

Miskolci Egyetem



GÉPÉSZMÉRNÖKI- ÉS INFORMATIKAI KAR
GYÁRTÁSTUDOMÁNYI INTÉZET

TDK DOLGOZAT
Automata Gyártósori Ciklusidő kiegyenlítés

KÉSZÍTETTE:
Hablik Zsolt
Gépészmérnök szakos hallgató

KONZULENS:
Dr. Sztankovics István
egyetemi docens

Miskolc, 2024.

Eredetiség nyilatkozat

Alulírott Hablik Zsolt, a Miskolci Egyetem Gépészmérnöki és Informatikai Karának hallgatója büntetőjogi és fegyelmi felelősségem tudatában kijelentem és aláírásommal igazolom, hogy ezt a dolgozatot saját magam készítettem, a benne leírt vizsgálatokat – ha ezt külön nem jelzem – magam végeztem el, és az ismertetett eredményeket magam értem el. Adatokat, információkat csak az irodalomjegyzékben felsorolt forrásokból használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Miskolc, 2024.04.03



.....
a hallgató aláírása

A konzulens nyilatkozata

Alulírott, Dr. Sztankovics István a Miskolci Egyetem Gyártástudományi Intézetének oktatója a

Automata Gyártósori Ciklusidő kiegyenlítés

című TDK dolgozatot beadásra alkalmasnak ítélem.

Miskolc, 2024. április 11.


.....
a konzulens aláírása

Tartalom

Előszó.....	5
1. Bevezetés a gyártósori ciklusidő kiegyenlítés alapjaiba.....	6
1.1 Miről szól a gyártósori ciklusidő kiegyenlítés, miért alkalmazzuk?.....	6
1.2 Manuális gyártósori ciklusidő kiegyenlítés bemutatása mintán keresztül	8
1.3 Gyártósori ciklusidő kiegyenlítés lehetőségei	10
1.4 Divergens gondolkodás a Túlélési torzítás elméletének bemutatásával	12
1.5 Gyakorlati példa a divergens gondolkodása	14
2. Tecnomatix Plant Simulation szoftver ismertetése.....	17
2.1 A szimuláció építőelemei és működése	18
2.2 Mintasor felépítése	20
2.3 Modell validálása	23
2.3.1 Kulcs mérőszámok	24
2.3.2 Modell validálása.....	24
2.4 Plant Simulation összefoglalva	26
3. Automata ciklusidő kiegyenlítő algoritmus.....	28
3.1 Összes lehetséges előfordulás matematikai modellje	28
3.2 Dolgozói eloszlástáblázat automatikus létrehozása	29
3.3 Táblázat optimalizálása.....	32
3.4 A dolgozó eloszlástáblázatot készítő algoritmus bemutatása	34
3.5 A szimulációs algoritmus bemutatása	37
4. Eredmények bemutatása	40
4.1 Szimuláció eredményei.....	40
4.1.1 Fejlesztési javaslat	42
4.1.2 Exp 2958 részletesebb elemzése.....	45
4.2 Számítási kapacitás csökkentés.....	45
5. Összefoglalás	47
6. Felhasznált irodalom.....	52

Előszó

A TKD dolgozatomban egy, a való életből vett példák keresztül bemutatom, hogyan lehet egy termelésőr ciklusidő kiegyenlítését elvégezni, majd ezt egy általam készített algoritmus segítségével fejleszttem.

Ehhez először bevezetem az olvasót az alapfogalmakba, bemutatom, hogyan tudunk egy ciklusidő kiegyenlítés feladatot önállóan elvégezni és azt, hogy ennek milyen hátrányai vannak. Kitérek a gondolkodásmódunk korlátaira és egy feladaton keresztül bárki letesztelheti azt, hogy képes-e *divergens módon*, új nézőpontok szerint megközelíteni egy feladatot.

Ezt követően a bemutatott feladatban elvégzett ciklusidő kiegyenlítést el fogom végezni automata módon is, ehhez a Simens Tecnomatix Plant Simulation programját választottam. A program használatával felépítem a Minta soromat és létrehozom azt a Plant Simulation metódust, amit képes lesz egy Kísérlettervezésbe futtatni a bemenő adatokat.

A sor validálása után elindítom ezt az algoritmust, ami el is végzi a ciklusidő kiegyenlítést. Kitérek arra, hogy ehhez milyen logika szerint építettem fel a programkódot, mi a mögötte lévő matematikai modell, hogyan lehetett gyorsítani a kód futását és erre miért volt szükség.

A hozott eredményekből látni lehet, hogy több olyan jó megoldás is lehetséges, amivel képes voltam a hatékonyságon növelni. Ezek között szerepel olyan, ami a valóságban nem kivitelezhető, vagy csak elméleti szinten működik, így ki kell választanom azt, ami az adott gyártási környezetben számomra a legmegfelelőbb. A több száz opciót le is szűkítem egyetlen lehetőségre, amit végül kicsit részletesebben is elemzek.

A dolgozatom végén bemutatom az algoritmus előnyeit, milyen esetekben és kik számára lehet hasznos. Kitérek a hátrányaira, arra, hogy milyen fejlesztési lehetőségeket rejt még magában az algoritmus, és hogy miért legyünk óvatosak a használatával.

1. Bevezetés a gyártósori ciklusidő kiegyenlítés alapjaiba

Első lépésben tekintsük át azt, hogy miért is van szükség a gyártósorok ciklusidő kiegyenlítésére, és hogy a folyamatmérnökök hogyan végzik ezt a napi munkájuk során. Tisztázzuk az idővel kapcsolatos alapfogalmakat, mert az a személyes tapasztalatom, hogy bár létezik erre a megfelelő szakmai nyelvezet, sajnos az egyes vállalati kultúrákba máshogy épült be, így cégenként más-más lehet ezek értelmezése. A dolgozatomban ezeket egységesen fogom kezelni, ezért szükséges ezek tisztázását.

Majd egy példán keresztül végig vezetem azt, hogyan kell a ciklusidő kiegyenlítést manuálisan megoldani és ennek milyen korlátjai vannak. A későbbiekben ezt a kézzel végzett kiegyenlítést fejleszttem egy gépi kódon keresztül úgy, hogy több szempontot megvizsgálva képesek legyünk a legjobb megoldás megtalálására.

1.1 Miről szól a gyártósori ciklusidő kiegyenlítés, miért alkalmazzuk?

A modern gyárakban a nagysorozatú vagy a tömegszerű gyártás megvalósítása kötött ütemű szerelősorokban történik, amivel egyenletes termelés biztosítható a vevői igénynek megfelelő ütemben. Ez a megoldás egyszerű és kedvező a termelő vállalatok számára, azonban egyből fel is veti a szűk keresztmetszet problémáját: a teljes termelősor ciklusideje legjobb esetben is csak olyan jó lehet, mint a szűk keresztmetszet ciklusideje.

Azaz a szűk keresztmetszet a folyamat leghosszabb ciklusidővel rendelkező művelete. Könnyű felismerni, ugyanis, ha a folyamat nem tartja az egydarabos anyagáramot, akkor a szűk keresztmetszet előtt felhalmozódik a gyártott termék. Ha pedig egydarabos az anyagáram, akkor a szűk keresztmetszeten dolgozó operátor megállás nélkül dolgozik, miközben az előtte és utána dolgozó operátorok rá várakoznak.^[1]

A célunk, hogy a termelősoron a munkatartalom egy előre meghatározott létszám között úgy legyen szétosztva, hogy az egyes szerelők által végzett összes munkatartalom ideje a vevői ütemidő alatt megtörténjen. Ezek alapján, hogy a dolgozat további részében egységes nyelvezetet használjuk, definiálni kell az alábbi idővel kapcsolatos fogalmakat, melyekhez a Lean szótárt vettem alapul:

Ütemidő: az az időtartam, amilyen gyakorisággal egy terméknek el kellene készülnie a vevői rendelések alapján. Tehát ez minden esetben számított, nem pedig mért tulajdonság! ^[2]

Azaz meg kell vizsgálnom, hogy mennyi idő áll rendelkezésemre a vevői igény teljesítéséig és ezt el kell osztanom a vevő által kért darabszámmal.

Példa: A vevő számára szükség van 5 000 db termékre, egy héten belül el kell készítenem és ehhez nem szeretnék több műszakos munkarendet alkalmazni az 1. egyenlet alapján

$$t_T = \frac{7 \text{ nap} * (480 \text{ perc munkaidő} - 45 \text{ perc szünet}) * 60 \text{ másodperc}}{5\,000 \text{ db}} \quad (1)$$
$$= \frac{182\,700 \text{ sec}}{5\,000 \text{ db}} = 36,54 \frac{\text{sec}}{\text{db}}$$

Ez azt jelenti, hogy egy, az időben már állandósult folyamatban, 36,54 másodpercenként el kell készülnie egy darabnak, ha ki akarom szolgálni a vevőt. Tovább vetítve ez azt jelenti, hogy a termelősor minden egységként tekintett részének ciklusidejének 36,54 másodperc alatt kell lennie.

Ciklusidő: azt az időtartamot nevezzük így, amennyi idő alatt a dolgozó egy ismétlődő műveletet elvégez. Mérésekor két egymást követő ciklusnak bármely két azonos pontja között eltelt időt vesszük figyelembe. Tehát ez mért érték és a valóságos folyamatunkat mutatja. [2] Ki kell azonban egészíteni azzal, hogy nem csak a dolgozók esetében lehet ciklusidőről beszélni, hanem az olyan berendezések esetén is figyelembe kell ezt venni, amelyek legalább félautomatikusan képesek dolgozni. Így külön beszélnünk kell gépi ciklusidőről is.

Gépi ciklusidő: A gépi idő és a gép működtetéséhez szükséges kézi idő összege. Ennyi időre van szükség, hogy egy termék ténylegesen elkészüljön a gépen. Gyakran hibásan gépi időnek nevezik. [2]

Vegyünk például egy teszt berendezést, amely különféle vizsgálatok sorával képes azt eldönteni, hogy az adott termék megfelel-e a vevői elvárásoknak. A dolgozó feladata a termék elhelyezése a berendezésben, majd a program már magától elindul, amint a dolgozó eltávolodik a munkatérből. Amennyiben a fenti példát követve egy tesztelési ciklus ideje nagyobb mint 36,54, úgy hiába van a folyamat összes többi ciklusideje ezen idő alatt, a vevőt nem fogjuk tudni kiszolgálni.

1.2 Manuális gyártósori ciklusidő kiegyenlítés bemutatása mintán keresztül

Amennyiben a ciklusidő kiegyenlítést egy már létező sorra kívánjuk végrehajtani, úgy az alábbi adatokra van szükségünk:

- VSM (Value Stream Map), azaz egy általános kép a gyártósor anyag és információáramlásáról
- Gyártósor Layout, azaz a munkahelyek elrendezése
- Munkahelyek száma
- Munkahelyi időadatok: kézi és gépi ciklusidő szétválasztva
- Vevői ütemidő
- Létszám kapacitás, hány fővel szeretnénk az állomáson dolgozni

Vegyünk azonban figyelembe azt, hogy ebben az esetben a ciklusidő egyenletes eloszlásán kívül semmilyen más paramétert nem tekintettünk a folyamat szempontjából fontosnak. Azonban előfordulhat, hogy több hasonló jó megoldás közül azt kell választani, ahol a termék átfutási ideje a legkevesebb vagy a dolgozók által megtett út minimális. Ezen előfordulások nem feltétlen fognak egy halmazt alkotni.

A szimulációt és automata sorkiegyenlítést is ugyanazokkal, a valóságon alapuló, de kitalált értékekkel fogom alkalmazni, mint amit a mostani példában meghatározok. Így a végén összehasonlíthatóvá válik a fejlődés.

Első lépésként vizsgáljuk meg az ütemidőt, az egyszerűség kedvéért ebben a példában legyen **75 sec/db** és **3 fő** áll a rendelkezésemre. Ez azt jelenti, hogy a munkatartalmat úgy kell szétosztani, hogy minden ciklusidő ez alatt legyen.

Amennyibe egy állomás ideje nagyobb, mint az ütemidő, úgy vagy megbontom az állomás munkatartalmát és két külön állomást hozok létre, vagy ha ez nem lehetséges az állomást duplikálni kell, mind addig még folyamatos áramlást feltételezve be nem fér az ütemidő alá. Jelen példában, ha van egy végátvevő berendezésem, aminek a ciklusideje 140 sec, úgy ezt le kell másolni. Reális esetben az állomás másolása esetén fejlesztéseket is eszközölünk rajta, ezért a második berendezés ciklusideje 90 sec lesz. Így viszont az átlagos ciklusidő számítása szükséges, amellyel a két állomást reprezentáljuk. A legkisebb közös többszörös 1260 sec, ennyi idő alatt a berendezés $14 + 9 = 23$ db-ot vesz át. $1260 / 23 = 54,78$ sec az állomások átlagos ciklusideje.

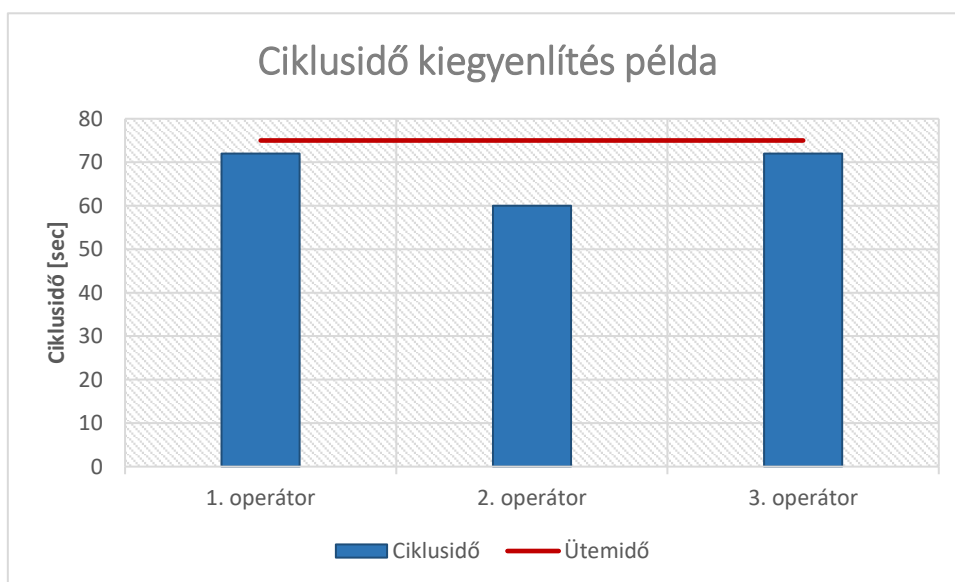
Most nézzük meg ugyanezt a dolgozókra vonatkoztatva az *1. ábrát* használva.

A kapott adatok alapján a három főnek az idejét úgy kell elosztanom, hogy minden dolgozó ciklusideje 75 sec alatt legyen, lehetőleg egymással kiegyenlítve. Ugyanis az 1.1 fejezetben említett szűk keresztmetszet hatására egydarabos áramlás esetén a legmagasabb ciklusidővel rendelkező állomás előtt fel fog torlódni az anyag, így a többi állomás várakozni kényszerül.

A táblázat utolsó oszlopába beírom, hogy az adott tevékenységet melyik dolgozó (operátor) végezze.

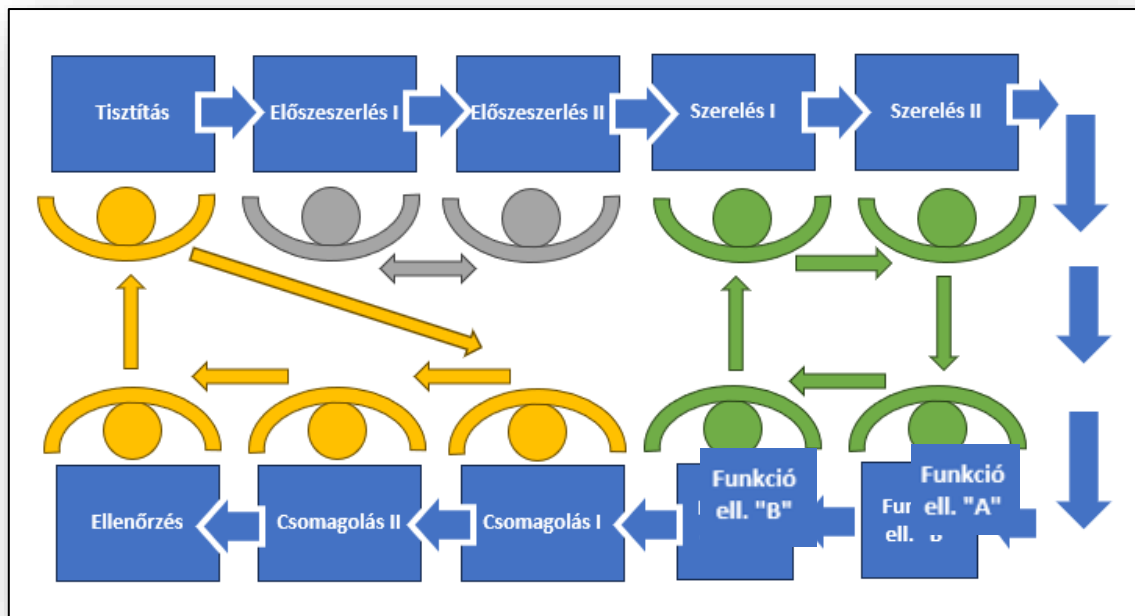
Sorszám	Állomás neve	Kézi Idő [sec]	Gépi Idő [sec]	Operátor
1	Tisztítás	18		1
2	Előszereles I	28		2
3	Előszereles II	32		2
4	Szerelés I	45		3
5	Szerelés II	15		3
6	Funkció ellenőrzés "A" berendezés	6	140	3
7	Funkció ellenőrzés "B" berendezés	6	90	3
8	Csomagolás I	25		1
9	Csomagolás II	24		1
10	Ellenőrzés, Adminisztráció	5		1

1. ábra: – Munkahely ciklusidő adatok (példa)



2. ábra – Kiegyenlített ciklusidő

A dolgozók elosztásának az eredményét a 2. ábrán szemléltetem. A diagramból látható, hogy teljesültek az elvárt feltételek. Minden dolgozó ideje 75 sec alatt van és közel egyforma a dolgozók terheltsége.



3. ábra – Példa soron a dolgozók mozgása

A 3. ábrán látható, hogy a dolgozók útvonala nem keresztezi egymást, nincs felesleges út. Így valós gyártási környezetben ez egy elfogadható megoldás lenne a ciklusidő kiegyenlítésre, amely adatokkal is megfelelően alá van támasztva.

1.3 Gyártósori ciklusidő kiegyenlítés lehetőségei

Az 1. ábrán látható, hogy több olyan munkafolyamat is van, amik párban szerepelnek. Ezek az Előszerszerelés, Szerelés, Funkció ellenőrzés, Csomagolás. Miért kell ezeket az állomásokat több részre bontani és miért nem lehet a teljes gyártósort egyetlen állomásból megoldani?

Ennek a kérdésnek alapvetően technológiai és folyamatszerkezési okai lehetnek. Technológiai oldalról egyszerűen olyan méretű gépekről, berendezésekről vagy szerelőasztalokról van szó, amik nem férnek el egyetlen állomáson. Szerelő munkaállomások esetén gondoljunk csak bele abba, hogy egy késztermék mennyi részegységből, alrészegységből és alkatrészből áll. Egy egyszerűbb elektromos kávéfőző is több mint 50 beépülő egységből áll össze, amik tárolása egy állomáson meg sem oldható, így mindenképpen több munkahelyből fog felépülni a gyártósorom.

Folyamatszervezés szempontjából pedig egyrészt célszerű a hasonló munkatartalmakat, amelye végén egy kézzel fogható egység keletkezik, egy munkaállomásra rendezni. Ezzel a folyamat átláthatóbb lesz. Így a nagyobb egységekre, mint Tisztítás, Előszerelés, Szerelés, Funkció Ellenőrzés, Csomagolás, Adminisztráció, készíték egy-egy különálló munkaállomást.

Másrészt ezekre elvégzem az *1.1 fejezetben* bemutatott példa szerint az ütemidő számítást, és ha azt veszem észre, hogy az állomás ciklusideje az ütemidő felett van, akkor azokat célszerűen több részre bontom. Ezzel elérem azt, hogy a dolgozói létszám módosításával és a dolgozók különböző állomásokhoz rendelésével az ütemidő alá tudjam hozni a dolgozók ciklusidejét.

Ehhez még vegyük figyelembe azt a tényt, hogy az ütemidő nem egy állandó érték. A vevői rendelések és a rendelkezésemre álló idő újraszámolásával folyamatosan változik. A gyártósor tervezésekor tudnunk kell azt, hogy milyen határok közé kell terveznem, így, ha azt látom, hogy egy állomás az ütemidő alatt van, de ahhoz közel, akkor célszerű egyeztetnem a vevői kapcsolattartóval a rendelési mennyiségek várható alakulásáról, vagy az esetleges időszakosan megnövekedett mennyiségekről.

Ezekkel a módosításokkal elérhető az, hogy a gyártósor rugalmasan tudjon alkalmazkodni a vevői rendelésekhez a dolgozói létszám változtatásával. Viszont ez a rugalmasság magában hordozza azt is, hogy minden egyes létszámba el kell készítenem a dolgozói ciklusidő kiegyenlítést is.

Az én Minta soromnál ezekből kifolyólag alakult úgy, hogy a főbb egységek mellett az egyes állomásokat szintén duplikáltam. Így keletkezett összesen 10 munkaállomás, ahova szeretném létrehozni a dolgozói műveletsorrendet.

Milyen lehetőségek állnak a rendezésemre ezen munkasorrendek kiegyenlítésére? Ebben a témakörben célszerű a LEAN szakterületén körülnézni, hiszen számos módszer foglalkozik a fenti kérdéssel és hasonló módszerekkel dolgozik, mint amit én is bemutatam: Neuron-háló modellek alkalmazásával és a szinergiák definiálásával előre jelezhető és optimalizálható a várható ciklusidő ^[3]. A LEAN szemlélet alkalmazásával és implementálásával a neurális hálózatok képesek a ciklusidő minimalizálására ^[4].

Másik módszer, ha a műveletsorrend kiegyenlítést egy szimulációs programra bizzuk, így a kísérlettervezés módszertanával és szimulációval elemezhető a kibocsátóképesség és növelhető a megbízhatóság ^[5]. Több olyan program is forgalomban van, amelyek képesek ez összetett szimulációt lefuttatni. Ha ebbe az irányba megyünk el, a kódolási szakmai tudásunk és a költségvetésünk szabhat határt az egyes változatoknak.

Olyan termelő soroknál, ahol a tömeggyártással vagy nagysorozat gyártással állítunk elő termékeket, lehetőségünk van a gyártási környezetben elérhető hatalmas adatmennyiség gyűjtésével és feldolgozásával (DATA mining alkalmazásával) azonosítani a kulcs teljesítménymutatókat, valamint az elért eredményekkel előre jelezhetőek azok értékei [6].

Haladóbb megoldás, hogy a ciklusidő minimalizálás elérhető megfelelő matematikai modell definiálásával és ezáltal az egyes műveletek úgy időzíthetők, hogy közbenső tárolók alkalmazása nem szükséges [7]. Megfelelő algoritmus definiálásával és kidolgozásával számolható nemcsak a minimális ciklus idő, hanem a gazdaságosan megtérülni képes ciklus idő is [8].

A dolgozatomban és a szimulációs programmal végzett kísérlettervezés mellett döntöttem, mert a mai modern számítógépen számítási kapacitása elegendő arra, hogy akár az összes lehetséges műveletssorrendet lefuttassuk. Arra, hogy miért kell az összes lehetőséget számba venni, nem pedig csak azt a párat vizsgálni, amit egy Lean szemléletű tervezés alkalmával megfelelőnek találok, a divergens gondolkodásra való hajlam adhat választ.

1.4 Divergens gondolkodás a Túlélési torzítás elméletének bemutatásával

Az 1.2-ben levezett példa jól mutatja, hogy milyen egyszerűen ki lehet egyenlíteni a ciklusidőket, ha csak ez a célunk. Azonban még ebben az esetben is vajon tudhatjuk biztosan azt, hogy a legjobb megoldást találtuk meg? Létezik-e esetleg egy olyan variáció, amit nem vettünk figyelembe?

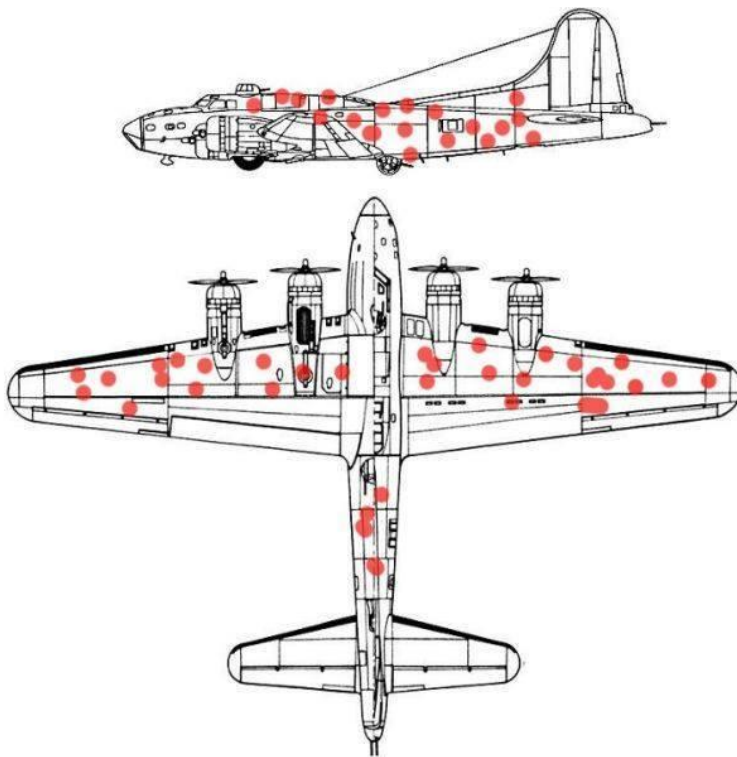
Ebben lehet segítségünkre a divergens gondolkodásmód. A divergens gondolkodásmód azt jelenti, hogy máshogyan, más megközelítésből, a standardoktól és a rutinoktól elvonatkoztatva nézzük a dolgokat. Meg kell tanulnunk másképp értelmeznünk egy probléma megközelítését. A divergens gondolkodásmód nem azonos a kreativitással, de kétségkívül ez az alapja.[9]

A következőkben tegyünk egy önvizsgálatot és teszteljünk magunkat arra vonatkozóan, hogy a saját gondolkodásmódunk mennyire ment át a rutinszerű munkavégzésbe és képesek vagyunk-e újszerűen látni a világot. Ehhez először bemutatom Wald Ábrahámot és az általa megfogalmazott Túlélési torzítás elméletét.

Wald Ábrahám egy méltatlanul ismeretlen magyar matematikus, aki a II. Világháborúban nagyban segített megfordítani az angliai légicsata kimenetét egyszerűen azzal, hogy a megszokottól eltérően nézett rá a problémára.

Wald az egyesült államokban egy alkalmazott matematikai bizottságnál kezdett el dolgozni, aminek a feladata olyan hadászati problémák megoldása volt, amelyekkel a komplexitásuk miatt a hadparancsnokok nem tudtak foglalkozni. Így került eléjük egy olyan feladat, amiben a bombázók hazatérési esélyeit, azaz a túlélésüket kellett növelni. Az hamar nyilvánvaló vált a hadvezetés számára, hogy a bombázók elsődleges célpontban számítanak, mindemelett elég nagyméretű gépekről volt szó, amelyek szabad szemmel is jól láthatóak. Így ezek a repülőgépek és a legénységük volt a legnagyobb veszélynek kitéve egy több órás, ellenséges terület felett történő bevetés esetén. Nagy szerencse kellett ahhoz, hogy a legénység túléljen egy ilyen bevetést, ezért a legapróbb matematikai előny is nagyban segíthette az akció sikerét.

A hadmérnökök tudták, hogy erősebb páncélzatra van szükség, azonban a teljes repülőgép páncéllal való ellátása a nagy súly miatt szóba sem jöhetett. Wald feladata volt azon pontok meghatározása, amelyeket meg kell erősíteni a túlélés érdekében. Segítségül a hadseregtől már kapott jegyzőkönyveket, amelyek a visszatérő gépeken vették fel a sérült pontokat és ezeket összesítették egy a 4. ábrához hasonló módon^[10].



4. ábra: Találatok a visszatérő repülőgépeken ^[11]

Látható volt, hogy a visszatérő gépeknél egy mintázat rajzolódott ki, miszerint a legtöbb golyó okozta sérülés a szárnyak mentén, a farokrészen és a törzs közepén helyezkedett el. Ebből a parancsnokság azt a következtetést vonta le, hogy a megerősített páncélzatot ezeken a

helyeken kell elhelyezni. Wald azonban rávilágított arra, hogy ezzel a gépek túlélési esélyeit egyáltalán nem növelnék.

De miért is? Mivel a visszatérő repülőgépeket vizsgálták így ezek a pontok azt mutatták meg, hogy a repülőgép mely részei erősek annyira, hogy találat esetén is visszatérhessen a bázisra. Sokkal többet mondd el az, hogy milyen részeken hiányzott a találat. A pilótafülke, a motorok és az irányításra szolgáló elemek. A helyes következtetés az, hogy nem a visszatérő, hanem a lezuhant gépeknek lenne szükség nagyobb védelemre. Azokon a pontokon a legsérülékenyebb a repülőgép, ahol a túlélők sértetlenek! Wald és a csapata bonyolult matematikai összefüggésekkel kiszámolta, hol kell a bombázókat extra védelemmel ellátni, ami így növelte a hazatérés esélyét.

A világ számára létrehozta a túlélési torzítás elméletét, amibe hajlamosak vagyunk nagyon sokszor beleesni. Bár a helyes adatokból dolgozunk, nem a helyes következtetést vonjuk le, mert túlságosan is a látható, egyszerűbb dolgokra koncentrálunk a nem látható, mélyebb értelmezést igénylő dolgok helyett.

Ez különösen igaz lehet, ha rutinszerűen oldunk meg feladatokat és már nem is akarunk más, újabb megközelítést meglátni. Az ilyen és ehhez hasonló esetekre adhat megoldást az, ha minden lehetséges megoldást megvizsgálunk, úgy ahogy azt az algoritmusomban is teszem majd a későbbiekben.

1.5 Gyakorlati példa a divergens gondolkodása

Most pedig teszteljük magunkat egy gyakorlati példával! Úgy építettem fel a fejezetet, hogy aki szeretne tudjon önállóan gondolkodni és valóban saját magától is meg tudja oldani a feladatot. Azért választottam ezt a példát, mert átfordítható termelési környezetre és értelmezhető ciklusidő kiegyenlítéses feladatként is.

A feladat:

A brémai muzsikások egy éjjel egy sebes patakhoz értek, amely felett egy hosszú, de szűk palló vezetett át. Teljes sötétség lévén csak úgy tudtak átkelni, hogy az egyetlen lámpásuknak mindig a pallón átkelőknél kellett lennie, s mivel az igen szűk volt, egyszerre csak két állat fért el egymás mellett (egymás hátán pedig nem mertek volna átkelni...). A macska egymagában 10 másodperc alatt végigszárguldott volna a pallón, a kutyanak ugyanez már csak 20 másodperc alatt sikerült volna. A kakas ellenben 50, végül a magát megmakacsoló szamár csak 100 másodperc alatt ért volna át egyedül. Amikor pedig két állat együtt haladt át a pallón, a gyorsabb mindig bevárta a lassabbat. Hány másodperc alatt érhettek át a leghamarabb?^[9]

Következő bekezdésben nézzük a megoldást, így kérem csak az olvasson tovább azonnal, aki nem szeretne ezen önállóan is elgondolkodni.

Logikusan úgy kezdjük a feladatot, hogy a visszahordási időt szeretnénk minimalizálni, ezért azt mondjuk, hogy a macska legyen az, aki segít az összes állat átkelésében. Ha így indultunk el, akkor valami hasonló megoldás születhetett:

- Macska + Kutya átkel 20 sec
- Macska vissza 10 sec
- Macska + kakas átkel 50 sec
- Macska vissza 10 sec
- Macska + Szamár átkel 100 sec

Ezen sorrend szerint haladva 190 sec alatt érnek át az állatok, ami egy logikusan alátámasztható és elfogadható megoldás.

De nem a legjobb megoldás!

A leggyorsabb esetben 170 sec alatt teljesíthető a feladat. Ez több mint 10%-os javulást jelent, ami egy valós termelősor esetén már költségmegtakarítás szempontjából is egy kiugró eredmény. Főleg annak a fényében, hogy ehhez a 10%-os javuláshoz nincs szükségünk új szerszámmra, berendezésre, sem más jellegű befektetésre. Egyszerűen a ciklusidő kiegyenlítést kell más nézőpont szerint elkezdni. Innentől ismét felvázolom a megoldást, így ismételtén arra kérem az olvasót, hogy csak abban az esetben folytassa azonnal, amennyiben nem szeretne önmagától gondolkodni ezen.

Bár logikus a visszahordási idő minimalizálása, nem ezen nyerünk a legtöbbet, hanem ha a két leglassabb tényezőt kezeljük egyként. Viszont a visszahordási idő is kulcs tényező lesz, ezért induljunk el hasonlóan:

- Macska + kutya átkel 20 sec
- Macska vissza 10 sec
- Kakas + Szamár átkel 100 sec
- Kutya vissza 20 sec
- Macska + Kutya átkel 20 sec

Ha alaposabban megnézzük a megoldást észre lehet venni, hogy nem csak időt sikerült megtakarítani, hanem a megtett távot is jobban szét tudtuk osztani.

Ezen felül meg kell jegyezni, hogy azért nehéz ez a gondolkodásmód mert több olyan elemet is tartalmaz, ami első ránézésre logikátlanak, sőt feleslegesnek is tűnhet: pl. az első és utolsó lépés ugyanaz, a Macska + Kutya együtt átkel.

Az a képesség, hogy így lássuk a dolgokat és más nézőpontot keressünk valójában mindenkinben megtalálható, csak tudunk kell átlépni a saját gondolkodásmódunk korlátait.^[9] Ez azonban nehezebb, mint elsőre hangzik.

Miért ne elégednénk meg a 190 sec-el mikor az egy **jó** megoldás, és nem is feltétlen tudjuk, hogy **nem a legjobb**?

Az általam bemutatott algoritmus segíthet ebben olyan módon, hogy egy termelősor elemzésekor megvizsgálja az összes lehetséges variációt! Lehetőségünk van emellett szabadon választható paraméterek megadására is, mint például az utak minimalizálása, így a kimeneten egy olyan táblázatot szolgáltat a felhasználó számára, amelyből ki tudja választani a számára legjobb megoldást.

2. Tecnomatix Plant Simulation szoftver ismertetése

A dolgozatomban a ciklusidő kiegyenlítésre a Siemens Tecnomatix Plant Simulation szoftverét használom. Ez a szoftver modellezi, szimulálja és elemzi a termelési rendszereket és ezek logisztikai folyamatait annak érdekében, hogy optimalizálni lehessen az anyagáramlást és az erőforrás felhasználást az üzem tervezésének minden szintjén, a globális létesítményektől a helyi üzemeken át a konkrét gyártósrig.^[12] A Siemens által készített példában egy üzem az 5. ábrán látható módon épül fel.



5. ábra – Tecnomatix Plant Simulationban készített gyártósor^[13]

Tehát a szoftver segítségével képesek vagyunk elkészíteni egy termelősor *digitális iker* mását. A digitális iker egy valós objektum vagy folyamat virtuális mását jelenti, ahol a virtuális másolat pontosan ugyanúgy viselkedik a külső behatásokra, mint a való világban létező testvére^[14]. Azaz egy jól lemodellezett folyamat, a validálást követően alkalmassá válik arra, hogy a termelősor zavarása nélkül a folyamatra vonatkozó fejlesztések és tesztek hajthassunk végre. Ezek a fejlesztések a való életbe átültetve is ugyanazokat az eredményeket kell, hogy hozzák **4%-os tűrésen belül**.

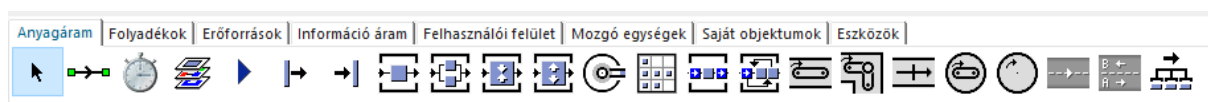
A szoftver tehát képes jól utánozni egy előre felépített gyártósor működését és számos beépített algoritmussal rendelkezik, amelyek a kiértékelést segítik, azonban önmagától az adott gyártósor ciklusidő kiegyenlítésére nem képes. Abban segítséget nyújt, hogy konkrét számadatokkal és vizuális megjelenítéssel támogatja a döntésünket, ha más-más dolgozói elrendezést szeretnénk vizsgálni. Ezt a képességét viszont sikeresen össze lehet hangolni azzal, hogy lehetőséget biztosít, sőt támogatja a saját algoritmusok és funkciók készítését, kódolással a SimTalk 2.0 objektum orientált programozási nyelvben.

Ezt kihasználva tudtam készíteni egy olyan funkciót a program adta eszköztár segítségével, ami beállítja és végig futtatja az összes lehetséges dolgozói elrendezést, majd egy diagramon összesíti az eredményeket.

2.1 A szimuláció építőelemei és működése

Első lépésként ismerkedjünk meg kicsit a programmal és nézzük meg, hogy milyen alap építőelemeket használhatunk egy termelősor modellezésére. Majd készítsük el a Minta sorunkat, amin tesztelhetjük a ciklusidő kiegyenlítő algoritmust. A gyártósor felépítésére használjuk a 6. ábrán látható eszközkészletet.

A gyártósor felépítéséhez első körben definiálni kell egy bemeneti pontot, ahol a termékek fizikailag is meg fognak jelenni. Ez a valós termelési környezetben annyit tesz, hogy a sornak egy információt átadnak, miszerint „Kezdjétek el gyártani az [A] terméket”.



6. ábra – Plant Simulation Anyagáram eszköztára

A bemeneti pontban a teljesség igénye nélkül meg lehet adni: milyen anyag keletkezzen (lehet definiálni terméket, termékcsaládot), milyen időközönként jelentkezzen az igény, alkalmazzon-e eloszlás függvényt, esetleg szállítási táblázat alapján időpontra meghatározva jelenyen meg anyag a bemeneten

Ezt követően munkáállomásokat tudunk lehelyezni. Több opció közül lehet választani, mi most a legegyszerűbbet használjuk. A munkáállomásoknál számunkra jelenleg a legrelevánsabb az *Idők* fül, azon belül is a *Feldolgozási idő*. Alapértelmezés szerint ez 1:00 perc, ahogy a 7. ábrán is látható, ezt kell minden állomás esetén testre szabni.

7. ábra – Állomások általános adatlapja

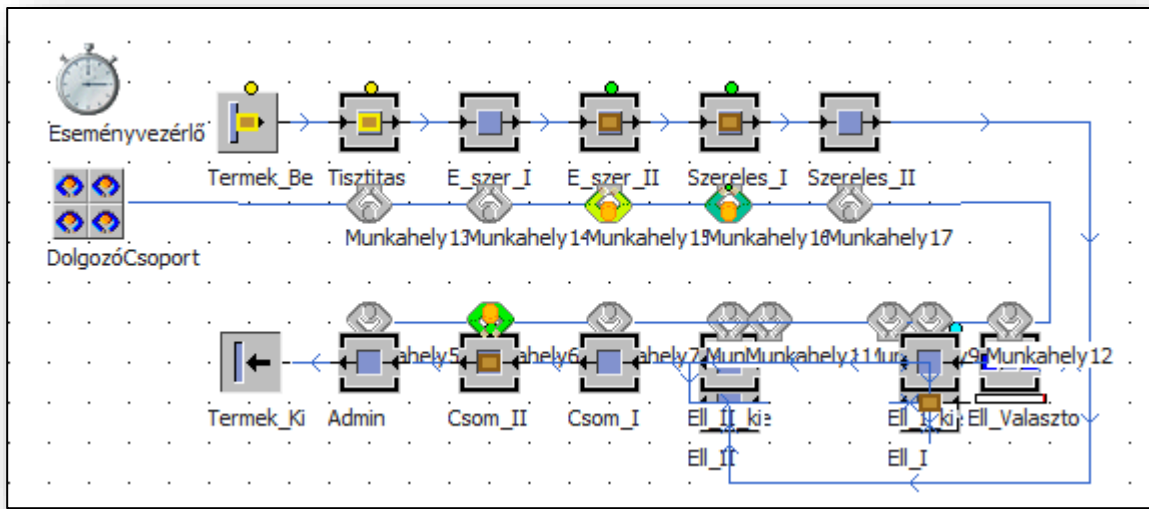
Az idő bevitele után dolgozókat tudunk hozzárendelni minden munkaállomáshoz, egyenként. Így lesz lehetőségünk arra, hogy több munkaállomást egy dolgozóhoz rendelve megkíséreljük az idők kiegyenlítését. Fontos eleme a szimulációnak a *DolgozóCsoport*, ahol azt tudom megadni, hogy az adott soron összesen hány fő dolgozik, milyen műszakbeosztás szerint történik a munkavégzés, valamint a tervezett leállásokat, szüneteket is meg lehet adni műszak szinten.

Fontos még megjegyezni, hogy a szimuláció **eseményvezért**, ami azt jelenti, hogy az idő csak akkor telik, amikor valamilyen esemény végrehajtása folyamatban van. Ebből adódóan a szimuláció futtatásakor és kiértékelésekor az alábbi tényezőkre figyelni kell:

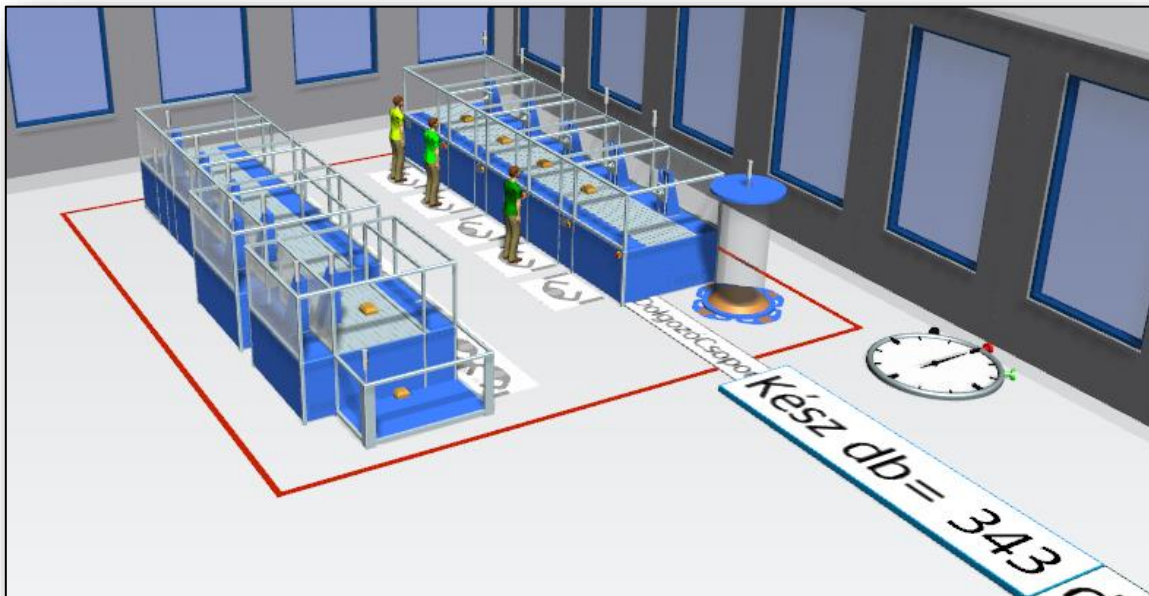
- ha 0 perc idővel adok meg egy folyamatot, az le fog futni, látható animációval végre fogja hajtani a program, de valójában a szimulációs idő ezen tevékenység alatt nem telik. Ez megtévesztő lehet.
- ha nem történik olyan esemény, amit a szimuláció végre tudna hajtani, leáll a program. Azaz nem fog telni az idő, és úgy tűnik, mint ha végzett volna a program futása, valójában azonban egy nem megfelelően lekezelt esemény hatására idő előtt megállt, mert nem volt számára feladat, amit elvégezzon, az eseményvezérlés nem kapott jelet.

2.2 Mintasor felépítése

Építsük fel a termelősort az 1.3 fejezetben ismertetett adatok alapján. Az előző pontban használt építőelemek használva a 8. ábrán megjelenített 2 dimenziós és a 9 ábra szerinti 3 dimenziós virtuális sort kapom. A program automatikusan elkészíti a 3D sort, aminek prezentációs és vizuális haszna van. A kísérlettervezés szempontjából számunka a 2D sor elegendő.



8. ábra – A felépített mintasor



9. ábra – A modell 3D megjelenítése

Láthatók benne az alapépítőelemek: A *Termek_Be* állomáson folyamatosan megjelenik a munkadarab, ami össze van kötve a termelő állomásokkal. Minden állomás előtt egy szürke operátor jelölő mutatja, azt, hogy az állomás végrehajtásához emberi tevékenység szükséges. A zöld operátor jelölők mutatják az aktív dolgozóinkat.

A program igyekszik mindent színekkel vizuálisan megjeleníteni, az állomás felett a zöld karika arra utal, hogy ott éppen munka folyik. Ha ez átvált sárgára, az jelöli, hogy az állomás már végzett, de valamiért várakozni kényszerül: foglalt az a hely, ahova a terméket továbbítani kell, vagy épp nincs megfelelő erőforrása a termék elkészítésére.

Ugyanígy színnel jelöltek a termék is, a sötétbarna szegéllyel keretezett barna téglalap. Ha a termék átvált sárga árnyalatra, az szintén azt jelzi, hogy várakozni kényszerül.

A folyamat végén a *Termek_Ki* állomáson hagyja el a termék a gyártósorunk. Ennek a ciklusideje 0, azért van rá szükség, hogy a program statisztikát tudjon szolgáltatni, valamint legyen egy hely, ahol a termék példányosításait megszüntetjük

A következő pontokban ismertetem azokat a peremfeltételeket, amik alapján a Minta gyártósort felépítem. Később ezeket lehet módosítani, finomítani, hogy valósabb modellt kapjunk, azonban a ciklusidő kiegyenlítéshez így is elegendő.

Peremfeltételek:

- egyetlen készterméket gyártunk, ami folyamatosan jelenik meg a *Termek_Be* állomáson
- egyetlen, 8 órás műszakot szimulálunk végig, amiben egy 20 perces és egy 10 perces szünet van tervezve. A szünetben a Funkcióellenőrző készülékekbe helyezett termékek ciklusa felügyelet nélkül végig futhat
- a funkció ellenőrző berendezések közül azt választjuk amelyik éppen szabad
- a funkció ellenőrző berendezések nem várják be egymást, így gyorsabb a termelés, de 1-1 darab esetén sérülni fog a FIFO sorrend (First In First Out, azaz az első darab amelyik belép a termelősorra, elsőként is fogja elhagyni azt. A példában ez sérül olyan módon, hogy a Csomagolási állomásra az a munkadarab ér korábban oda, amelyik később lett behelyezve a gyorsabb végátvevő berendezésbe)
- a funkció ellenőrző berendezésekből ugyanaz a dolgozó veszi ki a terméket, aki behelyezte

- anyagmozgatási és dolgozó mozgási idő nincs számolva az állomások között. Ennek oka, hogy nem mindegy melyik pontból melyikbe megy a dolgozó. Ezt az utat a program fogja lekezelni
- a műveleti időket az *1. ábra* alapján határoztuk meg
- a munkahelyek és a dolgozók között több-az-egyhez kapcsolatot határozzunk meg, de egy munkahely csak egyetlen dolgozóhoz tartozik (a program egyébként lehetőséget biztosítana arra, hogy legyen olyan közös munkaállomások, melyeken bármelyik dolgozó végezhet munkát)
- bár folyamat szinten nincs rá szükség, a szimuláció miatt építettem be egy puffert az ellenőrző berendezések elé: ennek oka, hogy a puffer fogja szortírozni, hogy melyik berendezéshez kerüljenek a termékek. Annak érdekében, hogy az egydarabos áramlás fennmaradjon úgynevezett reteszelési megoldással lehet zárni a puffert. Ez azt jelenti, hogy a Szerelés II és az azt követő állomáson összesen 1 db termék várakozhat. Így a folyamat nem sérül.
- egy munkahelyen egy időben csak egy dolgozó tartózkodhat
- az áttekinthetőség kedvéért a dolgozókat 3 különféle zöld színnel különböztetjük meg egymástól a programban, lásd *3. ábra*
- folyamatot modellezzük, így úgy számolunk, hogy a termelősor már előre fel van töltve anyaggal és folyamatában megy a szerelés. Műszak átadáskor sem kell leírítani a sort, így a következő műszak is előre feltöltött sorral tud kezdeni. Ez a programban beállítható úgy, hogy hagyunk a sornak egy feltöltési időt, amíg a statisztikai mutatókat nem veszi figyelembe
- egy darabos anyagáramlást alkalmazunk
- jelen modellben a selejtkezeléssel nem foglalkozunk, feltételezzük, hogy a funkció ellenőrzés minden esetben sikeres. Ennek oka, hogy a validálási folyamatot feleslegesen bonyolítaná túl a selejtezés
- a dolgozók hatékonyságát 100%-osnak feltételezzük egy Gauss eloszlás alapján, tehát a táblázatban megadott idő adatokat tartósan tudják teljesíteni, de lehetnek benne ingadozásokat

- a program lehetőséget ad különböző eloszlásfüggvények alkalmazására, ezek közül én a Gauss (normál) eloszlásfüggvényt választottam minden állomás esetén. Ehhez meg kell adni egy:
 - várható értéket, ami minden esetben az *1. ábrán* a munkaállomás ciklusideje
 - standard szórást, ami a mindenkori ciklusidő 10%-a
 - alsó határt, amit úgy számolok, hogy a várható értékből kivonom a várható érték 8%-át
 - felső határt, ami a várható érték 10-szerese
- Így egy eltolt normál eloszlás függvényt kapunk az állomásokra, ami azt jelenti, hogy a dolgozók a várható érték közelében végzik a leggyakrabban a tevékenységük, ettől az esetek kisebb részében gyorsabbak is lehetnek, viszont előfordulhatnak olyan eltérések is, amikor jelentősen lelassulnak (folyadékpótlás, dekoncentráltóság, fáradás, figyelem elvonása a környezet által)

2.3 Modell validálása

Valós gyártósorok esetén mindenképpen a validálás a következő lépés, ami azt jelenti, hogy egy műszakon keresztül mérjük a kulcs mérőszámokat, és a bemeneti paramétereket, majd a paramétereket beállítva a kulcs mérőszámok értékét kell kapnunk 4%-os tűrésen belül

Bemeneti paraméter lehet:

- tényleges munkavégzés ideje (nem tervezett leállások időtartama vagy tervezett, de a modellben nem standard leállás)
- napi aktuális selejtmutató
- valós gyártási terv (jelen esetben csak egy terméket gyártunk, de több termék esetén célszerű a valódi tervet betölteni)
- dolgozói létszám, ciklusidő kiegyenlítettség táblázat (mi majd ezt akarjuk a későbbiekben fejleszteni)
- valamint további tényezők, amelyeket bevittünk a modellbe és úgy gondoljuk, hogy célszerű az értéket változtatni

A kimeneti értékek, azaz a kulcs mérőszámok lehetnek:

- a műszakban legyártott jó darabszám
- termék átfutási idő
- dolgozói ciklusidő
- ciklusidő kiegyenlítettség mértéke

- dolgozók által megtett út (napi lépésszám)
- valamint a termelősor és a vállalat által meghatározott olyan mutatószám, ami befolyásolásával javulást érhetünk el a Kulcs mérőszámok rendszerében

2.3.1 Kulcs mérőszámok

A Kulcs teljesítménymutatók (KPI – Key Performance Indicator) azok a mérhető adatok, amik megmutatják, hogy a vállalat hogyan teljesít a céljai elérésében ^[15]. Ezen mutatók rendszeres áttekintése segít a szervezetnek eldönteni, hogy jó úton haladnak-e a korábban kitűzött célok elérése érdekében.

Egy jó KPI az alábbi tulajdonságokkal rendelkezik:

- Specifikus: meghatározza, hogy a KPI mit hivatott mérni és miért
- Mérhető: egyértelműen meghatározott a mérés módja
- Elérhető: realiztikus célnak kell hozzá tartozni, ami teljesíthető
- Releváns: a mérőszámnak előre kell mozdítani a vállalatot
- Időkeret: a múltbéli események alapján reális időkeretet állítsunk fel

Ezek alapján a Minta sorunk rendelkezzen három kulcsmérőszámmal:

- Darabszám: a 8 órás műszak alatt gyártott termékek száma. Ez legyen maximális.
- Átfutási idő: egy termék elkészüléséhez szükséges időtartam. Ez legyen minimális.
- Dolgozók által megtett távolság: legyen minimális

2.3.2 Modell validálása

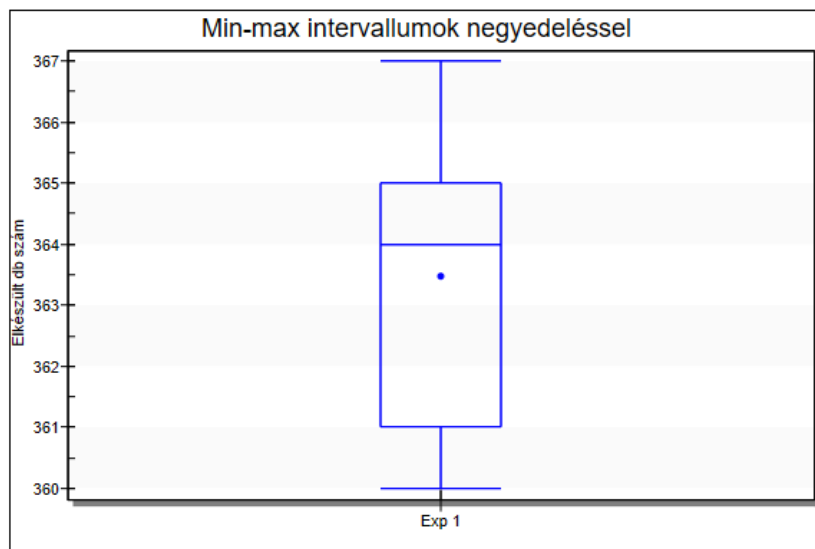
A Plant Simulationban lehetőség van úgynevezett Kísérlet Tervezésre, ahol a program automatikusan lefuttat egy általunk meghatározott számú műszakot, miközben a bementi paramétereket a szélső értékek között állítgatja. Nézzük meg, hogy a Kísérlettervezés milyen főbb folyamatokból áll:

Futtatásnak nevezem a szimuláció egy elindításától annak a befejezésig tartó esemény. Ez lehet egy műszak, egy nap, 1000 db, stb. Egy adott eredménye van, amit ki lehet értékelni.

A kísérlet több futtatást jelent, ugyanazokkal a paraméterekkel, csak a véletlenszerűen generált számok változnak. Jelen esetben pl. a dolgozói ciklusidő a fentebbe említett Gauss eloszlás szerint. Egy kísérletből már az eredmények összevethetőek és magállapítható az adott eredmény szórása.

A Kísérlet tervezés több kísérletet jelent, ahol már változhat egy vagy több előre meghatározott paraméter. Így a kapott eredmények a paraméterek függvényében összehasonlíthatóak és megállapítható az is, hogy melyik opció milyen szórással rendelkezik.

A modell egy 15 futtatást végrehajtott kísérlet után, a következő adatokat szolgáltatta:



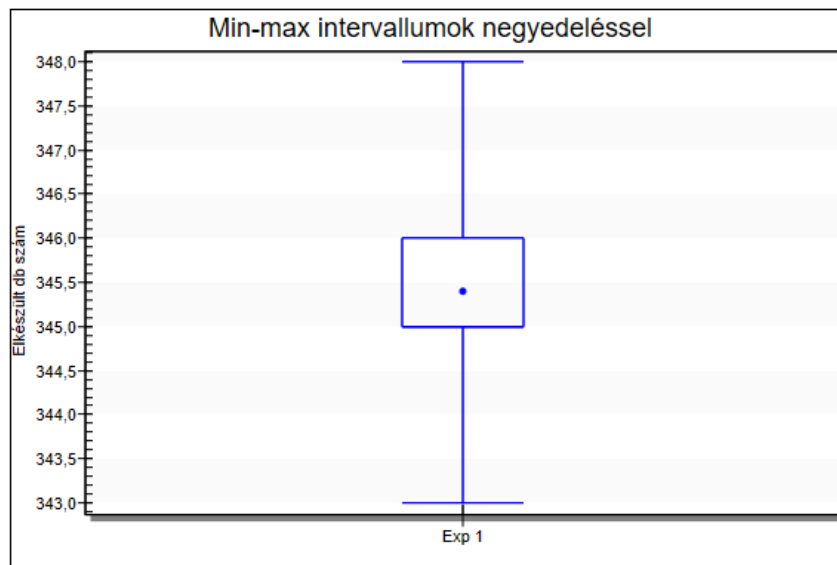
10. ábra – Minta sor eredménye a darabszámra

A 10. ábrán látható, hogy az átlagosan termelhető mennyiség 363,5 db volt. Az előzetesen lefektetett szabály szerint akkor jó a modell, ha 4%-os pontosságon belül dolgozik. Az 1. ábra adatai alapján a szűk keresztmetszet ideje 72 sec.

$$\text{Elvárt darabszám} = \frac{450 \text{ perc} \times 60 \text{ sec}}{72 \text{ sec}} = 375 \text{ db}$$

$375 \times 0,04 = 15$ db. Tehát 360 és 390 közötti darabszám esetén tekintem a modellt érvényesnek. A 10. ábra alapján ez teljesül, így a modellt érvényesnek tekinthetjük.

Vegyük azonban figyelembe, hogy sem a ciklusidő kiegyenlítésnél, sem a modellben nem számoltunk a mozgási idővel. Ha ezeket beprogramozzuk a modellbe, akkor a 11. ábra szerint alakul a darabszám:



11. ábra- Mozgási idővel módosított darabszám diagram

A további KPI-ok tekintetében:

- Darabszám: 345 db
- Átfutási idő: 12:01 perc
- Dolgozó 1 megtett távolság: 2620 méter
- Dolgozó 2 megtett távolság: 1467 méter
- Dolgozó 3 megtett távolság: 3565 méter

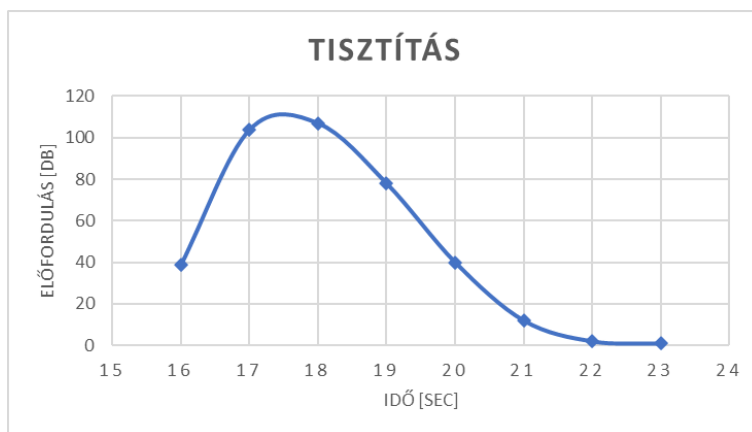
Ezeket az értékeket tekintjük a standardnak, amit az Automatikus sorkiegyenlítéssel fejleszteni szeretnénk.

2.4 Plant Simulation összefoglalva

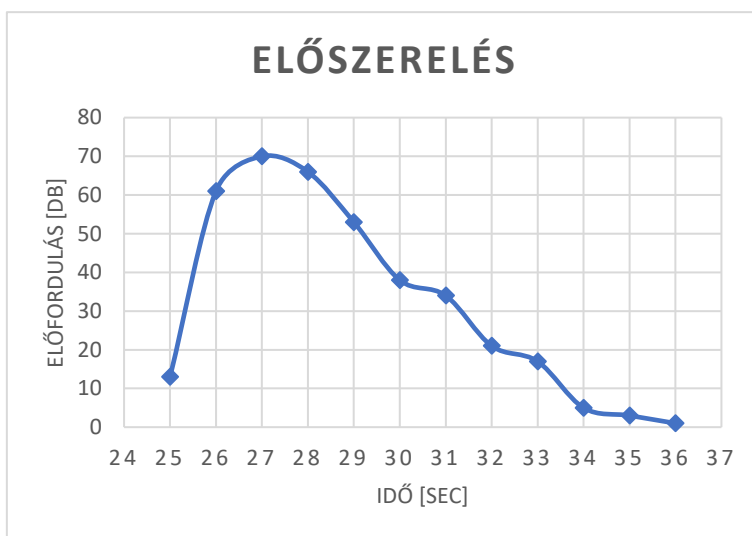
Miben is másabb a Plant Simulation mint egy hagyományos tábla és miért választottam, ezt a programot? Egyrészt arról ahogyan korábban már említettem, a ciklusidők esetén is be lehet állítani eloszlásfüggvényeket. A programot többször futtatva, kísérleteket hajtva végre, a szélsőértékek között véletlenszerűen határozza meg a ciklusidőket, így egy szélesebb képet kapunk a sor állapotáról.

Nem csak egy ideális állapotot figyelhetünk meg, hanem láthatjuk, hogy az általunk választott megoldásnak milyen szórása van akár az apróbb zavarokra, mikor a dolgozó kiesik a ciklusból, akár arra, ha még nem feltétlen dokumentálható módon, de megszakítja a munkavégzést azzal, hogy ha például felbontja a kulacsot és iszik pár kortyot. Ha ezzel nem számolunk, és csak egy idealizált állapotot veszünk alapul, lehet nem az elvárásoknak megfelelően működik a gyártósor.

Másrészről egy, a szimulációban jól felépített gyártósoron, nem kell foglalkozunk azzal, hogy a megtett távolságot külön-külön kiszámoljuk A-B; A-C, B-C állomások között ezt a program automatikusan intézi így sokkal gyorsabban tudunk több különböző esetet megvizsgálni.



12. ábra – Tisztítás művelet ciklusidő eloszlása egy futtatás alkalmával



13. ábra – Előszerelés I művelet ciklusidő eloszlása egy futtatás alkalmával

Példaként az, hogy az elvárásoknak megfelelően alakultak az eloszlásfüggvények. A Tisztítás esetén a várható érték 18 sec volt (12. ábra), még az Előszerelés I esetén 28 sec (13. ábra). Ezek előfordulása volt (közel) a legtöbb, de többször túl is lehetett ezt teljesíteni. Valamint látható, hogy volt bőven olyan zavar, mikroállás, ami feltarthatta a gyártósort.

Így a program valóban egy reálishoz közeli, átfogó képet tud adni egy valódi gyártósor működésével kapcsolatban, amit táblázatokkal nem lennénk képesek kezelni.

3. Automata ciklusidő kiegyenlítő algoritmus

A fentieket összefoglalva: amennyiben egy sorkiegyenlítés alkalmával nem vagyok biztos abban, hogy a legjobb megoldást találtam meg, szeretnék fejleszteni a gyártáson és nem tudom hogyan tegyem vagy olyan komplex a sorom, aminél nem tudom minden bemenő paraméter hatását figyelembe venni, akkor célszerű lehet az összes lehetséges előfordulást ellenőrizni.

Ehhez készítettem egy összetett algoritmust, ami két részből áll össze. Első körben létre kell hoznom a dolgozói eloszlástáblázatot az összes lehetséges előfordulásra. A fejezet elején ennek a matematikai modelljét mutatom be, hogy lássuk milyen adatmennyiséggel kell dolgoznunk egy-egy elemzés alkalmával, majd pedig az elkészült kódot.

Általánosságban elmondható, hogy a variációk száma munkahely és állomás függvénye, de minél magasabb ezek száma, annál több sorból fog állni a táblázat, ami így akár több ezer lehetséges variáció is lehet. Belátható, hogy ezt minden esetben kézzel létrehozni nagyon időigényes, valamint a hibázási lehetőség is nagy, így mindenképpen kell egy algoritmus, ami önállóan képes létrehozni az összes variációt. Következő lépésként egy logika mentén egyszerűsítettem az algoritmust, ezzel spórolva mind a táblázat elkészítési idejével mind pedig később a szimuláció futási idejével.

Másrészről a Plant Simulation szoftverben is el kellett készítenem azokat a kódokat, amik képesek feldolgozni a korábban létrehozott táblázatot és az alapján egymás után többször lefuttatja a szimulációt, úgy, hogy minden esetben a soron következő dolgozói eloszlást használja. Ebben definiálom azt, hogy hogyan rendelje az egyes dolgozókhoz az adott állomásokhoz. A futtatások végével pedig összesíti az adatokat, hogy el tudjam dönteni melyik opció az, ami a bementei paraméterek szerint számomra a ideális.

3.1 Összes lehetséges előfordulás matematikai modellje

Az összes lehetséges előfordulás matematikai, kombinatorikai, megfelelője az ismétléses variáció. Ha n db. egymástól különböző elem közül kiválasztunk k db.-ot úgy, hogy a kiválasztott elemek sorrendje is számít és ugyanazt az elemet többször is választhatjuk, akkor az n elem k -ad osztályú ismétléses variációját kapjuk.

Az n elem k -ad osztályú ismétléses variációk száma: n^k [16].

Tehát a dolgozókat tudom úgy tekinteni, mint a sorba rendezendő elemeket a munkahelyek függvényében: a dolgozók jelentik az egymástól különböző elemeket, fontos, hogy milyen

sorrendbe rendezem őket, és egy dolgozót több munkaállomáshoz is rendelhetek. Így, ha például 2 dolgozót szeretnék 3 állomásra elrendezni akkor azt $2^3 = 8$ különféle módon tehetem meg.

Variáció	1. Munkahely	2. Munkahely	3. Munkahely
1	1	1	1
2	1	1	2
3	1	2	1
4	1	2	2
5	2	1	1
6	2	1	2
7	2	2	1
8	2	2	2

14. ábra – 2 dolgozó 3 munkahelyre történő eloszlásának összes variációja

A Plant Simulation kísérlettervezője lehetőséget ad arra, hogy egy hasonló táblázatot betöltve végig futtasson egy Kísérlet sorozatot, ahol a táblázat minden egyes sora egy-egy kísérletet jelent az általunk megadott számban futtatva.

Ahhoz tehát, hogy az összes lehetőséget meg tudjam vizsgálni, első lépésként létre kell hoznom a fent dolgozói táblázatot. Ezt akár manuálisan is meg tudom tenni, de ahogy a fenti képletből is látható, a lehetséges variációk száma exponenciálisan növekszik. Ha 2 főt 8 állomás között szeretném elosztani már 256 sort kellene létrehoznom, de a Minta sort vizsgálva, ha 3 fő dolgozik egy 10 munkahellyel rendelkező gyártósoron akkor már 59 049 lehetséges variáció van. Ezt már táblázatkezelő programokban is sem lehet manuálisan lekövetni, ezért mindenképpen szükségünk van egy olyan algoritmusra, ami automatikusan előállítja számunkra a kívánt táblázatot. Ehhez nekem 2 bemenő paraméter kell megadnom: a dolgozók és a munkahelyek számát.

3.2 Dolgozói eloszlástáblázat automatikus létrehozása

Első lépésként matematikai szempontból tekintsük a dolgozókat biteknek. Így 2 dolgozó esetén 0 és 1 bitekről beszélhetek. Azaz a dolgozók eloszlását az állomások között a kettes számrendszerben lehet reprezentálni. Módosítsuk annyiban a 14. ábrán látható táblázat értékeit, hogy az 1. dolgozót „0”-val, a 2. dolgozót pedig „1”-el jelölöm. Valamint a variációk számozását kezdjem 0-ról.

Variáció	1. Munkahely	2. Munkahely	3. Munkahely
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

15. ábra -A 14. ábra átírva bináris számrendszerbe

Látható, hogy a két táblázat logikailag azonos, összesen 8 különböző esetet tudok vizsgálni csak a számozásban és a dolgozók megnevezésében van eltérés. Azonban a 15. ábra megfelel a kettes számrendszer természetes számainak növekvő sorrendbe állított sorozatának. Valamint a variációk oszlopban található tízes számrendszerbeli szám megfelel az adott szám bináris alakjának, ha a táblázat jobb oldali oszlopát tekintem 0. hatványnak, a középső oszlopot az 1. hatványnak a bal oldali oszlopot pedig a 2. hatványnak.

A 6. sor esetén $110 \rightarrow 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 4 + 2 + 0 = 6$

A dolgozói táblázat így leképezhető az adott számú dolgozónak megfelelő számrendszerrel, azaz pl. 3 fő esetén a hármas számrendszerben tudjuk létrehozni a Dolgozói eloszlástáblát. Ezt követően a dolgozói eloszlástábla létrehozható úgy, hogy minden számjegyhez hozzáadok 1-et.

Nincs más dolgom, mint a 3.1.1-ben használt képlettel kiszámítani, hogy összesen hány variáció van az adott esetben, majd egy programmal létrehozni 0-tól kezdve a kiszámolt értéktől eggyel kevesebb értékig létező minden egyes egész szám adott számrendszerbeli alakját és ezt egy táblázatban összefoglalni.

Ennek a táblázatnak a létrehozására így az Microsoft Excel programját választottam. Ennek oka, hogy univerzálisan elterjedt, könnyen beszerezhető és jól programozható. Létezik benne az ALAP függvény, ami pontosan azt csinálja amire nekünk szükségünk van: létrehozza egy decimális szám általunk kiválasztott számrendszerben létező alakját és még azt is meg lehet adni, hogy hány számjegyből álljon a létrehozott szám. A nem használt kitevők helyére 0-t ír, de minden szám hossza egységes lesz, így később könnyen kezelhető.

Az alap függvény felépítése a következő lesz:

$$" = ALAP(i; DolgozókSz\acute{a}ma; MunkahelyekSz\acute{a}ma)"$$

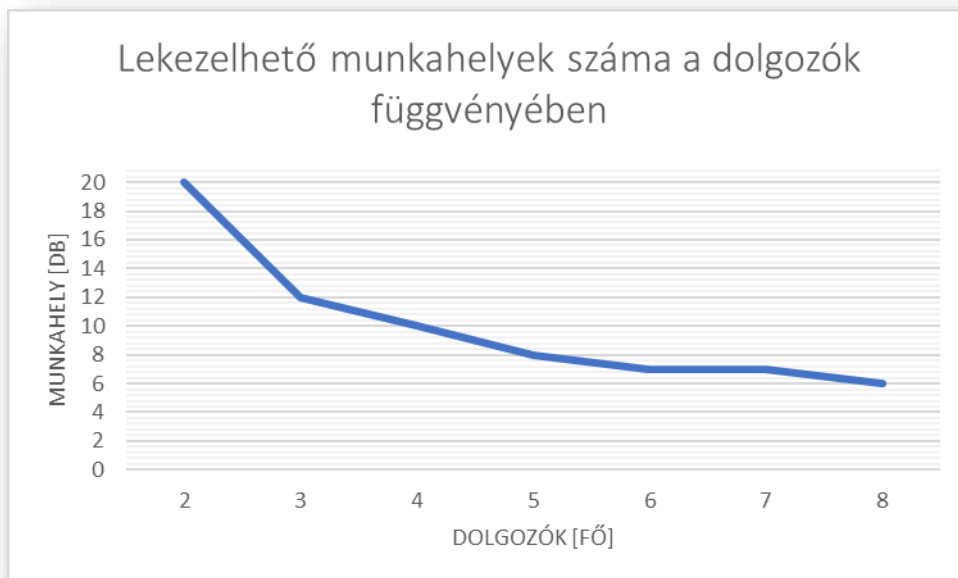
ahol 0-tól a $DolgozókSz\acute{a}ma^{MunkahelyekSz\acute{a}ma} - 1$ értékig minden egész számot felvesz.

Így a Minta sor példájánál maradva létre kell hoznom összesen $3^{10} = 59\,049$ számot. Az első szám a 0 lesz, az utolsó az 59 048.

A táblázat 10. eleme a következő lesz: „=ALAP(10;3;8)” eredménye pedig 00000101

Ez lefedi a korábbi pontokban említett minden lehetséges variáció lefuttatását és az így létrehozott táblázatot rá is lehetne engedni a Plant Simulationra, azonban a számítási kapacitással kapcsolatos előzetes elvárásaink túl optimistának bizonyultak. Egyrésről, ha a Kísérlettervezőben minden kísérlethez csak 5 futtatást rendelünk hozzá, az már közel 300 000 futtatást jelent, ami több óráig, akár napokig eltarthat. Valamint óriási mennyiségű adattal jár, amit már nem feltétlen tudunk akadás nélkül kiértékelni.

Másrésről bár az egy oszlopban elméletileg lekezelhető adatok száma 1 048 576, a gyakorlati tapasztalat az, hogy ebben az esetben már nagyban lassul a program működése. Át lehetne írni úgy, az algoritmust, hogy elkezdjen egy következő oszlopot használni, de már ez az adatmennyiség is megakasztja a hardvert, így más megoldást kell kitalálni.



16. ábra – Optimalizálás nélkül megadható munkahelyek száma a dolgozók függvényében

Másrésről viszont, ha ezt az adatmennyiséget tekintem maximálisnak, akkor nagyon beszűkíti a kezelhető munkahelyek számát. A 16. ábrán látható, hogy az egyes létszám

függvényében hány munkahely vizsgálata esetén érem el a maximálisnak definiált 1 millió variációt. Így 3 fő esetén még 12 munkahelyet le lehetne generálni, bár a szimuláció szempontjából már ez is 3 millió feletti futtatást jelentene 3 kísérletenként 3 futtatást számolva, azonban 7 fő esetén már csak 7 munkahelyet képes táblázatba rendezni, 8 fő esetén pedig már nem is értelmezhető: csak 6 állomást tud lekezelni, ami nem is egy életszerű példa. Így kell találni egy megoldást, amivel optimalizálni lehet a táblázatot.

3.3 Táblázat optimalizálása

A *16. ábra* kapcsán felmerülhet bennünk a kérdés, hogy 7 főt 7 állomásra elrendezve miért is kell egy 1 millió variációs táblázatot létrehozni, mikor ennek a kimenete gyakorlatilag egyetlen sor?!

A dolgozók eloszlásánál a *14. ábrát* szemlélve is feltűnhet, hogy feleslegesen vannak olyan sorok generálva, ahol nincs használva az összes munkaerő. Nem **legalább** három fővel, hanem **pontosan** három fővel terveztem a Minta sort is. Amennyiben szeretném megtudni azt, hogy két fővel hogyan működik a sor, arra külön kísérlettervezést létre lehet hozni, ahol 2 fő és 10 állomás van beállítva. Így viszont első körben az optimalizáló algoritmusunk számára meg tudjuk azt adni, hogy ha generált egy adott számrendszerbeli számot, de abban nem használja fel az összes lehetséges számjegyet, azaz a táblázatban nem használom fel az összes dolgozót, akkor azt az előfordulást be se írja.

Az összes lehetőség száma 7 fő és 7 állomás esetén már 7 ismétlés nélküli permutációs számára csökkenek, ami a faktoriálissal számolható. $7!=5040$ különböző lehetséges sorba rendezés. Ez az eredeti milliós nagyságrendhez képest jóval kedvezőbb futtatást eredményez, de a józan logika által meghatározott egyetlen lehetséges opcióhoz képest mégis sok. Miért is van ez? Hogyan lehetne ezen csökkenteni?

Vizsgáljunk meg a *14. ábrán* lévő táblázatban a 3. és 6. sort. Látható, hogy operátor szervezési szempontból lényegtelen, hogy milyen sorszámot adok az adott operátornak. Így több olyan variáció is szerepel a táblázatban, ami matematikailag különböző, sorszervezettség szempontjából viszont azonos. Ezeket vizuálisan az alábbi táblázatba foglaltam össze:

Variáció	1. Munkahely	2. Munkahely	3. Munkahely
1	1	1	1
2	1	1	2
3	1	2	1
4	1	2	2
5	2	1	1
6	2	1	2
7	2	2	1
8	2	2	2

17. ábra – Egyszerűsítési lehetőségek

A 17 ábrán pirossal sraffozva jelöltem azokat az opciókat, amik 2 fő esetén nem relevánsak, hiszen itt csak egyetlen dolgozót használunk fel a táblázatban. A többi esetben azonos színekkel jelöltem azokat a lehetőségeket, amiket, ha jobban megvizsgálunk annyiban térnek el egymástól, hogy milyen azonosítóval jelöltem a dolgozókat. Így ezek valóban egyformák, viszont mind a táblázat generálás idejét mind a szimuláció futási idejét feleslegesen terhelik le, így az első előfordulás után ezekkel már nem kell foglalkozni.

Észrevehetjük, hogy a táblázat jelen esetben szimmetrikus. Ez azonban csak a kettes számrendszer miatt van így, más számrendszerben ez nem törvényszerű. Viszont azt felhasználhatjuk, hogy az 1.Munkahelyen történő váltás után már nem fogunk tudni új, egyedi variációt felírni.

Logikusan belátható, hogy olyan mintázatot keresünk, ami csak egyszer fordul elő. Mivel növekvő számsorrendbe írtam fel a számokat, jobbról balra történik a számjegyek növelése. Ugyanúgy, mint a decimális számoknál is, a nulladik kitevőn lévők tagok vannak jobb oldalt, ezeket növelem a számrendszer elemeivel, majd, ha ezekből kifogytam növelem a következő kitevőt és újramezdem az alacsonyabb kitevők számlálását. Így, ha eljutottam addig a pontig, hogy a bal oldali oszlop kitevőjét kell emelnem, de a mintázat szempontjából már valójában az összes lehetséges variáción túl vagyok. Ez számunkra annyit tesz, hogy amikor az összes lehetőséget vizsgálom, eggyel kevesebb munkahelyre kell felírnom az alapképletemet, hiszen a bal oldali oszlopban tudom, hogy mindig „1” fog szerepelni. Tehát az operátor^{munkahely-1} képletet használhatom, ami a táblázatkészítés idejét csökkenti.

A következő feladatom, hogy a fent leírt logika szerint létrehozzak egy algoritmust.

3.4 A dolgozó eloszlástáblázatot készítő algoritmus bemutatása

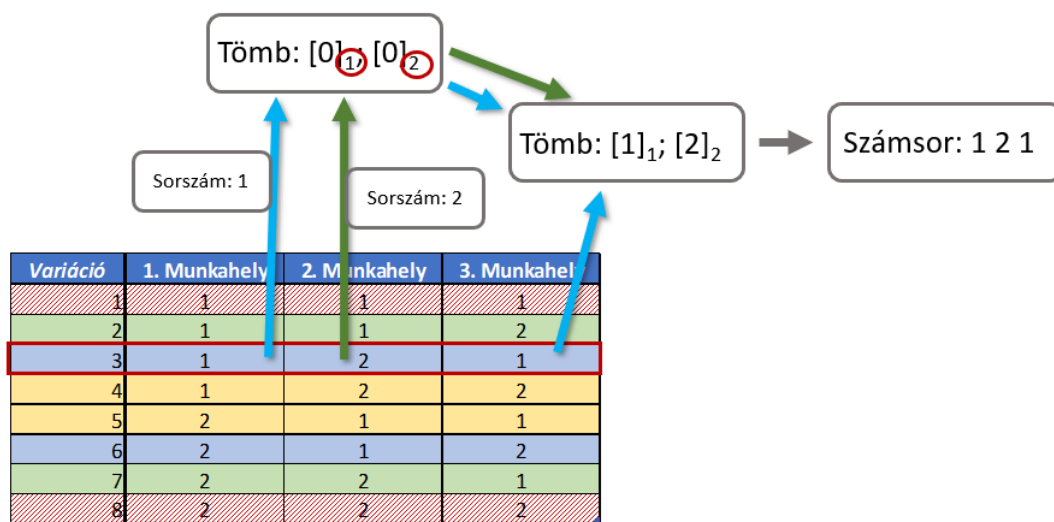
Az algoritmus létrehozására az Microsoft Excel beépített macro program modulját, a Visual Basicet használok. Mivel táblázatokban dolgozom kézenfekvő, hogy a táblázatkezelő programot használjam, de a fenti logikát már függvényekkel nem lehet lefedni, ezért elengedhetetlen a kódolás. Most azt fogom bemutatni, hogy milyen logika alapján hoztam létre a fenti algoritmust.

A számok generálása a 3.1.2 fejezet szerint meg is történik, azonban a mintázat felismertetése a programmal komolyabb feladat. Minden generált szám minden számjegyét egyesével meg kell vizsgálnom és az összkép alapján el kell dönteni, hogy ez a minta már előfordult-e korábban. Ez sok logikai vizsgálatot jelent, ezért is fontos, hogy az előző fejezetben le tudtuk csökkenteni a megvizsgálandó számok mennyiségét legalább egy kitevővel.

A minta ellenőrzéséhez minden sorban szereplő számkódot újra létrehozok, de egy általam meghatározott logikai feltételrendszer szerint. Ezek pedig a következők:

- létrehozok egy tömböt, ami annyi elemből áll, amennyi munkahelyet szeretnék vizsgálni, majd a tömb minden elemének az értékét 0-ra állítom
- létrehozok egy egyszerű változót, amit folyamatosan növekvő sorszámként használok, ennek a kezdeti értéke 1. Ez lesz a futó sorszámom.
- sorban megvizsgálom az adott számkód számjegyeit balról jobbra haladva
- az adott számjegyet a tömböm sorszámaként kezelem és megvizsgálom az ehhez tartozó értéket
 - o ha az ez érték 0, azaz még nem volt használatban, akkor nevesítem az értéket: a tömböm értékét beállítom a futó sorszámom értékére, majd a sorszámm értékét növelem 1-gyel
 - o ha az érték nem 0, azaz a számsor adott értékét már nevesítettem akkor egyszerűen ezt a nevet használok
- ezt követően a kapott számot beírom az új számsoron végére
- amikor a számkód minden számjegyre lefutott az ellenőrzés, megvizsgálom a tömböm minden elemét, ha találok benne 0-t, akkor az azt jelenti, hogy valamelyik számot nem kellett nevesítenem, azaz az eredeti számkódban valamelyik szám nem szerepelt. Ez pedig azt jelenti, hogy azt a számot nem használtam, tehát azt a dolgozót nem írtam be a táblázatba. Így ezt az eredményt nem fogom a végső táblázatba beírni, mert nem az adott létszámnak megfelelő
- másrésről megvizsgálom, hogy az újonnan létrehozott számkód magasabb értékű-e, mint a táblázatom utolsó sorába írt számkód

- ha az értéke magasabb, akkor ez az minta még nem fordult elő, így beírom az utolsó sorba
- ha az értéke alacsonyabb, akkor ezt azt jelenti, hogy ez a minta már valahol előfordult a táblázatomban, így nem írom be. Gyakorlatilag az utolsó lépés már csak egy egyszerűsítés a még rövidebb futási időhöz. Végig is nézhetném a már létrehozott kódokat, hogy az új kód egyezik-e valamelyikkel, de ezzel a lépéssel a számítások mennyiségét lehet csökkenteni



18. ábra: 1. példa a minta keresésre

Nézzük a gyakorlatban hogyan is működik, a 18. ábrán látható módon, vizsgálom a 3. sort. Ennek a sornak a bal első eleme: 1. Tehát megvizsgálom, hogy a tömböm [1] indexén milyen érték szerepel. Ez jelen esetben 0, így az [1] indexet értékét beállítom a futó sorszám értékére, ami jelenleg szintén 1, majd növelem a sorszám értékét. Ezt követően ezt az elemet beírom a számsorom végére. Jelenleg a számsorom szintén: 1.

Következő lépésként veszem a második elemet, ami 2. Ehhez megvizsgálom a tömböm [2] indexét, ami szintén 0, így beállítom ide a futó sorszám értékét, amit az előző alkalommal növeltem meg eggyel, így az már 2. Növelem a sorszám értéket, majd a tömb értékét beírom a számsor végére, ami jelenleg így néz ki: 12.

Veszem a soron következő elemet, ami 1. Megvizsgálom a tömböm értékét az [1] indexen, amit már nevesítettem, az értéke 1. Így nincs más dolgom, mint az ismert értéket beírom a számsor végére és így meg is kapom a végeredményt, ami: 121.

Tömb: [0]₁ [0]₂

Sorszám: 1

Sorszám: 2

Tömb: [2]₁; [1]₂

Számsor: 1 2 1

Variáció	1. Munkahely	2. Munkahely	3. Munkahely
1	1	1	1
2	1	1	2
3	1	2	1
4	1	2	2
5	2	1	1
6	2	1	2
7	2	2	1
8	2	2	2

Minden új sor vizsgálatakor a tömböm, a sorszám és a számsor értéke is visszaáll eredeti értékre. Ebben az esetben a sor bal első eleme 2, így a tömböm [2] indexét vizsgálom, ami 0, így beírom a futó sorszám értékét, az l -es értéket. Növelem a sorszám értékét, a kapott számot pedig beírom a számsorom végére, ami még most egyetlen számból áll: 2.

A harmadik esetben vizsgálom a $[2]$ -es indexet, de ehhez már tartozik egy l -es érték, így nem teszek mást, mint ez beírtom a számsor végére. Így előállt az előző feladatban is kapott: 121 .

4. Automata ciklusidő kiegyenlítő algoritmus

Ezt leködölni valamivel egyszerűbb, mint a fenti leírásban levezetni, az ehhez tartozó visual basic kód a következő:

```
For ii = 1 To allomas 'sorban megvizsgálom minden elemet
a = Mid(Cells(3, 3), ii, 1) + 1 'szöveggé tároltam, ezért kell kivágnom az adott elemet
If x(a) = 0 Then 'ha az x tömbben az adott elem nincs nevesítve, nevesítem
    x(a) = t
    t = t + 1
End If
StdMunka = StdMunka & x(a) 'beírom az x tömböm megfelelő elemét a sor végére
Next ii
```

A teljes kód természetesen ettől hosszabb, de ez a kódrészlet az érdemi része annak az algoritmusnak, amivel a futási idő csökkentést el lehetett érni.

3.5 A szimulációs algoritmus bemutatása

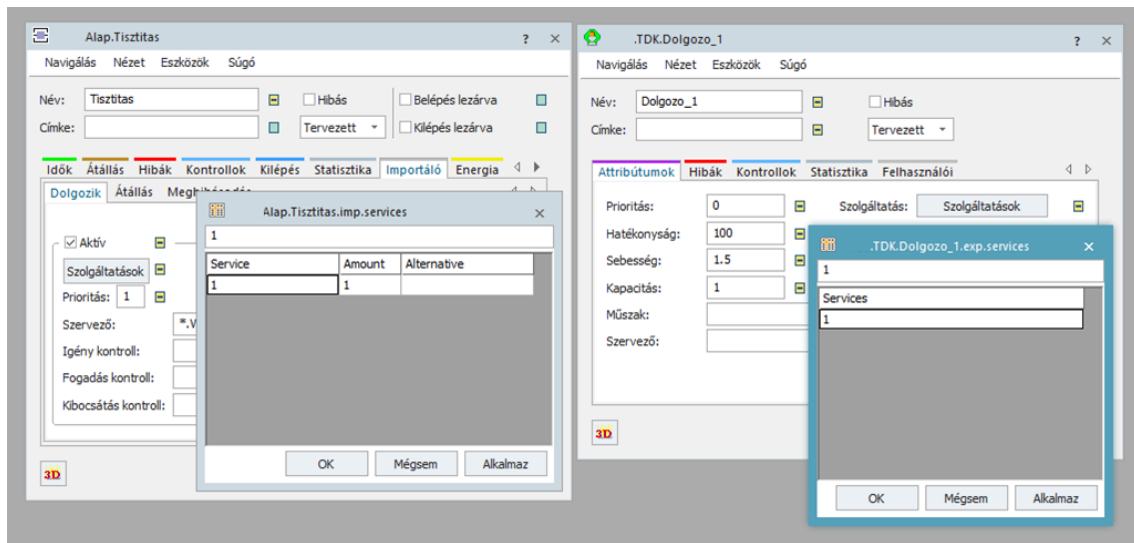
Az előző fejezet eredményeként létrejött egy a 19. ábrához hasonló táblázat, aminek az adatait fel kell töltenem a korábban létrehozott Minta soromba. A táblázatban minden oszlopban az szerepel, hogy melyik munkaállomáson melyik dolgozónak kell munkát végeznie.

A Plant Simulationban az egyes tevékenységek hozzárendelését szöveges formában találták ki, azaz egy dolgozóhoz hozzá tudok rendelni több feladatot, amit képes elvégezni, úgy, hogy a következő meghatározásokat használom pl. csavarozás, ragasztás, szegecseles majd az egyes munkaállomásokhoz is hozzá tudom rendelni, hogy ott milyen tevékenységet kell végrehajtani. Ha ez a két tevékenység egyezik, akkor a program a munkaállomásra behívja azt a dolgozót, aki az ott meghatározott feladatot el fogja tudni végezni.

Kicsit más gondolkodásmódban, de a dolgozói táblázatban meghatározott hozzárendelést ebben a formában meg tudom adni:

- minden állomáshoz egyetlen *service* feladatot rendelek
- az a feladat legyen, típuskonverzióval, annak a dolgozónak a száma, akit szeretném, hogy ott dolgozzon
- minden dolgozóhoz egyetlen *service* feladatot rendelek, ez pedig szintén típuskonverzióval a dolgozó száma. Azaz az 1. dolgozóhoz 1-es értéket, a 2. dolgozóhoz 2-es értéket és így tovább, rendelem hozzá
- a dolgozók értékei konstansok, azokat a program kísérletei során nem változtatom

- az állomások értékét kell kódba foglalni, hogy minden kísérlet előtt újradefiniálja



20. ábra – Munkahely-Dolgozó összerendelés

Ahhoz, hogy a fenti hozzárendelést lehető legegyszerűbben oldjam meg, létrehozok egy táblázatot, ami az állomások minden fontosabb adatát tartalmazza, így azt is, hogy az adott állomáson az adott kísérletben melyik dolgozónak kell munkát végeznie.

Alap.Allomasok						
0	object	boolean	object	string	boolean	boolean
1	KezTovabbitas	CelAllomas	Dolgozo	Std_Exp_be	Ugyv	
2	.Alap.Tisztitas	false	1	true		
3	.Alap.E_szer_I	false	2	true		
4	.Alap.E_szer_II	false	3	true		
5	.Alap.Szeres_I	false	3	true		
6	.Alap.Szeres_II	false	Alap.Ell_Valaszt	3	true	
7	.Alap.Ell_I_be	false	3	true		
8	.Alap.Ell_I_ki	false	3	false		true
9	.Alap.Ell_II_be	false	3	true		
10	.Alap.Ell_II_ki	false	3	false		true
11	.Alap.Csom_I	false	3	true		
12	.Alap.Csom_II	false	Alap.Admin	3	true	
13	.Alap.Admin	false	3	true		
14	.Alap.Ell_Valaszt	false		false		

21. ábra – Állomás adatok

Ezt a táblázatot tudom adatbázisként használni és a munkaállomásra mint, kulcs, tudok hivatkozni. A szimulációban beállítható egy függvény, ami minden egyes futtatás indítás előtt végrehajtódik. Abban a függvényben egyértelműen megtörténhet a hozzárendelés, azaz az állomásra hivatkozva meg tudom találni, hogy hányas számú dolgozónak kell ott lennie, ezt az értéket pedig beállítom a munkahely *service* paraméterébe. Azt már azt elértem, hogy ha manuális

átírom

21. ábrán szereplő táblázat 3. oszlopának az adatait, akkor azt az adott futtatás indításakor az állomásokhoz rendeli.

Most elő kell vennem a Kísérlettervezőt, ahol be kell állítanom:

- a bemeneti változók definiálást, vagyis azt, hogy kísérletről kísérletre mi változzon a programban
- az elemzések definiálását, azaz azt a táblát, ami alapján változtatja a bemeneti értékeket
- a kimeneti értékeket, amikor majd eredményt szeretnék látni

A bementi változókhoz beírom a 21. ábrán szereplő táblázat 3. oszlopát. Tehát azt kérem a programtól, hogy minden kísérlet előtt töltsen fel az új dolgozói eloszlást, majd az indításkor automatikusan lefutó függvénnyel ezt hozzá is rendelem az állomásokhoz.

Az elemzések táblázatba betöltöm a táblázatkezelőben létrehozott sorokat, amit az előzőekben már olyan formátumban alkottam meg, hogy egymással kompatibilisek legyenek és egy egyszerű másolás művelettel másolható legyen a bemenő adat.

A kimeneti értékekhez pedig azokat a változókat adom meg, amiknek az eredményét a szimulációk végén ki szeretném értékelni:

- Elkészült darabszám -> létező funkció, a *Termek_Ki* objektumot elhagyó elemek
- Dolgozók által megtett út -> szintén létező statisztikai változó. Minden operátor objektum rendelkezik egy változóval, ami figyeli a megtett utat. Ezt dolgozóként külön hivatkozom meg
- Átlagos átfutási idő -> ez nem létező megfigyelő, ezért ezt létre kell hoznunk. A Plant Simulationban az elemek rendelkeznek egy olvasható változóval, amit bekérve megkapjuk az adott elem létrehozásának az idejét. Az elkészülés, azaz a kiemenet idejét tudom minden egyes elemnek, hiszen az az időpont amikor megérkezett a *Termek_Ki* állomásra. Így az elkészülés időpillanatából kivonva a létrehozás, azaz a belépés időpillanatát, megkapom az adott elem soron töltött idejét, azaz az átfutási időt. Ezt táblázatba rendezem az összes elkészült elemre, összegzem, majd a futtatás végén elosztom az elkészült darabszámmal, akkor kapok egy Átlagos átfutási időt.

Ezek alapján elkészítettem a Kísérlettervezőt, amit elindítottam. A teljes futási időtartam kicsivel kevesebb mint másfél óra volt. Ebből nagyságrendileg 10 perc volt az, ami a Dolgozói táblázat elkészítéséhez kellett. A maradék időt a 9330 kísérlet 3 futtatásban történő elvégzése ölelte fel.

4. Eredmények bemutatása

Ebben a fejezetekben nézzük meg, hogy az algoritmussal és a szimulációval milyen eredményeket lehet elérni a Minta gyártósoron. Ezt két részre bontottam, először bemutatom, a program milyen eredményeket hozott, ténylegesen sikerült-e javítani a ciklusidő kiegyenlítésén a korábban megadott KPI-ok vonatkozásában? Vajon használható-e az Automata Ciklusidő kiegyenlítő algoritmus?

Ezt követően pedig azt elemzem, hogy a táblázat készítő algoritmus segítségével mennyi számítási kapacitást lehet megtakarítani, és ez számunkra, órába lefordítva mennyi hasznos időt jelent.

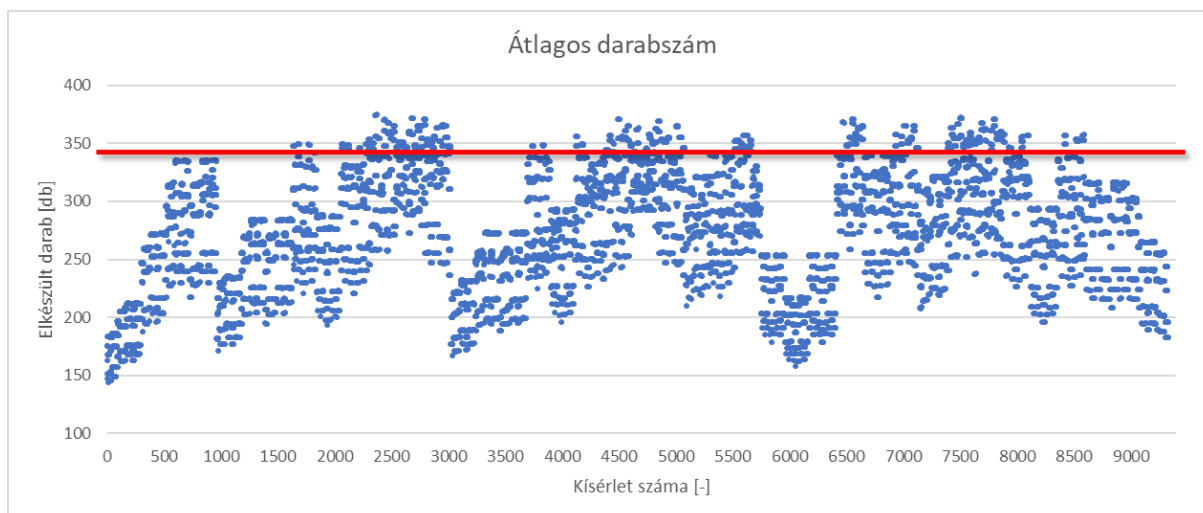
4.1 Szimuláció eredményei

Az eredmények ellenőrzéséhez először bizonyosodjunk meg arról, hogy a szimulációnk még mindig jól működik és visszakeresve az általunk beállított dolgozó eloszlást ugyanazt az eredményt hozza, mint amit eddig is.

Ez az Exp 7117 néven futó kísérlet volt, és a következő eredményeket hozta:

- Elkészült termék átlagosan: 346 db
- Átfutási idő 12 perc
- Dolgozók által megtett út:
 - o 2629 méter
 - o 1472 méter
 - o 3568 méter

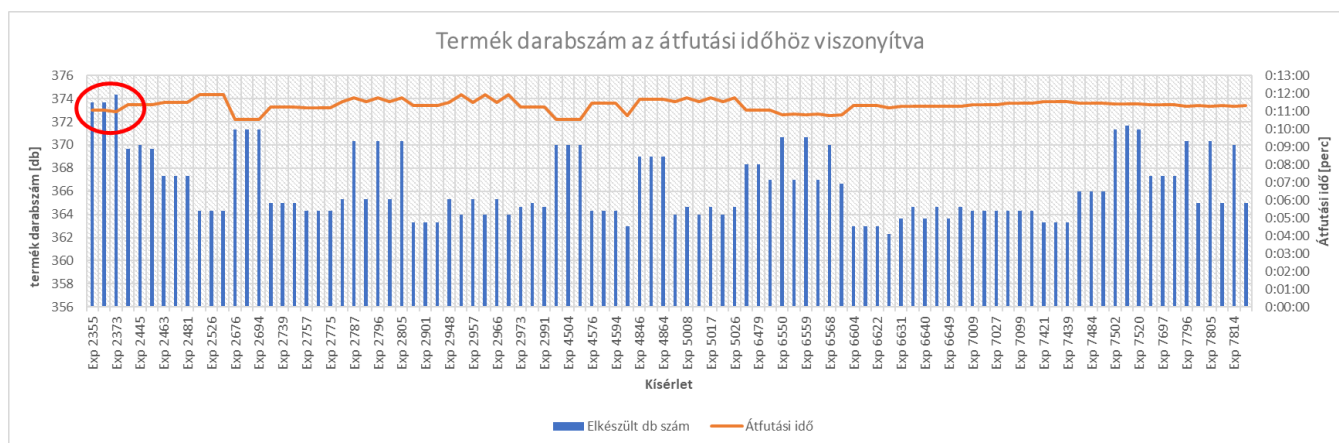
Tehát ugyanazokat az eredmények állnak rendelkezésünkre, mint eddig is. Most nézzük meg, hogy a legyártott termékek vonatkozásában találunk-e jobb eredményt:



22. ábra – A kísérlet eredménye a darabszám függvényében

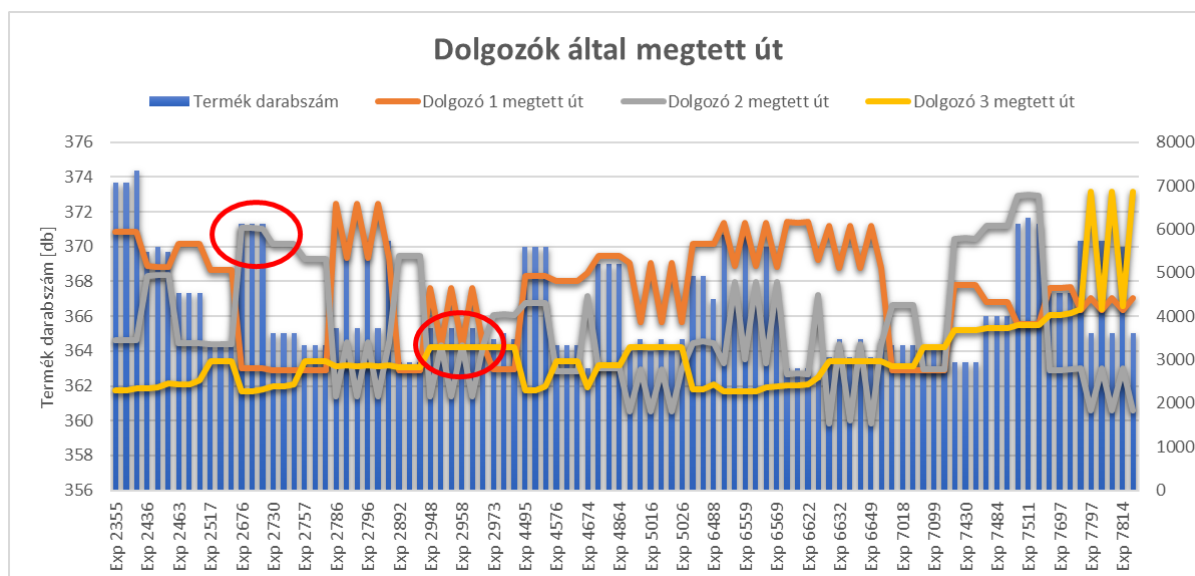
A 22. ábrán piros vonallal jelöltem be a közelítőleg a 345 db-ot és azt látom, hogy több olyan kísérlet is volt, ami ezek felett tudott teljesíteni.

Most szűkítsük az adatokat csak azokra a kísérletekre, amik számunkra eredményesek voltak. Ez 594 esetben volt lehetséges, ezért még ezt is szűkítsük le olyan módon, hogy csak azokat az esetekre nézzük, amik legalább 5%-os javulást eredményeztek. Majd elemezzük ezeket az eredményeket a többi KPI tekintetében.



23. ábra – Darabszám átfutási idő függvényében

A 23. ábrát megvizsgálva látható, hogy az átfutási időben nincs különösebb változás, ebből a szempontból bármelyik dolgozó eloszlás jó lehet, ami több darabszámot eredményezett. Ezért, ha ez a fő szempont akkor célszerű lenne a pirossal bekarikázott kísérletek közül választani, hiszen itt értük el a legtöbb darabszámot.



24. ábra – Dolgozók által megtett út a darabszám függvényében

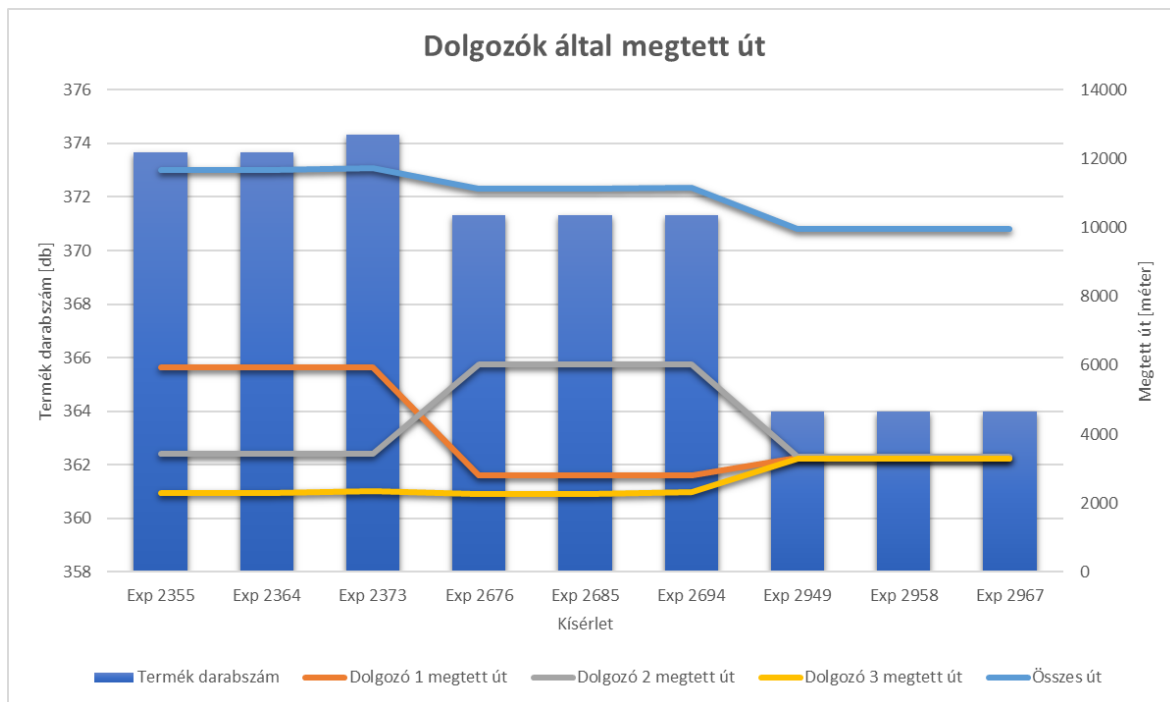
Ha viszont inkább az a célunk, hogy a dolgozók terhelést kiegyenlítsük és ezért hajlandóak vagyunk darabszámot áldozni, akkor a 24.ábra alapján célszerű lehet az ott pirossal bekarikázott kísérlet közül választani.

Ez innentől kezdve már kicsit vállalati politika is, hogy a cég melyik változatot támogatja jobban. Megéri-e hosszútávon a bevétel kieséssel járó opció választása, ha ezzel a dolgozók kényelmesebben dolgoznak? Esetleg tudjuk-e valamilyen módon forgatni a dolgozókat, hogy ne kelljen egy adott dolgozót terhelni?

4.1.1 Fejlesztési javaslat

Szűkítsük tovább a lehetőségeket és hozzunk egy döntést arra vonatkozóan, hogy a Minta sorunkat hogyan fejlesszük, mi legyen az új dolgozói eloszlás standard? Azt már megállapítottuk, hogy az átfutási idő közel állandó minden esetben, így nézzük csak a dolgozók által megtett út oldaláról. Tekintsük azt, hogy a dolgozókat lehet forgatni a műszakban, így, ha egy dolgozónak többet is kellene mozognia, akkor azt sem egész műszakon keresztül végzi. Ez alapján jelenítsük meg a teljes megtett utat is.

A következő, 25. ábrán már csak azokat az eseteket jelenítettem meg, amik a felső ábrákon a prioritizált piros körökben szerepeltek. Így a lehetséges eseteket 9 előfordulásra sikerült szűkítenem.



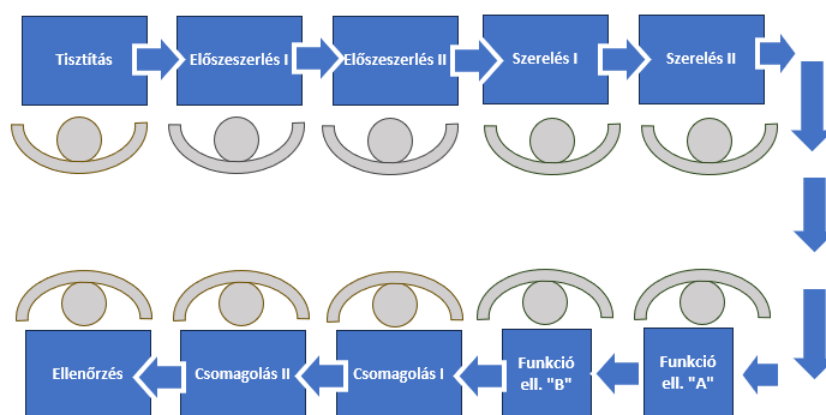
25. ábra – 9 esetre szűkített

Nézzük meg, hogy a Dolgozói eloszlás táblázatban ezek az esetek mit jelentenek:

	Tisztítás	ElőszereI és I	ElőszereI és II	Szerelés I	Szerelés II	Funckió ellenőrzés "A" berendezés	Funckió ellenőrzés "B" berendezés	Csomagolás I	Csomagolás II	Ellenőrzés, Adminisztráció
Exp 2355	1	1	2	3	1	1	3	1	2	2
Exp 2364	1	1	2	3	1	1	3	2	2	2
Exp 2373	1	1	2	3	1	1	3	3	2	2
Exp 2676	1	1	2	3	2	2	3	1	1	2
Exp 2685	1	1	2	3	2	2	3	2	1	2
Exp 2694	1	1	2	3	2	2	3	3	1	2
Exp 2949	1	1	2	3	3	3	1	1	2	2
Exp 2958	1	1	2	3	3	3	1	2	2	2
Exp 2967	1	1	2	3	3	3	1	3	2	2

26- ábra – Dolgozói eloszlástábla a legjobb 9 esetre

Szürkével jelöltem a 26. ábrán azokat a részeket, amit mind a 9 esetben megegyeznek, ezekkel nem kell foglalkoznunk a döntés alkalmával. Ha tovább vizsgáljuk a lehetőségeket azt vesszük észre, hogy bár az első három esetben értük el a legnagyobb darabszámot, itt volt az 1. dolgozó által megtett út is a legnagyobb. Látható, hogy nem véletlenül, hiszen a sor minden részén megfordul és a duplikált állomások közül mindegyiknél végez valamilyen munkát. Hiába lennének darabszám szempontjából ezek az esetek a kedvezőbbek, a sok egymást keresztező út miatt vesszük ezek alkalmazását.



27. ábra- A sori elrendezés a dolgozók kijelölése nélkül

Tovább vizsgálva, azokat az opciókat, amelyek jók lehetnek:

- Exp 2676: kompromisszumos megoldás a darabszám és a megtett út tekintetében, bár több keresztezési pont itt is van
- Exp 2685: lényegesen több keresztezési pont van, ezért kizárom
- Exp 2694: kompromisszumos megoldás a darabszám és a megtett út tekintetében, bár több keresztezési pont itt is van
- Exp 2949: Dolgozók által megtett útra optimalizált változat és bár van keresztezés, az nem zavaró módon fordul elő
- Exp 2958: Dolgozók által megtett útra optimalizált változat és bár van keresztezés, az nem zavaró módon fordul elő
- Exp 2967: lényegesen több keresztezési pont, ezért kizárom

Tehát, ha az volt a célom, hogy beruházási költség nélkül egy élhető ciklusidő eloszlást találjak a soron, ami darabszám növekedéssel jár, attól függően, hogy a másodlagos célom volt-e a dolgozók által megtett út kiegyenlítése vagy sem, akkor az alábbi esetek bármelyikét választhatom:

	Tisztítás	Előszerszerelés I	Előszerszerelés II	Szerelés I	Szerelés II	Funckió ellenőrzés "A" berendezés	Funckió ellenőrzés "B" berendezés	Csomagolás I	Csomagolás II	Ellenőrzés, Adminisztráció	Elkészült db szám	Átfutási idő [sec]	Dolgozó 1 megtett út [méter]	Dolgozó 2 megtett út [méter]	Dolgozó 3 megtett út [méter]
Exp 2676	1	1	2	3	2	2	3	1	1	2	371	632	2815	6035	2276
Exp 2694	1	1	2	3	2	2	3	3	1	2	371	632	2815	6014	2326
Exp 2949	1	1	2	3	3	3	1	1	2	2	364	715	3323	3344	3288
Exp 2958	1	1	2	3	3	3	1	2	2	2	364	715	3323	3344	3288

28. Ábra – Választható eloszlások a ciklusidő kiegyenlítéseként

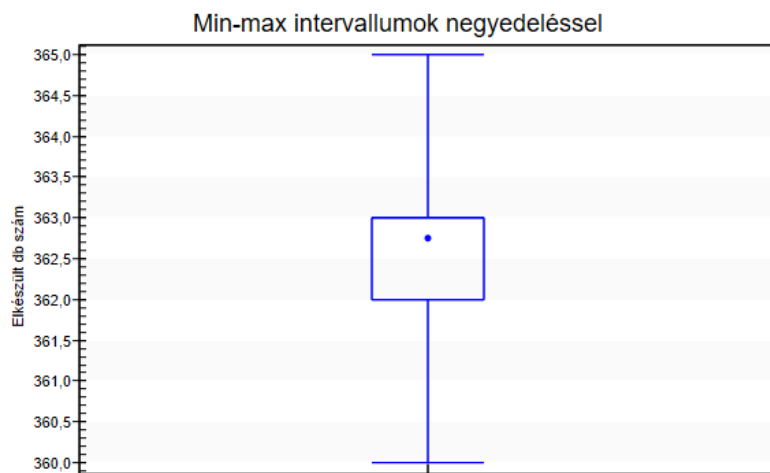
Én ebből az Exp2958-at választottam. Bár a cél a darabszám növekedés volt, ezt ésszerű határok között kell elérni és az első két esetben a 2-es Dolgozó útját több helyen is keresztezik és irreálisan nagy utat kell bejárnia egy műszak alatt.

A 2949 és 2958 közül az döntött, hogy ebben az esetben a folyamat végső szakaszát, így a teljes csomagolást egy ember végzi.

4.1.2 Exp 2958 részletesebb elemzése

Így az általunk választott opcióra már van értelme egy több kísérletből álló Kísérlettervet futtatni. Ennél az egy kísérletnél már be lehet állítani a futások számát egy nagyobb értékre, hogy a szélsőértékek között egy szélesebb spektrumban lássam a lehetőségeket.

A futások számát felemeltem 100 db-ra, ebben az esetben a futási idő 20 másodperc volt és a következő eredményt adta.



29. ábra – Exp2958 elkészült darabszám, 100 futtasás alkalmával

Az elkészült darabszám a beállított eloszlásgörbék szerint várhatóan 360 és 365 között ingadoznak., mint az a 29.ábárn is látható. Átlagosan viszont 362-363 darabszám érhető el. Megállapítható, hogy a rendszernek nincs jelentős ingadozása, a program által számított szórás is 1,12 db, így elfogadható a fenti eredmény, ezzel együtt az új dolgozói eloszlás.

4.2 Számítási kapacitás csökkentés

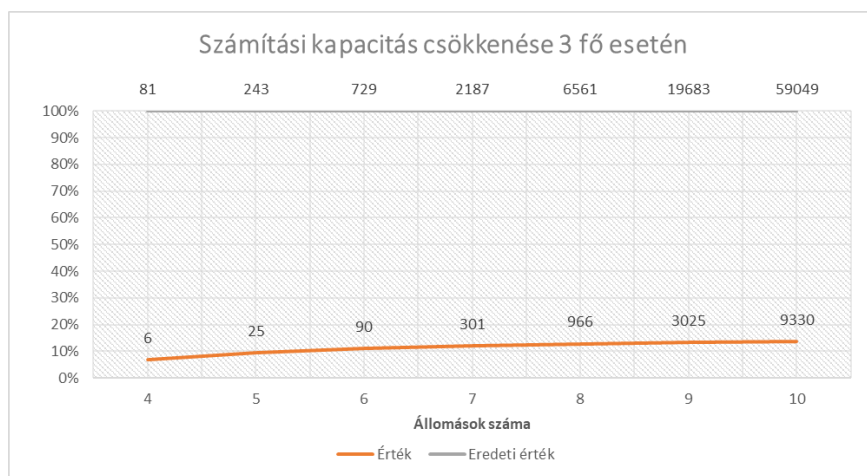
A Minta példasorban szereplő esetben a 3 főt 10 állomásra kombinatorikai számítások alapján 59049 különféle módon tudom elhelyezni. Lefuttatva az algoritmust ez a szám lecsökkent 9330 db-ra, ami közel 85%-os számítás csökkenés.

Processzor, memória és a szimuláció összetettsége függvényében változó időt vehet igénybe a számítási művelet, most fogadjunk el egy olyan kiinduló értéket, hogy 10.000 szimuláció futtatása nagyságrendileg 1 órát vesz igénybe. Így, ha a 9330 lehetséges esetet minimum 3 szimulációs kísérletben futtatom, akkor az 3 órán keresztül fog tartani. Még ugyanez az eset az egyszerűsítő algoritmus nélkül közel 18 órát venne igénybe.

Így elsőre a 18 óra sem egy olyan érték, amit nem lehetne kivárni, de vegyük figyelembe, hogy mindösszesen 3 fővel dolgoztam. Minél nagyobb a létszámom, vagy több a munkahely annál több esetet kell megvizsgálnom. Illetve vállalat függő, de nem feltétlen rendelkezik a vállalat egy 24 órás futó számítógéppel, amit magára lehetne hagyni. Így a 18 óra 2 egész napos munkaidőt vesz igénybe, ami mellett a mérnök, aki futtatja a programot tud dolgozni, de 3D tervező szoftvert már nem feltétlen bír el a processzor.

Ezzel szemben egy 3 órás futtatás idejére könnyebben nélkülözhető a számítógép.

Diagramba összefoglaltam, hogy milyen futási számmal lehet számolni, ha maradunk annál az opciónál, hogy 3 fő dolgozik a soron, de különböző számú munkahelyre kell őket elosztani. Ha át akarjuk váltani időre, akkor használjuk a 10.000 futtatás 1 óra közelítést.



20. ábra – Számítási kapacitás csökkenése 3 fő esetén

A 20. ábrán látható diagrammon a sárga vonal jelöli, hogy az eredeti variációk számának hány százalékát kell a szimulációban lefuttatni, felette az érték pedig a futások darabszámát mutatja. A felső szürke vonal jelöli az egyszerűsítés nélküli 100%-ig halmazott állapotot, valamint az ehhez tartozó futási számot. Ezek alapján elmondható, hogy a megtakarítás értéke egy logaritmikus skálán halad és közelít a 15% -os várható érték irányába.

Azaz nagyságrendileg a futási időt 85%-al tudjuk csökkenteni a fenti algoritmust használva. Ez különösen fontos akkor, amikor nem csak egy adott kísérlettervezést akarunk végrehajtani, hanem több opciót is szeretnénk megvizsgálni

5. Összefoglalás

Összegezve azt, amit fentebb a programmal el lehet érni, megállapítható, hogy az általunk meghatározott fő mutatószámok közül sikeresen tudtunk a darabszámon növelni. A dolgozók által megtett út is egyenletesen lett elosztva, azonban árnyalja a képet, hogy az általuk megtett összes út jelentősen növekedett.

	Eredeti	Elért	Megjegyzés
Darabszám [db]	345	362	5%-os növekedés
Átfutási idő [perc]	12:01	11:55	Nincs jelentős változás
Dolgozó 1 által megtett út [m]	2620	3323	Bár 2 dolgozó esetén többlet utat jelent, ez darabszám növekedést eredményezett
Dolgozó 2 által megtett út [m]	1467	3344	
Dolgozó 3 által megtett út [m]	3565	3288	

21. ábra – KPI-ok összegzése

A 21. ábrán látható, hogy az Automata Ciklusidő Kiegyenlítő algoritmus működik és bár nem feltétlen a való életből vett példán keresztül mutattam be a működését, az elv az jó és bármelyik termelősorra alkalmazható, ahol több operátort kell elosztani a munkaállomások között.

Az elsődleges célunk a darabszám növelése volt, és ez jár olyan kompromisszummal, hogy a dolgozók által megtett út növekedett. Mindazonáltal igyekeztem azt az opciót választani, ahol ez nem túlzó. Meg kell jegyezni, hogy ha azt akartam volna megnézni, hogy az összes mozgási időt nem növelve milyen lehetőségeim vannak, a fenti 9330 adatból ezt is meg tudtam volna keresni.

Az alábbiakban a 22. ábrán szemléltetve, egy SWOT analízissel vizsgáljuk meg, hogy mit jelent számunkra az algoritmus használata, mikor és kiknek ajánlott?

Előnyök:

- Ahogyan a dolgoztam elején egy egyszerű önvizsgálattal is bemutattam, az emberek jelentős része standardokban és megszokáson alapuló munkavégzésekben gondolkodik, nehezen vonatkoztat el a már bejáratott úttól és nem próbálunk ki új lehetőségeket. Milyen új meglátást nyitott ki számunkra az algoritmus? Jelen esetben, hogy ha kicsit rugalmasabb és engedik, hogy a dolgozók útvonala minimálisan keresztezze egymást, akkor ezzel darabszám növekedést tudnak elérni, úgy, hogy nem volt műszaki beruházási költsége az új folyamatnak. A megtérülése azonnali és 100%-os.
- Minden lehetőséget megvizsgál a program így a számunkra legkedvezőbb lehetőséget tudjuk választani, attól függően, hogy mi a célunk. Emlékezzünk vissza, több mint 500 olyan esetet listáztam ki, ahol darabszám javulást lehetett elérni, és ezt szűkítettem első körben 100 majd 9 végül 4 lehetőségre, melyből már kiválasztottam azt, ami a vállalat víziója szerint a legkedvezőbb.
- A modul könnyen implementálható egy másik szimulációba. Maga a kísérlettervezés és a hozzá tartozó pár objektum, ami a lényegét adja. Ezek egyszerű másolással és a hivatkozási út megadásával modúlárisá teszik az algoritmust.
- Relatív gyors eredményre képes és ha az első kör után leszűkítjük azokra az esetekre, amik számunkra kedvezőnek tűnnek. A paraméterek finomításával már csak ezen eseteket vizsgálva széles körben gyors elemzéseket tudunk végrehajtani.
- A Dolgozói táblázat elkészítését elég egyetlen egyszer futtatni, hiszen az csak az állomás és a dolgozók számától függ. Amennyiben ezt a táblázatot elmentjük, gond nélkül alkalmazható egy másik ugyanilyen létszámmal és állomásszámmal működő termelősorra is, akkor is, ha annak a sornak az értékárama teljesen eltérő. Ezzel a futási idő tovább rövidülhet.

Hátrányok:

- Kompromisszumot kellett kötni a dolgozók útját illetően, nem találtunk olyan megoldást, hogy a darabszámot növeljük, de ne legyenek benne keresztező utak.
- Nem sikerült a dolgozók útját lerövidíteni, sőt 2 esetben többet is kell mozogniuk, bár a műszakban termelhető darabszám így javítható volt.
- Nem szabad minden eredményt elhinni, amit a program ad. Kritikus és szakértő szemmel meg kell vizsgálni, hogy el tudjuk dönteni megvalósítható-e az opció. Tehát nem automatizálható teljes mértékben, kell a mögöttes szaktudás, hogy a döntést meghozzuk.
- Amennyiben a dolgoznál a soron következő állomásra még nem érkezett meg a darab, de közben más állomáson képes lenne munkát végezni, akkor nem fog a soron következő állomáson dolgozni, hanem arra a feladatra ugrik, amit el tud végezni. Ez befolyásolhatja mind az elért darabszámot, mind a megtett utat, de nem életszerű.
- Jelenleg 10 főig képes mind a két algoritmus dolgozni. Ennek oka, hogy a dolgozók számától függő számrendszert választottam, tehát 16 fő esetén a hexadecimális számrendszerben kellene leképeznem a táblázatot, ami viszont a nem numerikus karakterek miatt jelenleg nem lehetséges. Az indexelésnél és a végső szám összehasonlításánál is gondot okozna. Ha szükséges, módosításokkal és a kód fejlesztésével mind a két algoritmus bővíthető tízes számrendszer feletti leképzésre is, de erre eddig nem volt igény így nem fektettem bele plusz energiát.
- A táblázat készítés ideje is képes akár napokra is elnyúlni, ha már 8 feletti létszámot kell elosztani értelemszerűen 8-tól több állomásra. Azonban, ahogy ezt korábban megjegyeztem ez egyszeri futtatást jelent, amint ez a táblázat rendelkezésre áll, el lehet menteni és legközelebb már nem kell létrehozni.

Lehetőségek:

- Az általam meghatározott mutatószámok csak példa jellegűek voltak, a programmal képesek vagyunk bármilyen, a számunkra fontos tényezőt elemezni. Lehetett volna vizsgálni selejt kezelést egy külön állomáson, több típus gyártása esetén egy optimális tervet vagy akár gépmeghibásodást és javítási időt, ha szeretnénk volna
- Keresztező utak figyelése, nagyobb hangsúlyt fektetni arra, hogy a dolgozók által megtett út a minimális legyen. Ehhez akár layout módosítást is alkalmazni. Ezt a programmal lehet tesztelni, majd a termelő soron bevezetni.
- Lehet finomítani a termelősoron nem hivatalos pufferek beépítésével. Pl.: egy valós soron, ha a dolgozó látja, hogy pár másodpercen belül kapni fog terméket, akkor azt megvárja és nem kezd bele másik állomás feladatába. Ezt egy szimulációba beépített, de a valóságban nem létező egy darabos puffer létrehozásával lehet finomítani.
- Az általunk kapott eredmények figyelembevételével tudunk javaslatokat tenni, a köztes pufferekre, a layoutra, a gyártási tervre, amit a valós termelő sorra is át lehet vezetni.

Veszélyek:

- A szoftver tanulása autodidakta módon vagy az interneten elérhető hivatalos anyagokból is elsajátítható, azonban a használata igényel némi jártasságot. Az igazi mélységét akkor adja, amikor már függvényeket is képesek vagyunk benne létrehozni, és táblázatból, automatikusan állítani az egyes paramétereket. Pont a kezdeti sikerek miatt könnyen elbízhatjuk magunkat, a használatával kapcsolatban, a kezdeti fázisban éppen ezért érdemes óvatosan bánnunk az eredményekkel.
- A szoftver olyan opciókat is kínál, amik a való életben nem kivitelezhetőek ezért mindenképpen át kell gondolni milyen fejlesztést választunk.
- Minden dolgozót egyforma képességekkel ruháztunk fel. A valóságban viszont előfordulhat, hogy egyik dolgozó az adott állomáson jobban, még a másikon kevésbé hatékonyan tud dolgozni. Ez a szimulációban kezelhető és futtatható, azonban a Dolgozói táblázat egyszerűsítésnek így nincs értelme, hiszen az 1-2-1 minta innentől nem lesz egyenértékű 2-1-2 mintával.



22. ábra – SWOT analízis

Összeségben azzal zárnám, ami minden programra és gépre érvényes: ha szakértő módon, intelligensen és kellő kritikával használjuk, akkor képes a program arra, hogy segítse a termelésünk fejlődését és olyan megoldásokat kínáljon, aminek a befektetési költsége közel nulla, mégis a fő mutatószámaink javulását eredményezi.

6. Felhasznált irodalom

- [1] Kaizen Pro: Lean szótár - szűk keresztmetszet. Internetes
forrás: <https://leanszotar.hu/page.php?151>, megtekintés dátuma: 2024. 03. 04.
- [2] Kaizen Pro: Lean szótár - idővel kapcsolatos fogalmak. Internetes forrás:
<https://leanszotar.hu/page.php?84>, megtekintés dátuma: 2024. 03. 04.
- [3] Pengjie Gao, Junliang Wang, Ray Zhong, Jie Zhang, Neuron synergy based explainable neural network for manufacturing cycle time forecasting, Journal of Manufacturing Systems, Volume 71, 2023, Pages 695-706
- [4] S.R. Vijayakumar, P. Suresh, Lean based cycle time reduction in manufacturing companies using black widow based deep belief neural network, Computers & Industrial Engineering, Volume 173, 2022, 108735
- [5] Jad Imseitif, He Tang, Mike Smith, Throughput Analysis of Manufacturing Systems with Buffers Considering Reliability and Cycle Time Using DES and DOE, Procedia Manufacturing, Volume 39, 2019, Pages 814-823
- [6] Y. Meidan, B. Lerner, M. Hassoun, G. Rabinowitz, Data Mining for Cycle Time Key Factor Identification and Prediction in Semiconductor Manufacturing, IFAC Proceedings Volumes, Volume 42, Issue 4, 2009, Pages 217-222
- [7] Czeslaw Smutnicki, Minimizing cycle time in the manufacturing system based on the flow of various jobs, IFAC Proceedings Volumes, Volume 42, Issue 4, 2009, Pages 1137-1142
- [8] Alexandros Koulouris, Georgios P. Georgiadis, An exact algorithm for calculating the minimum and feasible ranges of cycle time in periodic scheduling with shared resources, Computers & Chemical Engineering, Volume 175, 2023, 108286
- [9] Mensa HungarIQa - Horváth Zoltán Henrik, Néhány ravaszabb feladat avagy egy pályaaorientációs nap margójára. Internetes forrás: <https://mensa.hu/asztallap/nehany-ravaszabb-feladat-avagy-egy-palyaorientacios-nap-margojara/>, megtekintés dátuma: 2024. 03. 06.
- [10] Schrödinger kutyája - A túlélési torzítás. Internetes forrás:
https://schrodingersdawg.blog.hu/2017/07/18/a_tulelesi_torzitas, megtekintés dátuma 2024. 03. 06
- [11] II. Világháború fegyverei - Történetek- Wald és a túlélési torzítás. Internetes forrás:
https://2vilaghaborufegyverei.blog.hu/2021/04/30/tortenet_wald_es_a_tulelesi_torzitas, megtekintés dátuma: 2024. 03. 06

- [12] Siemens – Tecnomatix Plant Simulation. Internetes forrás:
<https://plm.sw.siemens.com/en-US/tecnomatix/products/plant-simulation-software/> ,
megtekintés dátuma: 2024. 03. 07
- [13] Siemens - What's new in Plant Simulation?. Internetes forrás:
<https://blogs.sw.siemens.com/tecnomatix/whats-new-in-plant-simulation-august-2021/> ,
megtekintés dátuma: 2024. 03. 07
- [14] ictglobal - Tölgyes László András, A digitális iker technológia és a metaverzum.
Internetes forrás: <https://ictglobal.hu/iparagi-megoldasok/a-digitalis-iker-technologia-es-a-metaverzum/> , megtekintés dátuma 2024. 03. 07
- [15]. Forbes - What Is A KPI? Definition & Examples. Internetes forrás:
<https://www.forbes.com/advisor/business/what-is-a-kpi-definition-examples/> , megtekintés dátuma 2024. 03. 12
- [16] Mateking – Ismétléses variáció. Internetes forrás: <https://www.mateking.hu/matematika-kepletyujtemeny/ismetleses-variacio> , megtekintés dátuma 2024. 03. 13