Óbudai Egyetem
Neumann János Informatikai Kar
Alkalmazott Informatikai Intézet

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

ASZTALOSIPARI SZOFTVER TEZÉSE ÉS FEJLESZTÉSE LAPSZABÁSMINTA OPTIMALIZÁLÁS FUNKCIONALITÁSSAL

Szerző: **Hodák Bence**
mérnök informatikus, BSc IV. évfolyam

Konzulens: **Balázs Elemér**
egyetemi gyakornok

TARTALOMJEGYZÉK

Bevezetés 4
1. Lapszabászat 5
1.1. Lapszabász gépek 5
1.2. Szabásminta 6
2. Hasonló szoftverek 7
2.1. BIPPO 7
2.2. PaneCutter 8
2.3. AMORF 10
2.4. Optimik 4 11
2.5. Szoftverek összehasonlítása 13
3. Optimalizáló algoritmusok 15
3.1. 2-fázisú Guillotine vágás 16
3.2. 3-fázisú Guillotine vágás 17
3.3. Sarokba helyezés 18
3.4. Sík felosztása 19
3.5. Heurisztikus megoldások 20
3.5.1. BL (Bottom-Left) algoritmus 20
3.5.2. BLF (Bottom-Left-Fill) algoritmus 21
3.6. Módszerek áttekintése 21
4. Irodalomkutatás áttekintése 22
5. Rendszerterv 23
5.1. Backend 23
5.1.1. Alkalmazás 23
5.1.2. Adatbázis 24
5.2. Frontend 24
6. Optimalizáló komponens 25
6.1. Működése 25
6.2. Kimenet értékelése 27
7. Megvalósítás 29
7.1. Az alkalmazás felülete 31
7.2. PDF szabásminta 33
8. Eredmények értékelése 34

8.1. Optimalizálás értékelése 34
8.2. Szabásminta értékelése 46
8.3. Funkcionalitás áttekintése 46
8.4. Összehasonlítás hasonló rendszerekkel 47
Összefoglalás 48
Irodalomjegyzék 49

BEVEZETÉS

Dolgozatomban egy olyan asztalosipari szoftvert tervezek és valósítok meg, amely támogatja a napi ügyintézési folyamatokat (ügyfelek, rendelések nyilvántartása, raktárkészlet nyilvántartása) és az ezek alapján létrejött termékleírásokból szabásmintát generál a megadott paraméterek alapján.

A szabásminta generálás folyamata során kétdimenziós optimalizálást elvégezve, szabásminta létrehozása a cél, amely tartalmaz a célgép függvényében vízszintesen, vagy függőlegesen átmenő vágásokat, amelyen az elhelyezett szükséges alkatrészek pozíciója, a legkisebb anyagveszteséget generálja, valamint felhasznál korábbról maradék alapanyagokat.

Az alkalmazásból a szabásminta online kép formájában, illetve PDF formátumban lesz megtekinthető, valamint kinyerhető. Cél mindezt egy modern szemléletű rétegzett webalkalmazásban megvalósítani.

1. LAPSZABÁSZAT

Az aktuális fejezet az [1] forrás alapján került kidolgozásra.

A program működéséhez fontos körbejárni a lapszabászat témakörét. Meg kell nézni a legelterjedtem módszereket. Ki kell emelni a paraméterek közül, hogy milyen termékeket szeretnénk feldolgozni, milyen vágási megoldást szeretnénk használni.

A legszélesebb körben használt anyagok az alábbiak:

- laminált faforgácslapok
- HDF és MDF farostlemezek
- furnérozott lapok
- lombos és tűlevelű fajokból készült táblásított anyagok
- rétegelt lemezek

Ezeket azért szükséges kiemelni, mert van közöttük olyan, amin látszik a fának az erezete, így nem mindegy a szabásmintán az alkatrész iránya. Ilyen alapanyag lehet a *laminált forgácslap*, *furnérozott lap,* valamint a *rétegelt lemez*.

1.1. Lapszabász gépek

A legtöbb a lapszabász gép körfűrész lappal, vagy szalagfűrésszel működik, ami azt jelenti a gyakorlatban, hogy a szabásminta kialakításánál figyelembe kell venni, hogy valahol a bútorlap szélén el kell kezdeni a fűrészelést, így szükség van bemeneti pont(ok)ra, ahol elkezdhető a vágás. Szükség lesz kimeneti pont(ok)ra is, mivel megfordulni, illetve fordulni nem tudunk, valamint visszamenni a vágaton sem célszerű, hiszem megsérthetjük az anyagot.

Ellentétben a CNC, és egyéb maró gépekkel, itt pontosan meg kell tervezni, hogy hol haladjanak a vágások. A lehető legjobb elrendezés, lapszabászgéppel nem biztos, hogy a valóságban kivágható úgy, mint a rajzon.

Ma az iparban a leggyakrabban vízszintes és függőleges lapszabász gépeket használnak.

A **vízszintes lapszabászgép** (*1.1. ábra*), a hagyományos körfűrészek tovább fejlesztett verziója. A fűrészlap fix pozícióban található, és egy szerkezet segítségével a bútorlap táblát mozgatjuk. Előnye, hogy rugalmasabb, az egyedi szabásmintákhoz alkalmasabb.

1.1. ábra – Vízszintes lapszabás gép (forrás [2])

A **függőleges lapszabászgép** (*1.2. ábra*), abban különbözik, hogy a munkadarab van fix pozícióban, és fűrészlap hordozó vágóegység mozog. Kevésbé rugalmas, leginkább derékszögű alkatrészek kivágására alkalmas.

1.2. ábra - függőleges lapszabász gép (forrás [3])

1.2. Szabásminta

Olyan szabásmintát kell előállítani, amelyből az alkatrészeket, az azok megfelelő elrendezése miatt, ki lehet vágni körfűrészes vagy szalagfűrészes lapszabász géppel, és nem csak CNC géppel, valamint figyelembe veszi a felhasznált anyag mintázatát (a fa erezetének irányát az alkatrészek forgathatósága miatt), a fűrészlap szélességét (mekkora távolságra legyenek egymástól az alkatrészek a fűrészlap mérete miatt keletkező pontatlanság elkerülése miatt).

A szoftver által előállított szabásmintán szükséges az optimálisan elrendezett alkatrészek rajzán a hozzájuk tartozó méreteket, és igény szerint az alkatrész nevét, a beazonosíthatóság miatt megjeleníteni.

2. HASONLÓ SZOFTVEREK

A piacon több lapszabász program is található, különböző funkciókkal, az alábbi fejezetben ezek kerülnek bemutatásra.

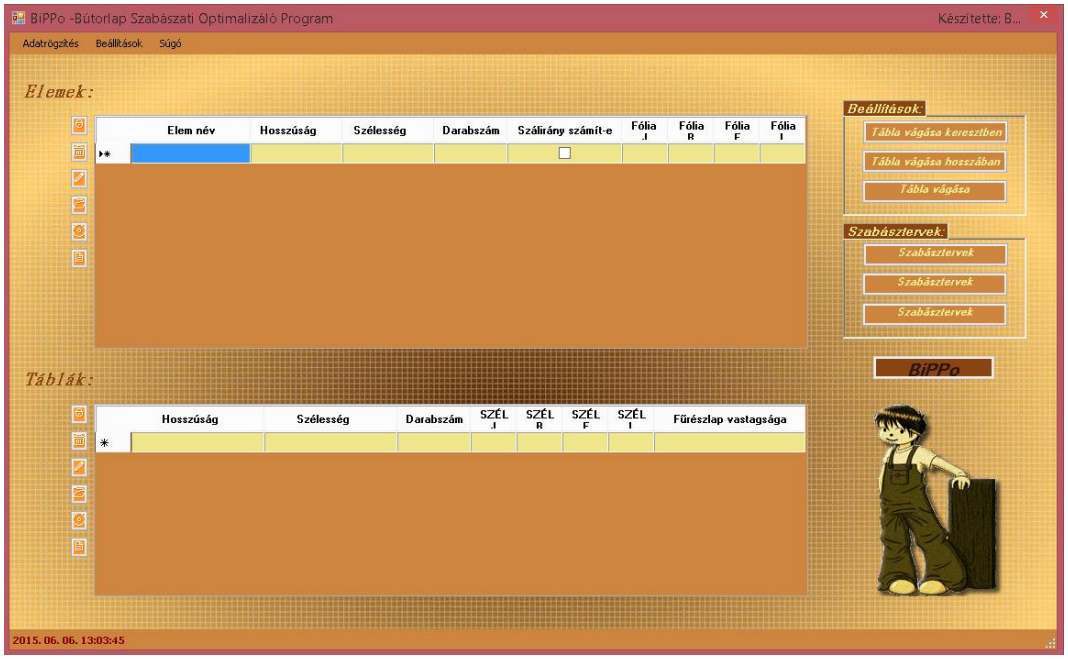
2.1. BIPPO

„A BIPPO egy olyan bútorlap-szabászati optimalizáló program, amely lehetővé teszi a gyors, hiba nélküli, kevert szálirányú optimalizálást és a grafikus, méretarányos szabászati terv megjelenítését.” [4]

Kipróbálás nem sikerült, nincs elérhető demó vagy próbaverzió.

A szoftver a weboldala szerint [4] az alábbi funkciókkal rendelkezik:

- elem és tábla méret manuális bevitel
- elem és tábla méret Microsoft Excel fájlokból történő beolvasása
- bevitt adatok exportálása
- függőleges vágásra optimalizálás
- méretarányos és léptékhelyes szabászterv megjelenítés
- sok paraméter megadható (fűrészlap vastagsága, irány) (*2.1. ábra*)



2.1. ábra - BIPPO kezelőfelülete (forrás: [4])

2.2. PaneCutter

A PaneCutter szoftverről az információk a [5] forrásból származnak.

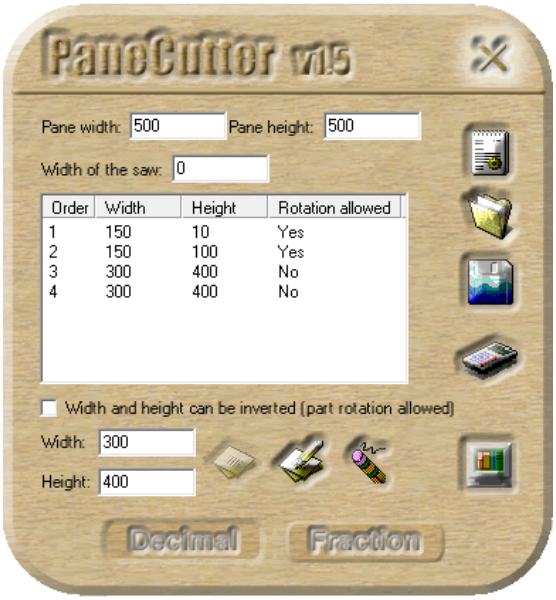
„A szoftver a lap alakú alapanyagok (laminált lap, MDF lap, üveglap, fémlemez stb.) vágási tervének automatikus generálására készült, bútor készítő-, üveges-, és egyéb műhelyek számára, amelyek termékskálája széles vagy állandóan változik.” [5]

A szoftver ingyenesen kipróbálható. Elérhető egy ingyenes demó, amiben 4 alkatrész méreteit adhatjuk meg.

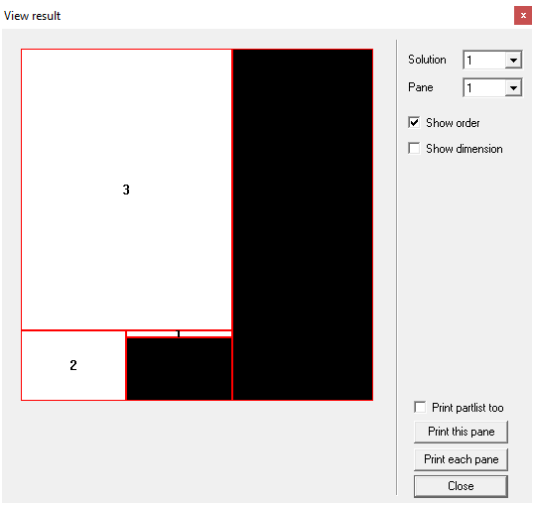
A hivatalos oldaluk leírása alapján rekurzív algoritmussal keresik meg az optimális vágási tervet. Ezzel törekednek a lehető legkevesebb veszteségre. Kimenetként megkapjuk az első optimális megoldást, majd utána választhatunk a többi jó megoldás közül.

A szoftver az alábbi funkciókkal rendelkezik:

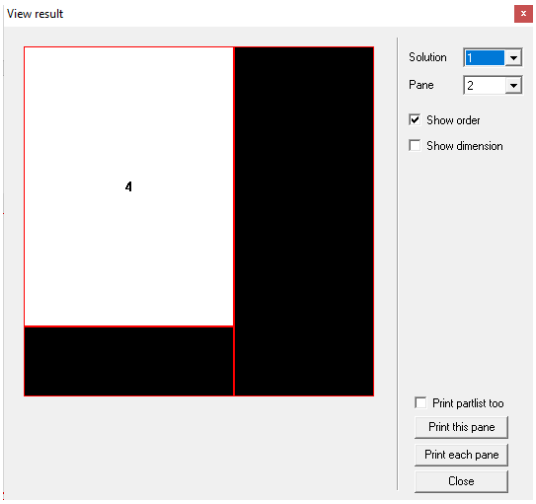
- elem és tábla méret manuális bevitel
- bevitt adatok és a szabásminta exportálása (PDF kiment) (*2.5. ábra*)
- megadható az alkatrész iránya (*2.2. ábra)*
- több megoldás közül választhatunk
- méretarányos és léptékhelyes szabászterv megjelenítés
- kimeneten nincsenek méretek (2*.3. ábra, 2.4. ábra és 2.5. ábra)*



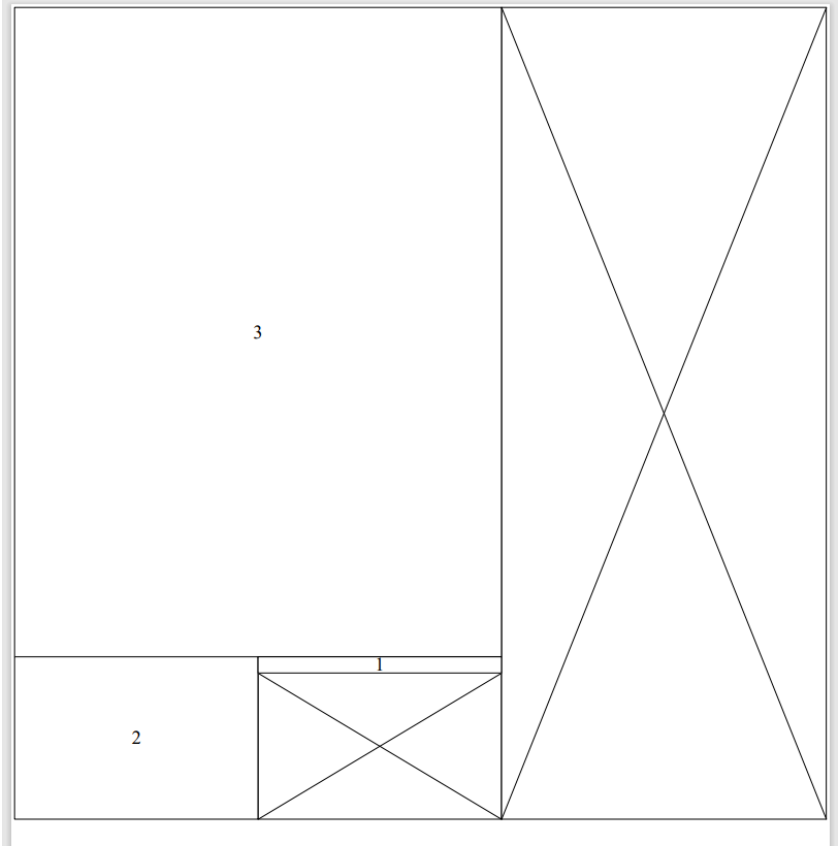
2.2. ábra - PaneCutter kezelőfelület, teszt bemenettel (saját ábra)



2.3. ábra - PaneCutter teszt kimenet, első táblára (saját ábra)



2.4. ábra - PaneCutter teszt kimenet, második táblára (saját ábra)



2.5. ábra - PaneCutter teszt kimenet PDF formátumban, első táblára (3.3 ábra) (saját ábra)

2.3. AMORF

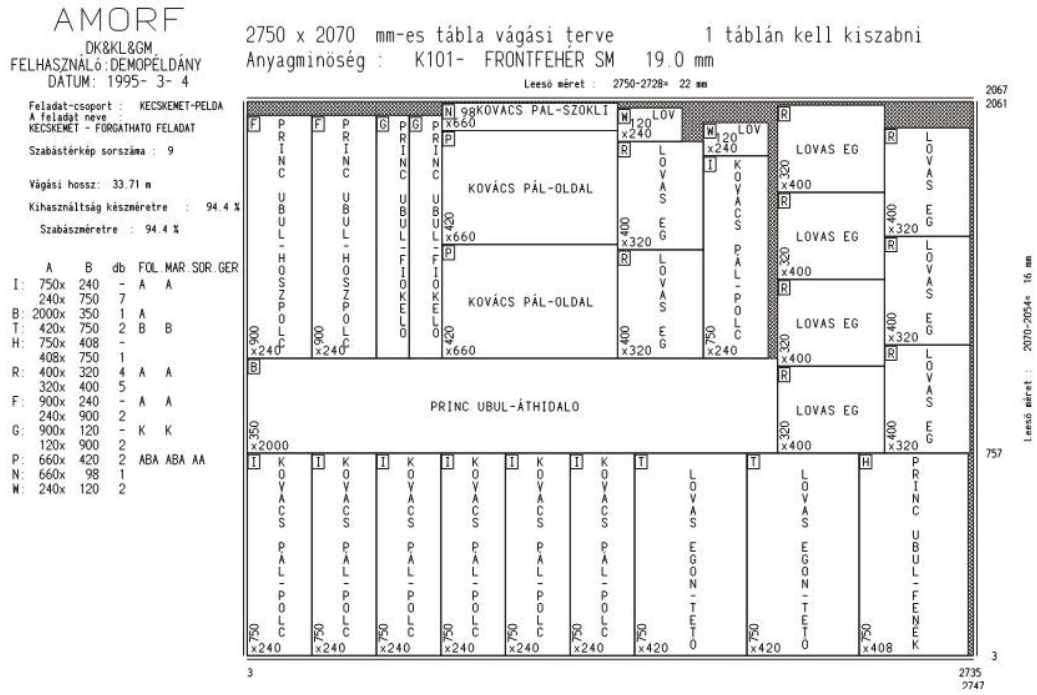
„Az AMORF szoftver optimális szabásterveket gyorsan, automatikusan állít elő, elenyésző szabási veszteséggel és a maradékok halmozódásának kiküszöbölésével.” [6]

A szoftvert kipróbálni nem sikerült, nincs elérhető demó vagy próba verzió.

Megadható paraméterek: szálirány, szélezés (hossz- és keresztirányban), szabász- és készméret eltérése, maradékok készlete (nem kötelező), egész táblák készlete (készlethiány esetén fiktív készletet).

A szoftver az alábbi funkciókkal rendelkezik:

- manuális adatbevitel
- szálirány kezelése
- készlet adatbázis
- több megoldás az utolsó táblán
- mindig van átmenő vágás
- mintán méretek, és alkatrész azonosítók (*2.6. ábra)*
- szabási veszteség figyelembevétele
- szabási terv exportálható automata szabászgépre



2.6. ábra - AMORF kimenet (forrás [6])

2.4. Optimik 4

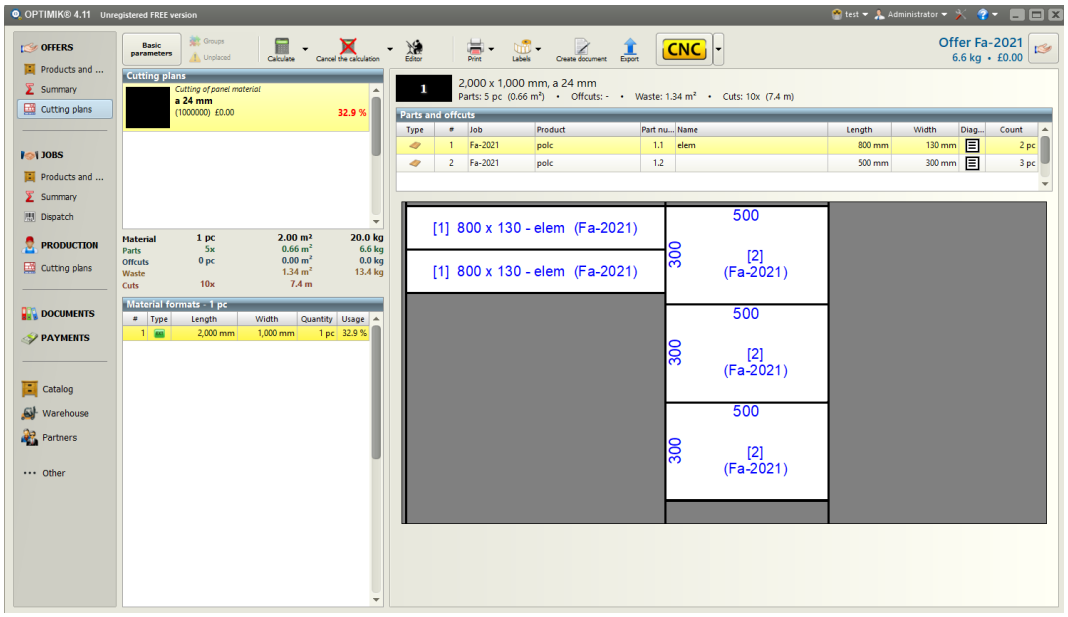
Az Optimik 4 a weboldala [7] szerint egy modern többfunkciós lapszabász alkalmazás, amelyben többek között lehetőség van projekteket létrehozni, raktárkészletet és pénzügyeket vezetni.

A szoftver ingyenesen kipróbálható.

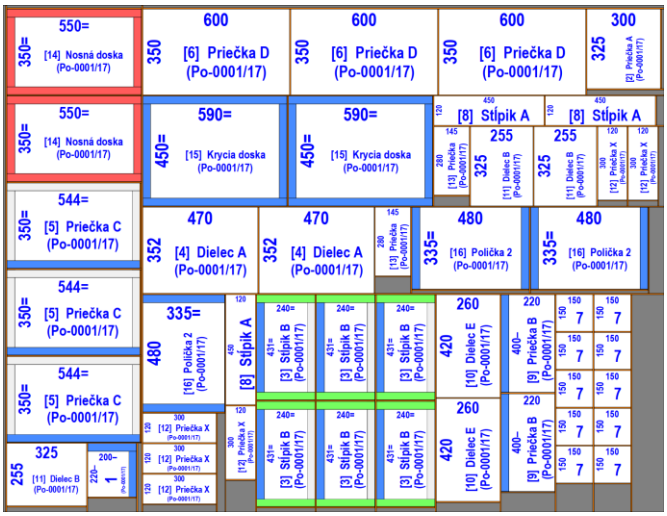
Az alkalmazás segítségével, létrehozhatunk projekteket (bútorokat) majd hozzáadhatjuk az alkatrészeket, beállíthatjuk az árakat hozzá, az alapanyagokat. A szabásmintáknál beállíthatunk több paramétert is (például vágási módszert) (*2.7. ábra*).

A szoftver az alábbi funkciókkal rendelkezik:

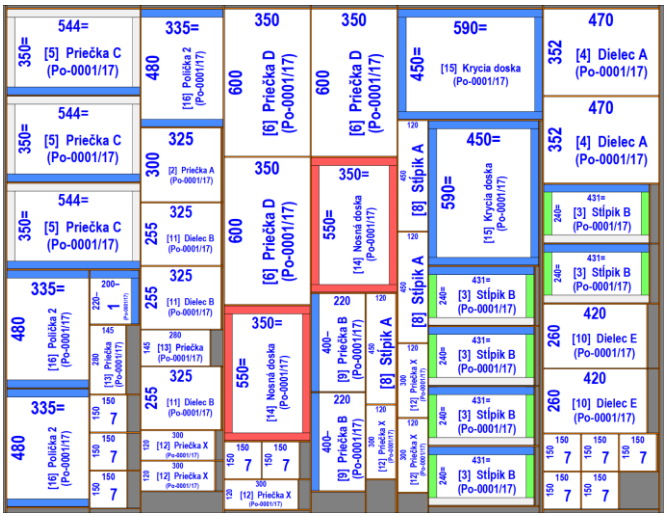
- manuálisan bevihető adatok
- meglévő adatbázis importálása
- projektek létrehozása
- képek hozzárendelése a projektekhez
- képek hozzárendelése anyagokhoz
- adatbázis (raktárkészlet, kiadás, bevétel, dokumentumok tárolása)
- nem szögletes alkatrészek készítése is
- raktárkészletből felhasználás
- többféle vágási irány (*2.8. ábra, 2.9. ábra és 2.10. ábra)*



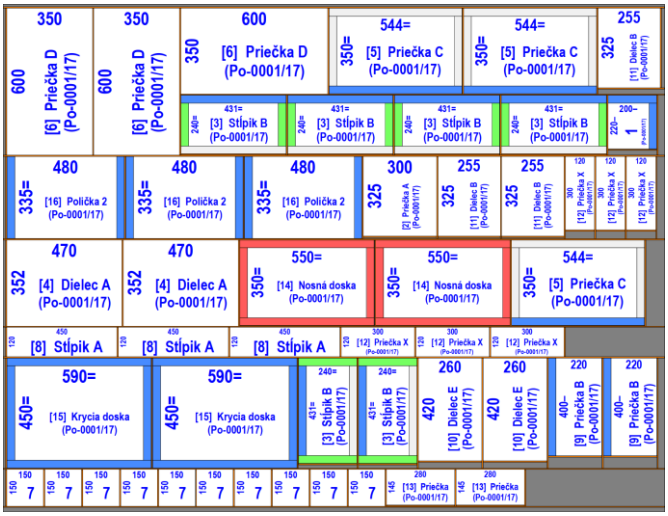
2.7. ábra - Optimik 4 felhasználói felület teszt bemenettel, és szabásmintával (saját ábra)



2.8. ábra – Optimik 4 szabásminta többirányú vágással (forrás: [8])



2.9. ábra – Optimik 4 szabásminta keresztirányú vágással (forrás: [8])



2.10. ábra – Optimik 4 szabásminta hosszirányú vágásokkal (forrás: [8])

2.5. Szoftverek összehasonlítása

A rendszerek összehasonlítását a *2.11. táblázat* alapján végeztem el.

	BIPPO	***PaneCutter***	***AMORF***	***Optimik 4***
Adatbevitel módja	Manuálisan, Excel tábla importálásával	Manuálisan	Manuálisan	Manuálisan, Adatbázis importálásával
Kimenet	Képernyőn megjelenő, nyomtatható formában	Képernyőn megjelenő, nyomtatható formában	Képernyőn megjelenő, nyomtatható formában, Automata szabászgépre küldve	Képernyőn megjelenő, nyomtatható formában
Szabásminta	Alkatrészek irányának megtartásával	Több opció, Méretek nélkül, Alkatrészek irányának megtartásával	Több opció, keresztirányú, hosszirányú vágással, méretekkel, Alkatrészek irányának megtartásával	Több opció, Keresztirányú, hosszirányú, többirányú vágással, Méretekkel, Alkatrészek irányának megtartásával
Funkciók	Csak szabászminta optimalizálás	Csak szabásminta optimalizálás	Szabásminta optimalizálás, raktárkészlet kezelés	Szabásminta optimalizálás, Raktárkészlet kezelés, Pénzügy kezelés, Dokumentum kezelés

2.11. táblázat - Funkciók összehasonlítása

Összevetve a különböző szoftverek adatbeviteli módjait, ez minden esetben manuálisan megtehető, azonban a négyből kettő szoftvernél, lehetőség van valamilyen egyéb úton is bevinni az adatokat a programba.

A kimenetét a szoftvereknek, vagyis a szabászmintát megjeleníthetjük a képernyőn, választhatunk több jó megoldás közük, majd nyomtatható formában exportálhatjuk, minden esetben. Az AMORF szoftverből, azonban használhatjuk a mintát automata szabászgépre küldve is.

A szabásminták közül a szoftverek több optimális opciót is felkínálnak. Az optimalizációt az alkatrészek dimenziójának megtartásával hozzák létre. Az AMORF, és Optimik 4 szoftver

esetén lehetőség van specifikálni a főbb vágási irányokat is. Ez azt jelenti, hogy megadhatjuk, hogy a hosszabb vágások vízszintesen vagy függőlegesen legyenek.

Funkciókat tekintve, elmondható, hogy a háttérmunkát segítő funkciókkal vannak ellátva. A raktárkészlet kezelés funkciót, kiemelném, mert a korábbi szabásmintákból keletkező nagyobb maradékok, egy új szabásminta létrehozásakor felhasználhatóak, így visszaszorítva a maradékok mennyiségének túlzott megnövekedését.

3. OPTIMALIZÁLÓ ALGORITMUSOK

Az alábbi fejezetben a feladat megoldására alkalmas módszerek és problémák kerülnek feldolgozásra a [9] -es, és [10] -es forrás alapján.

Ezt a problémát az irodalom, úgynevezett kétdimenziós *cutting stock*, vagy kétdimenziós *bin packing* problémának nevezi, ami nemcsak a bútorlap szabászatban, hanem az üveg, acél vágásban, vagy a konténer pakolásnál is megjelenő probléma.

A cél, hogy minél több különböző méretű téglalapokat vágjunk ki, nagyobb méretű téglalapokból(alapanyagból), táblákból úgy, hogy a lehető legkevesebbet használjuk az utóbbiból.

Meg van adva véges számú n_E téglalap, $E = \{1, \dots, n_E\}$, minden elemnek van egy magassága $h_i \in \mathbb{N}^+$, szélessége $w_i \in \mathbb{N}^+$, és hogy mennyi darabra van belőle szükségünk $d_i \in \mathbb{N}^+$. Az alapanyag magassága $H \in \mathbb{N}^+$, szélessége $W \in \mathbb{N}^+$. Az elemeket, ebben az esetben 90°-ban forgathatónak tekintjük, ami azt jelenti, hogy hozzárendelünk minden téglalaphoz egy elforgatott változatot.

A cél, hogy megtaláljuk az optimális P vágási mintát, amin a téglalapok nem fedik egymást, a lehető legkevesebb táblát használtuk fel, valamint a veszteség is minimális.

Erre a problémára adnak megoldásokat a determinisztikus 2- és 3-fázisú Guillotine vágások, a Sarokba helyezés algoritmusa, a Sík felosztás, valamint a heurisztikus genetikus algoritmus alapú módszerek.

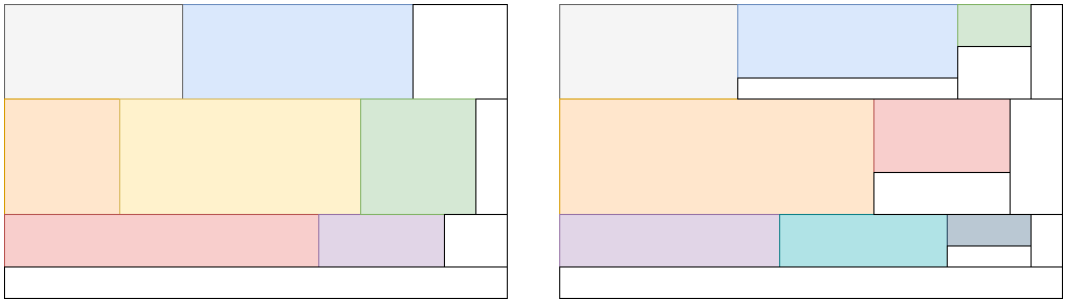
3.1. 2-fázisú Guillotine vágás

Az alábbi fejezet a [9] -es forrás alapján készült.

A 2-fázisú guillotine vágás esetében, a vágás kettő fázisból áll, vízszintes és egy függőleges vágásból.

Először redukáljuk a problémát kettődimenziósból egydimenzióssá. Ezt úgy hajtjuk végre, hogy az egyik oldalan mentén csíkokra osztjuk a táblát, úgy, hogy ráférjenek, azokra a csíkokra, a többi elhelyezendő elem. Ezután a csíkokat egydimenziós *cutting stock* problémaként elhelyezzük.

Ez a megoldás, néhány esetben működik, de az olyan esetekben, amikor nincs azonos oldalhosszúságú elem, rengeteg maradék keletkezik (*3.1. ábra*). Erre lehet megoldás a 3-fázisú Guillotine vágás.



3.1. ábra - Példa 2-fázisú Guillotine vágásra, vannak azonos oldalhosszúságú elemek (balra) és különböző oldalhosszúságú elemek (jobbra) (saját ábra, [9] -es forrás alapján)

3.2. 3-fázisú Guillotine vágás

Ez a fejezet a [10] -es forrás alapján készült.

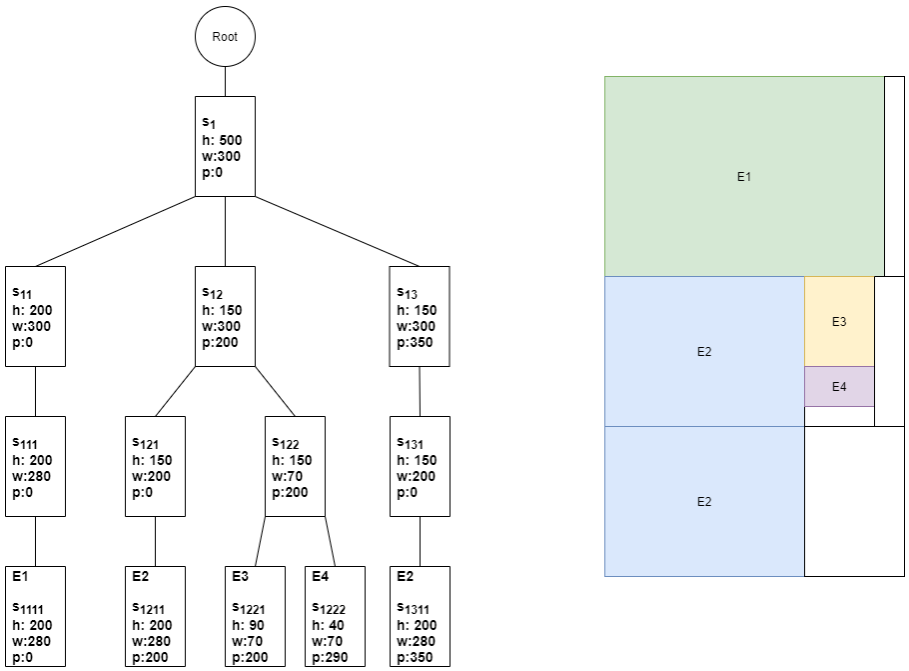
A 3-fázisú guillotine vágás esetében, a vágás 3 fázisból áll, az első, és a harmadik horizontális, a második vertikális. A vágási minta egy vágási fa alapján kerül meghatározásra.

Az alapanyag tábla, bal felső sarka a (0; 0), a jobb alsó, pedig (H; W) koordinátával rendelkezik. Minden egyes felhasznált táblán a maradék csík magassága a (0; 0) koordinátától ρ_j $j = 1, \dots, n$, ha nincs maradék, $\rho_j = 0$, c(P) a legnagyobb maradék, melyet az (1) egyenlettel határozhatunk meg.

$$c(P) = \min\left(n - \frac{max_{j=1,\dots,n}\rho_j}{H}\right) \tag{1}$$

Az algoritmus a vágási fát, a következő módon állítja elő. Létrehozzuk s_1-et, ez a tábla, amiből kiindulunk, ez a fának a gyökere. Aztán vízszintesen vágunk, ezek lesznek az s_{1n} táblák. Ezután függőlegesen vágunk, majd megint vízszintesen. Ezekből lesznek s_{1nm} és s_{1nmk} táblák. Ebből megkapjuk az optimális elhelyezést.

Vegyünk egy példát. A tábla, amivel dolgozunk 500 cm magas és 300 cm széles. Szükségünk van egy 200 cm magas és 280 cm széles elemre (E1), kettő 150 cm magas, és 200 cm széles elemre (E2), egy 90 cm magas és 70 cm széles elemre (E3), valamint egy 40 cm magas és 70 cm széles elemre (E4). Erre a példára látható vágási fa, és ebből kialakított vágási minta a *3.2. ábrán*.

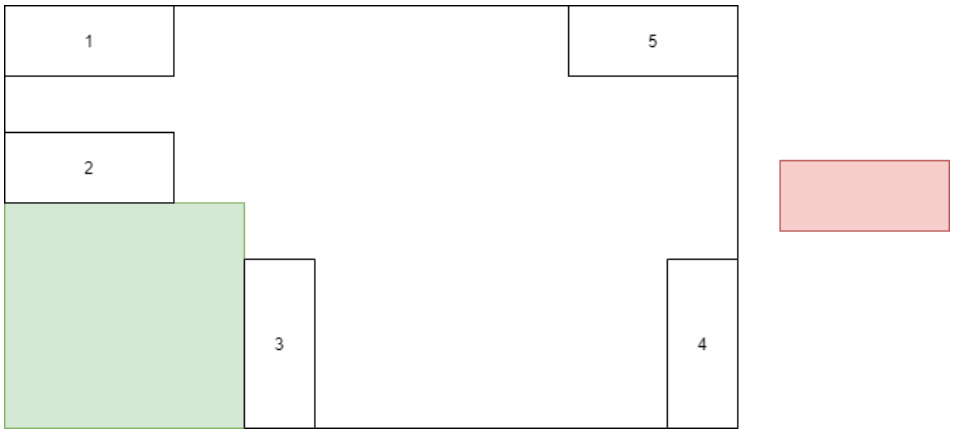


3.2. ábra - Egy vágási fa (balra) és az ebből kialakított vágási minta (jobbra) (saját ábra, [10] -es forrás alapján)

3.3. Sarokba helyezés

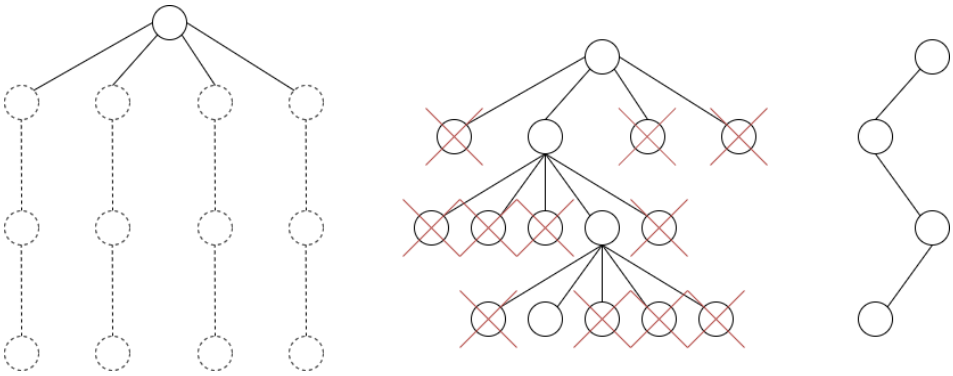
Ebben a fejezetben a [11] -es forrás került felhasználásra.

A sarokba helyezés módszer, egy másik stratégia lehet. Sorba rendezzük az alkatrészeket valami alapján meghatározott jóság szerint. Majd ebből kiválasztva a legjobb elemet, az alapanyag tábla, egyik sarkába helyezzük. Saroknak azt a helyet tekintjük, amikor az alkatrész kettő, különböző oldalhoz ér hozzá, ez lehet a tábla két oldala, vagy a tábla egyik oldala, és egy másik alkatrész egyik oldala vagy két alkatrész egy-egy oldala. Mivel induláskor a táblánk üres, 4 szabad sarka van egy téglalapnak, így négy lehetőségünk van, azonban, ha lehet forgatni az elemeket, de az alapot nem, akkor nyolc. Mivel nem tudjuk, melyik megoldás lesz optimális, ezért az összes lehetőség szerint visszük tovább a megoldást. A következő legjobb elem kiválasztása után, annak is keresünk kell egy sarkot. Azonban, ebben az esetben, már 5 olyan hely van, amit saroknak tekinthetünk, ami az alkatrész forgathatósága miatt 10-et jelent (*3.3. ábra*). Szükség van, egy szabályra, ami alapján szűkíteni lehet a lehetőségeket.



3.3. ábra - öt pozíció, a következő alkatrész elhelyezésére. (saját ábra, [11] -es forrás alapján)

Válasszuk, ki azokat a pozíciókat, amiknél, a legkisebb a távolság egy másik alkatrész felé. Majd itt is leágaztatjuk a megoldási lehetőségeket, és visszük tovább. A végén, pedig összehasonlítjuk a megoldásokat, és kiválaszthatjuk a jó megoldásokat.



3.4. ábra - Megoldás fa a sarokba helyezés módszeréhez (saját ábra, [11] -es forrás alapján)

3.4. Sík felosztása

Az alábbi módszer egy rekurzív algoritmus, ami saját ötletemen alapul és a működését tekintve hasonló a már feldolgozott megoldásokhoz.

Szélességük alapján csökkenő sorrendbe helyezzük az alkatrészeket.

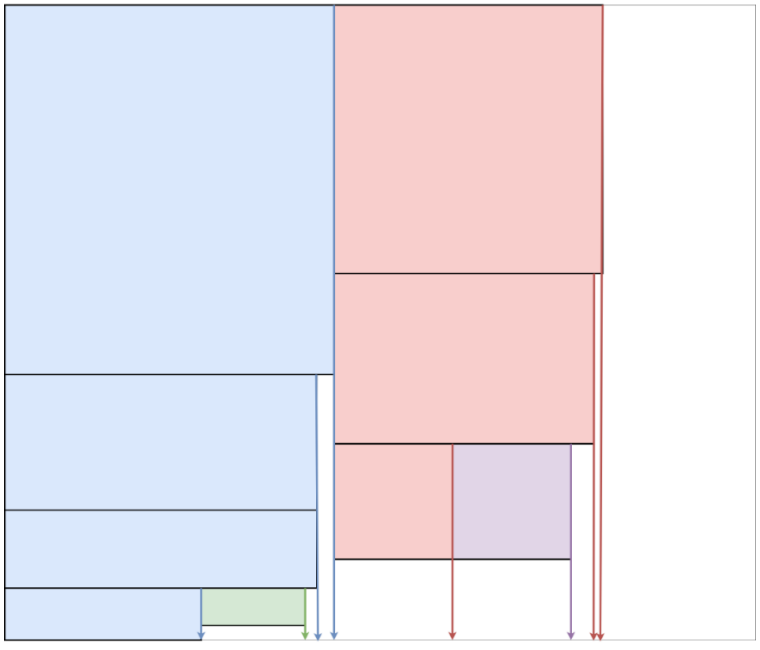
A legszélesebb alkatrész a bal felső sarokba, helyezzük, majd lefelé meghosszabbítva jobb oldalát, két szabad területet kapunk, egyet a lehelyezett alkatrész alatt, a másikat mellette.

Ezután kezdődik el az első fázis: elkezdjük a legelső elem alatti területre elhelyezni az alkatrészeket, mindig úgy, hogy a jobb oldali oldalukat meghosszabbítva kapott alattuk lévő területet használjuk, egészen addig, amíg el nem fogy a hely. Amennyiben elfogyott, akkor az utoljára lehelyezett elem melletti síkon lerakunk egy következő elemet.

Abban az esetben, hogyha alá elfér egy következő alkatrész akkor, azzal épp úgy, mint az első fázisnál, elkezdjük a következő fázist, és rakjuk egymás alá az alkatrészeket, amíg van hely. Amikor elfogyott megnézzük az utoljára lehelyezett elem melletti részt, és indítjuk a következő fázist.

Amennyiben nem tudunk oda elemet elhelyezni, akkor megnézzük az utolsó előtti lehelyezett elemet, és azzal indítjuk a következő ciklust.

Addig lépünk vissza és indítjuk a következő rekurziót backtrack jellegűen, amíg el nem fogynak a lehelyezendő elemeink. A működése a *3.5. ábrán* látható



3.5. ábra – Az algoritmusom működése, kék (első fázis), zöld második fázis), piros (harmadik fázik), lila (negyedik fázis) … (saját ábra)

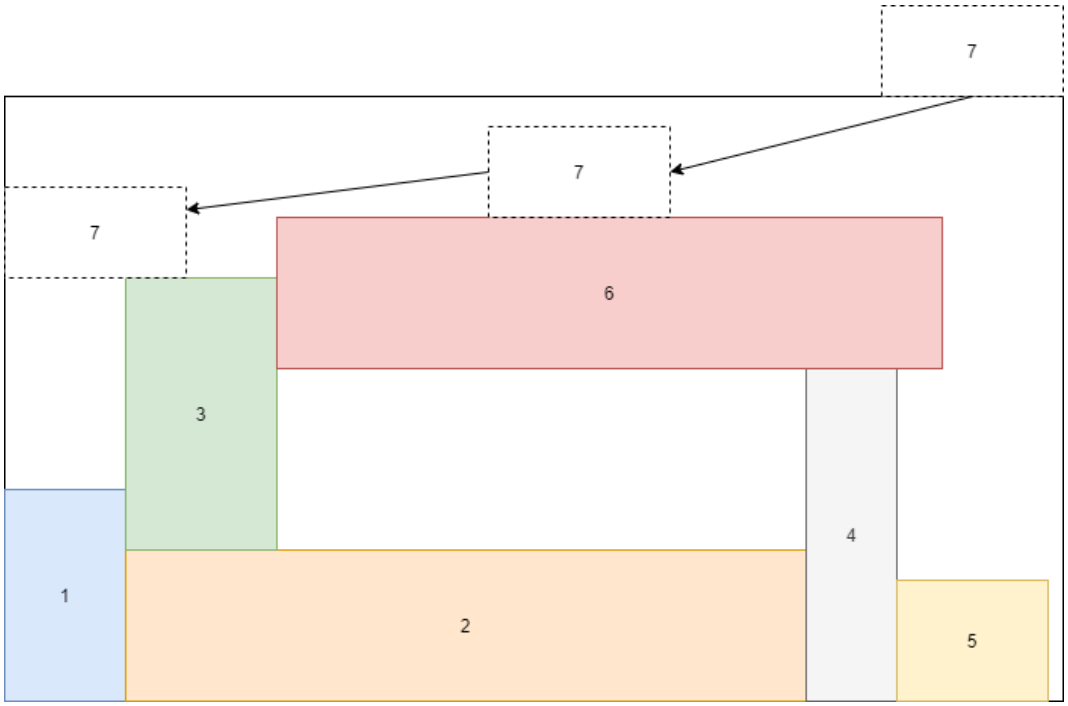
3.5. Heurisztikus megoldások

Ebben a fejezetben a [12] -es forrás került felhasználásra.

Míg az előző megoldásokban, a Guillotine vágáshoz hasonló megoldásokat láthattunk, ebben az esetben, a klasszikus megoldásoktól eltérőek kerülnek bemutatásra. Ilyen megközelítést képeznek a generikus algoritmusok használata az ismertetett problémára.

3.5.1. BL (Bottom-Left) algoritmus

„Bal-lent" algoritmus, azt jelenti, hogy a jobb felső részén kezdjük el az ideális pozíció megkeresését, és folyamatosan haladunk a bal alsó sarok irányába, hogy minél baloldalibb, és lentebbi pozíciót találhassunk az alkatrésznek genetikus algoritmus felhasználásával.



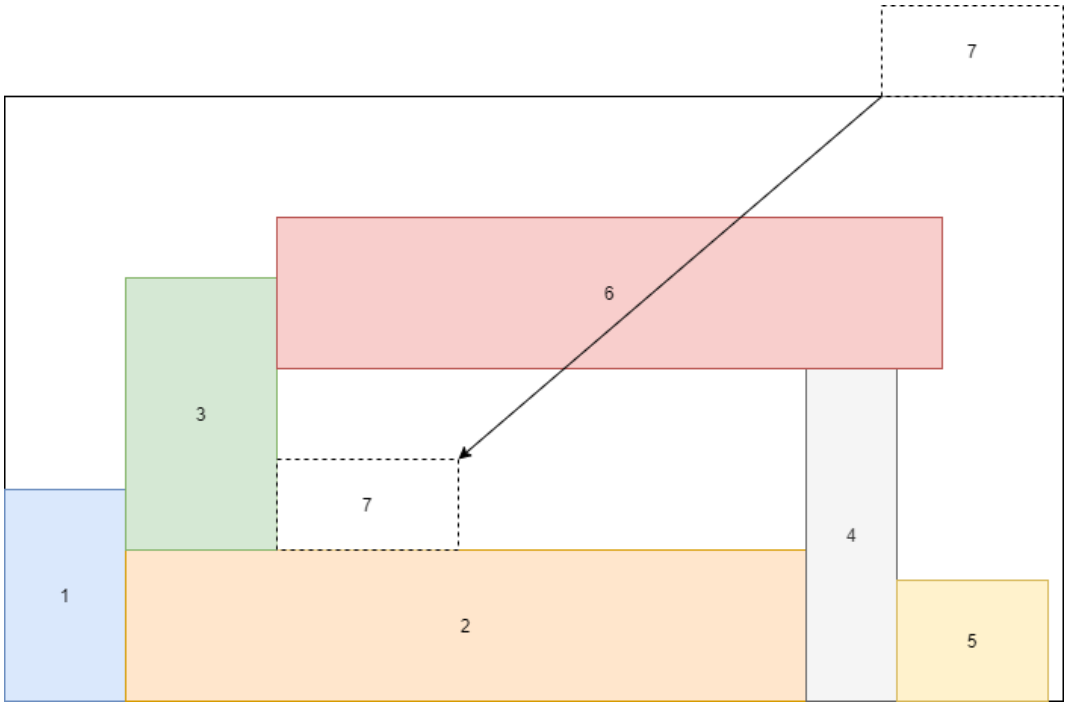
3.6. ábra - - BL algoritmus (saját ábra, [12] -es forrás alapján)

Láthatjuk, hogy ez nem fog minden esetben, ideális elrendezést adni (3.*6. ábra*), mivel nagy területű üres mezőket, hoz létre, ami az algoritmus működése miatt, nem kerül kitöltésre. Mivel genetikus algoritmusról van szó, ezért fontos a futási időt megnézni. Ha N-nek nevezzük az összes lehelyezett elemet, akkor a futási idő $O(N^2)$.

A nem megfelelő helykihasználásra ad megoldást, ennek a tovább fejlesztett változata a *BLF algoritmus.*

3.5.2. BLF (Bottom-Left-Fill) algoritmus

„Bal-lent-kitöltő” algoritmus, ami azt jelenti, hogy a meglévő lyukakat próbálja meg kitölteni, a minél baloldalibb, és lentebb szemlélet szerint (*3.7. ábra*).



3.7. ábra BLF algoritmus (saját ábra, [12] -es forrás alapján)

3.6. Módszerek áttekintése

A fentebb taglalt módszereket áttekintve, azt láthatjuk, hogy számos módszer van a problémánk megoldására, azonban végig gondolva nem mindegyik lesz számunkra megfelelő, mivel, az 1.2. fejezetben leírt okok miatt, átmenő vágásokra van szükségünk, olyanokra, amik a táblát kisebb táblákra osztják, majd még kisebb táblákra, amíg ki nem vágtuk az alkatrészeket. Éppen ezért mindenképpen a Guillotine vágást, vagy ahhoz hasonló módszeren alapuló megoldást szükséges használni. A 3.3. fejezetben leírt módszer azért nem lesz ideális, mert nem garantált az átmenő vágás, a 3.5. fejezetben taglalt megoldások szintén nem lesznek megfelelőek számunkra, mert minden futásnál más eredményt adnak, valamint itt sem garantált az átmenő vágás minden esetben. A másik három a 3.1., 3.2., és 3.4. fejezetben taglalt determinisztikus stratégiák lesznek kielégítőek számunkra az optimalizáció folyamán, ezek közül a módszerek és algoritmusok közül kell választanunk.

4. IRODALOMKUTATÁS ÁTTEKINTÉSE

Az irodalomkutatás során, számos hasonló programot, és a problémára megoldást kínáló technikát megvizsgáltam.

Arra a következtetésre jutottam a kutatásomból, hogy egy olyan programra van szüksége az iparnak, amiben ezeknek a programoknak a hasznos funkcióit egységesítem. Megemlítendő, hogy fájlból beolvasva is lehessen az adatokat bevinni, hogy egy-egy projektet el tudjunk menteni, későbbi újra felhasználásra. Szükséges kezelni a fűrész által okozott pontatlanságát, az elemek forgathatóságát, elmenteni, a vágások után keletkező maradék darabokat újbóli felhasználásra, a felesleges maradék képzés minimalizálásának érekében.

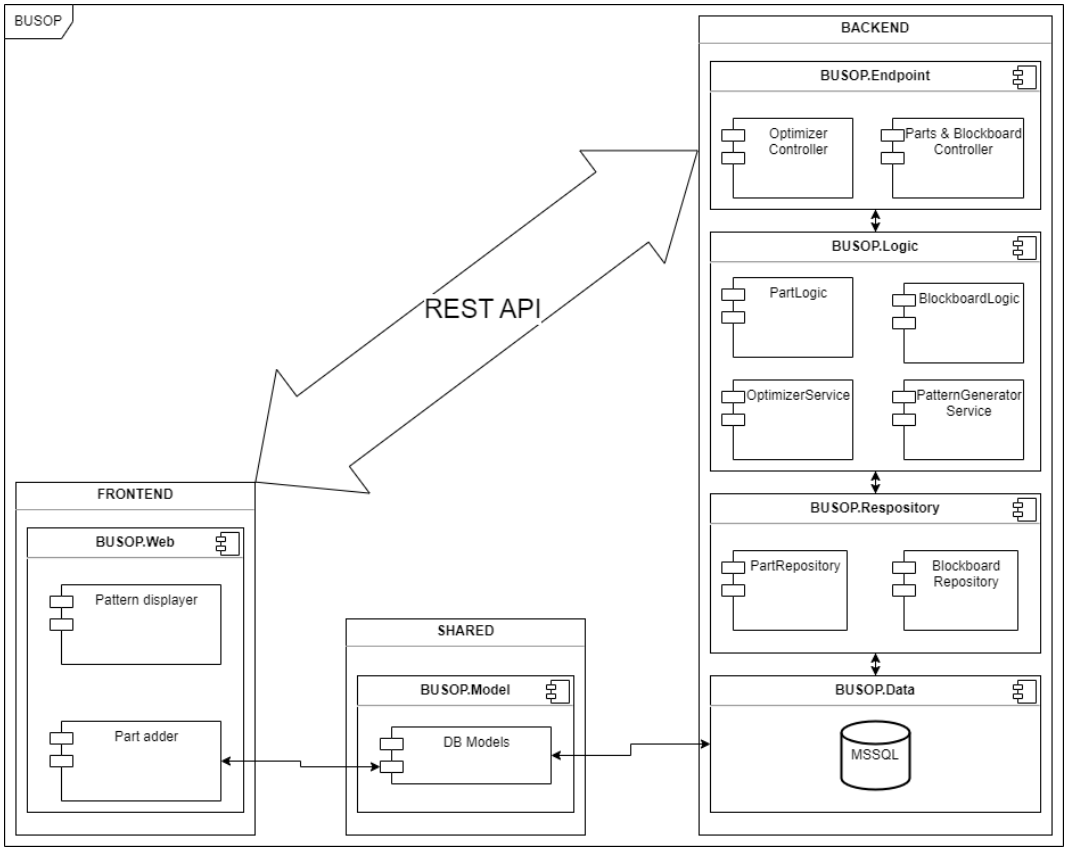
A szabásmintát tekintve, arra a következtetésre jutottam, hogy fontos megjelölnünk az elemeket rajta, valamint a méreteit feltüntetni. Megkötés továbbá, hogy nyomtatható formában ki lehessen exportálni, valamint, hogy a programból megnyitva választani lehessen több optimális megoldás közül, hogy a felhasználó el tudja dönteni számára melyik a legjobb.

Az optimalizáló módszereket tekintve, azt a konklúziót vontam le, hogy a legcélszerűbb egy Guillotine vágáson alapuló algoritmus lesz, hiszen ennél a módszernél mindig van átmenő vágás, ami azért fontos, mert a lapszabász géppel nem tudunk visszamenni, valamint fordulni.

A cél egy egyszerű felületű alkalmazás, ami csak jól behatárolt és releváns funkciókkal rendelkezik, gyors és optimális megoldást ad a problémára.

5. RENDSZERTERV

A szoftvert BUSOP fantázianévvel kereszteltem el, ami a Bútorlap szabásminta optimalizáló program rövidítése. A BUSOP szoftvert, többrétegű alkalmazásként az egyetemen tanult módszerek alapján terveztem meg. Két nagy részből fog állni az egész alkalmazás, valamint egy közös komponensből, amit a *5.1. ábrán* látható.



5.1. ábra – Szoftver felépítése (saját ábra)

A rendszer az alábbi alfejezetekben kerül bemutatásra.

5.1. Backend

A háttérbeli logika felbontható az alkalmazásra és az adatbázisra.

5.1.1. Alkalmazás

A *backend* részt C# nyelven fogom elkészíteni .NET 6.0-val. Ez a rész terveim szerint állni fog, egy **data** rétegből, ami az adatbázist fogja tartalmazni, lokális MSSQL-t. Az adatbázis modellje egy **shared** projectben lesz eltárolva. A backend rész tartalmazni fog egy **repository** réteget, ami majd a data és a logic réteget köti majd össze. A **logic** rétegben egyszerű CRUD műveletek lesznek megvalósítva, illetve ebben lesz implementálva az optimalizáló algoritmus és a szabásminta grafikus megjelenítését létrehozó szolgáltatás. Az API **endpoint** ASP.NET segítségével lesz elkészítve, ami a logic metódusait fogja majd meghívni és eredményeiket továbbítani.

5.1.2. Adatbázis

Adatbázisként a már fentebb említett lokális MSSQL-t fogok használni. Minden alkatrész téglalapként lesz kezelve és az alábbi mezőkkel fog rendelkezik az adatbázisban:

Név	Leírás	Típus
id	alkatrész azonosítója	string
projectid	alkatrész alapanyaga	string
material	projekt azonosító	string
name	alkatrész neve	string
blockboard	melyik bútorlap táblából kell kivágni az alkatrészt	int
x	bútorlap táblán x koordináta	int
y	bútorlap táblán y koordináta	int
width	alkarész szélessége	int
height	alkatrész magassága	int
isRotatable	forgatható-e	bool

5.1. táblázat -Alkatrészeket tartalmazó tábla felépítése

5.2. Frontend

A webalkalmazás *frontend* részét Blazor segítségével fogom megvalósítani. A frontendhez is C#-ot és .NET 6.0-át fogok használni. Két főbb komponensből fog állni. Az egyik segítségével a felhasználó fel tudja majd vinni a kivágandó alkatrészeket, valamint törölni tudja őket az adatbázisból. A másik komponens a szabásminta megjelenítéséért lesz felelős. Ez a szoftver rész használni fogja a **shared** komponenst az adatmodellekhez. Az űrlap megjelenítéséhez Bootstrapet fogok használni.

6. OPTIMALIZÁLÓ KOMPONENS

Az optimalizáció megvalósítására a 3.4. fejezetben leírt módszert fogom majd megvalósítani és ebben a részben kifejteni, ezt a 3.1. fejezetben leírt két-fázisú Guillitone vágás megvalósított algoritmusával fogom a dokumentum végén összehasonlítani.

6.1. Működése

Az algoritmus először elforgatja az elemeket, amelyeknek forgatását engedélyezte a felhasználó úgy, hogy a hosszabb oldala legyen vízszintesen.

Ezután ezen oldal szerint csökkenő sorrendbe rendezi őket, majd elkezdődik a rendezésük a kettő dimenziós térben, az alábbi módon.

A megírt függvény rekurzívan működik. Bemeneti paraméterei az alábbiak: a síkidomot amire optimalizálni kell az adott elemet, az optimalizálandó, rendezett gyűjteményt, amiben az elhelyezendő elemek vannak, egy kimeneti üres gyűjteményt, amibe a felhasznált elemek kerülnek át, valamit egy szintén üres gyűjtemény, amibe, a fel nem használt területek lesznek kigyűjtve.

Ezek után egy for ciklus segítségével megkeresi a legnagyobb arra területre elférő alkatrészt, majd odahelyezi. Ezek után az alatta lévő téglalapot, és a mellette lévő téglalapot elmenti, majd azokra újra meghívja saját magát a függvény. Amennyiben nincs már elhelyezhető elem a paraméterként kapott területre, akkor az a terület elmentésre kerül. A működését a *6.1. algoritmus* mutatja be.

Két sorrendben történhet a rendezés. Vízszintesen (*6.2. ábra*), először a jobb oldali területre próbálja elhelyezni elemeket, vagy Függőlegesen (6*.1. ábra* és *6.3. ábra*), azaz az alatta lévő téglalapra próbálja először az alkatrészeket elhelyezni. Mikor lefutott a függvény, akkor a maradékokat tartalmazó gyűjteményből kiszedjük a nulla területű maradékokat.

Több táblára történő optimalizáláshoz a legfelső szintet egy while ciklusba ágyazom, ami addig indítja el újra a rekurzív optimalizálást, amíg van fel nem dolgozott elem. Bevezetésre került egy változó, ami a tábla sorszámát menti el.

Maradék feldolgozásánál, a fentebb leírt while ciklusban a maradékokat tartalmazó adatbázisból kinyert paraméterekkel indul el a legfelső szintű rekurzív optimalizáló függvény.

A fűrészlap vastagsága miatt keletkező veszteség kezelésére úgy lehet bővíteni az algoritmust, hogy az alkatrész lehelyezése után a mellette lévő területből a veszteség mértékét kivonjuk, a fűrészlap szélességével csökkentjük szélességét. Az alkatrész alatti területnél pedig a magasságát csökkentjük a fűrészlap vastagságával.

6.1. Algoritmus BUSOP_Optimalizáló

Bemenet: *referenciaként* tábla – **T**, *referenciaként* elemek – **T lista**, *referenciaként* elrakott_elemek – **T lista**, *referenciaként* maradék_elemek – **T lista**, tábla_szám - **szám**

```
1:  függvény BUSOP_Optimalizáló(tábla, elemek,
    elrakott_elemek, maradék_elemek, tábla_szám)
2:     ciklus i 0-tól elemek hosszáig
3:        aktuális_elem ← elemek[i]
4:        ha ¬(aktuális_elem.X = 0 és aktuális_elem.Y = 0
          és aktuális_elem.Szélesség = 0 és
          aktuális_elem.Magasság = 0) és
          aktuális_elem.Szélesség < tábla.Szélesség és
          aktuális_elem.Magasság < tábla.Magasság akkor
5:           elemek \ aktuális_elem
6:           aktuális_elem.X ← tábla.X
7:           aktuális_elem.Y ← tábla.Y
8:           aktuális_elem.Tábla_szám ← tábla_szám
9:           vertikális_maradék_tábla ← T (X: tábla.X +
             aktuális_elem.Szélesség, Y: tábla.Y,
             Szélesség: tábla.Szélesség +
             aktuális_elem.Szélesség, Magasság:
             tábla.Magasság)
10:          horizontális_maradék_tábla ← T (X: tábla.X, Y:
             tábla.Y + aktuális_elem.Magasság, Szélesség:
             aktuális_elem.Szélesség, Magasság:
             tábla.Magasság + aktuális_elem.Szélesség)
11:          elrakott_elemek ∪ aktuális_elem
12:          BUSOP_Optimalizáló(vertikális_maradék_tábla,
             elemek, elrakott_elemek, maradék_elemek,
             tábla_szám)
13:          BUSOP_Optimalizáló(horizontális_maradék_tábla,
             elemek, elrakott_elemek, maradék_elemek,
             tábla_szám)
14:       elágazás vége
15:    ciklus vége
16:    ha tábla ∉ maradék_elemek akkor
17:       maradék_elemek ∪ tábla
18:    elágazás vége
19: függvény vége
```

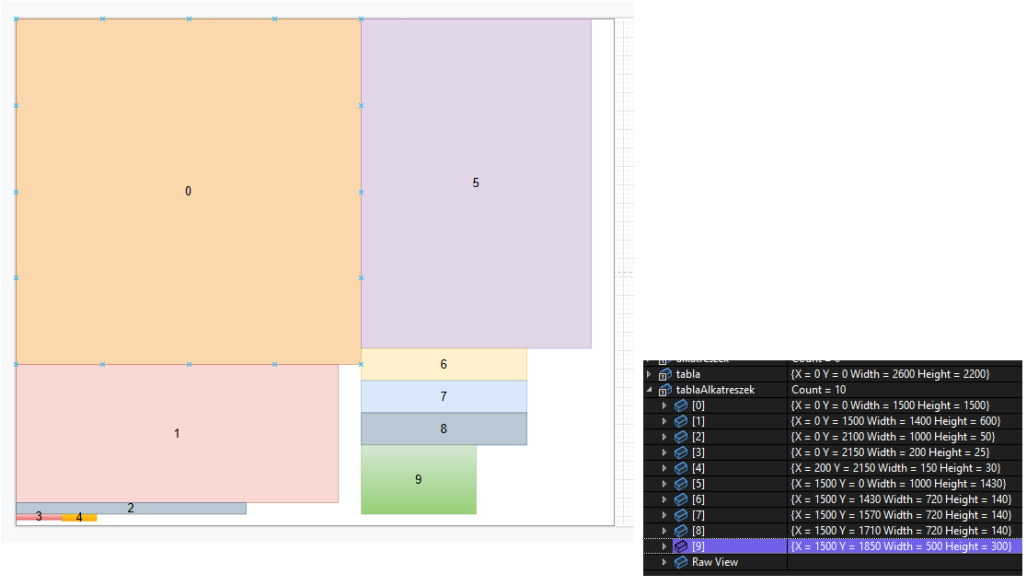
T: Olyan objektum, ami rendelkezik x, y koordináta, magasság, szélesség és tábla_szám tulajdonsággal.

6.2. Kimenet értékelése

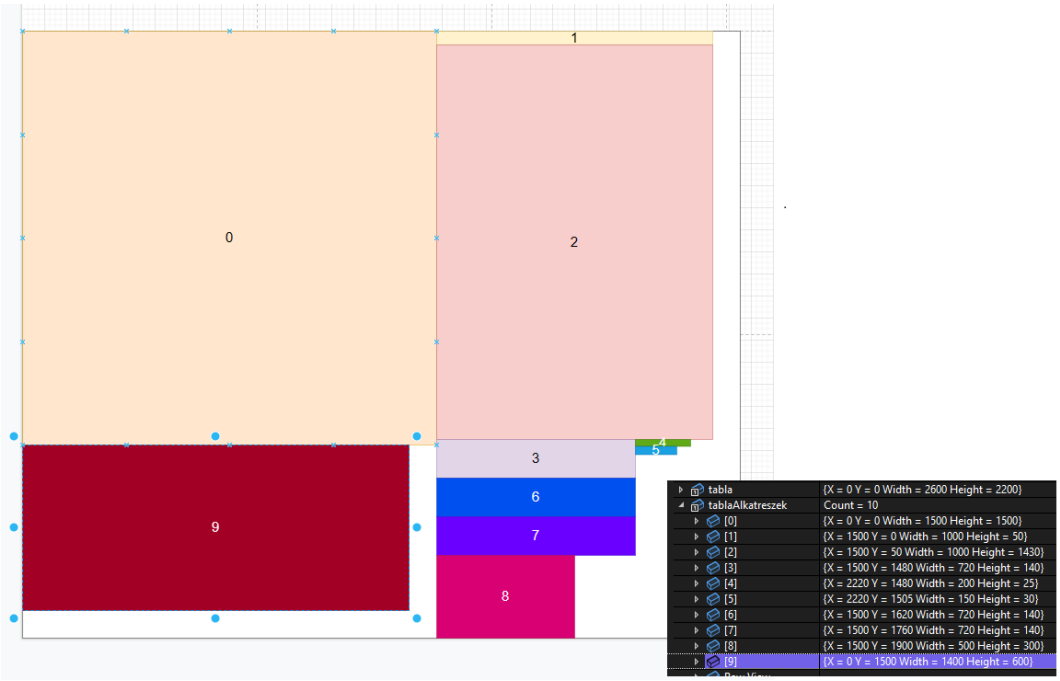
Az optimalizáló sikerességének értékelése az alábbi módon történik. Az egész maradék területét elosztja a felhasznált téglalap alakú alapanyagok területével. Ezek után minden maradék rész területét elosztja a legnagyobb alkatrész területével, és ezeket összeadja, majd átlagot számol. Végül ezzel az átlaggal megszorozza az első eredményt (*2. képlet*).

$$eredmeny = \frac{T_{maradék}}{T_{egész}} \times \frac{\sum_{i=0}^{n} \frac{T_{maradek_i}}{T_{alkatrész_{max}}}}{n} \tag{2}$$

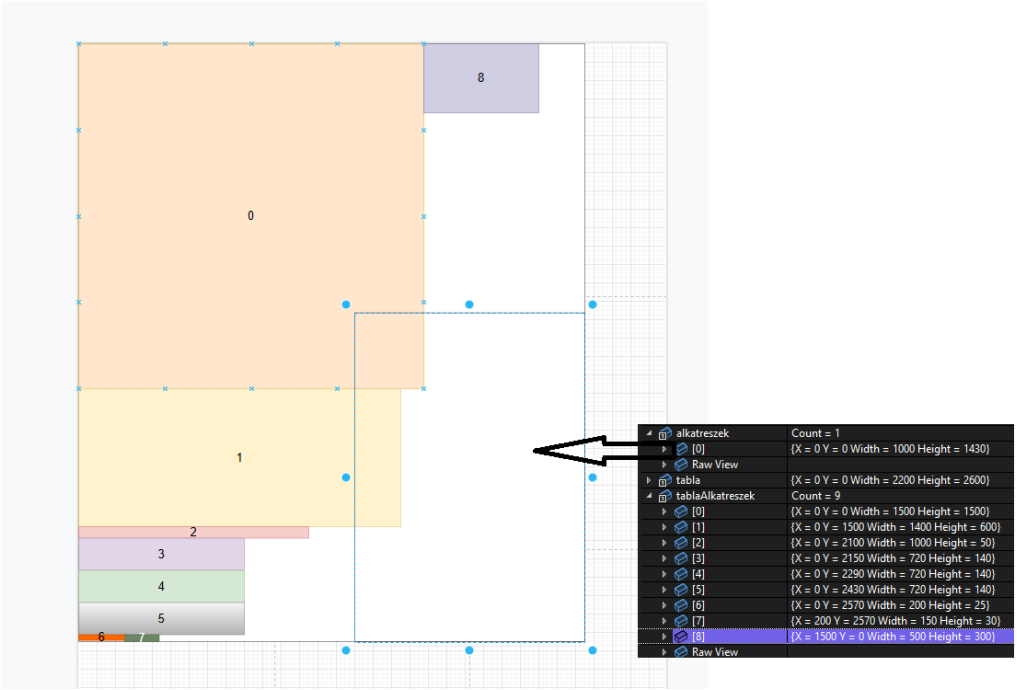
Ha az egész alapanyagot felhasználtuk, akkor 0 lesz az eredmény, minél több fel nem használt területünk, és ezek minél több alapanyag táblán helyezkednek el, annál nagyobb lesz ez az érték az egyenlet a *(2) képleten* látható.



6.1. ábra – Függőleges rendezés eredménye 1 (saját ábra)



6.2. ábra – Vízszintes rendezés eredménye 2 (saját ábra)



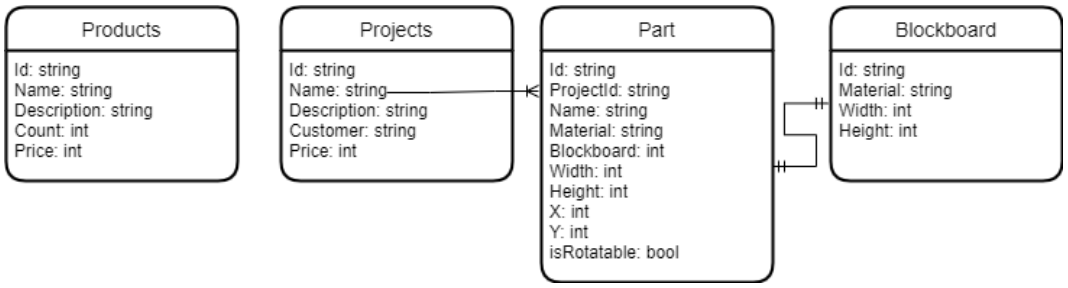
6.3. ábra – Függőleges rendezés eredménye 3 (saját ábra)

7. MEGVALÓSÍTÁS

Ebben a részben a rendszerterv alapján megvalósított megoldások, valamint a közben felhasznált megvalósítást segítő technológiák kerülnek tárgyalásra.

A **shared** komponens **model** rétegben kerültek megvalósításra a különböző termékek köztük, a kivágandó alkatrészek és a maradék táblák leíróinak adatbázis modelljei, amik felhasználásra kerülnek mind a frontend, mind a backend rész által, illetve a különböző időleges adattárolásra és továbbításra használt objektumok leírása is. Az adatbázis tábla leírások a következő fejezetben a Backend/Data réteg leírásában lesz tárgyalva.

A **backend** komponens **data** rétegben a lokális adatbázis fájljai találhatóak meg. Ennek létrehozásához a Data storage and processing Visual Studio kiegészítő telepítése volt szükséges. Így lehetővé vált, hogy lokális MSSQL szerver fusson az alkalmazás futtatásakor.



7.1. ábra – Adatbázis táblák (saját ábra)

A **repository** rétegben valósítja meg a Shared/Model réteg által leírt adatbázismodellek alapján az adatbázis táblalétrehozást, valamint a komplexebb logikához szükséges primitív lekérdező, módosító és törlő műveletek alapját, ezzel kapcsolatot hoz létre az üzleti-logika és az adatbázis között.

A **logic** réteg vagy más néven üzleti-logika több szolgáltatást megvalósít.

Felhasználja a repository réteg adatbázis műveleteit és megvalósít egy adatbázis kezelő rendszert, ami következő fejezetben említésre kerülő, endpoint réteg által lett kivezetve.

Ebben a rétegben kerültek megvalósításra az optimalizáló szolgáltatások is melyek létrehozzák a szabásmintát, a repository réteg által szolgáltatott adatok segítségével. Ezen szolgáltatások be- és kimenetei egy interfész segítségével kerültek leírásra így egyszerűen minimális változtatással kiválasztható a számunkra leginkább megfelelő algoritmust használó szolgáltatás. Az optimalizáló algoritmusok bemeneteként várja az optimalizálandó alkatrészeket tartalmazó listát, valamint három paramétert, amik a maradékok —a szabásmintát az alkatrészek elhelyezése után maradó fel nem használt alapanyag darabok— adatbázisban történő elhelyezését, az adatbázisban levő maradékok felhasználását szabályozzák és az utolsó paraméter segítségével a fűrészlap vastagságát lehet beállítani.

Itt került továbbá implementálásra az optimalizálás által adott megoldás vizualizálását végző szolgáltatás is. Az előző bekezdésben említett interfész által leírt kimenet alapján a System.Drawing.Common MIT licensz alapján felhasználható NuGet csomag segítségével egy,

illetve több Bitmapet generál, a szolgáltatást használó művelet igényei szerint. Ha a frontendi megjelenítésre szeretnénk ezt használni, akkor egy az összes táblát tartalmazó Bitmapet generál, PDF készítés esetében minden táblát egy külön Bitmapként egy listába helyezve generálja le.

Megrendeléskezelő szolgáltatás segítségével, projekteket vihetünk fel a rendszerbe, amiben eltárolhatjuk a folyamatban lévő ügymeneteket.

A raktárkészlet nyilvántartó szolgáltatás segítségével, nem alkatrész típusú adatokat vihetünk be a rendszerbe, és kezelhetjük ezeket.

Fontos megemlíteni azt, hogy mivel System.Drawing.Common csomag, kizárólag Windows operációs rendszeren futtatható, ezért ennek a NuGetnek a használata limitálja az alkalmazást futtató környezetet.

Az **endpoint** réteg tartalmazza az API végpontokat. A rétegben 5 kontroller került megvalósításra.

- PartController: Az alkatrészek adatbázisműveleteit (CRUD) tartalmazó kivezetés
- BlockboardController: A szabásminta létrehozás során keletkező maradék alapanyag darabok adatbázisműveleteinek (RD) kivezetése
- OptimizerController: Az optimalizáló komponens elindítását végzi, a frontend számára továbbítja a logic réteg szabásminta generáló szolgáltatása által létrehozott bitmap szabásmintát, valamint a letölthető PDF formába összecsomagolja és szintén továbbítja azt a frontend felé
- StockController: A többi termékleíró raktárkészletnyilvántartó (CRUD) kivezetése
- ProjectController: A szabásmintákhoz és termékleírókhoz vásárlót vagy megrendelőt hozzárendelő kivezetése, és ennek adatbáziskezelő logikájának kivezetése.

Az API végpontok dokumentálása Swagger segítségével történt meg, amihez szükség volt a Swashbuckle.AspNetCore MIT licenszes NuGet csomag telepítésére. A PDF összeállításához a PdfSharpCore, szintén MIT licenszű — .NetFrameworkhöz készült PdfSharp felhasználói által .NetCorera portolt megvalósítása — csomagot használtam.

A frontend komponens elkészítéséhez a BlazorWebAssembly technológiát használtam. Az oldalak kinézetét Bootstrap segítségével készítettem el.

A webalkalmazás több oldalból áll.

A **PartAdder** oldalon lehetséges a szabásminta alapjául szolgáló téglalap alakú alkatrészeket felvinni, és elmenteni az adatbázisba. Valamint törölni is lehetséges őket.

A **CuttingPattern** oldalon a szabásminta megjelenítése történik, ami a projectId alkatrész jellemző megadása után, valamint a generálás paramétereinek beállítása után képként megjeleníti az ahhoz a projectId-hoz tartozó szabásmintát. A PDF gombra kattintva PDF fájlként elkezdi a böngésző letölteni a szabásmintát, amely minden oldalon tartalmaz egy alapanyag táblát, megjelölve méretét, a táblán az alkatrészek ajánlott elhelyezését, rajtuk a

méretük és nevükkel. Az alapanyag tábla alatt látható, hogy hányadik táblát láthatjuk, mi a projekt azonosítója, illetve a generálás napjának dátumát.

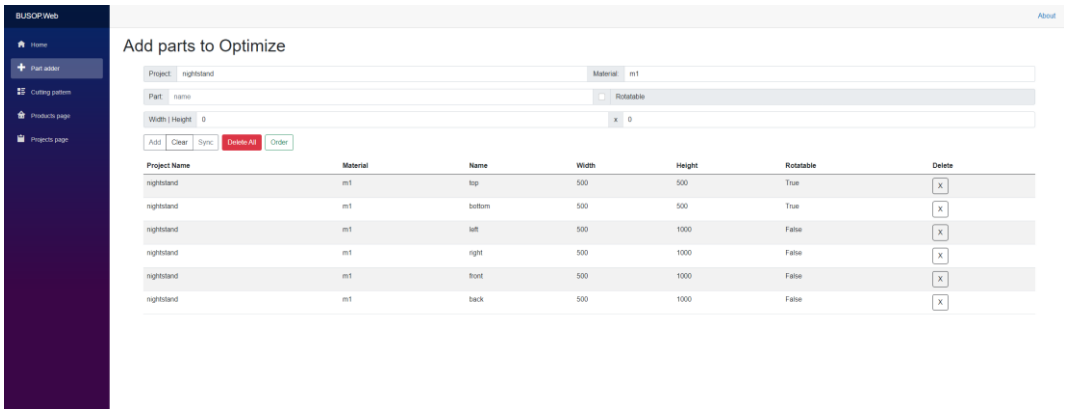
Products oldalon a raktárkészlet nyilvántartó rendszer kezelhető.

Projects fülön pedig a megrendeléseket lehet nyilvántartani, felvinni a vevőt, megrendelés részleteit, szabásmintát hozzárendelni.

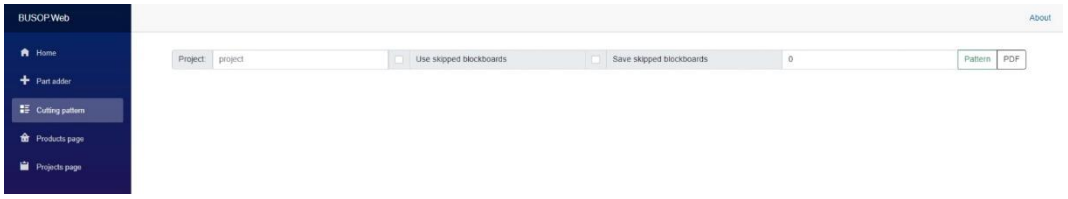
A különböző szolgáltatásokat tartalmazó osztályokat dependency injection segítségével kerülnek példányosítva.

7.1. Az alkalmazás felülete

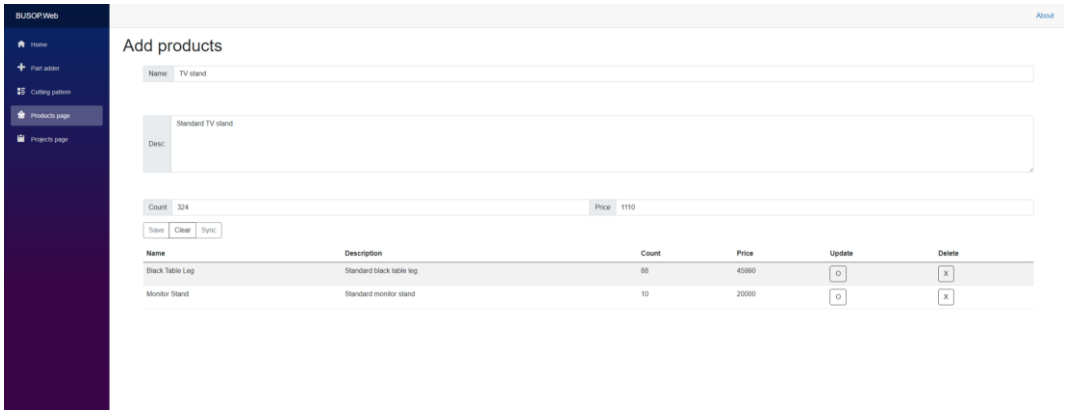
Az alábbi alfejezetben a megvalósított alkalmazás néhány felülete látható.



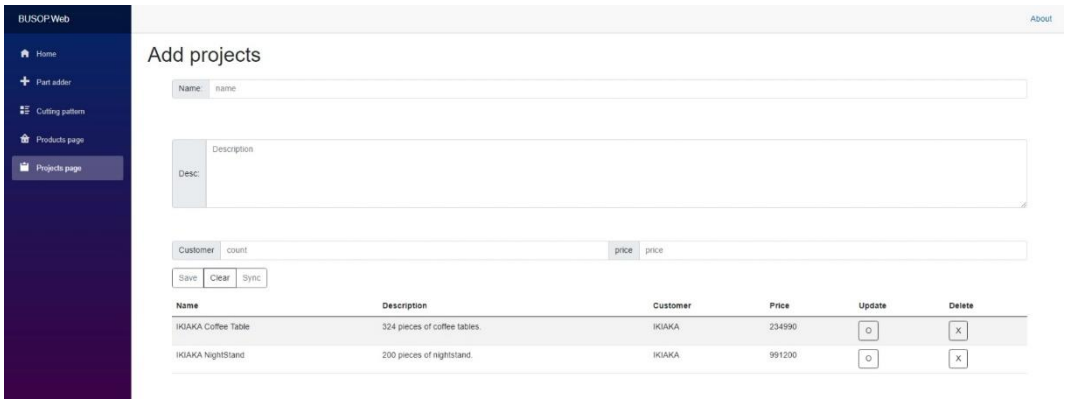
7.2. ábra - Alkatrész felvivő űrlap az alkalmazásból (saját ábra)



7.3. ábra - Szabásminta beállításait bekérő űrlap(saját ábra)



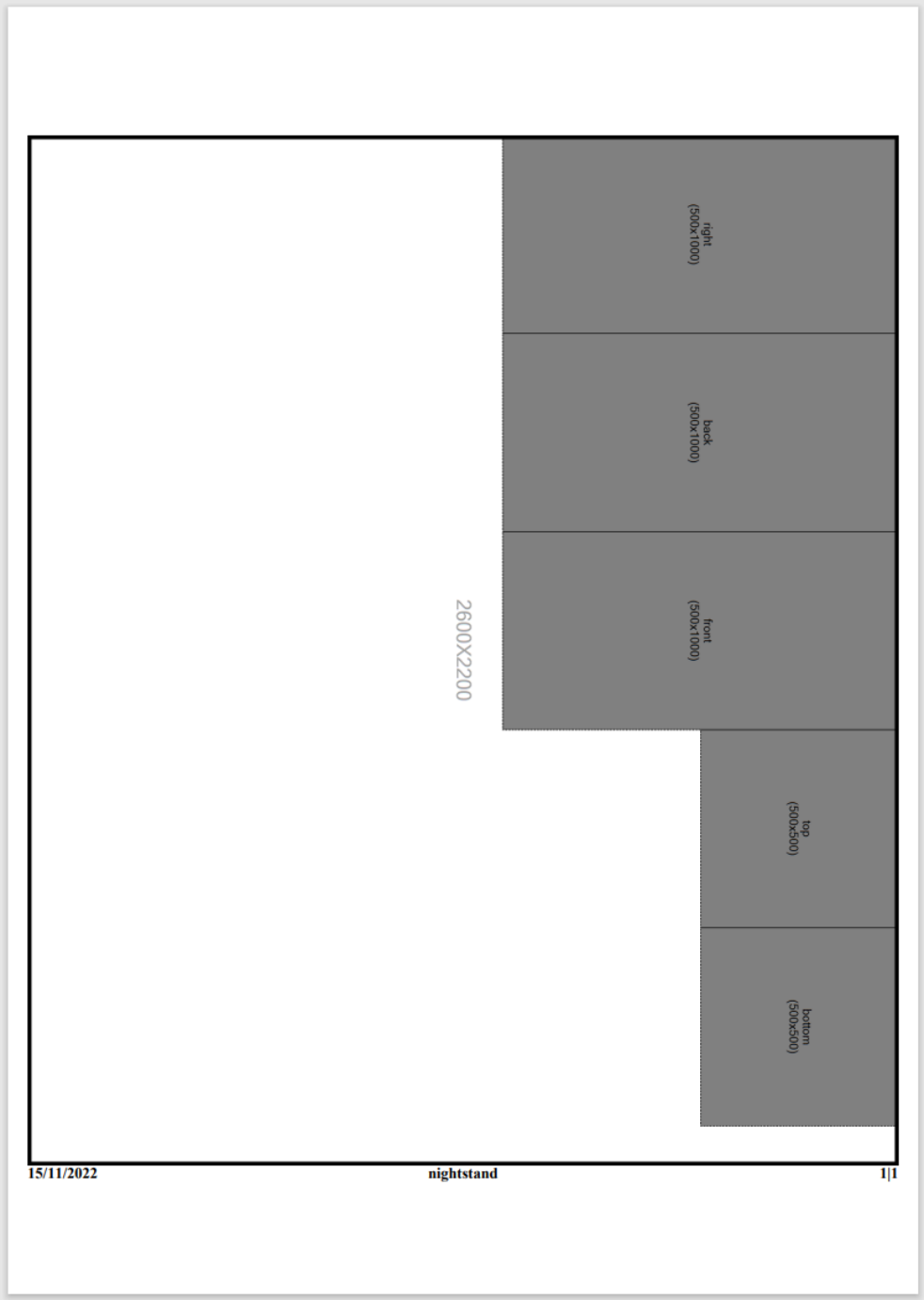
7.4. ábra - Raktárkészlet kezelő oldal (saját ábra)



7.5. ábra - Project menedzsment oldal (saját ábra)

7.2. PDF szabásminta

Az alábbi alfejezetben a szoftver által létrehozott PDF formátumú szabásmintáról lesz szó. Megtalálható rajta a generálás dátuma, a tábla mérete, az alkatrészek mérete és neve, a projekt neve, valamint az alapanyag sorszáma ez látható a *7.6. ábrán*.



7.6. ábra - PDF szabásmint (saját ábra)

8. EREDMÉNYEK ÉRTÉKELÉSE

A kész szoftvert elemezve megállapítható, hogy a kitűzött céloknak megfelelő funkciókkal rendelkező modern architektúrájú, és korszerű technológiát használó szoftvert kaptunk. A funkciókat tekintve, elmondható, hogy teljesíti a követelményekben leírtakat, valamint az elvártaknak megfelelően szabásmintákat generál a bevitt adatokból, melyeken láthatóak a korábban megfogalmazott információk.

8.1. Optimalizálás értékelése

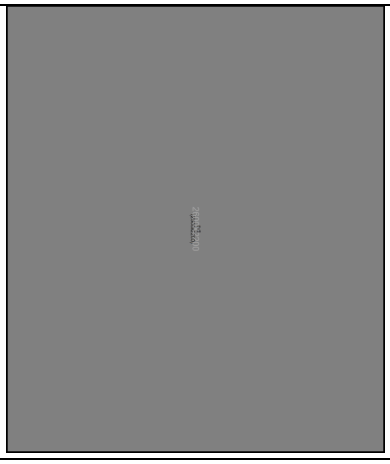
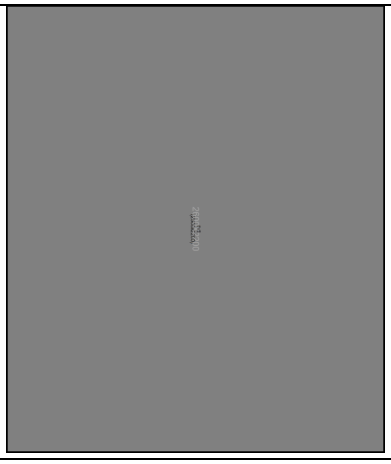
Optimalizálásra két algoritmus került megvalósításra a szoftverben.

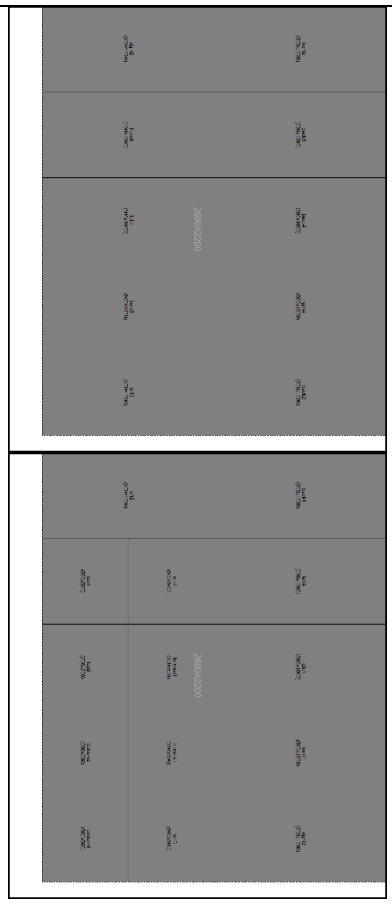
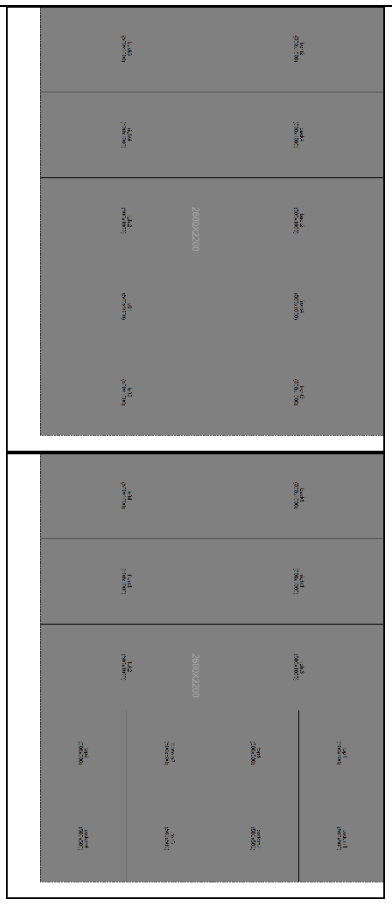
Az első a saját magam által megalkotott sík felosztása módszer alapján működő optimalizáló, a másik pedig a két-fázisú guillotine vágás algoritmusa. A két módszer által adott megoldásokat a 6.2. fejezetben leírt képlet által adott eredmény segítségével hasonlítottam össze, aminek végeredményét az alábbi táblázatban láthatjuk a kisebb érték jelenti a nagyobb jóságot.

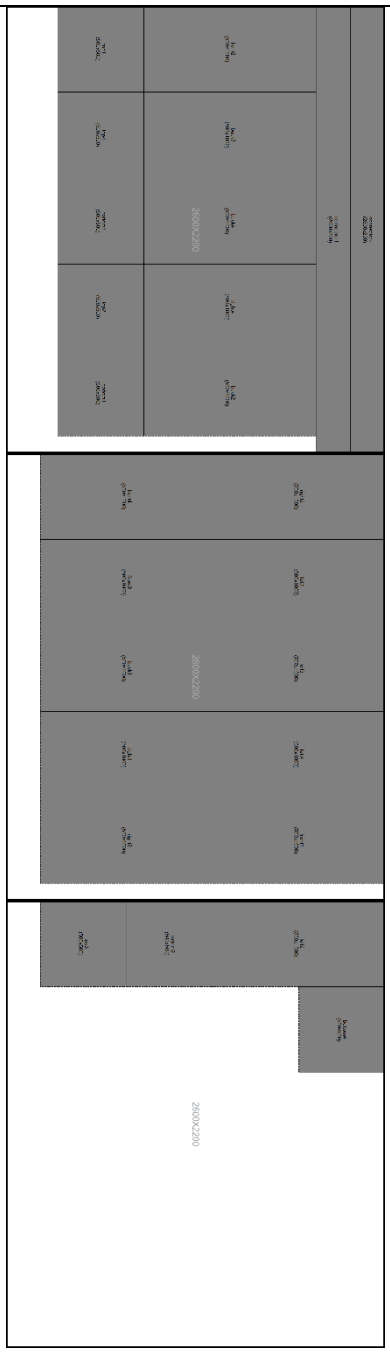
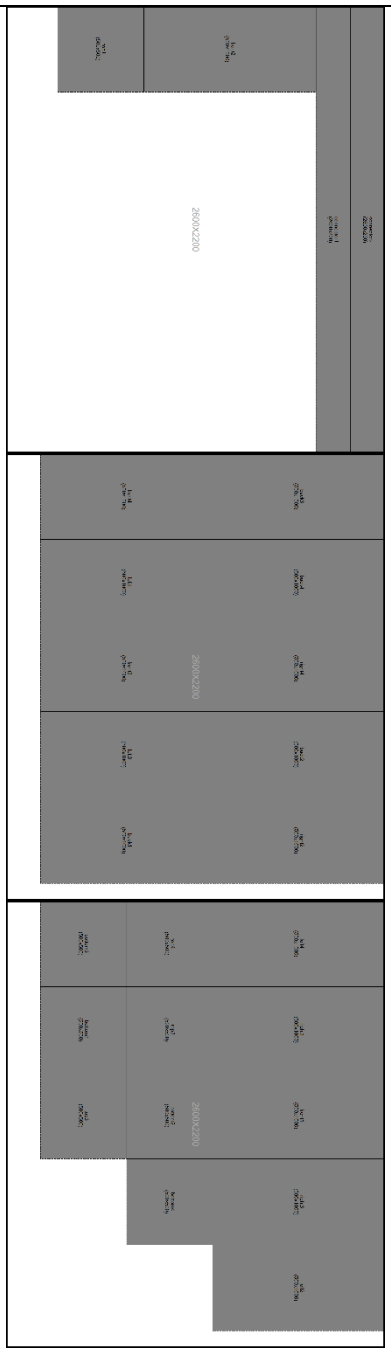
Maradék alapanyagok felhasználása nélkül (alapanyag mérete: 2600x2200)			
	Teszt eset	Sík felosztása	Két-fázisú guillotine vágás
1.	Egész alapanyagot kitöltő alkatrész (2600x2200)	0	0
2.	Két darab egész alapanyagot kitöltő alkatrész (2db 2600x2200)	0	0
3.	Két darab különböző állású az egész alapanyagot kitöltő forgatható alkatrész (2600x2200, 2200x2600)	0	0
4.	Egy alapanyagot nem kitöltő egy alkatrész (2000x2000)	0.30745	0.30745
5.	Alapanyagnál nagyobb egy alkatrész (3000x3000)	nincs szabásminta	nincs szabásminta
6.	Két darab az alapanyagnál nagyobb alkatrész (2db 3000x3000)	nincs szabásminta	nincs szabásminta
7.	Két darab különböző állású az egész alapanyagnál nagyobb forgatható alkatrész (3000x4000, 3000x4000)	nincs szabásminta	nincs szabásminta
8.	Két darab téglatest, aminek oldalai elférnek egy táblán (4db 500x500, 8db 500x1000)	0.02703	0.02703

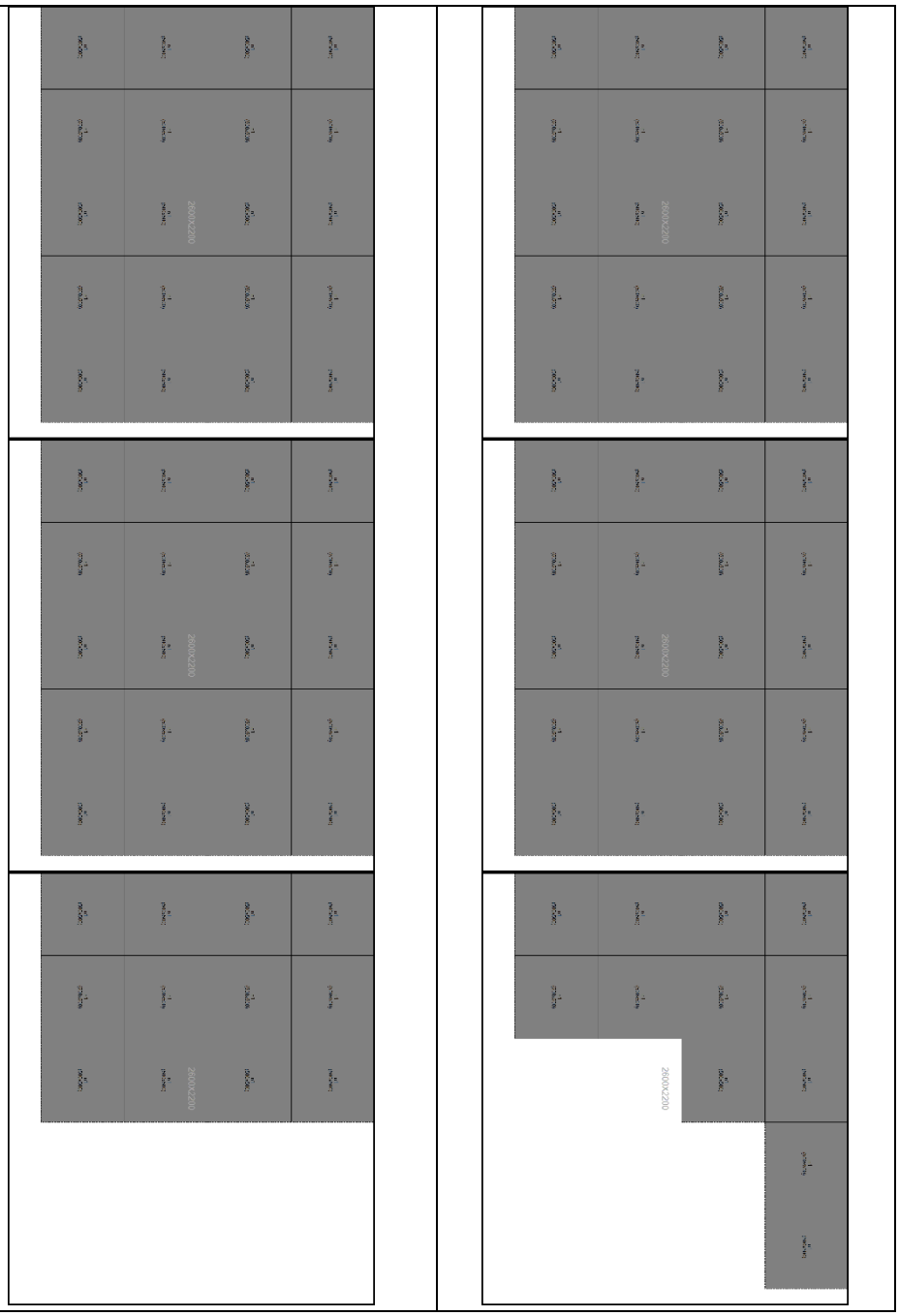
9.	Négy darab téglatest, aminek oldalai elférnek két tábla alapanyagon (8db 500x500, 16db 500x1000)	0.02149	0.02149
10.	Négy darab téglatest, aminek oldalai elférnek két tábla alapanyagon és egy hosszú alkatrész, ami elfér az egyik felhasznált alapanyag táblán (8db 500x500, 16db 500x1000, 2600x200)	0.00897	0.31039
11.	Négy darab téglatest, aminek oldalai elférnek két tábla alapanyagon és két hosszú alkatrész, amik elférnek a felhasznált alapanyag táblán (8db 500x500, 16db 500x1000, 2db 2600x200)	0.24409	0.24409
12.	Ötven kettő darab azonos méretű alkatrész (52db 500x500)	0.10263	0.10263
13.	Két különböző méretű alkatrészekből több darab (13db 500x1000, 7db 500x500)	0.12971	0.12971
14.	Két darab téglatest, aminek oldalai elférnek két tábla alapanyagon és egy hosszú alkatrész, ami elfér az egyik felhasznált alapanyag táblán (4db 500x500, 8db 500x1000, 2600x200)	0.15386	0.96233

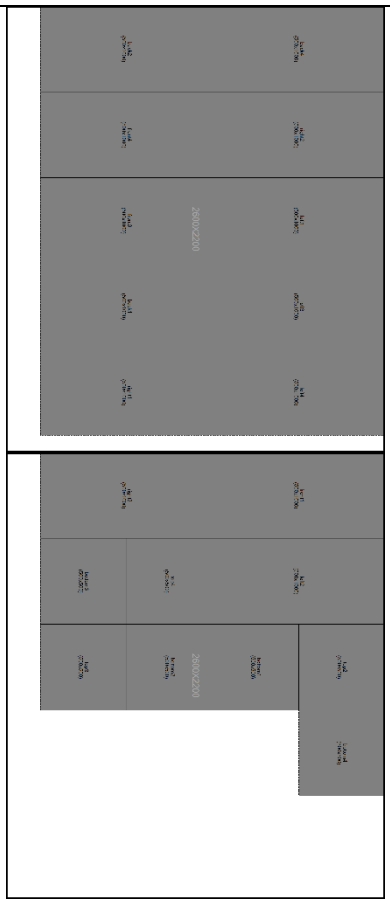
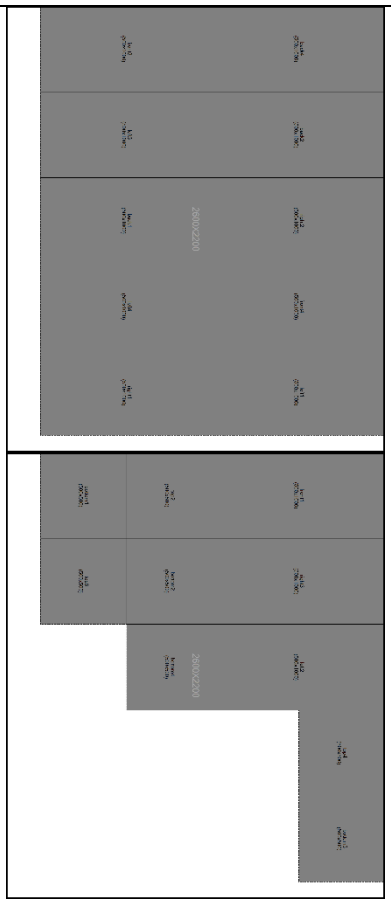
8.1. táblázat – Első tesztesetek leírása

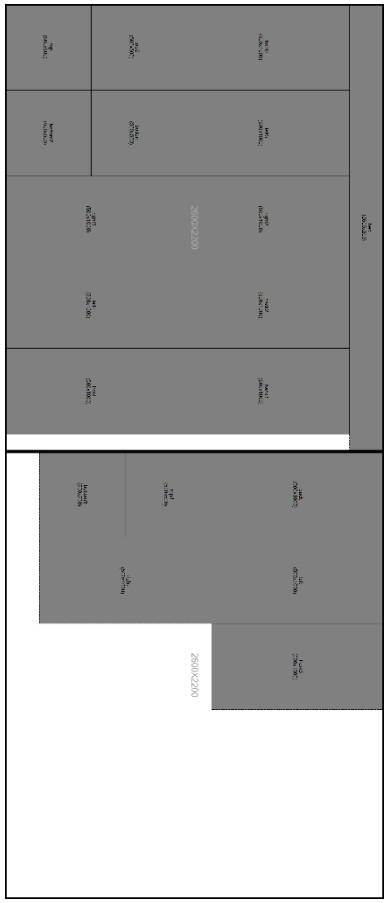
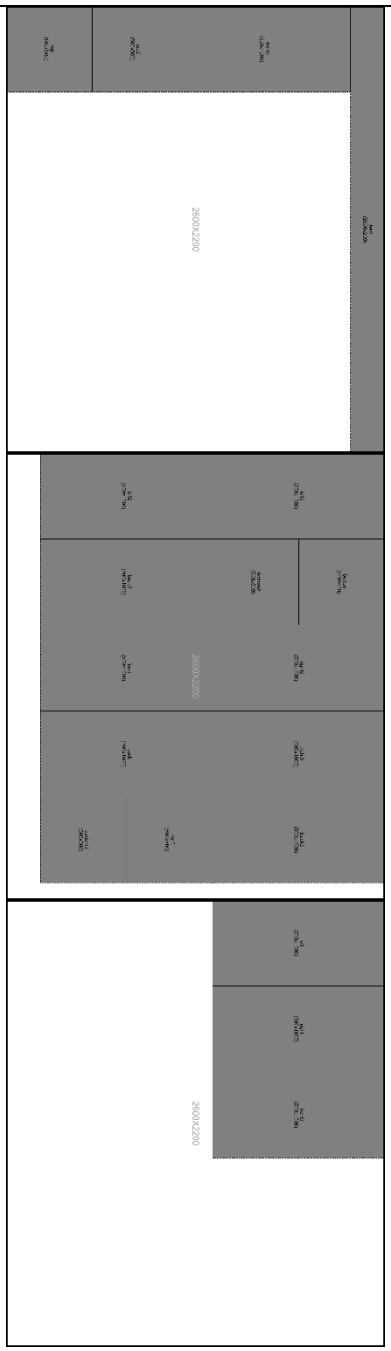
Szabásminták		
	Sík felosztása módszer	Két-fázisú guillotine
1.		

9.			

11.		

12.		

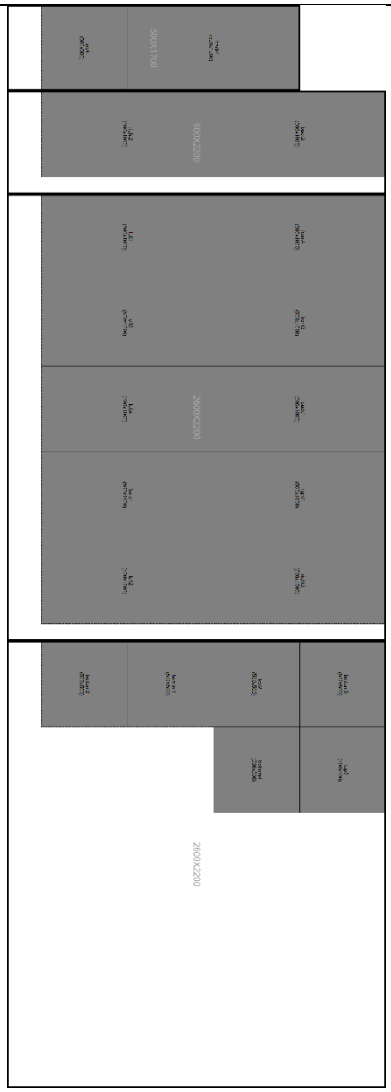
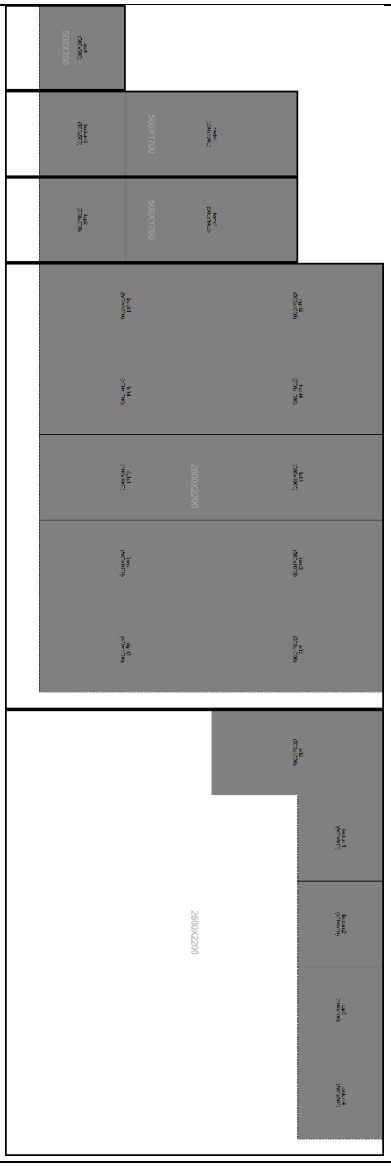
13.		

14.		

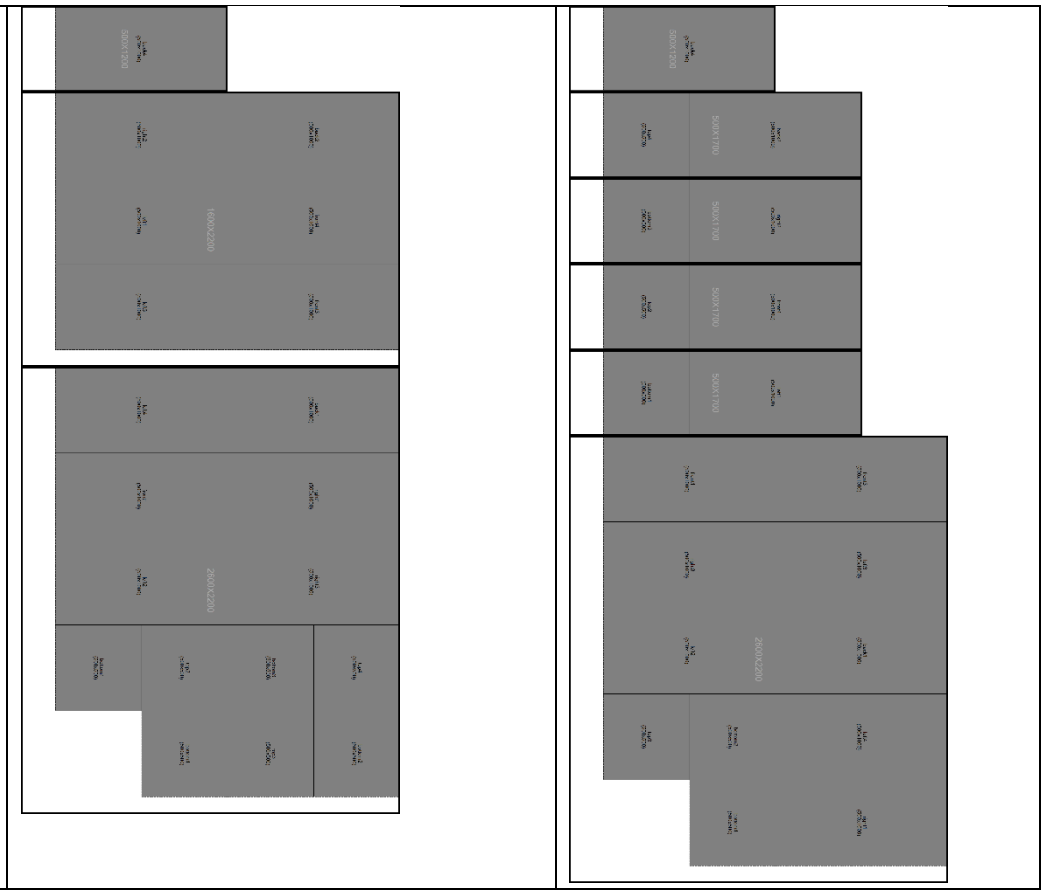
8.2. táblázat – Az első tesztesetek adataival generált szabásminták bizonyos eseteknél

Maradéktábla felhasználással			
	Teszt eset	Sík felosztása	Két-fázisú Guillotine
I.	Két különböző méretű alkatrészekből több darab a saját maradékát felhasználva (13db 500x1000, 7db 500x500)	0.32246	0.60320
II.	Két különböző méretű alkatrészekből több darab kétszeri futtatásból származó maradékokkal (13db 500x1000, 7db 500x500)	0.09667	0.09022
III.	Ötven kettő darab azonos méretű alkatrész a saját maradékainak felhasználásával (52db 500x500)	0.35853	0.34857
IV.	Ötven kettő darab azonos méretű alkatrész kétszeri futtatás maradékainak felhasználásával (52db 500x500)	0.21510	0.21259

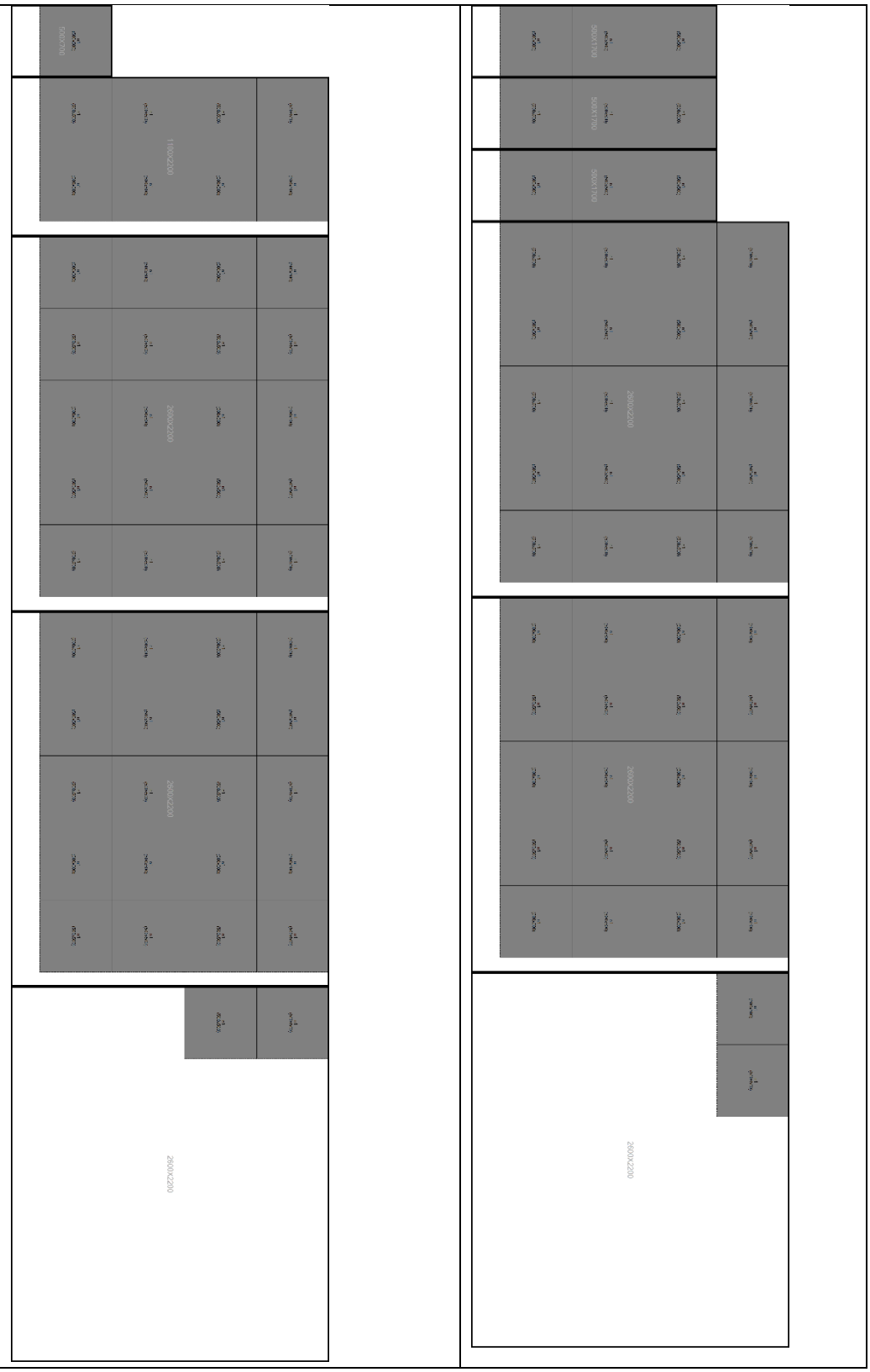
8.3. táblázat – Második teszt teszteseteinek leírása

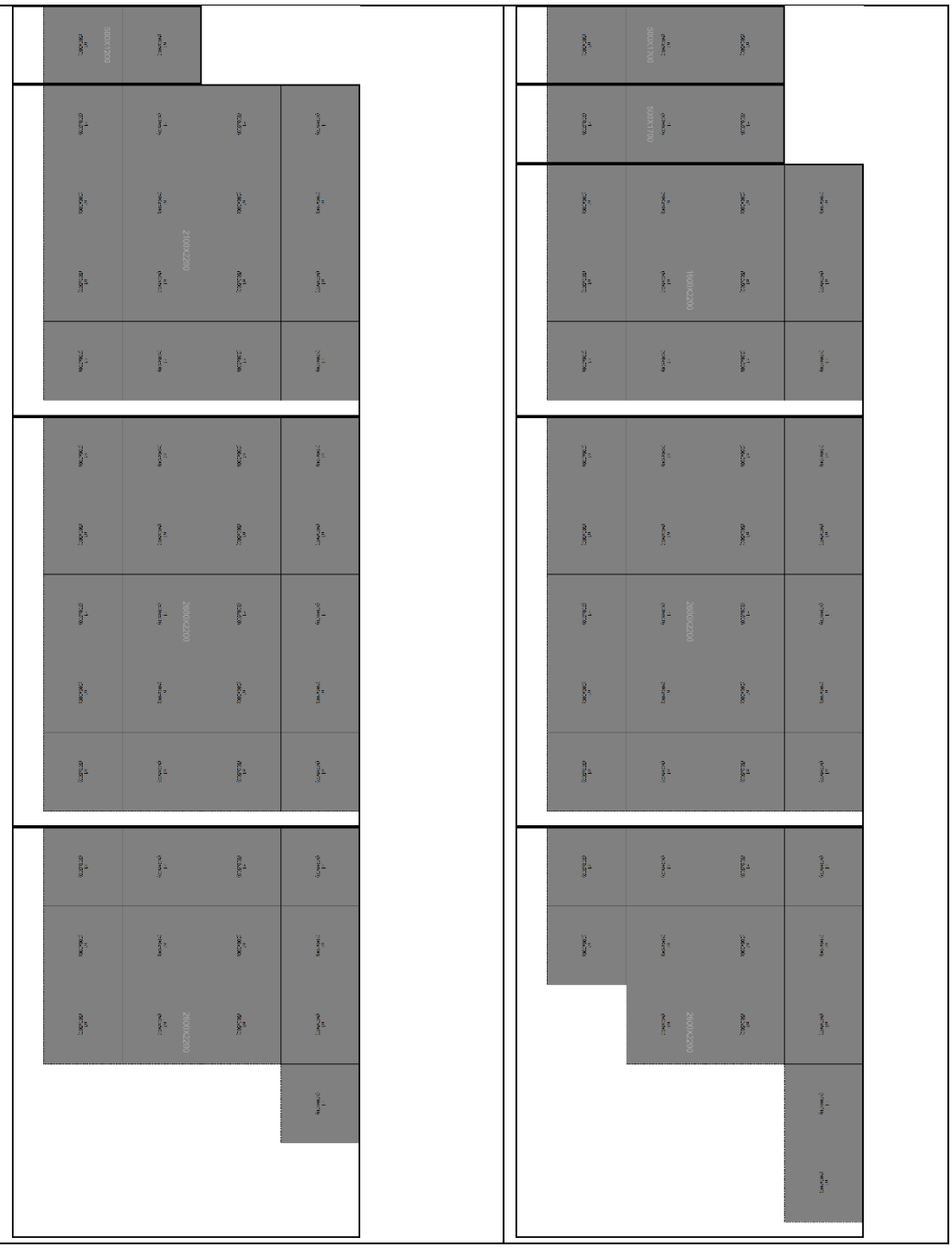
Szabásminták		
	Sík felosztása	Két-fázisú Guillotine
I		

II



III.



IV.	

8.4. táblázat – A második teszt testesetei alapján generált szabásminták

A szabásmintálat látva és megtekintve a jóság értékeket, azt láthatjuk, hogy a két módszer hasonló eredményeket ad a legtöbb esetben, a felhasznált alapanyag mennyisége azonos, és az elrendezésben a különbség az utolsó néhány elem pozíciója különbözik.

Azonban vannak olyan esetek 11, 12, 14 amikor a szabásminta elrendezése sokkal hatékonyabb, a saját módszeremet használva.

Megállapítható, hogy mindkét módszer hatékony megoldást ad, olyan esetekben amikor az alkatrész fajtákból több egyforma, ugyanakkor a Guillotine vágás problémába ütközik, ha egy nagy mértékben eltérő paraméterekkel rendelkező alkatrész kerül az optimalizálandó alkatrészek közé.

A saját algoritmusom, jobban kezeli, ezt az esetet, mivel a másik megoldás algoritmusa csak akkor lesz effektív, ha második fázisban az első fázisban lerakott kezdőelemhez hasonló méretűek az alkatrészek.

A maradékot felhasználó esetekben, szintén hasonló eredményt kapunk.

Összeségében elmondható, hogy maradék felhasználás nélkül eredményesebb a saját algoritmusom, viszont a maradék felhasználásnál ilyen következtetés nem vonható le. Mivel az előző optimalizálás maradékait használjuk fel, a sík felosztása módszer néhány esetben kevesebb alapanyagot használ fel, kevesebb lesz a maradék, így ez azt eredményezheti, hogy a maradék felhasználásával, több új alapanyagra van szükség, amiből több maradék keletkezik. Mindazonáltal figyelembe kell venni, hogy a saját algoritmusom jobbnak tekinthető elrendezése, a maradék alapanyag táblákat nagyobb hatásfokkal tudja feldolgozni.

8.2. Szabásminta értékelése

A szabásmintát megnézve megállapíthatjuk, hogy láthatóak rajta a követelményekben leírtak, valamint PDF formátumban letöltve nyomtatásra kész állapotban kapjuk meg azokat. Minden felhasznált tábla utólagos átméretezés nélkül, kinyomtatható egy A4-es méretű lapra, valamint több tábla felhasználása sem okoz gondot a sorszámozásuk miatt. A fűrészlap vastagságának beállításával, pedig az alkatrészek között pont akkora távolság jelenik meg, amennyi veszteséget okoz a vágás.

8.3. Funkcionalitás áttekintése

Funkciókat tekintve, az alkalmazás alkalmas a szükséges alkatrészek felvitele után szabásmintát generálni, amit képként, PNG formátumban, illetve PDF formában a felhasználó számára rendelkezésre bocsát. Fontos megemlíteni, még, hogy nem minden esetben tud az alkalmazás optimális eredményt adni és az alkatrészeket csak a webalkalmazás partAdder oldalán található űrlap segítségével lehetséges betáplálni.

A raktárkészet és megrendelés kezelő komponensek ellátják a feladatukat, azonban megvalósításuk, feltehetőleg nem fed le minden való életben létező igényt. Ennek oka, hogy a szoftver tervezésekor és a követelmények megállapításakor a hangsúly az optimalizáló komponensre került. Az igények felmérésre sem kerültek.

A maradékokat kezelő rendszer működését tekintve egyetlen limitációval rendelkezik, nem kerülnek automatikusan szelektálásra a benne tárolt és nyilvántartott elemek, így ezt manuálisan kell elvégezni, illetve sok szabásminta generálás után rengeteg felhasználásra alkalmatlan maradék lesz eltárolva.

8.4. Összehasonlítás hasonló rendszerekkel

A hasonló rendszerekkel foglalkozó fejezben megvizsgált szoftverekkel összevetve, elmondható, hogy több funkcióval rendelkezik, mint a megtekintett szoftverek nagyrésze, azonban ezeknek egyike sem annyira mély, és kidolgozott. A szabásminta generáló komponenst tekintve egy szinten van a többi a szoftverrel. A szoftverem egy átmenetet képez tudásban és lehetőségekben a PaneCutter és az Optimik4 alkalmazás között, mivel előbbi szabásminta generálása volt a leginkább kezdetleges, utóbbi pedig rendelkezett az én szoftverem funkcióival, azonban előbbinél az itt leírt alkalmazás szabásminta generálása jobbnak tekinthető, illetve több algoritmust tud használni, utóbbi egyéb funkcióit ugyan tartalmazza, de nem olyan kidogozottan.

Továbbfejlesztési lehetőségként meg kell említeni, a .CSV vagy egyéb beviteli forrásként kezelhető fájlbeolvasás funkció lefejlesztését.

ÖSSZEFOGLALÁS

A dolgozatban egy asztalosipari szoftvert terveztem és valósítottam meg, aminek során megvizsgálásra került a bútorlapszabászat, az abban használt alapanyagok és technológiák, összehasonlítottam a piacon található létező alkalmazásokat. Ezek alapján a leírások alapján létrejött egy követelményrendszer, az itt megtervezendő és megvalósítandó szoftverre.

Különböző heurisztikus és determinisztikus optimalizáló algoritmusokat tekintettem meg, amelyek közül egy, valamint egy saját magam által kitalált algoritmus megvalósításra és implementálásra került a szoftverben.

Elkészítettem egy rendszertervet a fentebb említett kutatások és kritériumok szerint, a szoftver megvalósítására, ami alapján egy modern webalkalmazás fejlesztettem.

Az elkészült szoftver ezután tesztelésre került. Több teszteset segítségével a szabásminta generálás funkció és a maradék alapanyagok felhasználásával történő szabásminta generálás jósága lett megvizsgálásra. A kész szoftver funkcióinak tudása, és limitációit végül összevettem a megvizsgált szoftverekkel.

IRODALOMJEGYZÉK

[1] Faipari és szabászgép működési információk: (https://faipar.hu/hirek/gep-es-szerszam/9203/fueggoleges-vagy-vizszintes)
(utolsó megtekintés: 2022.05.01)

[2] Vízszintes lapszabász gép: (https://paliszander.hu/robland-z3200z300x3-lapszabaszgep-44-kw-15736)
(utolsó megtekintés: 2022.05.01)

[3] Függőleges lapszabász gép: (https://etalon.hu/termek/etalon-fuggoleges-lapszabaszgep/)
(utolsó megtekintés: 2022.05.01)

[4] BIPPO szoftver weboldala: (https://www.berenyisoft.hu/bippo/)
(utolsó megtekintés: 2022.05.01)

[5] PaneCutter szoftver weboldala: (http://www.panecutter.eu/HU/index.htm)
(utolsó megtekintés: 2022.05.01)

[6] AMORF szoftver weboldala: (http://amorfszoft.hu/amorf_b.html)
(utolsó megtekintés: 2022.05.01)

[7] Optimik 4 szoftver weboldala: (https://www.optimik.com/description.html)
(utolsó megtekintés: 2022.05.01)

[8] Optimik 4 szabásminták (https://www.optimik.com/plan.html)
(utolsó megtekintés: 2022.05.01)

[9] P. C. Gilmore, R. E. Gomory (1965). Multi-Stage Cutting Stock Problems of Two or More Dimensions. Operations Research. 13. 10.1287/opre.13.1.94.

[10] F. Dusberger, G. R. Raidl: Solving the 3-Staged 2-Dimensional Cutting Stock Problem by Dynamic Programming and Variable Neighborhood Search, Electronic Notes in Discrete Mathematics, Volume 47, 2015, Pages 133-140, ISSN 1571-0653,

[11] M. Chen (2008). A Greedy Algorithm with Forward-Looking Strategy, Greedy Algorithms, Witold Bednorz (Ed.), ISBN: 978-953-7619-27-5, InTech, Available from: http://www.intechopen.com/books/greedy_algorithms/a_greedy_algorithm_with_forward-looking_strateg

[12] E. Hopper, B.C.H Turton: An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem, European Journal of Operational Research, Volume 128, Issue 1, 2001, Pages 34-57, ISSN 0377-2217,