítható. Egy kettévágás után a keletkező, több szobát magukban foglaló darabokat ***szuperszobá***knaknak hívjuk.



3.4. ábra. Alaprajzok típusai

A 3.4. ábrán látható alaprajzok közül az (a) szétvágható és mozaik, a (b) mozaik, de nem szétvágható, a (d) se nem szétvágható, se nem mozaik. A (c) szétvágható, és a szobák falainak minimális eltolásával általában megszüntethető az elfajultság. Ekkor egy vele ekvivalens alaprajzot kapunk, amely már mozaik. A falakat csak abban a kivételes esetben nem tudjuk eltolni, ha minden szobát pontosan kitölt a benne található modul. Ettől az esettől most eltekintünk, és az ilyen alaprajzokat mozaiknak tekintjük. A fentiek alapján az alaprajzok típusaira a 3.5. ábrán látható tartalmazások teljesülnek.



3.5. ábra

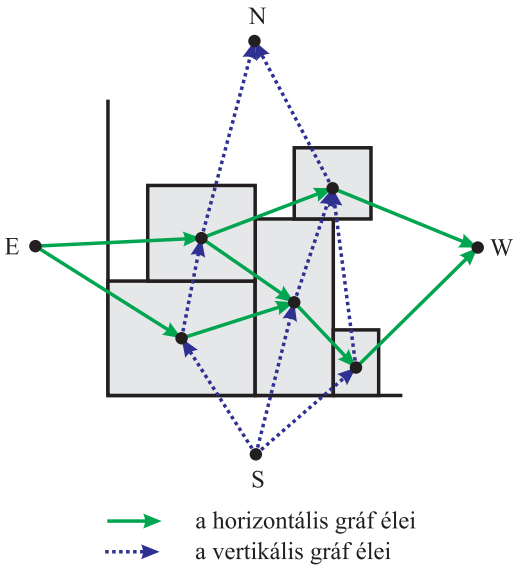
Egy pakolást aszerint nevezünk szétvághatónak, mozaiknak illetve általánosnak, hogy melyik az a legszűkebb alaprajztípus, amely konstruálható hozzá. Szétvágható pakolás esetén az egy szuperszobába tartozó modulokat ***szupermodul***nak nevezzük.

3.3. Kényszergráf

Egy adott, pakoláshoz tartozó horizontális (vertikális) ***kényszergráf*** (*constraint graph*) egy aciklikus irányított gráf, melyet a következőképpen konstruálhatunk meg. Minden modulnak megfelel egy csúcs. Egy csúcsot és a neki megfelelő modult ugyanazzal a betűvel jelöljük. Legyen a és b két modul. Az x-tengelyre (y-tengelyre) vett vetületük közös részét jelölje $P_x(a,b)$ ($P_y(a,b)$). Az a csúcsból a b csúcsba akkor vezet él, ha az a modul a b modultól balra (lejjebb) helyezkedik el, és a következők teljesülnek:

(i) $P_y(a,b)$ ($P_x(a,b)$) több mint egy pontból áll,
(ii) nem létezik olyan i, az a-tól jobbra (feljebb) és a b-től balra (lejjebb)

elhelyezkedő modul, melyre $P_y(a,b) \cap P_y(a,i)$ $(P_x(a,b) \cap P_x(a,i))$ több mint egy pontból áll.



3.6. ábra. Kényszergráf

A gráfhoz hozzáadunk egy forrás-, valamint egy nyelőpontot. A forrásból minden olyan, modulnak megfelelő csúcsba vezet él, melyeknek nincs bejövő éle. A nyelőbe minden olyan, modulnak megfelelő csúcsból vezet él, melynek nincs kimenő éle.

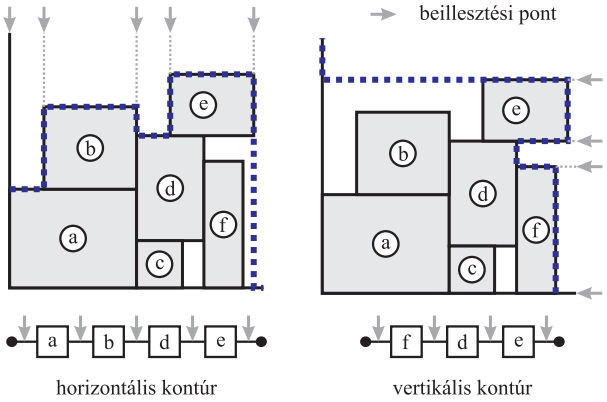
3. Állítás. *A kényszergráf síkbarajzolható.*

Bizonyítás. A fenti feltételek azt jelentik, hogy csak olyan modulok között vezethet él, melyek vagy érintkeznek (horizontális gráf esetén) egy függőleges oldalon, vagy nem érintkeznek ugyan, de az egyik jobb és a másik bal oldala között található egy sáv, amelyben nincsen másik modul. Így ha a gráf pontjait a modulok középpontjának feleltetjük meg, az élek vagy a modulokon belül haladnak, vagy a sávokban. Világos, hogy mindig behúzhatóak úgy, hogy ne metsszék egymást. A gyökér- és a nyelőpont éleire ez nyilvánvaló. □

Ezek alapján a gráfnak legfeljebb $3|V| - 6$, azaz n modul esetén $3n$ éle van.

3.4. Kontúr struktúra

Egy adott pakoláshoz tartozó horizontális (vertikális) ***kontúr*** (*contour*) a pakolást felülről (jobb oldalról) határoló ortogonális töröttvonal által érintett modulok listája, az y-tengelytől (x-tengelytől) kezdve. A horizontális (vertikális) töröttvonal kezdőpontja, illetve függőleges (vízszintes) szegmenseinek megfelelő pozíciók alkotják a ***beillesztési pont***ok (*insertation point*) halmazát.



3.7. ábra. Kontúr struktúra

Ha egy n modulból álló pakolást az üres pakolástól indulva a modulok egymás utáni, a pakolás tetejéhez egy megadott x-koordinátában és a lehetséges legkisebb y-koordinátában történő hozzáillesztésével építünk fel, akkor a keresett y-koordináta a pakolásban már szereplő, az x-tengelyre az új modullal átfedő vetületet alkotó modulok felső oldalainak maximális koordinátája lesz. A keresett y-koordináta kiszámolásához minden lépésben a már elkészített pakolás összes modulját végig kell nézni, így ez $O(n^2)$ időt vesz igénybe. Ugyanez a helyzet a jobb oldalhoz való illesztéskor, megadott y és megkeresendő minimális x-koordináta esetén.
A következő állítás szerint speciális esetekben a modulok számában lineáris időben is elvégezhető az összes beillesztés.

4. Állítás. *[3] n modul egymás utáni, a kontúr adott beillesztési pontjában történő beillesztése, valamint a kontúr felülírása $O(n)$ időt vesz igénybe.*

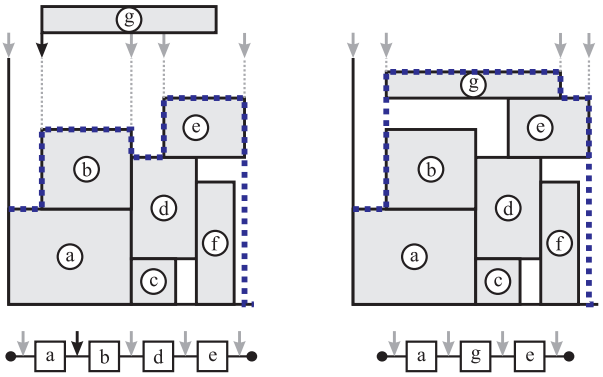
Bizonyítás. Minden modul beillesztésekor pontosan a kényszergráfban a hozzá tartozó csúcs bejövő éleinek megfelelő modulokat vizsgáljuk meg, így az összes beillesztés a kényszergráf éleivel arányos időben, $O(n)$-ben elvégezhető. □

Az 1. algoritmus a horizontális kontúrt írja felül az argumentumaként kapott modullal. A kontúr egy modulokból álló láncolt lista 0-tól indexelve. A beillesztési pontok értelmezhetők úgy, hogy ezek közül melyik modul bal felső sarkának x-koordinátájában kell a modult beilleszteni. Ennek a modulnak az indexe az algoritmus második argumentuma, illetve az utolsó beillesztési pont (amikor a ház alsó falához illesztjük a modult), a lista hosszával megegyező indexnek felel meg. A harmadik argumentum az az x-koordináta, ami elé eső modulokat a kontúrban vizsgálni kell. Az algoritmus azzal a modullal tér vissza, melynek felső falához a legkisebb y-koordinátában a beilleszteni kívánt modult illeszteni lehet.

Algorithm 1 Update(module, insIndex, maxX)

```
whereMax ← empty module;
maxY ← 0;
while insIndex < contour.Count and contour[insIndex].X < maxX do
  if maxY < contour[insIndex].Y + contour[insIndex].Height then
    whereMax ← contour[insIndex]
    maxY ← whereMax.Y + whereMax.Height
  end if
  if contour[insIndex].X + contour[insIndex].Width ≤ maxX then
    remove element from contour at insIndex
  end if
end while
insert module into contour at insIndex
return whereMax
```

Az algoritmust használva a beillesztendő modul koordinátái a következőképp kaphatók meg.



3.8. ábra. Beillesztés a horizontális kontúr segítségével

modul.X ← horizontalContour[insIndex].X
belowModule ← horizontalContour.Update(module, insIndex, module.X + module.Width)
modul.Y ← belowModule.Y + belowModule.Height

Megjegyzés. Az horizontális kontúrt használó algoritmus - mint láthattuk - mindig azokat a modulokat vizsgálja, melyek a beszúrandó modullal a vertikális kényszergráfban össze vannak kötve, és ezek közül azzal tér vissza, mely a felső oldala mentén érintkezik a beszúrandó modullal, vagy ha a modul a ház alsó falával érintkezik, akkor egy üres modullal. Ha az üres modult megfeleltetjük a kényszergráf gyökerének, és a beszúrandó modulnak megfelelő csúcsot mindig az algoritmussal kapott modulnak megfelelő csúccsal kötjük össze, akkor egy leghosszabb utak feszítőfáját kapjuk a vertikális gráfban.

4. fejezet

Reprezentációk

Mielőtt elkezdenénk megkeresni az optimális pakolást n adott modul esetén, szükségünk van valamilyen matematikai modellre, mely a pakolást jól leírja. Az itt ismertetett reprezentációk a pakolásokat sorozatokkal, illetve fákkal kódolják. Azonban vannak más kódolások is, például az általános típusú pakolásokat leírni képes BSG (*Bounded sliceline grid*), mely a modulok egymáshoz képesti viszonyait például egy négyzetrácsban ábrázolja.
A később ismertetett algoritmusok mind a kombinatorikus keresések családjához tartoznak, melyekben a keresési teret a reprezentáció kódjaiként értelmezzük. Egy kódot ***megvalósítható***nak (*feasible*) nevezünk, ha tartozik hozzá pakolás. A feladat megtalálni azt a megvalósítható kódot, mely a minimális területű pakolást írja le. Így az optimális megoldás megtalálásának lehetősége szempontjából fontos, hogy a soron következő reprezentációk megfelelnek-e az alább definiált fogalomnak. [11]

3. Definíció. Azt mondjuk, hogy a keresési tér ***P-megengedett*** (*P-admissible*), ha az alábbi négy tulajdonság teljesül:

(i) a keresési tér véges,
(ii) a benne szereplő összes kód megvalósítható,
(iii) minden kód kiértékelése és a hozzá tartozó pakolás realizációja lehetséges polinom időben,
(iv) a legjobbnak talált kódhoz tartozó pakolás optimális megoldása a téglalappakolási problémának.

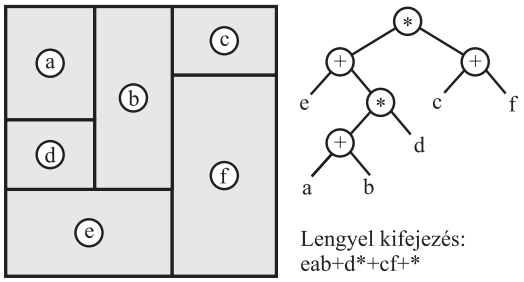
A szétvágható illetve mozaik struktúrák reprezentációi eleve nem lehetnek P-megengedettek, mivel nem teljesítik a (iv) feltételt. A (ii) feltétel sérülése esetén a kereső algoritmusok folytonos működése szakad meg, hiszen ekkor a kiértékelés előtt vagy ellenőrizni kell a kód megvalósíthatóságát, vagy a rossz kódok kiértékelését kell megszakítani. Ilyen lesz például az itt tárgyalt

reprezentációk közül az O-fa. Az (i) és (iii) feltétel szükségessége egy jó reprezentációhoz nyilvánvaló.

4.1. Szétvágható struktúrák reprezentációi és általánosításaik

4.1.1. Szétvágó fa (Slicing tree) és lengyel kifejezés (Polish expression

4. Definíció. A ***szétvágó fa*** egy bináris fa, melynek minden belső csúcsában a $+$ és a $*$ szimbólumok valamelyike szerepel, levelei pedig az $1 \ldots n$ egész számokkal címkézettek.



4.1. ábra

A szétvágó fa előállítása az alaprajzból úgy történik, hogy minden lépésben keresünk egy horizontális vagy vertikális szétvágó egyenest, és ennek mentén kettévágjuk az alaprajzot. Horizontális (vertikális) kettévágás esetén a szétvágó fa gyökerébe a $+$ ($*$) operátor kerül, és a bal oldali részfa fog megfelelni az alsó (bal oldali), a jobb oldali pedig a felső (jobb oldali) szuperszobának. Ezután az eljárást iteráljuk a szuperszobákon és a megfelelő részfákon. A 4.1. ábrán egy szétvágható alaprajz, valamint a hozzá tartozó szétvágó fa látható.

5. Állítás. *Egy n modulból álló pakoláshoz tartozó különböző szétvágó fák száma $O(n!\, 2^{3n-3}/n^{1.5})$*

Bizonyítás. Láttuk, hogy különböző alakú, n levelű bináris fák száma C_{n-1}, ez az 1. tétel szerint nagyságrendben $O(2^{2n-2}/(n-1)^{1.5}$. A belső csúcsok száma $n-1$, és minden belső csúcs $+$ vagy $*$ címkéjű lehet, ez 2^{n-1} lehetőség. Az n levélen a modulok $n!$ sorrendjének bármelyike állhat. □

A szétvágó fa inorder bejárásával egy aritmetikai kifejezést kapunk, példánkban az $(e+((a+b)*d))*(c+f)$-et. Ennek a postfix alakja a lengyelforma, a példában: $eab+d*+cf+*$, ami a postorder bejárásnak felel meg.

5. Definíció. Egy $b_1 b_2 \dots b_m$ $0-1$ sorozatot ***ballot sorozat***nak hívunk, ha minden prefixében a 0-k száma kevesebb, mint az 1-esek száma.

6. Definíció. [1] Egy $\alpha_1 \alpha_2 \dots \alpha_{2n-1} \in \{1,2,\dots,n,+,*\}^{2n-1}$ sorozat egy n modulhoz tartozó ***lengyel kifejezés*** pontosan akkor, ha a következők teljesülnek:
(i) i pontosan egyszer szerepen benne minden $1 \leq i \leq n$-re
(ii) $\sigma(\alpha_1)\sigma(\alpha_2)\dots\sigma(\alpha_{2n-1})$ ballot sorozat, ahol

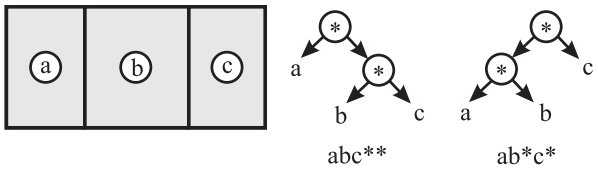
$$\sigma : \{1,2,\dots,n,+,*\} \to \{0,1\}$$

$$\sigma(i) = 1, \quad 1 \leq i \leq n$$

$$\sigma(+) = \sigma(*) = 0$$

A pakoláshoz tartozó lengyel kifejezést megkaphatjuk a szétvágó fa postorder bejárásából, vagy a szétvágó fa előállításához hasonlóan is. Az első horizontális (vertikális) szétvágáskor a következő feljegyezzük az $AB+$ ($AB*$) kifejezést, ahol A az alsó (bal oldali), B pedig a felső (jobb oldali) szuperszobát jelzi. A többi lépésben egy szuperszoba kettévágásakor a neki megfelelő betűt a kifejezésben lecseréljük a keletkező sorozatra.

A szétvágó fák és a lengyel kifejezések között kölcsönösen egyértelmű megfeleltetés van (így a lehetséges variációik száma is megegyezik). Egy n modulból álló alaprajzhoz azonban több szétvágó fa és lengyel kifejezés is tartozhat. Ennek az oka az, hogy amikor több szétvágási lehetőség is van, akkor bármelyiket választhatjuk.



Egy szétvágó fához a hozzá tartozó pakolást rekurzívan konstruálhatjuk meg a 4. algoritmussal. A könnyebb olvashatóság kedvéért vezessünk be néhány jelölést. A `csúcs.Module` alakú kifejezések a csúcs által reprezentált modulra vonatkoznak. Az algoritmus 13. illetve 17. sorában szereplő $\leftarrow +$ növelést jelent. A növelés szupermodul esetén az összes benne található modulra vonatkozik. A 19. sorban bal gyerekhez illetve jobb gyerekhez tartozó modul hozzáadása M-hez szupermodul esetén az összes benne található modul hozzáadását jelenti.
Az algoritmus végén a szétvágó fa egyetlen pontból, az eredeti gyökérből

fog állni, az ehhez tartozó szupermodulban pedig a modulok a kiszámított koordinátákkal szerepelnek. Ha a szétvágó fából alaprajzot szeretnénk konstruálni, az algoritmusból a 12. és 16. sorban található maximumszámítás felesleges, ekkor a két érték megegyezik.

Algorithm 2 StToPlacement(root)

```
1: if leftChild is not leaf then
2:    StToPlacement(leftChild)
3: end if
4: if rightChild is not leaf then
5:    StToPlacement(rightChild)
6: end if
7: M ← new supermodule
8: M.X ← leftChild.Module.X
9: M.Y ← leftChild.Module.Y
10: if root's label is + then
11:    M.Height ← leftChild.Module.Height + rightChild.Module.Height
12:    M.Width ← max(leftChild.Module.Width, rightChild.Module.Width)
13:    rightChild.Module.Y ← + leftChild.Module.Height
14: else {root's label is ∗}
15:    M.Width ← leftChild.Module.Width + rightChild.Module.Width
16:    M.Height ← max(leftChild.Module.Height, rightChild.Module.Height)
17:    rightChild.Module.X ← + leftChild.Module.Width
18: end if
19: add leftChild.Module and rightChild.Module to M
20: delete root's children
21: root.Module ← M
```

4.1.2. Normált lengyel kifejezés (Normalized Polish expression)

7. Definíció. [1] Egy szétvágó fát ***aszimmetrikus***nak nevezünk, ha nincs olyan csúcsa, hogy a csúcs és a jobb gyereke is ugyanazt az operátort tartalmazza.

A nemelfajult szétvágható alaprajzok és az aszimmetrikus szétvágó fák között kölcsönösen egyértelmű megfeleltetés áll fenn. Ugyanis az aszimmetrikus szétvágó fák annak felelnek meg, hogy ha több horizontális kettévágás is kínálkozik, akkor azok közül mindig a legfelsőt, ha több vertikális kínálkozik, azok közül mindig a jobb szélsőt választjuk.

Megjegyzés. Az aszimmetrikus fák ugyan megszüntetik a több horizontális és a több vertikális szétvágási lehetőségből fakadó redundanciát, de nem szüntetik meg azt, aminek az oka, hogy mindkétféle szétvágás lehetséges. Ha ilyen helyzet áll fenn, választhatnánk például mindig a horizontális szétvágást, és így a szétvágó fa elfajultság esetén is egyértelmű lenne. Ez a választási preferencia azonban nem azonosítható jól a fa valamely tulajdonságával.

A normált lengyel kifejezések az aszimmetrikus szétvágó fák postorder bejárásának felelnek meg, így a választási feltételek itt is jól megfogalmazhatóak.

8. Definíció. Az olyan lengyel kifejezéseket, melyekben nem áll egymás mellett közvetlenül két megegyező operátor (++ vagy $**$) ***normált lengyel kifejezés***nek nevezzük. [1]

6. Állítás. *Egy n modulból álló pakolást $2n!\,A_n$, különböző normált lengyel kifejezés írhat le, ahol A_n az n. szuper Catalan-szám.*

Bizonyítás. A kifejezésen belül a modulcímkék $n!$ féle sorrendben állhatnak. Ha a címkéket nem különböztetjük meg, akkor a kifejezések alakja $2A_n$ féle lehet, lásd a 3. állítás bizonyítását. □

Egy (normált) lengyel kifejezéshez tartozó pakolást a 3. algoritmussal állíthatunk elő. A növelések az előző fejezetben elmondottakhoz hasonlóan itt is a szupermodul összes elemére vonatkoznak, illetve alaprajz előállítása esetén a maximumszámítások nem szükségesek. Az algorimus végén a verem egyetlen szupermodult tartalmaz, melyben a modulok a kiszámított koordinátákkal találhatóak meg.

4.1.3. Rendezett szétvágó fa (Slicing Ordered Tree)

9. Definíció. Egy fát a következő tulajdonságokkal egy n szobából álló szétvágható alaprajzhoz tartozó ***rendezett szétvágó fá***nak nevezünk.

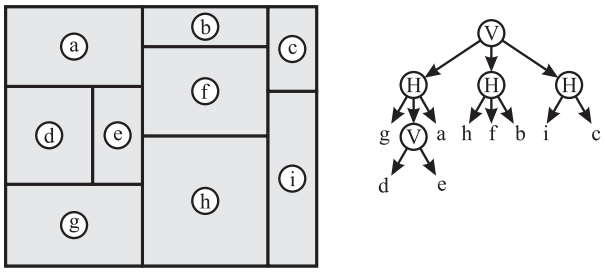
(i) A fa kitüntetett gyökérponttal rendelkezik, és n címkézett levele van.
(ii) Minden belső pontnak legalább két gyereke van.
(iii) Minden belső pont a v és h címkék valamelyikével címkézett.
(iv) Minden levél címkéje különböző.
(v) Minden belső pont címkéje különbözik a szülője címkéjétől.

Egy adott, nemelfajult alaprajzhoz tartozó rendezett szétvágó fát rekurzívan konstruálunk meg úgy, hogy az első lépésben az összes lehetséges kettévágást elvégezzük. Ha ezek horizontálisak (vertikálisak) voltak, akkor a fa gyökerébe H-t (V-t) írunk, és a szétvágások után keletkező szuperszobák balról

Algorithm 3 PeToPlacement(expression, modules[1...n])

```
1:  S ← empty stack
2:  for all e in expression do
3:    if e is module label then
4:      S.Push(modules[e])
5:    else {e is operator}
6:      M_R ← S.Pop()
7:      M_L ← S.Pop()
8:      M ← new supermodule
9:      M.X ← M_L.X
10:     M.Y ← M_L.Y
11:     if e is + then
12:       M.Height ← M_L.Height + M_R.Height
13:       M.Width ← max(M_L.Width, M_R.Width)
14:       M_R.Y ← + M_L.Height
15:     else {e is ∗}
16:       M.Width ← M_L.Width + M_R.Width
17:       M.Height ← max(M_L.Height, M_R.Height)
18:       M_R.X ← + M_L.Width
19:     end if
20:   end if
21:   add M_L and M_R to M
22:   S.Push(M)
23: end for
```

jobbra (lentről felfelé) vett sorrendben lesznek a gyerekek. Ezután az eljárást a szuperszobákon és a nekik megfelelő csúcsból mint gyökérből induló részfákon iteráljuk egészen addig, amíg minden szuperszoba egyetlen szobából nem áll. A 4.2. ábrán egy nemelfajult, szétvágható alaprajz és a hozzá tartozó rendezett szétvágó fa látható. A konstrukcióból adódóan a nemelfajult



4.2. ábra. Rendezett szétvágó fa

szétvágható alaprajzok és a rendezett szétvágó fák kölcsönösen megfeleltethetőek egymásnak. Így az összes lehetséges, n darab címkézett szobához tartozó alaprajz, illetve n modulból álló pakolás száma megegyezik az n modulcímkét tartalmazó szétvágó fák számával. Továbbá, mivel az egy-egyértelmű megfeleltetés a normált lengyel kifejezések és a nemelfajult szétvágható alaprajzok között is fennáll, az n szobához tartozó normált lengyel kifejezések száma is ennyi.

3. Tétel. *[7] A n szobát tartalmazó nemelfajult szétvágható alaprajzok és az ezeket reprezentáló rendezett szétvágó fák száma $n!\,2A_n$, ahol A_n az n. szuper Catalan-szám.*

Bizonyítás. A modulcímkék a fákon belül $n!$ féle sorrendben állhatnak.
Az egy rögzített permutációhoz rendezett szétvágó fák száma - ha a vágásokat jelző csúcsok címkéit nem különböztetjük meg - megegyezik a megengedett zárójelezésekkel. Feleltessük meg a vágásokat jelző csúcsokat a gyerekeiket magukban foglaló zárójelpároknak. Így a 4.2. ábrán látható fa például a $((g(de)a)(hfb)(ic))$ zárójelezésnek felel meg, amiből szintén egyértelműen visszafejthető a szétvágó fa. A 2. tétel szerint így a szétvágó fák száma nem megkülönböztetett vágáscímkékkel A_n. A vágásokhoz tartozó csúcsok címkéi szintenként váltakoznak, így ezeket egyértelműen meghatározza a gyökér címkéje, ami kétféle lehet, tehát a lehetőségek száma $2A_n$. □

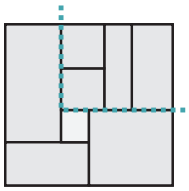
A rendezett szétvágó fához tartozó pakolás létrehozásának algoritmusa nagyon hasonlít a szétvágó fáéhoz, azzal a különbséggel, hogy itt a bal gyerek, jobb gyerek sorrend helyett az első gyerek, második gyerek, stb. sor-

rendben építjük fel a rekurziót. Az algoritmus leírásában az ott imertetett egyszerűsítéseket alkalmazzuk.

Algorithm 4 SotToPlacement(root)

```
1:  for all child of root do
2:    if child is not leaf then
3:      SotToPlacement(child)
4:    end if
5:  end for
6:  for all child of root except first child do
7:    M ← new supermodule
8:    M.X ← firstChild.Module.X
9:    M.Y ← firstChild.Module.Y
10:   if root's label is + then
11:     M.Height ← firstChild.Module.Height + child.Module.Height
12:     M.Width ← max(firstChild.Module.Width, child.Module.Width)
13:     child.Module.Y ← + firstChild.Module.Height
14:   else {root's label is ∗}
15:     M.Width ← first.Width + child.Module.Width
16:     M.Height ← max(first.Height, child.Module.Height)
17:     child.Module.X ← + first.Module.Width
18:   end if
19:   add firstChild.Module and child.Module to M
20:   delete root's first child {second child is the first now}
21:   root.firstChild.Module ← M
22: end for
```

4.1.4. Általánosított lengyel kifejezés (Generalized Polish expression)



4.3. ábra. Sarokvágás

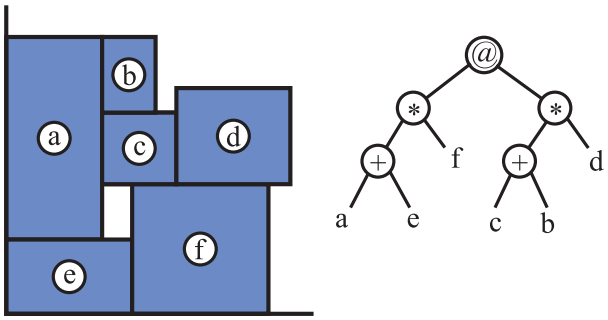
Annak érdekében, hogy a lengyel kifejezés az általános alaprajzokat is kezelni tudja, a horizontális és vertikális kettévágási lehetőség mellé bevezetünk egy új kettévágási módszert, a sarokvágást, mely ⌞ alakú vágást jelent. Könnyen látható, hogy egy általános alaprajzon, illetve a pakoláson e három közül valamelyiket mindig el lehet végezni. A sarokvágáshoz tartozó operátort ***sarokoperátor***nak (*corner operator*) nevezzük, és a @ karakterrel jelöljük.

10. Definíció. Egy $2n-1$ hosszú sorozat egy n modulhoz tartozó ***általánosított lengyel kifejezés***, ha a lengyel kifejezés definíciójában az alaphalmazhoz a @ jelet hozzávéve, $\sigma(@)$-t nullának definiálva az elmondottak teljesülnek.

7. Állítás. *Egy n modulhoz tartozó pakolást $O(n!\,2^{2n-2}3^{n-1}/n^{1.5})$ különböző általánosított lengyel kifejezés reprezentálhat.*

Bizonyítás. A bizonyítás az 5. állítás bizonyításával teljesen analóg. □

Egy általános pakoláshoz tartozó általánosított lengyel kifejezést a szétvágható pakoláshoz tartozó lengyel kifejezés mintájára kaphatunk meg.



4.4. ábra

A visszafejtés a horizontális és vertikális operátor esetén teljesen ugyanúgy működik. Sarokoperátor esetén a következő modult két, a már a pakolásban levő modul oldalai által meghatározott sarokba tesszük a következő preferenciák alapján [2]:

(i) válasszuk az első olyan sarkot, amely oldalainak mérete pontosan megfelel az elhelyezendő modul oldalainak méretének,
(ii) válasszuk az első olyan sarkot, amelyben a modul elhelyezése nem növeli meg a pakolás területét.

Látható, hogy egy sarokoperátor pontos megadásához szükség van a sarkot megadó modulok megnevezésére is, például a 4.4. ábrán látható pakoláshoz és szétvágó fához a $ae+f*cb+d*@_{(a,f)}$ általánosított lengyel kifejezés tartozik. Emiatt egy általánosított lengyel kifejezésből a pakolás előállítása az eddigieknél jóval bonyolultabb algoritmust igényel.

Megjegyzés. A normált lengyel kifejezés mintájára megalkothatjuk az általánosított normált lengyel kifejezést. A horizontális kettévágások közül válasszuk mindig a legfelsőt, a vertikálisak közül a jobb szélsőt, a sarokvágások közül pedig azt, amelynél a sarok origótól való távolsága a legnagyobb.

4.2. Mozaik struktúrák reprezentációi és általánosításaik

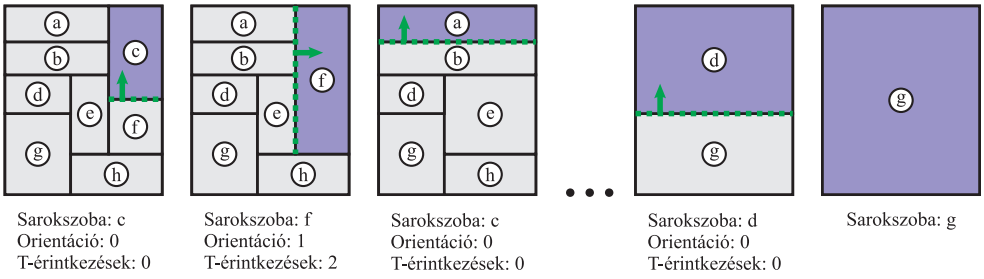
4.2.1. Sarokszoba-lista (Corner Block List)

11. Definíció. [3] Egy szobát ***sarokszobá***nak (*corner block*) nevezünk, ha az alaprajz jobb felső sarkán helyezkedik el.

8. Állítás. *[3] Minden alaprajzhoz pontosan egy sarokszoba tartozik.*

Egy sarokszoba bal alsó sarkát tartalmazó T-érintkezés 90°-kal ($\vdash$), vagy 180°-kal ($\perp$) elforgatott lehet csak.

12. Definíció. [3] A szoba orientációját vertikálisnak illetve horizontálisnak nevezzük, és 0-val vagy 1-gyel jelöljük, annak megfelelően, hogy a bal alsó sarkot tartalmazó T-érintkezés 90°-kal vagy 180°-kal van elforgatva.



4.5. ábra. Sarokszobák törlése

A sarokszoba-listát a sarokszobák egymást követő törlésével állítjuk elő. Ha az aktuális sarokszoba orientációja 0, a sarokszoba alsó falát a felső falba "húzzuk", ha az orientáció 1, a bal falat "húzzuk" a jobb oldali falba.

Lépés	S	L	T
1	c	0	0
2	f	1	110
3	a	0	0
4	b	0	0
5	e	0	0
6	h	1	10
7	d	0	0
8	g		

Minden lépésben feljegyezzük a sarokszoba címkéjét, orientációját, valamint a szobához tartozó T-érintkezéseket oly módon, hogy leírunk annyi egyest, ahány T-érintkezést az összehúzott fal tartalmazott, valamint egy 0-át, melynek szeparációs szerepe van. A 4.5. ábrán egy alaprajz és a sarokszobák törlésének első néhány és az utolsó két lépése látható. Ha az alaprajz már csak egy szobából áll, akkor ennek a szobának az orientációja lényegtelen (ezért nem is definiáljuk), valamint ekkor a T-érintkezések száma is mindig 0, így ezt sem szükséges feljegyezni. A példa lépéseihez tartozó adatok a mellékelt táblázatban láthatóak. Az S oszlop a sarokszobák címkéit,

az L az orientációikat, a T a hozzájuk tartozó T-érintkezéseket tartalmazza. Az oszlopokat alulról felfele összeolvasva kapjuk az alaprajzhoz tartozó (S, L, T) sarokszoba-listát. A fenti példához tehát a következő hármas tartozik: $(S, L, T) = (gdhebafc$,0100010,0100001100).

9. Állítás. *Adott n szobából álló alaprajzot leíró (S, L, T) sarokszoba-lista tárolásához legfeljebb $3(n-1) + n\lceil \log n \rceil$ bitre van szükség.*

Bizonyítás. A szobák címkéi legyenek egész számok. Ekkor egy címke legfeljebb $\lceil \log n \rceil$ biten tárolható, így S tárolásához legfeljebb $n\lceil \log n \rceil$ bitre van szükség. L egy $n-1$ hosszú $0-1$ sorozat, így $n-1$ biten tárolható. T-ben az utolsót kivéve minden szobához tartozik egy 0 szeparáló karakter, ez $n - -1$ bit. Ezenkívül a vertikális orientációjú szobákhoz együttesen legfeljebb annyi T-érintkezés tartozhat, mint ahány horizontális orientációjú szoba van és fordítva. Így T-ben összesen legfeljebb $n-1$ egyes lehet, azaz T leírható $2(n-1)$ bittel. □

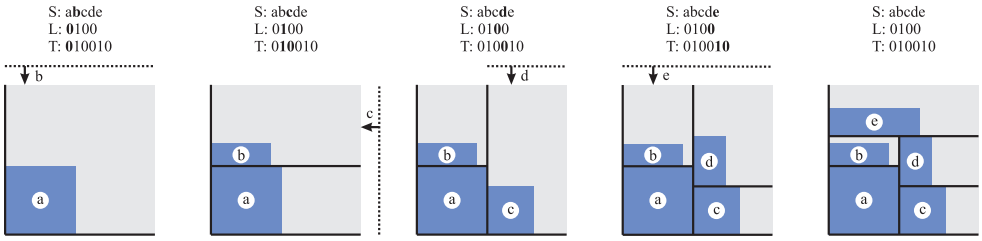
10. Állítás. *Egy n szobából álló alaprajzhoz $O(n!\, 2^{3n-3}/n^{1.5})$ különböző sarokszoba-lista tartozhat.*

Bizonyítás. A különböző S-ek száma $n!$, míg a különböző L-eké 2^{n-1}.
T legfeljebb $2n-2$ hosszú lehet, és semelyik prefixe sem tartalmazhat több 1-est, mint 0-át, ugyanis minden 0 egy szobához tartozik, és minden szoba beillesztésével egy új T-érintkezést hozunk létre, míg az egyesek a felhasznált T-érintkezéseket jelzik, és semelyik lépésben nem használhatunk fel több T-érintkezést, mint amennyit létrehoztunk. Az 1. tétel szerint a lehetséges T-k száma $O(2^{2n-2}/n^{1.5})$. □

Egy sarokszoba-listából a hozzá tartozó alaprajzot a sarokszobáknak az alaprajzba történő egymás utáni beszúrásával állítjuk elő. Első lépésben létrehozzunk az alaprajzot úgy, hogy azt kitölti az első szoba. Ezután minden lépésben S következő eleme lesz az aktuálisan beszúrandó szoba, L következő eleme a szoba orientációját, és T-ben a következő 0-ig szereplő elemek (a 0-t is beleértve), a szoba által fedett T-érintkezések száma. Ha az orientáció 0, akkor vesszük ház felső falának azt a szegmensét, mely a jobb oldali faltól kezdődik, egy, a felső falra merőleges falban végződik, és pontosan annyi T-érintkezést tartalmaz a belsejében, amennyi a T-érintkezés információ egyeseinek a száma. Ezt a szegmenst eltoljuk lefelé, és a keletkező helyre kerül a beszúrandó szoba. Ha az orientáció egy, hasonlóan járunk el a ház jobb oldali falának a felső faltól kezdődő szegmensével.

A sarokszoba-listával pakolást a következőképpen tudjuk előállítani. Minden modul címkéje megegyezik az őt tartalmazó szoba címkéjével, és minden

modult az őt tartalmazó szoba bal alsó sarkában helyezünk el. Az első lépésben az első szoba kitölti az első síknegyedet. Ezután minden lépésben a beillesztendő modul orientációjának megfelelően a ház jobb oldali/felső falának megfelelő szegmensét addig toljuk balra/lefelé, amíg a szegmens alatti szobákban levő modulok felső oldalánál maximuma engedi. A 4.6. ábrán egy példa látható.



4.6. ábra. Modulok pakolása sarokszoba-listával

A pakolás előállítását a 4.6. algoritmus szemlélteti. Itt T i-edik eleme T-nek $i+1$. modulhoz tartozó részsorozatát jelenti, tehát ha például a 3. modulnak két T-érintkezést kell lefednie, akkor $T[2] = 110$. Minden tömb indexelése 0-tól kezdődik, a tömb hossza a benne tárolt modulok számát jelenti. A `modules` tömb tartalmazza a modulokat, elemeire a modulok nevével hivatkozunk. A kontúrbeli maximum megtalálásához a 13. oldalon található 1. algoritmust használjuk. Mivel itt a beszúrt modulhoz tartozó szoba a modul méretétől függetlenül fedi a megadott T-érintkezéseket, ezért minden esetben végignézzük a kontúr összes hátralevő elemét, és mindet ki is töröljük, valamint a lépés végén az aktuálisan nem használt kontúr végére is beszúrjuk az adott elemet.

Láthatjuk, hogy a mozaik alaprajzok és a sarokszoba-listák között kölcsönösen egyértelmű megfeleltetés van. Így a sarokszoba listák segítségével meg tudjuk határozni az n szobából álló mozaik pakolások pontos számát.

4. Tétel. *Az n szobából álló mozaik alaprajzok száma B_n, ahol B_n az n. Baxter-szám. [7]*

Bizonyítás. Jelöljük $F_n(i,j)$-vel az olyan n szobából álló alaprajzok számát, melyeknél a ház felső falán i, a jobb falán j T-érintkezés helyezkedik el. Így a következő megállapításokat tehetjük:

(i) $F_1(0,0) = 1$, ez az 1 szobából álló pakolás.
(ii) $F_1(i,j) = 0$, ha $i > 0$ vagy $j > 0$, ugyanis egy 1 szobából álló alaprajz esetén a ház semelyik fala sem tartalmazhat T-érintkezést.

Algorithm 5 CblToPlacement(modules, S, L, T)

```
Initialize horizontalContour with modules[S[0]]
Initialize verticalContour with modules[S[0]]
i ← 0;
while i + 1 < n do
  if L[i] = 0 then
    insIndex ← horizontalContour.Length - T[i].Ones - 1
    modules[S[i]].X ← horizontalContour[insIndex].X
    whereMax ← horizontalContour.Update(modules[S[i]], insIndex, ∞)
    modules[S[i]].Y ← whereMax.Y
    verticalContour.Add(modules[S[i]])
  else
    insIndex ← verticalContour.Length - T[i].Ones - 1
    modules[S[i]].Y ← verticalContour[insIndex].Y
    whereMax ← verticalContour.Update(modules[S[i]], insIndex, ∞)
    modules[S[i]].X ← whereMax.X
    horizontalContour.Add(modules[S[i]])
  end if
end while
```

(iii) $F_n(0,0) = 0$, ha $n > 1$, ugyanis legalább két szobából álló pakolás esetén a ház vagy a jobb vagy a felső falán kell hogy tartalmazzon legalább egy T-érintkezést.

(iv) $F_n(i,j) = 0$, ha $i+j \geq n$, ugyanis egy n szobából álló pakolás legfeljebb $n-1$ T-érintkezést tartalmazhat összesen.

(v) $F_n(i,j) = 0$, ha $i < 0$ vagy $j < 0$, hiszen a T-érintkezések száma nem lehet negatív.

Tegyük fel, hogy már ismerjük azon n szobából áll alaprajzok számát, melyeknek felső falán i, jobb falán pedig több mint j T-érintkezés helyezkedik el, valamint azokét is, melyeknek felső falán több mint i, jobb falán pedig j T-érintkezés helyezkedik el. Így már megkaphatjuk az olyan $n+1$ szobából állókét, melyeknek felső falán pontosan $i+1$, jobb falán pontosan $j+1$ T-érintkezés található. Ugyanis ha beillesztünk egy 0 orientációjú szobát egy j T-érintkezést a jobb falán és több mint i-t a felső falán tartalmazó alaprajzba, akkor ez a művelet a jobb falon levő T-érintkezések számát eggyel növeli, így $j+1$ T-érintkezést kapunk, míg a felső falon levők közül pontosan annyit fedünk le vele, hogy a megmaradó T-érintkezések száma $i+1$ legyen, tehát $i+1$ esetén 0-t, $i+2$ esetén 1-et, stb. Hasonló mondható el az 1 orientációjú szoba beillesztésekor is a felső falon i, a jobb falon legalább $j+1$ T-érintkezést

tartalmazó alaprajzba. A fentiek alapján a következő rekurzió adódik:

(vi) $F_{n+1}(i+1, j+1) = \sum_{k=1}^{\infty} F_n(i+k, j) + F_n(i, j+k)$

Az összegzést nyugodtan végezhetjük a végtelenig, hiszen (iv) alapján a tagok egy idő után 0-k lesznek. A fentiek alapján $F_n i, j$ megegyezik a 2. állítás T_{n-1}-évél.
Világos, hogy az összes, n szobából álló alaprajz $M(n)$ száma megkapható, ha a felső és jobb falon levő T-érintkezések i és j száma szerint összegzünk az összes lehetséges módon, így

$$M(n) = \sum_{i,j \geq 0} F_n(i, j) = \sum_{i,j \geq 0} T_{n-1}(i, j) = B_n$$

□

4.2.2. Általánosított sarokszoba-lista (Extended Corner Block List)

13. Definíció. Egy olyan modult, melynek a szélessége és a magassága is nulla, ***hamis modul***nak nevezünk.

Egy ilyen modult tartalmazó szoba pont az üres szobának felel meg. Ezek bevezetésével a sarokszoba-lista már az általános alaprajzokat is kezelni tudja. Az általánosított sarokszoba-lista előállítása az alaprajzból, a sarokszoba-lista előállításával analóg módon történik, azzal a különbséggel, hogy az üres szobák nem rendelkeznek egyedi címkével. Ha egy üres szobát törlünk, akkor az F címkét jegyezzük fel S-be, L-be ugyanúgy a szoba orientációja, T-be az általa fedett T-érintkezések számának megfelelő $0-1$ sorozat kerül.

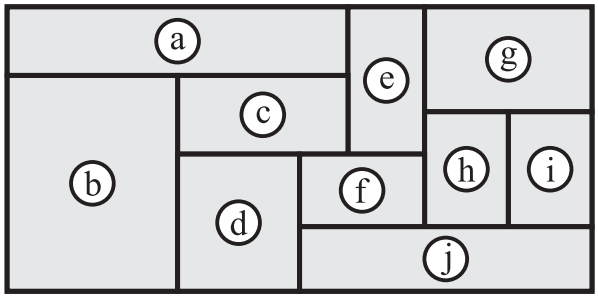
Az általánosított sarokszoba-listából a pakolást hasonlóan készítjük el, mint a sarokszoba-listából. Az üres modulok beszúrásának célja a T-érintkezések számának manipulálása, így ezekkel ugyanúgy felül kell írni a kontúrokat. Az egyetlen különbség az, hogy itt a `while` ciklus nem n-ig megy, hanem addig, amíg n darab modulcímkét fel nem dolgoztunk. Ezután már csak üres szobák következhetnének, ezeket pedig figyelmen kívül lehet hagyni.

4.2.3. Q-sorozat (Q-sequence)

14. Definíció. [5] Egy szobát ***farkszobá***nak (*tail room*) nevezünk, ha az alaprajz jobb alsó sarkában helyezkedik el. Egy nem-farkszoba jobb alsó sarkát tartalmazó T-érintkezés "szárát" a szoba ***elsődleges fal***ának nevezzük.

Egy nem-farkszoba elsődleges falát - ha a fal vertikális (horizontális) - jobb oldalról (alulról) érintő szobákat a szoba ***asszociált szobái***nak nevezzük. Ezek közül a legfelsőt (bal szélsőt) a szoba ***rákövetkező***jének mondjuk.

Egy mozaik alaprajzhoz tartozó Q-sorozatot a következőképp állítunk elő. Leírunk annyi R-t, ahány szoba érinti a pakolás bal falát, majd annyi B-t, amennyi szoba érinti a pakolás felső falát. Ezután minden lépésben veszünk egy szobát, és leírjuk a címkéjét. Ha az elsődleges fala horizontális, akkor a címkét annyi B követi, amennyi az asszociáltjai száma. Vertikális fal esetén ugyanennyi R-t jegyzünk fel. Az eljárást a bal felső szobával kezdjük, és minden lépésben az épp aktuális szoba rákövetkezőjével folytatjuk. A 4.7 ábrán egy alaprajz és a hozzá tartozó Q-sorozat látható.



*RRBBB*a*BB*b*RR*c*R*e*BB*d*RR*f*RR*g*BB*h*Ri*B*j*

4.7. ábra. Q-sorozat

A Q-sorozathoz tartozó RQ-sorozat (BQ-sorozat) az a sorozat, melyet Q-ból az összes B (R) törlésével nyerünk. Az RQ-sorozatban illetve a BQ-sorozatban az R-ek, illetve a B-k nyitó, és a modulcímkék záró zárójelre való cserélésével kapott két sorozatot a Q-sorozathoz tartotó ***zárójelrendszernek*** (*parenthesis system*) nevezzük.

15. Definíció. [5] Egy, a szobacímkéket és az R, B karaktereket tartalmazó sorozat egy n szobából álló alaprajzhoz tartozó ***Q-sorozat***, ha a következők teljesülnek:

(i) Minden olyan részsorozat, mely két szobacímke között helyezkedik el, legalább egy B vagy R karaktert tartalmaz.
(ii) A sorozathoz tartozó zárójelrendszerek érvényes zárójelezések.

A Q-sorozat és a sarokszoba ugyanazon az elven alapul. Ha a farkszobának a bal alsó sarokban elhelyezkedő szobát nevezzük ki, ennek megfelelően az elsődleges falat a bal alsó sarkot tartalmazó T-érintkezés jelöli ki, és az

asszociált szobák ettől balra (az R címke helyett L-et használunk) vagy lefele helyezkedhetnek el. A rákövetkező szoba vertikális elsődleges fal esetén az asszociált szobák közül a legfelső, horizontális fal esetén a legbalrább levő lesz. Ha az így kapott Q'-sorozatot megfordítjuk, a modulcímkék sorrendje a sarokszoba-lista S sorozatának felel meg. Egy modulcímkét megelőző B vagy L sorozat hossza a modul által fedett T-érintkezések száma, a betűjel pedig az orientációt mondja meg, a B a vertikális (0) orientációnak, az L a horizontális (1) orientációnak felel meg.
A Q-sorozatnak a sarokszoba-listához hasonlóan létezik általánosítása [6]. A sarokszoba-listához való hasonlóság miatt a Q-sorozattal és kiterjesztésével bővebben nem foglalkozunk.

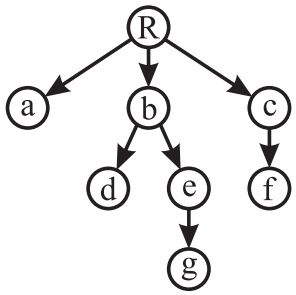
4.3. Általános struktúrák reprezentációi

4.3.1. O-fa (O-Tree)

16. Definíció. Az O-fa egy kijelölt gyökérponttal rendelkező rendezett irányított fa, ahol a rendezettség alatt rendezett DFS-bejárást értünk. A bejáráshoz kijelölünk egy preferenciát, például hogy mindig a "legbaloldalibb" részfát járjuk be először.

Az O-fa felfogható a bináris fa általánosításaként. Egy O-fában minden csúcsnak akárhány gyereke lehet, de a gyerekek sorrendje itt is lényeges. A bal gyerek, jobb gyerek helyett itt első, második, harmadik, stb. gyereket különböztetünk meg.

Egy $n+1$ csúcsú O-fát egy $2n$ hosszú $0--1$ sorozattal (T), valamint a gyökértől különböző csúcsok címkéinek n hosszú sorozatával (Π) kódulunk a következőképp. A DFS bejárás során előrelépéskor T-hez hozzáírunk egy 0-t, Π-hez pedig annak a csúcsnak a címkéjét, amelybe beléptünk. Hátralépéskor T-hez hozzáírunk egy 1-est. O-fa kódolására a 4.8. ábrán látható egy példa.



4.8. ábra

11. Állítás. *Adott $n+1$ csúcsú O-fához tartozó (T, Π) pár tárolásához $2n + n\lceil \log n \rceil$ bitre van szükség.*

Bizonyítás. A modulok címkéi legyenek egész számok. Ekkor egy címke tárolásához legfeljebb $\lceil logn \rceil$ bitre, így T tárolásához legfeljebb

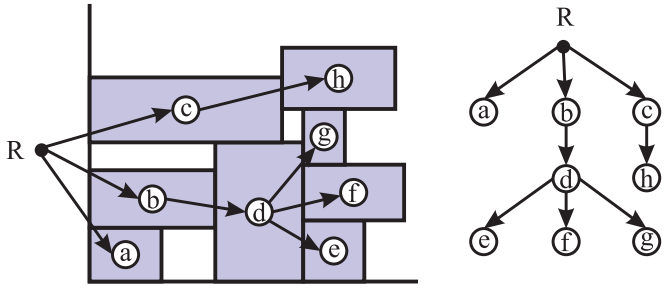
$n\lceil \log n \rceil$ bitre van szükség. Π $2n$ hosszú $0-1$ sorozat, így $2n$ biten tárolható. □

12. Állítás. *Adott gyökérpont esetén a különböző $n+1$ csúcsú O-fák száma:* $O(n!\,2^{2n}/n^{1.5})$.

Bizonyítás. A különböző Π-k száma $n!$.
T olyan $0-1$ sorozat, melynek semelyik prefixe nem tartalmazhat több 1-est, mint 0-t, hiszen az azt jelentené, hogy a DFS-bejárás során van olyan pillanat, ameddig többször léptünk vissza, mint előre. Továbbá a 0-k és az 1-esek száma megegyezik, mert a bejárást a gyökérben fejezzük be akkor, amikor már nem tudunk bejáratlan pontba lépni. Így az 1. tétel szerint a különböző O-fák száma $O(n!\,2^{2n}/n^{1.5})$.

□

O-fával LB-kompakt pakolásokat reprezentálhatunk. Egy adott pakoláshoz tartozó horizontális O-fát a következőképp kaphatunk meg. A pakolás bal oldalán felveszünk egy új pontot − ez lesz a gyökérpont, mely a ház bal oldali falát reprezentálja −, valamint minden modulnak megfelel egy pont a fában. Legyen a és b két csúcs a fában. a-ból b-be akkor vezet él, ha az a modul jobb oldalának (illetve ha a a gyökérpont, akkor a ház bal oldali falának) és a b modul bal oldalának x-koordinátája megegyezik. Ha egy pontból több pontba is vezet él, akkor ezek közül a fában a legbaloldalibb a legalsó modulhoz tartozik és így tovább. A horizontális O-fából a pakolást úgy kaphatjuk



4.9. ábra. Horizontális O-fa

meg, hogy a gyökérpontból indulva DFS-bejárással bejárjuk a fát. Minden a modulra álljon a $\Psi(a)$ halmaz azokból a modulokból, melyek a előtt vannak T-ben, és az x-tengelyre vett vetületük átfedő a-val. Ha az a csúcs szülője a b, akkor az a modul x és y koordinátája a következő lesz:

$$x_a = x_b + w_b$$

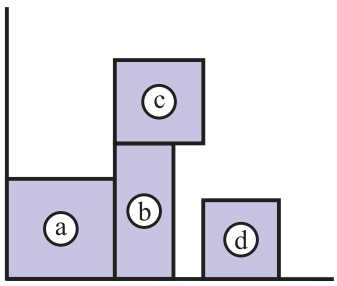
$$y_a = \max_{i \in \Psi(a)} y_i + h_i$$

A 6. algoritmus a horizontális O-fa pakolássá alakítását mutatja be. Vertikális O-fából hasonlóan készíthetjük el a pakolást a modulokhoz tartozó x és y koordináta, a magasság és szélesség, valamint a horizontális és verikális kontúr szerepének felcserélésével.

Algorithm 6 HorizontalOTreeToPlacement(modules, (T, Π))

```
insertationIndex ← 0
verticalOTree ← new tree with n + 1 nodes, no edges
for i = 0 to n do
  modules[Π[i]].X ← modules[Π[i]].Parent.X+modules[Π[i]].Parent.Width
  whereMax ← horizontalContour.Update(modules[Π[i]], insertationIndex, modules[Π[i]].X + modules[Π[i]].Width)
  modules[Π[i]].Y ← whereMax.Y + wheremax.Height;
  add edge from whereMax.Label to modules[Π[i]].Label in verticalOTree;
  insertationIndex ← index of modules[Π[i]].Parent in horizontalContour
end for
return verticalOTree;
```

Nem minden O-fát kódoló (T, Π) pár határoz meg LB-kompakt pakolást. Tekintsük például a következő horizontális O-fát: (T, Π) = (*abcd*,00100111). Ehhez az ábrán látható pakolás tartozik. Ez B-kompakt, de nem LB-kompakt. Általánosságban is egy horizontális O-fához tartozó pakolás mindig B-kompakt, egy vertikális O-fához tartozó pakolás mindig L-kompakt. Az O-fa által meghatározott pakolást a 7. algoritmussal transzformálhatjuk LB-kompakttá.

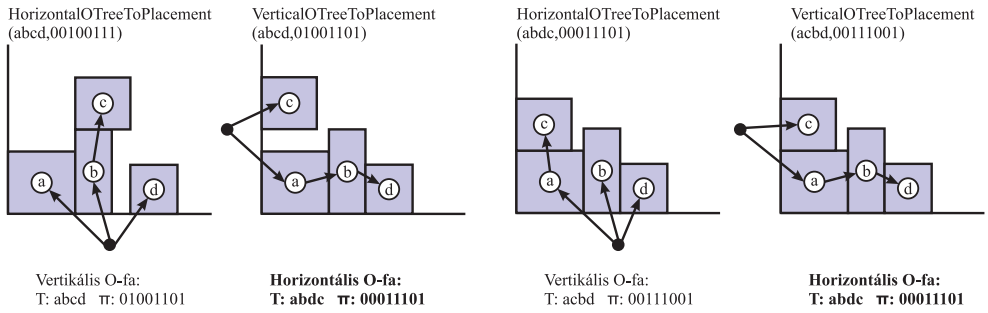


Horizontális O-fa:
T: abcd π: 00100111

Algorithm 7 OTreeToLBCompactPlacement(modules, T, Π)

```
changed = true;
while changed do
  (T_v, Π_v) = HorizontalOTreeToPlacement(modules, T, Π)
  (T_h, Π_h) = VerticalOTreeToPlacement(modules, T_v, Π_v)
  if (T_h, Π_h)=(T, Π) then
    changed = false
  end if
end while
```

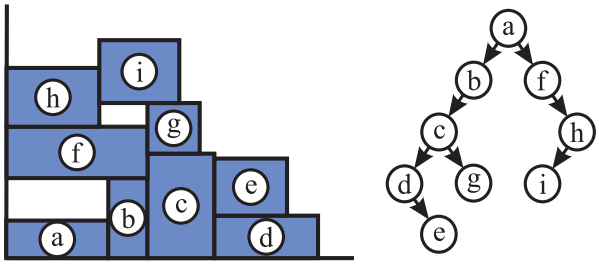
Az algoritmus lépéseit a 4.10. ábrán látható példa szemlélteti. Látható, hogy minden lépés vagy balra vagy lefele tolja el a mozdítható modulokat egész addig amíg vannak ilyenek.



4.10. ábra

4.3.2. B*-fa (B*-Tree)

A B*-fa egy bináris fa, ahol a mélységi bejárás sorrendje az O-fához hasonlóan lényeges. Itt is mindig a bal oldali részfát járjuk be elsőként.
Egy adott n modulból álló LB-kompakt pakoláshoz a hozzá tartozó n csúcsú horizontális B*-fát a következőképp konstruáljuk meg. A bal alsó sarokban levő modul felel meg a gyökérnek. Ezután az előbb említett DFS-sorrendben meghatározzuk a bal és jobb gyerekeket. Egy a csúcsnak a b bal gyereke, ha a b modul a-nak bal oldali szomszédja, még nem jártunk be, és ha több ilyen van, akkor ezek közül a legalsó. Egy a csúcsnak a b jobb gyereke, ha a b modul az a fölött helyezkedik el, az x-koordinátájuk megegyezik, és ha az a modulnak van bal szomszédja, akkor annak a felső oldalának y-koordinátája nagyobb, mint a b y-koordinátája. Ha több ilyen van, akkor ezek közül a legalsó. A konstrukcióból adódóan minden LB-kompakt pakoláshoz pontosan egy B*-fa tartozik. A 4.11. ábrán egy LB-kompakt pakolás és a hozzá



4.11. ábra. Horizontális B*-fa

tartozó B*-fa látható.
A B*-fából a pakolást úgy kapjuk, hogy a DFS-sorrendnek megfelelően helyezzük el a modulokat. A gyökér koordinátája (0,0) lesz, ezután minden b csúcsra

(i) ha b az a csúcs bal gyereke, akkor $x_b = x_a + w_a$
(i') ha b az a csúcs jobb gyereke, akkor $x_b = x_a$
(ii) $y_b = \max_{i \in \Psi(b)} y_i + h_i$, ahol $\Psi(b)$ azokból a modulokból áll, melyek a DFS-sorrendben b előtt helyezkednek el, és az x-tengelyre vett vetületük átfedő.

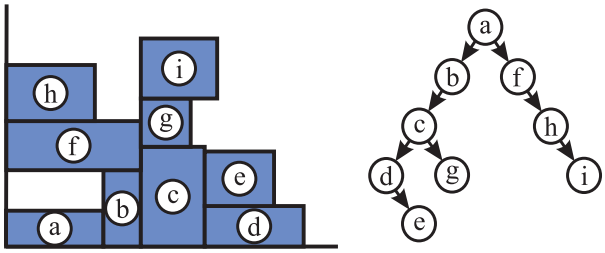
A pakolást a 8. algoritmus állítja elő.

Algorithm 8 HorizontalBTreeToPlacement(root)

```
Initialize horizontalContour with root.Module
S ← empty stack
S.Push(root)
while S.Count > 0 do
  node ← S.Peek()
  if node has left child and node.LeftChild is not visited then
    insertationIndex ← index of node.Module in horizontalContour
    node.LeftChild.Module.X ← node.Module.X + node.Module.Width
    whereMax ← horizontalContour.Update(node.leftChild.Module,
    insertationIndex, node.leftChild.Module.X + node.leftChild.Module.Width
    node.LeftChild.Module.Y ← whereMax.Y + whereMax.Height
    S.Push(node.LeftChild)
  else if node has right child and node.RightChild is not visited then
    insertationIndex ← index of node.Module in horizontalContour
    node.RightChild.Module.X ← node.Module
    whereMax ← horizontalContour.Update(node.RightChild.Module,
    insertationIndex, node.RightChild.Module.X + node.RightChild.Module.Width
    node.RightChild.Module.Y ← whereMax.Y + whereMax.Height
    S.Push(node.RightChild)
  else {node is leaf or already visited}
    S.Pop()
  end if
end while
```

Az O-fához hasonlóan egy B*-fához sem tartozik feltétlenül LB-kompakt pakolás. Tekintsük például a 4.11. ábra kis módosításával keletkező a 4.12.

ábrán látható esetet.



4.12. ábra. Nem LB-kompakt pakolást reprezentáló B*-fa

Így B*-fák esetén is szükség van a pakolás kompakttá tételére, mely most kicsit bonyolultabb, mint az O-fák esetén. Az y-koordináták kiszámítására B*-fáknál is a kontúr struktúrát használjuk. Itt a horizontális fából készülő pakolás előállítása során a vertikális B*-fa gyökere megegyezik a horizontális B*-fa gyökerével, és egy új, a gyökértől különböző a modul beillesztésekor a következő esetek állhatnak fenn:

(i) a bal gyerek és a maximális y-koordinátát meghatározó modul üres modul

(ii) a bal gyerek és a maximális y-koordinátát meghatározó modul a nemüres b modul

(iii) a jobb gyerek és a maximális y-koordinátát meghatározó modul nemüres b modul

Az (i) esetben a beillesztendő a modul a vertikális B*-fa gyökerének jobb gyereke lesz, illetve ha ez a pozíció már foglalt, akkor a jobb ágon a modult addig kell lefelé csúsztatni, amíg nem foglalt pozíciót találunk. Az (ii) és az (iii) esetben az a modul a b modul bal gyereke lesz, illetve ha ez a pozíció már foglalt, akkor a b modul bal gyerekének jobb gyereke, ennek a foglaltsága esetén az előbb említett csúsztatást kell végrehajtani. Ha a kiindulási fa vertikális, akkor vertikális kontúrt használunk, és a horizontális fát az x és y koordináta szerepének felcserélésével állítjuk elő.

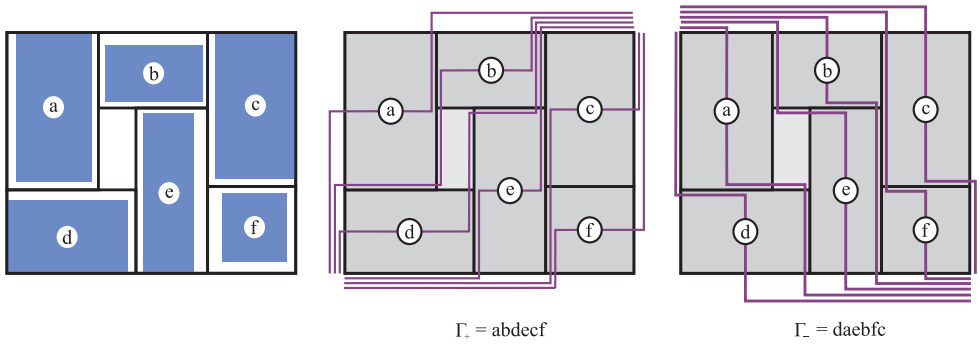
4.3.3. Modulsorozat-pár (Sequence Pair)

A $(\Gamma_+, \Gamma_)$ modulsorozat-pár kettő, a modulok címkéinek permutációjából álló sorozat, mely a modulok horizontális és vertikális sorrendjének információját hordozza.

13. Állítás. *Egy n modulból álló pakolást kódoló modulsorozat-pár tárolásához legfeljebb $2n\lceil \log n \rceil$ bitre van szükség.*

14. Állítás. *Egy n modulból álló pakolást leíró különböző modulsorozat-párok száma $(n!)^2$.*

Tekintsünk egy pakolást és egy hozzá tartozó alaprajzot. Válasszuk ki az alaprajz egy nemüres szobáját, és a szoba középpontjából indulva készítsünk el négy töröttvonalat. Az első töröttvonal haladjon felfele, amíg nem ütközik a szoba falába, majd balra a szoba fala mentén, amíg nem ütközik egy merőlegeses falba, innen ismét felfelé, és így tovább, amíg el nem érkezik a ház bal felső sarkához. A többi töröttvonal hasonló módon készül, irányaik: le-jobbra, jobbra-fel és balra-le. [11]
Az első és második töröttvonal együttesen alkotja a szoba ***negatív helyzetvonal***át (*negative locus*), a harmadik és negyedik töröttvonal pedig a ***pozitív helyzetvonal***át (*positive locus*). Készítsük el az alaprajz összes nemüres szobájához tartozó pozitív és negatív helyzetvonalakat a 4.13. ábrán látható módon.



4.13. ábra. Pozitív és negatív helyzetvonalak

5. Tétel. *[11] Semelyik két negatív helyzetvonal nem metszi egymást. Semelyik két pozitív helyzetvonal nem metszi egymást.*

Az előző tétel következménye, hogy a pozitív és negatív helyzetvonalak sorba rendezhetőek. A helyzetvonalakat a hozzájuk tartozó modul/szoba címkéjével jelölve, a pozitív helyzetvonalak bal felső saroktól jobb alsó sarokig vett sorrendje (Γ_+) és a negatív helyzetvonalak bal alsó saroktól jobb felső sarokig vett sorrendje (Γ_-) által alkotott (Γ_+, Γ_-) pár a pakoláshoz tartozó modulsorozat-pár.

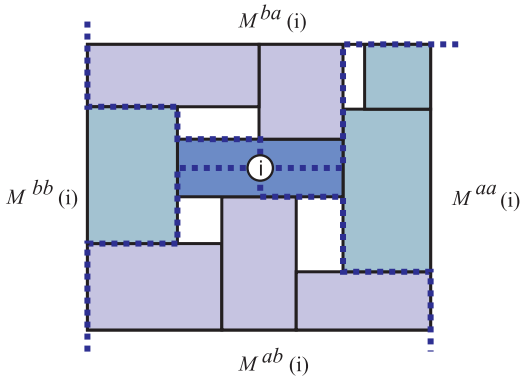
Minden modulhoz definiáljunk négy halmazt a következőképp [11]:

$\mathcal{M}^{aa}(i) = \{j | j$ az i után van Γ_+-ban és Γ_--ban is$\}$
$\mathcal{M}^{bb}(i) = \{j | j$ az i előtt van Γ_+-ban és Γ_--ban is$\}$

$\mathcal{M}^{ab}(i) = \{j | j$ az i után van Γ_+-ban és az i előtt van Γ_--ban$\}$
$\mathcal{M}^{ba}(i) = \{j | j$ az i előtt van Γ_+-ban és az i után van Γ_--ban$\}$

Ezek a halmazok a 4.14 ábrán látható módon a következő tartalommal bírnak:

$\mathcal{M}^{aa}(i)$ halmaz elemei az i modultól balra helyezkednek el
$\mathcal{M}^{bb}(i)$ halmaz elemei az i modultól jobbra helyezkednek el
$\mathcal{M}^{ab}(i)$ halmaz elemei az i modultól lefele helyezkednek el
$\mathcal{M}^{ba}(i)$ halmaz elemei az i modultól felfele helyezkednek el



4.14. ábra. A helyzetvonalak geometriai jelentése

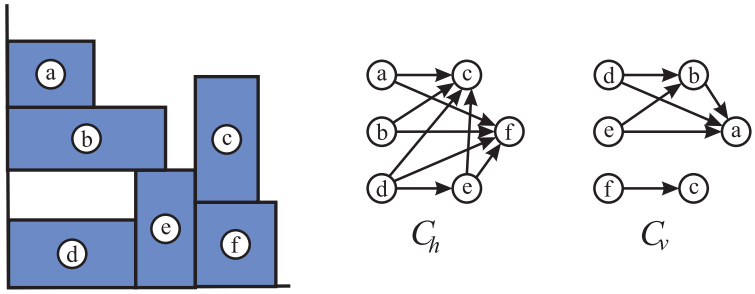
Készítsük el a G_h és G_v gráfokat a következőképpen. Mindkét gráfban minden modulnak megfelel egy csúcs, valamint mindkét gráfhoz hozzáveszünk egy forrás- és egy nyelőpontot. G_h-ban minden i modulnak megfelelő pontba vezet él az összes olyan modulnak megfelelő pontból, mely eleme $\mathcal{M}^{aa}(i)-$nek. G_v-ben minden i modulnak megfelelő pontba vezet él az összes olyan modulnak megfelelő pontból, mely eleme $\mathcal{M}^{ab}(i)$-nek. A forrásból minden pontba, a nyelőbe minden pontból vezet él. Az így kapott G_h és G_v gráf a pakoláshoz tartozó horizontális, illetve vertikális kényszergráf tranzitív lezártja. Így G_h-ban a leghosszabb utak a pakolás moduljainak x-koordinátáit, G_v-ben az y-koordinátáit adják. A nyelőbe vezető leghosszabb utak a ház szélességének és magasságának felelnek meg.
A leghosszabb utak algoritmusa $O(n^2)$ nagyságrendű, így a pakolás előállítása a modulsorozat-párból ennyi időt vesz igénybe.

4.3.4. Tranzitív lezárt gráfok (Transitive Closure Graphs)

17. Definíció. Egy $D = (V, E)$ irányított gráf ***tranzitív lezártján*** azt a $D' = (V, E)$ irányított gráfot értjük, melyben minden olyan a és b csúcs között vezet él, melyek között vezet irányított út D-ben.

Egy pakolás a moduljára azt mondjuk, hogy horizontális (vertikális) relációban van a b modullal, ha az a modul a b-től balra (lejjebb) helyezkedik el, és az y-tengelyre (x-tengelyre) vett vetületük átfedő. Azt mondjuk, hogy az a modul diagonális relációban van a b modullal, ha az a modul a b-től balra helyezkedik el, és a velütelük az y-tengelyre nem átfedő. A diagonális relációt horizontálisnak fogjuk tekinteni, kivéve akkor, ha létezik vertikális relációk egy sorozata a diagonális relációban álló modulok között. Ekkor a diagonális relációt vertikálisnak tekintjük. [12]

Ezek alapján konstruáljuk meg a pakoláshoz tartozó horizontális (C_h) és vertikális (C_v) tranzitív lezárt gráfot. Mindkét gráfban minden modulnak megfelel egy csúcs, és egy a csúcsból a b csúcsba akkor vezet él C_h-ban (C_v-ben), ha az a modul horizontális (vertikális) relációban áll a b-modullal.
A tranzitv lezárt gráfokból a pakolást a leghosszabb-út algoritmussal kap-



hatjuk meg, az x-koordinátákat a C_h-beli, az y-koordinátákat a C_v-beli leghosszabb utak eredményezik. Így a pakolás előállítása $O(n^2)$ időt vesz igénybe.
A tranzitív lezárt gráfok és a modulsorozat-pár előnyös tulajdonságait kombinálja a TCG-S reprezentáció. [13]

Reprezentáció	Tároláshoz szükséges bitek száma	Különböző reprezentációk száma	Dekódolás futásideje	Pakolás típusa	P-megengedett
ST/PE	-/$2n-1+n\lceil\log n+2\rceil$	$O(n!\,2^{3n}/n^{1.5})$	$O(n)$	szétvágható	nem
NPE	$2n-1\lceil\log n+2\rceil$	$O(n!\,2^{3n}/n^{1.5})$	$O(n)$	szétvágható	nem
GPE	$2n-1+n\lceil\log n+3\rceil$	$O(n!\,2^{2n}3^{n-1}/n^{1.5})$	N/A	általános	nem
CBL	$3(n-1)+n\lceil\log n\rceil$	$O(n!\,2^{3n}/n^{1.5})$	$O(n)$	mozaik	nem
ECBL	N/A	$O(C^n_{\lceil\lambda n\rceil}n!\,3^{3\lceil\lambda n\rceil-4}/\lceil\lambda n\rceil^{1.5})$	$O(\lceil\lambda n\rceil)$	általános	nem
QS	N/A	$O(n!\,2^{3n}/n^{1.5})$	$O(n)$	mozaik	nem
EQS	N/A	$O(2^{6n}(2n)!\,/n!)$	$O(n)$	általános	N/A
TBS	N/A	$O(n!\,2^{3n}/n^{1.5})$	$O(n)$	mozaik	nem
OT	$2n+n\lceil\log n\rceil$	$O(n!\,2^{2n}/n^{1.5})$	$O(n)$	LB-kompakt	nem
BT	N/A	$O(n!\,2^{2n}/n^{1.5})$	$O(n)$	LB-kompakt	nem
SP	$2n\lceil\log n\rceil$	$(n!)^2$	$O(n^2)$	általános	igen
BSG	N/A	$O(n!\,C(n^2,n))$	$O(n^2)$	általános	igen
TCG	N/A	$(n!)^2$	$O(n^2)$	általános	igen
TCG-S	N/A	$(n!)^2$	$O(n\log n)$	általános	igen

4.1. táblázat. Reprezentációk összehasonlító táblázata (az adatok egy része [14]-ből származik)

5. fejezet

Algoritmusok

5.1. Mohó algoritmus

A mohó algoritmus minden lépésben egy már meglevő pakoláshoz ad hozzá egy új modult a következőképp. Kezdetben az üres pakolásból indulunk ki, és ebbe illesztjük be az első modult a lehető legoptimálisabban. Ez minden esetben a (0,0) koordinátán való beillesztést jelenti. Ezután minden lépésben az adott reprezentáció által biztosított beillesztési pontok mindegyikén beillesztjük a következő modult, és a lehetőségek közül a legjobbat megtartjuk. Arra számítunk, hogy más-más sorrendben beillesztve a modulokat különböző jóságú megoldásokat kapunk. A későbbiekben látni fogjuk, hogy ez tényleg így is van, és hogy a teszteredmények alapján mely beillesztési sorrendek vezettek általában optimálisabb megoldásokhoz. A mohó algoritmus pszeodokódja a 9. algoritmus. Az algoritmusban terület szerinti minimalizálás helyett választhatunk kerület szerinti minimalizálást is. Az első esetben az eredmény alakja olyan téglalap lesz, amelynek az egy oldala jóval hosszabb, mint a másik, míg kerület szerinti minimalizálás esetén közel négyzetes.

Látható, hogy az algoritmus használhatóságát nagyban befolyásolja, hogy az adott reprezentáció beillesztési pontjait mennyire jól tudjuk meghatározni, illetve a beillesztési pontok száma befolyásolja a futásidőt. A következőkben ezzel foglalkozunk.

Normált lengyel kifejezés

Egy üres normált lengyel kifejezésbe egyféleképpen tudunk beilleszteni egy modulcímkét. Ha a kifejezés legalább egy elemet tartalmaz, akkor egy modulcímke és egy operátor beillesztésére van szükség. A modulcímkét a kifejezés bármelyik pontjába beszúrhatjuk. Ha a kifejezés $2k-1$ elemű (ekkor k modulcímkét tartalmaz), akkor ez $2k$ beillesztési pontot jelent.

Algorithm 9 Greedy(modules, moduleOrder)

```
Input: modules[0..n-1] - array of modules, with width and height
       moduleOrder[0..n-1] - array of labels of modules
Output: placement of n modules
  for all i moduleOrder do
    minArea ← ∞;
    for all insertationPoint in representation do
      representation.Insert(insertationPoint, modules[i]);
      placement ← representation.ToPlacement;
      if placement.Area < minArea then
        minArea ← placement.Area;
      end if
      representation.Remove(insertationPoint, modules[i]);
    end for
  end for
  return placement
```

Egy operátor beillesztésénél le kell ellenőrizni, hogy megmarad-e a ballot tulajdonság. A megmaradást feltéve, két modulcímke közé horizontális illetve vertikális operátort is beilleszthetünk, egy modulcímke és egy operátor közé csak az operátortól különböző operátort. Két operátor közé nem lehet operátort beilleszteni, hiszen ekkor akár horizontális, akár vertikális operátort illesztenénk be, az az egyik szomszédjával megegyezne, és így a kapott kifejezés már nem lenne normált.

O-fa

Egy k modult tartalmazó, tehát $k+1$ csúcsú O-fának csak a külső pontjaiba tudunk egyszerűen egy új modulnak megfelelő csúcsot beilleszteni. Ez a művelet megfelel annak, hogy a $2k$ hosszú T-be a lehetséges $2k+1$ hely valamelyikére beszúrunk egy 01-et, és az ennek megfelelő helyre Π-be beszúrjuk a következő modul címkéjét. Mivel T-ben a beszúrt 01-et megelőző 0-k száma a beszúrt modul előtt bejárandó csúcsok számát jelzi, az új modult Π-ben pontosan annyi modulnak kell megelőznie, ahány 0 volt a beszúrt 01 előtt T-ben. Az algoritmus során a beillesztési pontok száma

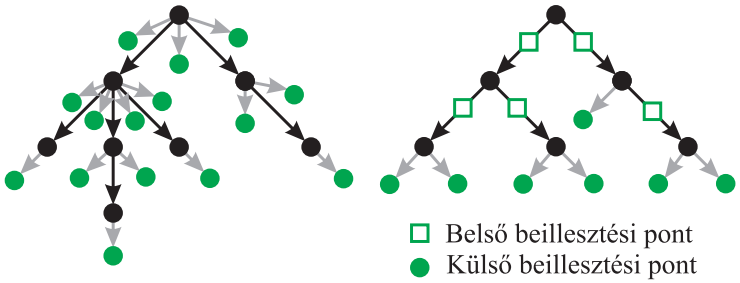
$$\sum_{k=0}^{n-1} 2k+1$$

Egy $k+1$ modulhoz tartozó O-fához tartozó pakolás $O(k+1)$ idő alatt konstruálható meg, így az algoritmus futásideje

$$O\left(\sum_{k=0}^{n-1}(2k+1)(k+1)\right) = O(n^3)$$

B*-fa

B*-fa esetén az O-fától eltérően belső pozíciókba is tudunk úgy csúcsot beszúrni a reprezentáció megvalósítása, azaz a bináris fa struktúra miatt. A beszúrási pozíciók az 5.1. ábrán láthatóak.



5.1. ábra. Beillesztési pontok O-fa, illetve B*-fa esetén

Modulsorozat-pár

Egy k modulból álló modulsorozat-pár mindkét tagjába $k+1$ helyre illeszthetjük be az új modult. Az első beillesztés az üres pakolásba, az utolsó az $n-1$ modulból álló pakolásba történik, így az algoritmus során a beillesztési pontok száma:

$$\sum_{k=0}^{n-1}(k+1)^2$$

Mivel egy $k+1$ modulhoz tartozó modulsorozat-párhoz tartozó pakolás $O((k+1)^2)$ idő alatt konstruálható meg, így az algoritmus futásideje:

$$O\left(\sum_{k=0}^{n-1}(k+1)^4\right) = O(n^5)$$

5.2. Lokális keresés és rokon algoritmusai

Az ilyen típusú algoritmusok mindig a keresési tér egy pontjából indulnak - esetünkben egy n pontú pakolást leíró reprezentáció egy érvényes kódjából -, és valamilyen szisztéma szerint átlépnek egy szomszédos pontra. Ezért mindenek előtt definiálnunk kell, hogy a különböző reprezentációk esetén mit értünk szomszédosság alatt.

Normált lengyel kifejezés

18. Definíció. Operátorok egy sorozatát ***lánc***nak (*chain*) nevezzük, ha benne minden szomszédos elemek különböző.

Világos, hogy egy lánc csak $+*+*+*\ldots$ vagy $*+*+*+\ldots$ alakú lehet. Egy lánc komplementere alatt azt a láncot értjük, melyet az eredeti láncból az a $+$ operátor $*$-ra, és a $*$ operátor $+$-ra cserélésével nyerünk.
Definiáljunk három műveletet.

M1. ψ-ban két modulcímkét megcserélünk.
M2. ψ-ban egy láncot komplementálunk.
M3. ψ-ban egy operátort és modulcímkét megcserélünk.

Az M1 és M2 művelet elvégzése biztosan normált lengyel kifejezést eredményez, míg M3 elvégzése során előfordulhat, hogy a kapott sorozat vagy megegyező szomszédos operátorokat tartalmaz, vagy nem ballot sorozat.
Két, n modulhoz tartozó normált lengyel kifejezést szomszédosnak tekintünk, ha az érvényes M1, M2 vagy M3 műveletek valamelyikével egymásba vihetők. Belátható az is, hogy ezen műveletek alkalmazásával bármely, n modulhoz tartozó normált lengyel kifejezésből bármely másikba el lehet jutni.

Általánosított lengyel kifejezés

Az általánosított lengyel kifejezés szomszédait a normált lengyel kifejezés szomszédaival teljesen megegyező módon hozzuk létre. Az M1, M2, M3 művelet megegyezik az előbb definiáltakkal, annyi különbség van csupán, hogy itt a láncok háromféle operátort tartalmazhatnak, ezért a komplementálás műveletét felül kell definiálnunk. Általánosított lengyel kifejezés esetén egy lánc komplementerén azt a láncot értjük, melyben minden operátort lecserélünk egy vele nem megegyezőre.

Sarokszoba-lista

Sarokszoba-lista esetén a szomszédossági műveletek:

M1. Két elem cseréje S-ben, azaz két modulcímke cseréje.
M2. L-ben egy 0 elem 1-re cserélése vagy fordítva, azaz egy modul orientációjának cseréje.
M3. Egy 1 elem beszúrása T-be, vagy egy 1 elem eltávolítása T-ből, azaz a T-érintkezések számának megváltoztatása.

Az M3 művelet esetében beszúráskor a művelet érvényességét a normált lengyel kifejezés műveletérvényesség-ellenőrzésétől eltérően csak $O(n)$ időben tudjuk végrehajtani, ugyanis az érvényesség ekkor nem csupán a megelőző tagoktól függ, hanem a későbbiek érvényben maradását is befolyásolja.

O-fa

O-fa esetén a szomszédokat úgy kapjuk, hogy egy csúcsot kitörlünk, majd a mohó algoritmusnál ismeretetett beillesztései pontok egyikében beillesztjük.

B*-fa

B*-fa esetén két szomszédossági műveletet definiálunk:

M1. Két csúcs cseréje.
M2. Egy csúcs törlése, majd beillesztése a mohó algoritmusnál ismertetett módon.

Itt törlés esetén a törlendő a csúcs a gyerekek száma szerint háromféle lehet.

(i) Az a csúcs levél. Ekkor az a csúcsot egyszerűen töröljük
(ii) Az a csúcsnak egy gyereke van. Ekkor az a csúcsot töröljük, és a gyerekét az a csúcs szülőjéhez kapcsoljuk bal illetve jobb gyerekként, annak megfelelően, hogy az a bal vagy jobb gyerek volt.
(iii) Az a csúcsnak két gyereke van. Ekkor kiválasztjuk az egyik gyerekét, és az a címkéjét, valamint ennek a gyereknek a címkéjét felcseréljük, és az így előállt (i)-(iii) helyzetnek megfelelően töröljük az a csúcsot.

Modulsorozat-pár

Modulsorozat-pár esetén a szomszédossági műveletek:

M1. Két modulcímke cseréje Γ_+-ban vagy Γ_--ban.
M2. Két modulcímke cseréje Γ_+-ban és Γ_--ban is.

5.2.1. Lokális keresés

A lokális keresés a keresési tér egy pontjából indulva végignézi a pont szomszédait, és ha talál köztük jobbat, átlép rá. Az átlépés történhet azonnal, ahogy jobb szomszédot talált, vagy az összes szomszéd átnézése után a legjobb szomszédot választva. Ha a keresési tér pontjainak nagy számú szomszédjuk van, akkor az összes szomszéd helyett vizsgálhatjuk a szomszédoknak csak egy halmazát. Az algoritmus leáll, ha egy olyan pontra lép, melynek már nem létezik jobb szomszédja.
A 10. algoritmus egy, az összes szomszédot megvizsgáló, és ezek közül a legjobbra átlépő lokális keresés algoritmusa, melyet a keresési tér egy véletlen pontjából indítunk.

Algorithm 10 LocalSearch(modules)

```
Input: modules[0..n-1] - array of modules, with width and height;
Output: placement of n modules
  issue ← random issue of Representation for n modules;
  existsBetter ↔ true
  placement ← Representation.ToPlacement(modules, issue)
  while existsBetter do
    existsBetter ← false
    for all iss in issue.neighbors do
      pl ← ToPlacement(modules, iss)
      if pl.Area < placement.Area then
        placement ← pl;
        nextIssue ← iss;
        existBetter ← true;
      end if
    end for
    issue ← nextIssue;
  end while
  return placement
```

5.2.2. Tabu keresés

A tabu keresés a lokális kereséshez hasonlóan a keresési tér egy véletlen pontjából indul. Ugyanúgy átléphet az összes vagy néhány véletlenszerűen választott szomszéd közül a legjobb szomszédra, vagy az összes közül az első jobbra. A különbség, hogy tárol egy tabu listát, amely a már megtett lépéseket tartalmazza, és ezekre nem lép többé vissza. Valamint, ha nem talál

Algorithm 11 TabuSearch(modules)

```
Input: modules[0..n-1] - array of modules, with width and height;
Output: placement of n modules
  issue ← random issue of actual representation for n modules;
  i ↔ k * n
  tabuList ← empty list
  while i > 0 do
    worseArea ← ∞
    placement ← Representation.ToPlacement(modules, issue)
    for all iss in issue.neighbors do
      pl ← ToPlacement(modules, iss)
      if pl.Area < placement.Area and pl.Area is not tabu then
        placement ← pl;
        nextIssue ← iss;
        tabuArea ← pl.Area
        i ← i+1
        break
      else
        if pl.Area < worseArea then
          nextIssue ← iss;
          worseArea ← pl.Area
          tabuArea ← pl.Area
        end if
      end if
    end for
    i ← i-1
    issue ← nextIssue;
    add tabuArea to tabuList
  end while
  return placement
```

jobb szomszédot, akkor a lehetséges rosszabb szomszédok közül a legkevésbé rosszra lép át. Ha valamelyik tabu lépés már régen történt, azt a listából törölni lehet. Az algoritmusnak sokféle leállási feltételt adhatunk, ezek közül az egyik legegyszerűbb az iterációk számának korlátozása.
Esetünkben a tabu lista a már megtalált területeket tartalmazza, ilyen területű szomszédra nem lehet átlépni. A tabu listából sose törlünk, és a szomszédok közül mindig az első legjobbra lépünk át, ha van ilyen, ha pedig nincs akkor a legkisebb területűre a rosszabbak közül. Az algoritmus leáll, ha már a k-szor a modulok számának megfelelő rosszabb lépést tett. Normált lengyel kifejezés esetén k-t 5-nek, sarokszoba lista esetén 10-nek, O-fa, B*-fa és modulsorozat-pár esetén 1-nek választottam.

5.2.3. Szimulált hűtés

A szimulált lehűtés általános algoritmusához a keresési tér és a szomszédossági struktúra megatározásán kívül a következő adatokra van szükség:

- T_0 a kezdeti hőmérséklet
- T_{frozen} a leállási hőmérséklet
- $E(x)$ a rendszer energiáját (esetünkben a keresési tér adott pontjához tartozó megoldás jóságát) kiszámító függvény
- $f(x)$ a hőmérséklet hűlését meghatározó függvény
- k az egy hőmérsékleten elvégzendő iterációk száma
- b Boltzmann konstans, melynek az értékét úgy választjuk meg, hogy a kezdeti hőmérsékleten a rosszabb megoldások elfogadásának valószínűsége közel 1 legyen

Az általános algoritmust a 12. pszeudokód írja le. A leállási feltétel kiegészíthető időkorláttal, vagy például azzal, hogy találtunk-e olyan megoldást, ami az optimálistól csak kis százalékban tér el. A `for` ciklus valahány, a szomszédok számától függő iterációt végez ugyanazon a hőmérsékleten, ezután a hőmérséklet csökkentése következik. Látható, hogy egy véletlenül választott szomszédot elfogadunk, ha jobb, vagy bizonyos valószínűséggel elfogadjuk, ha rosszabb. Az algoritmus megjegyzi a legjobb megtalált eredményt, és ezzel tér vissza.
Az algoritmus megvalósításakor a következő értékeket választottam:

- T_0 40000
- T_{frozen} 0.1 sarokszoba-lista, normált lengyel kifejezés, O-fa és B*-fa esetén, 500 modulsorozat-pár esetén

Algorithm 12 SimulatedAnnealingGeneral

$P \leftarrow$ initial point of solution space
$P_{best} \leftarrow P$
$T \leftarrow T_0$
while $T > T_{frozen}$ **do**
  for $i = 0$ **to** k **do**
    $Q \leftarrow$ random neighbor of P
    $\Delta E \leftarrow E(Q) - E(P)$
    if $\Delta E < 0$ **or** $e^{\frac{-\Delta E}{bT}} > random(0..1)$ **then**
      $P \leftarrow Q$
    end if
    if $E(P) < E(P_{best})$ **then**
      $P_{best} \leftarrow P$
    end if
  end for
  $T \leftarrow f(T)$
end while
return P_{best}

$E(x)$ $A_{act} - A_{next} + P_{act} - P_{next}$, ahol A_{act} és P_{act} az aktuális kód által leírt pakoláshoz tartozó ház területe illetve kerülete, A_{next} és P_{next} az épp kiértékelt random szomszéd megfelelő értékei

$f(x)$ $0.99 * x$ (lineáris hűtés)

k 1000 sarokszoba-lista, normált lengyel kifejezés, O-fa és B*-fa esetén, 500 modulsorozat-pár esetén

b 99%−os elfogadási valószínűséget elvárva, $b = \frac{-\Delta}{40000 * \ln 0.99}$, ahol Δ a kiindulási kód 2-szer a modulok száma darab random szomszédja közül a nála nagyobb energiájúaknak megfelelő energianövekedések átlaga.

6. fejezet

Megvalósítás és eredmények

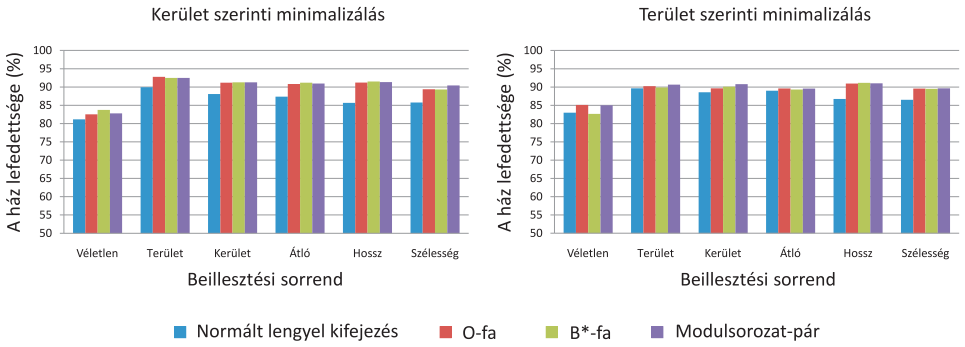
Az ismertetett algoritmusok mindegyike, valamint a reprezentációk közül a normált lengyel kifejezés, a sarokszoba-lista, az O-fa, a B*-fa, és a modulsorozat pár lett megvalósítva. A sarokszoba-listához nem sikerült könnyen kezelhető beillesztési pontokat találni, ezért ehhez a reprezentációhoz nem választható a mohó algoritmus. Az eddig elhangzottak illusztrálására készült program C# nyelven .NET keretrendszer segítségével íródott. A fejlesztésre és tesztelésre használt gép egy Intel Core i3-330M-es, mely 4GB memóriával rendelkezik.
Tíztől száz modulig minden tízzel osztható modulszámra az összes lehetséges algoritmus-reprezentáció-egyéb opciók lehetőséget háromszor terveztem tesztelni véletlen generált modulokra (egyenletes eloszlású oldalhosszakkal), de az ötven modulhoz tartozó tesztek befejezése után a lokális keresésre és a tabu keresésre vonatkozó tesztelést abbahagytam a kevéssé jó lefedettségi arányok, vagy az algoritmus lassúsága (esetleg mindkettő) miatt. Hasonló okok miatt nincsenek teszteredmények 80 modultól a szimulált hűtés esetén O-fára, B*-fára és modulsorozat-párra.

6.1. Összehasonlítás

Mohó algoritmus

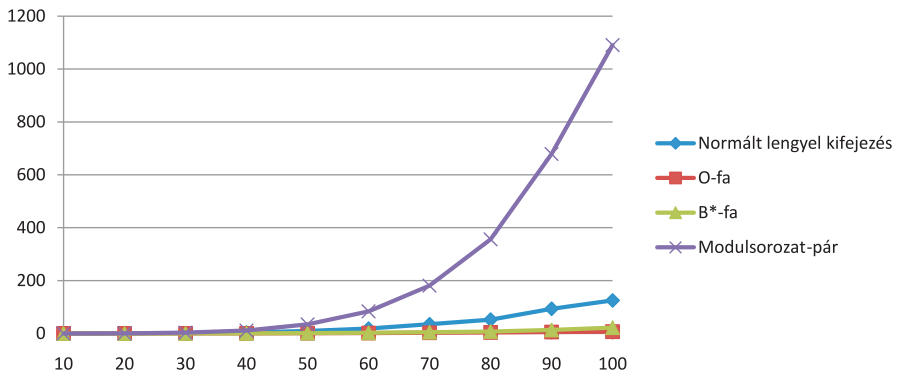
Mohó algoritmus esetén a választható beillesztési sorrendek a véletlen, valamint a terület, kerület, átló, szélesség valamint magasság szerinti nem csökkenő sorrendek lettek. Minden esetben választható terület illetve kerület szerinti minimalizálás. A különböző beillesztési sorrendeknek illetve minimalizálásnak megfelelő ház-lefedettségi százalékok a 6.1. ábrán láthatóak. Ennél az összehasonlításnál nem lett figyelembe véve a modulok száma, bár ennek növelésével a kihasználtság - akár az optimális megoldások esetén -

növekszik. Ha a különböző reprezentációkat tekintjük, nem meglepő, hogy a



6.1. ábra. A mohó algoritmus területkihasználási adatai

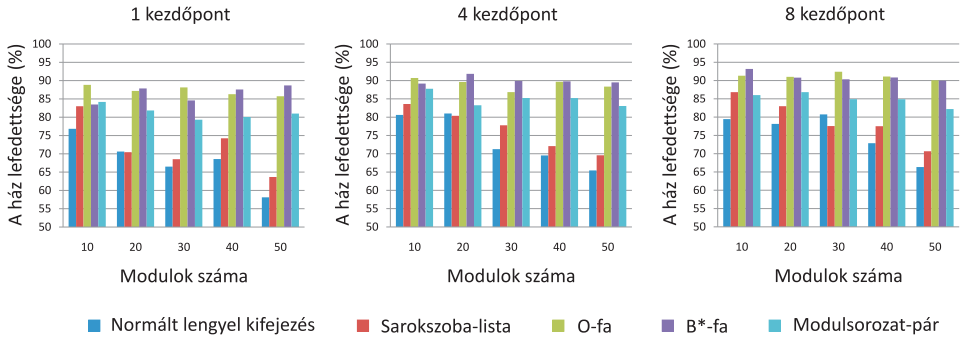
normált lengyel kifejezéssel rosszabb eredményre jutunk, hiszen ez csak szétvágható struktúrákat tud kezelni, míg a többi általánosat vagy legalábbis LB-kompaktat. Érdekesebb kérdés a beillesztési sorrend szerinti különbség. Tekintsük innentől a 90% körüli fedettségi arányokat jónak. A grafikon alapján a véletlen beillesztés elvethető. A többi lehetőség között az átlagos adatok alapján nem látszik számottevő különbség, azonban a mindkét oldalhosszat egyaránt figyelembe vevő beillesztási sorrendek (terület, kerület és átló) megfelelőbbek.
A futásidő kérdését tekintve az előző fejezetekben ismertetett nagyságrendek alapján várt eredményeket kapjuk. Ott bővebben nem lett tárgyalva a mohó algoritmus nagyságrendje normált lengyel kifejezés esetén, de kiszámítható, hogy mivel a normált lengyel kifejezésből a pakolás előállítása $O(n)$ időt vesz igénybe, az algoritmus nagyságrendje $O(n^4)$. A futásidőre vonatkozó teszteredményeket a 6.2. ábra grafikonja szemlélteti.



6.2. ábra. Mohó algoritmus futásidő

Lokális keresés

Mivel ez az algoritmus csak lokális optimumokat talál meg, és onnan nem vizsgálódik tovább, nem meglepő, hogy nem kapunk a használatával kiemelkedően jó eredményeket. A lefedettségi adatok a 6.3. ábra diagramjain láthatóak. Az algoritmus választható opciója a kezdőpontok száma. A több



6.3. ábra. A lokális keresés területkihasználási adatai

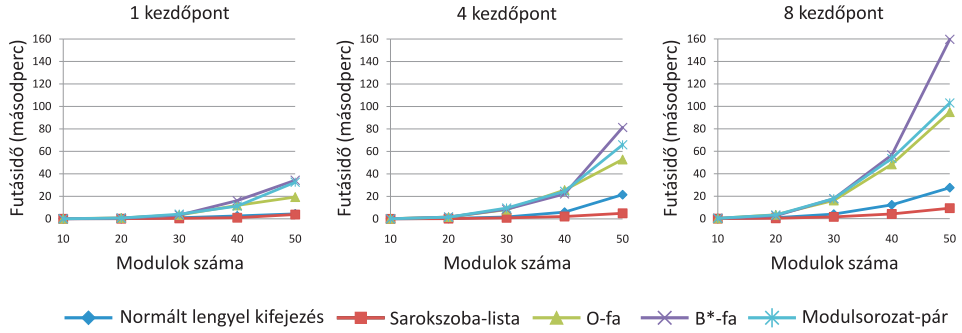
pontból indított lokális keresés könnyen párhuzamosítható, így a megvalósításban több kezdőpont esetén ez a megoldás szerepel. A tesztelésre használt gép processzora két magos, de mindkét mag virtuálisan ketté van osztva, így a gép egyszerre négy szálat tud futtatni. Így a négy pontból indított lokális kesesés futási kevesebb lesz, mintha egy szálon futtatnánk egymás után a négy keresést, de nem ennek a futásidőnek a negyede. Mivel az algoritmus leállása a lokális optimum megtalálásához, nem pedig valamilyen iterációszámhoz kötött, a futásideje – a reprezentáció dekódolásának idején kívül – attól függ, hogy milyen hosszú, jó szomszédokból álló utak vezetnek ki a kiindulási pontból. A futásidők a 6.4. ábrán láthatóak. Megfigyelhető, hogy például az $O(n)$ dekódolási idejű B*-fára alkalmazott lokális keresés hosszabb ideig fut, mint az $O(n^2)$-es modulsorozat-párra alkalmazott.

Tabu keresés

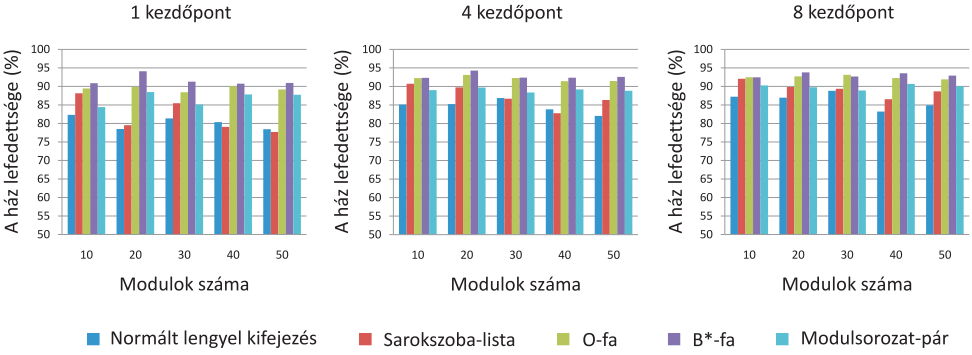
A tabu kesesés futásidejére és párhuzamosíthatóságára a lokális keresésnél elmondottak érvényesek. A konstrukcióból adódóan nem meglepő, hogy a tabu keresés megjavítja a lokális kereséssel kapott lefedettségi adatokat, ám annál jóval lassabb. A kapott eredményeket a 6.6. és a 6.5. ábra tartalmazza.

Szimulált hűtés

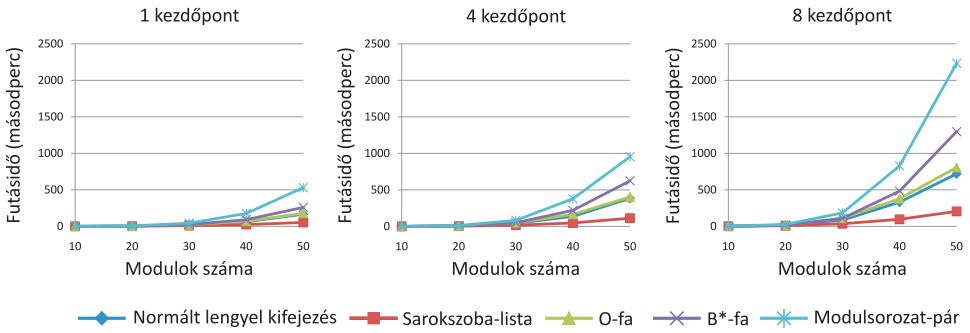
A szimulált hűtés az idézett cikkekben leggyakrabban javasolt megoldás volt. Meg kell említeni, hogy ezek mindegyikében engedélyezve volt a modu-



6.4. ábra. Lokális keresés futásidők

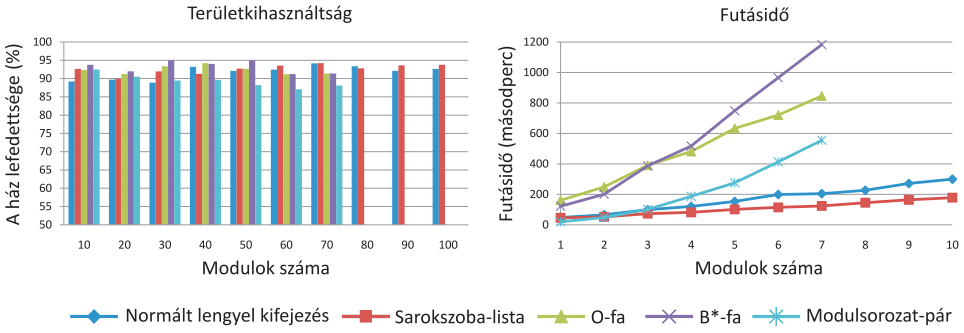


6.5. ábra. A tabu keresés területkihasználási adatai



6.6. ábra. Tabu keresés futásidők

lok forgatása mint szomszédossági művelet, Sprite képek esetében azonban nem lehet a kisebb képeket elforgatni, így a forgatást az algoritmusok során nem engedtem meg. Az előző fejezetben ismertetett választásokkal a 6.7. ábrán látható eredmények adódtak. Az algoritmus ugyan nem túl gyors, de elég jó eredményeket ad.



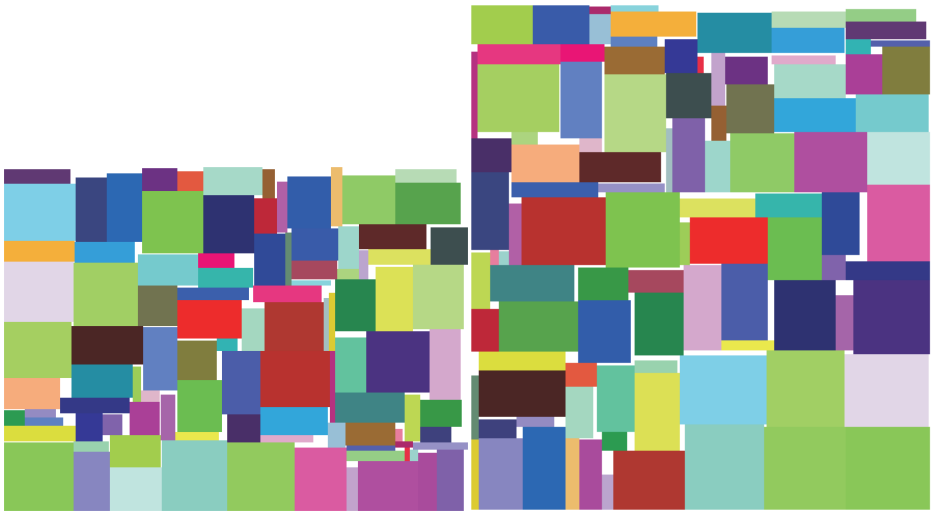
6.7. ábra. Szimulált hűtés adatok

6.2. Néhány pakolás

Most lássunk néhány, a programmal kapott konkrét pakolást és a hozzájuk tartozó adatokat. Elsőként a 6.8. és a 6.9. ábrán a mohó algoritmus által 100 modulra adott három eredmény látható, melyeken jól megfigyelhető a kerület és terület szerinti minimalizálás közti különbség.

A 6.10. ábrán két, a szimulált hűtéssel kapott pakolás látható az előzőekkel megegyező a 100 modulra.

Végül érdekességképp következzen egy, a 6.11. ábrán látható 500 modulból álló pakolás, melyet a mohó algoritmus adott eredményül O-fa reprezentáció esetén.



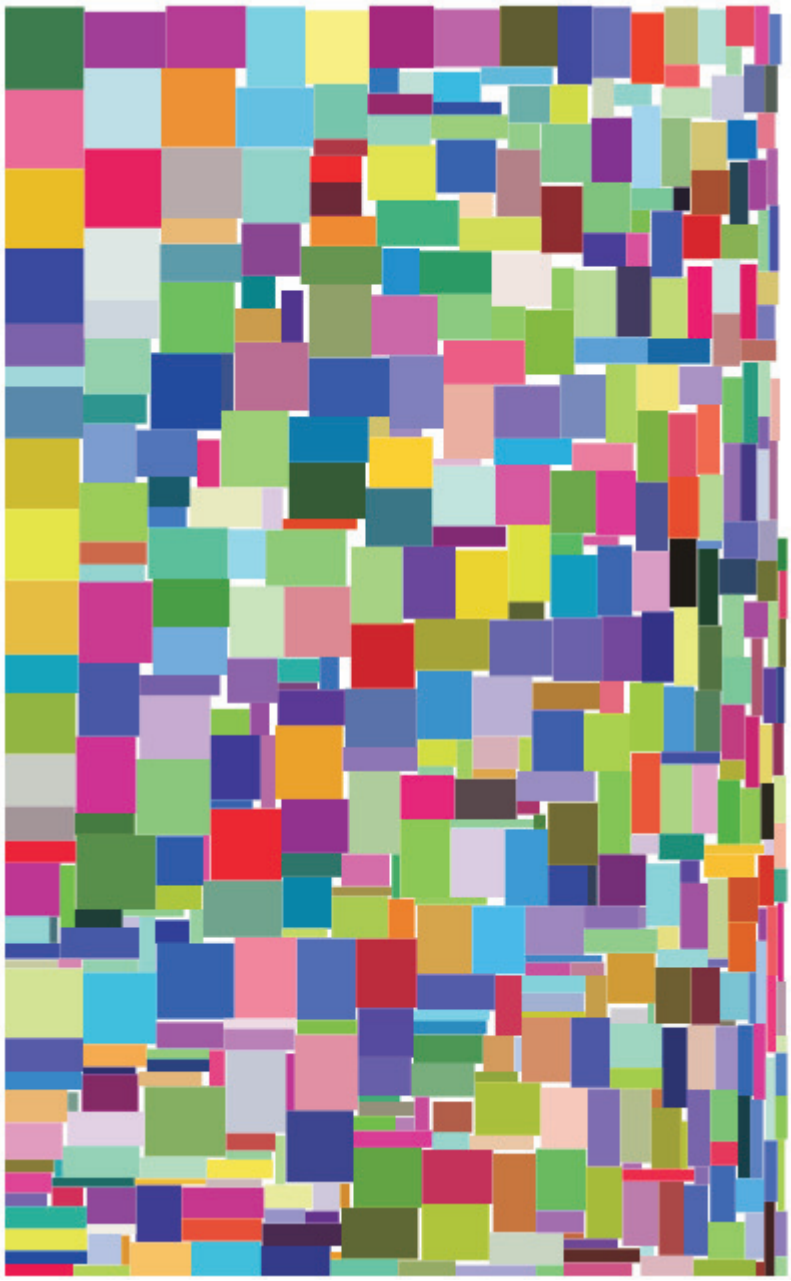
(a) B*-fa, 148s, 94.99% (b) Modulsorozat-pár, 1131s, 95.78%

6.8. ábra. Mohó algoritmus, kerület szerinti minimalizálás

6.9. ábra. Mohó algoritmus, terület szerinti minimalizálás, B*-fa, 151s, 93.95% (kicsinyített kép)

(a) Normált lengyel kifejezés, 316s, 90.78% (b) Sarokszoba-lista, 177s, 94.04%

6.10. ábra. Szimulált hűtés



6.11. ábra. Mohó algoritmus, kerület szerinti minimalizálás, O-fa, 1026s, 96,63%

7. fejezet

Összefoglalás

Láthattuk, hogy pakolások reprezentációjára sok megoldás született az évek során. Még megemlítünk kettőt, a Yao, Chen, Cheng és Graham által bevezetett iker bináris fákat (*twin binary trees*) [7], illetve ennek általánosítása Young, Chu és Shen által, az iker bináris sorozatokat (*twin binary sequences*) [8].
A Reprezentációk fejezet végén található a 4.1. táblázat adataiból is látszik, hogy a bevezetőben említett probléma megoldására nem valószínű, hogy a modulsorozat-pár lesz a legjobb választás, hiszen találhatunk olyan reprezentációt is, melynek dekódolási ideje pusztán lineáris a modulok számában. El kell mondani azonban, hogy a modulsorozat-párnak (és a többi P-megengedett reprezentációnak) sok előnye van a többivel szemben. Ezek jól kezelik például az előre meghatározott helyre teendő modulokat, amire sok probléma megoldása esetén szükség van.
A tesztek alapján B*-fát és az O-fát használva mohó algoritmussal és kerület szerinti minimalizálással, terület vagy kerület szerinti beillesztési sorrendben rövid idő alatt kifejezetten jó megoldásokat kapunk a problémára. Mivel úgy gondolom, hogy ez a dolgozat akkor válik teljessé, ha az elején felvetett Sprite kép feladatra gyakorlati megoldás is születik, így érdekességképpen erre is készítettem egy programot, melynek megvalósításához az előbb említettek közül az O-fát választottam. A mohó algoritmus során az elemek terület szerinti nem csökkenő sorrendben kerülnek beillesztésre. A program előállítja a Sprite kép használatához szükséges CSS-fájlt is.
A http://people.inf.elte.hu/kacpaat/ weboldalról a dolgozathoz készített programok letölthetőek.

Irodalomjegyzék

[1] D. F. Wong, C. L. Liu: *A new algorithm for floorplan design*, Proc. 23rd ACM/IEEE Design Automation Conf. (1986)

[2] Chang-Tzu Lin, De-Sheng Chen, Yi-Wen Wang: *GPE: A New Representation for VLSI Floorplan Problem*, IEEE International Conference on Computer Design(2002)

[3] Xianlong Hong, Gang Huang, Yici Cai, Jiangchun Gu, Sheqin Dong, Chung-kuan Cheng, Jun Gu: *Corner block list: An effective and efficient topological representation of non-slicing floorplan*, ICCAD (2000)

[4] Shuo Zhou, Sheqin Dong, Xianlong Hong, Yici Cai, Chung-kuan Cheng, Jun Gu: *ECBL: An Extended Corner Block List with Solution Space including Optimum Placement*, ISPD (2001)

[5] Keishi Sakanushi, Yoji Kajitani: *The quarter-state sequence (Q-sequence) to represent the floorplan and applications to layout optimization*, IEEE APCCAS (2000)

[6] Changwen Zhuang, Keishi Sakanushi, Liyan Jin, Yoji Kajitani: *An Enhanced Q-Sequence Augmented with Empty-Room-Insertion and Parenthesis Trees*, DATE (2002)

[7] Bo Yao, Hongyu Chen, Chung-Kuan Cheng, Ronald Graham: *Revisiting Floorplan Representations*

[8] Evangeline F. Y. Young, Chris C. N. Chu, and Zion Cien Shen: *Twin Binary Sequences: A Nonredundant Representation for General Nonslicing Floorplan*, IEEE TCAD (2002)

[9] Pei-Ning Guo, Toshihiko Takahashi, Chung-Kuan Cheng, Takeshi Yoshimura: *Floorplanning Using a Tree Representation*, IEEE Transaction On Computer-aided Design of Integrated Circuits and System, Vol. 20, No. 2 (2001)

[10] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, Shu-Wei Wu: *B*-Trees: A New Representation for Non-Slicing Floorplans*, Proc. DAC (2000)

[11] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani: *Rectangle-Packing-Based Module Placement*, ICCAD (1995)

[12] Jai-Ming Lin, Yao-Wen Chang: *TCG: a transitive closure graph-based representation for non-slicing floorplans*, DAC (2001)

[13] Jai-Ming Lin, Yao-Wen Chang: *TCG-S: Orthogonal Coupling of P*-admissible Representations for General Floorplans*, DAC (2002)

[14] Tung-Chieh Chen, Yao-Wen Chang: *Floorplanning (Electronic Design Automation. Synthesis, Verification, and Test. Chapter 10)*, Elsevier Science and Technology (2009)