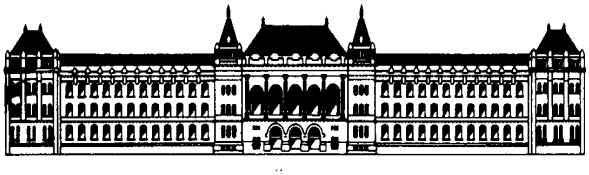


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Elektronikai Technológiai Tanszék

Tudományos Diákköri Konferencia Dolgozat

A raktári tárhelykijelölési probléma megoldása genetikus algoritmussal

VONYÓ PÉTER ANDRÁS

HUOKFQ

2022

Tartalom

Absztrakt

A logisztika a civilizációk velejárója. Már a római hadseregben is voltak „logisták” akik a légiók ellátásáról gondoskodtak. A bizánci birodalom írásaiban is definiálták a logisztika feladatait. Az elmúlt évtizedekben jelentősége felértékelődött. Ennek számos oka van, mint például a szétterülő ellátási láncok, a kiszolgálási szinttel támasztott magas elvárások és a modern értékalkotási láncok szükségletei.

A logisztika fogalma két nagyobb részre bontható. Az egyik a fizikai mozgásokkal (vállalati telephelyen belüli un. intralogisztika és vállalati telephelyen kívüli un. extralogisztika), a másik pedig a fizikai folyamatokhoz szükséges információáramlással foglalkozik. Az intralogisztikai folyamatok közül a legfontosabb a komissiózás, mert egy raktár költségeinek körülbelül 50%-át teszi ki. A komissiózás támogatására több módszer és technológia létezik (pl. több lépcsős komissiózás, komissiózó gépek, pick by light rendszerek stb). Az általam kiválasztott probléma azzal foglalkozik, hogy az áru olyan tárhelyre kerüljön, ahonnan hatékonyan megoldható a kigyűjtés a listán szereplő többi termék függvényében. Ezt hívja a szakirodalom tárhely-kijelölési problémának (storage location assignment problem-SLAP), melynek jó megoldása, jelentős mértékben csökkenti a komissiózó ágens (gép/ember) tárhelyek közötti mozgási időszükségletét.

Kutatásaim során részletesen vizsgáltam a komissiózás kérdéskörét, kiemelten a SLAP problémát. Megvizsgáltam az iparban jelenleg használt, legelterjedtebb módszereket, továbbá vizsgáltam tanulmányokat az elmúlt 5-10 évben megjelent új megközelítésekről. Készítettem egy saját, genetikus algoritmusokon alapul SLAP megoldó algoritmust, melynek teljesítményét több tesztesetet vizsgálva összehasonlítottam az alábbi módszerekkel:

- a ma leginkább elterjedt „véletlen betárolás” módszerével, ami kontrollként is szolgál,
- a Pareto-elv alapján működő „ABC” betárolási stratégiával, és
- egy datamining módszereken alapuló heurisztikus megközelítéssel.

Kutatásaim során Ch. Kamset és P. Meesuk 2014-ben megjelent raktármodelljét használom, melyre számos tanulmány hivatkozik. Java környezetben implementáltam a raktármodellt és a betárolási stratégiákat, melyeket aztán különböző paraméterekkel rendelkező raktáregyedeken vizsgáltam. A heurisztikus megközelítésnél a datamining feladatokat R-ben végeztem, egy ún. apriori algoritmus segítségével. Ezen apriori algoritmus használatához minősített internetes forrásokat használtam.

Az általam használt árucikkeket és megrendelési adatokat az R-szoftver „Groceries” nevű adatbázisa szolgáltatja. 165 terméket és ezen termékekhez tartozó több ezer megrendelést tartalmaz ez az adatbázis. Ezen adatokat bontottam „leave one out” keresztvalidációs logika mentén tanító- és teszthalmazokra, és teszteltem le az egyes betárolási stratégiákon. Teszteket végeztem kigyűjtési távolság és kigyűjtési idő minimalizálására egyaránt.

Alapvetően horizontális komissiózási példákat vizsgáltam, vagyis a komissiózás döntően egy vagy két szintről zajlott, így a komissiózási időszükséglet is kétdimenziós mozgásokra korlátozódik. A szinteken kívül vizsgáltam, hogyan hat az eredményekre az állványsorok sorszám/sorhossz arány változtatása.

A genetikus algoritmus tesztelése során számtalan érdekes jelenség mutatkozott, a populáció nagyság, a mutációs paraméterek, az elitizmus és a belterjesség tekintetében. Ezekből a tapasztalatokból egyre hatékonyabb algoritmusokat lehet létrehozni.

Kigyűjtési távolságok minimalizásánál az ABC elemzés volt a legsikeresebb, a genetikus algoritmus akkor tudott jobb eredményt elérni, amikor a tárhelyek száma többszörösen meghaladta az árucikkek számát. Azonban az összetettebb kigyűjtési idő minimalizálásánál már jelentősen hatékonyabb volt a genetikus algoritmusom.

A kutatások folytatására számos tervem van. A modell és az algoritmusok nagyobb, valós, céges megrendelési adathalmazokon való tesztelésével még jobban megismerhetem a megoldási lehetőségek tulajdonságát és hatékonyságát. Továbbá a jövőben vizsgálnám a neurális hálók alkalmazását a problémára.

1. Bevezető

1.1. A logisztika szerepe napjainkban

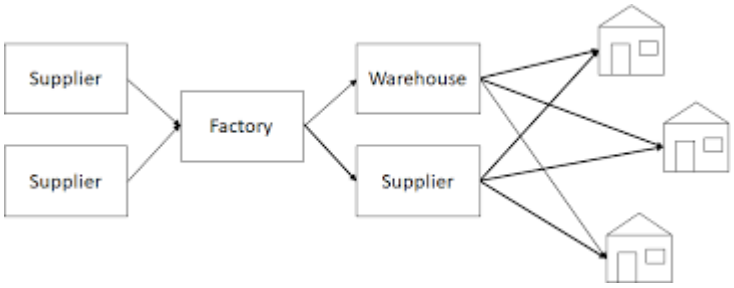
Napjainkban a világgazdaság és globalizálódó értértékalkotási láncok egyre nagyobb elvárásokat támasztanak a logisztikai rendszerekkel szemben, úgy mint:

- nagy szállítási gyakoriság jellemzően kisebb tételekben,
- alacsony logisztikai költségek,
- rövid átfutási idők,
- magas rendelkezésre állás stb.

Ezek az elvárások megkövetelik a logisztikai rendszerek fejlesztését géptechnológiai, szervezéstechnikai és informatikai téren egyaránt. Számos példát találhatunk arra, hogy azok a vállalatok, amelyek ezeket a kihívásokat hatékonyan oldják meg, jelentős előnyt szereznek a versenytársaikkal szemben.

1.2. Az ellátási láncokról és a raktári folyamatokról

Modern világunk vérkeringését a logisztika által működtetett ellátási láncok adják. Az ellátási lánc (supply chain) „egy olyan hurok, mely a vevőnél kezdődik, és nála fejeződik be” [1]. A vevőnél kezdődik, hiszen ő határozza meg, hogy milyen termékekre/szolgáltatásokra van szükség. Ez az információ az adott termékek gyártóin majd azok beszállítóin keresztül lépcsőzetesen egyre „távolabb” kerül a primer vevőtől egészen a nyersanyagok szintjéig. Onnan aztán anyagáramlás indul el, mely folyamat során az anyagok számos cégen, műveleten keresztülmennek, mígnem végeredményként a vevő megkapja az általa kívánt terméket [1].



1. ábra: Ellátási lánc, forrás:https://s3.amazonaws.com/assets.datacamp.com/production/course_8835/slides/chapter1.pdf

Az ellátási láncban fontos szerepe van a raktáraknak. A raktárak két alapvető feladata, hogy áthidalják az igény és az előállítás közti időbeli és térbeli különbségeket, Hiszen nem valószínű, hogy ha nekem szükségem van valamire, akkor az ott abban a pillanatban kerül legyártásra, amikor én azt szeretném, de jó eséllyel van a közelben egy vállalat, amely rendelkezik készleten azzal a termékkel, amit én szeretnék megkapni [2].

A lean filozófia és a JIT elterjedésével az ellátási lánc szereplői igyekeznek egyre kisebb készleteket tartani, aminek hatására jelentősen csökkentek a raktárkészletek. Ugyanakkor semmilyen iparágban sem lehet teljesen raktármentes működést kialakítani,

hiszen mindig lesznek ingadozások az egyes logisztikai rendszerekben, valamint mindig lesznek olyan kritikus fontosságú termékek, melyek rendelkezésre állása 100%-közeli kell, hogy legyen [2] [3].

A raktárkészletek méretének csökkenése nem jelenti a logisztikai rendszereken áramló áruk volumenének csökkenését. Épp ellenkezőleg, a rendszereken áramló áruk mennyisége nő, a készletek forgási sebessége pedig felgyorsul, így nincs szükség állandó nagy készletre. Ugyanakkor ez a felgyorsult forgási sebesség nagy plusz terhet jelent a raktári folyamatoknak, melyek így rákényszerülnek hatékonyságuk javítására.

A legfontosabb raktári folyamat a komissiózás, melynek költségei a raktári költségek felét is kitehetik. A költségek mellett kritikus a folyamat átfutásiidő-szükséglete, hiszen a megrendelés-teljesítések átfutási idejének jelentős hányadát is ez teszi ki. Ezen okok miatt kiemelten szükséges a komissiózási folyamatok hatékonyságának javítása. Dolgozatom fő célja, hogy a komissiózó rendszerek munkáját segítsem, hatékonyságát növeljem egy azokat támogató új, hatékonyabb megközelítéssel [2].

1.3. Személyes motiváció

Abban, hogy a téma kutatásába kezdjek, két tényező játszott szerepet. Az egyik ok, hogy logisztikában dolgozó mérnökként az egyes folyamatok rendszerszintű optimalizálása a fő feladatom, és kiemelten érdekelnek az informatikában továbbá a modern algoritmusokban rejlő lehetőségek. A másik ok, hogy egy külső cég felvetette számomra a tárhelykijelölési problémát, mint kutatandó kérdéskört, és minél inkább beleástam magamat a témába, annál inkább megfogott.

2. Szakirodalmi áttekintés

2.1. Komissiózás, betárolás és tárolás

A raktári folyamatok kihatással vannak az egész ellátási láncra. Három fontos raktári folyamatot érdemes kiemelni: a betárolás, a tárolás és a komissiózás. Általánosságban a legfontosabb a komissiózás, ami az áruk kigyűjtésével foglalkozik. Nagyban befolyásolja a komissiózás teljesítményét az áruk raktárba történő betárolása, hiszen a betárolás határozza meg a későbbi komissiózás által érintett lokációkat. A tárolás módja, a tárolási rendszer pedig, meghatározza ezen folyamatokat, kezdve az anyagmozgatáshoz szükséges eszköztől, az alkalmazható szervezéstechnikai megoldásokon át, a fizikai paraméterekig, hogy mekkora távolságokkal és férőhelyekkel rendelkezik a rendszer [4] [5].

A raktári komissiózás kérdéskörével először T. Gudehus [6] foglalkozott a 70-es években. A komissiózási rendszerekben számos vizsgálandó kérdés felmerül, mint például:

- a komissiózó rendszer működési stratégiája,
- a komissiózási rendszer teljesítménye,
- a leadóhelyek pozíciója és a keresztfolyosók számának hatása a komissiózási időszükségletekre,
- a komissiózási útvonalak,
- a tárolási technológia,
- az anyagmozgatási technológia.

Magyar nyelven a témáról legátfogóbb mű Dr. Prezenski József Logisztika I-II című könyvei [7] [8]. Továbbá Dr. Bóna Krisztián *Anyagmozgatási és raktározási folyamatok* című jegyzete is bepillantást nyújt a témába[4].

2.2. Genetikus algoritmusok

Az optimalizálás a logisztika alapvető feladata. Több paraméter változtatása által kínált számos lehetőség közül ki kell választani a legmegfelelőbbet. A lehetséges megoldások alkotják az un., keresési teret (search space). A célállapotot pedig a célfüggvény felírásával és megoldásával próbáljuk meg meghatározni. [9]

Az 1950-es 60-as években matematikusok kezdték el kutatni az evolúciót úgy, mint egy nagyon sok paraméterrel rendelkező problémára való optimalizációs folyamatot. Az 1990-es évek elején pedig kialakult az evolúciós számítás tudományterülete, melynek három fő változata van: [10]

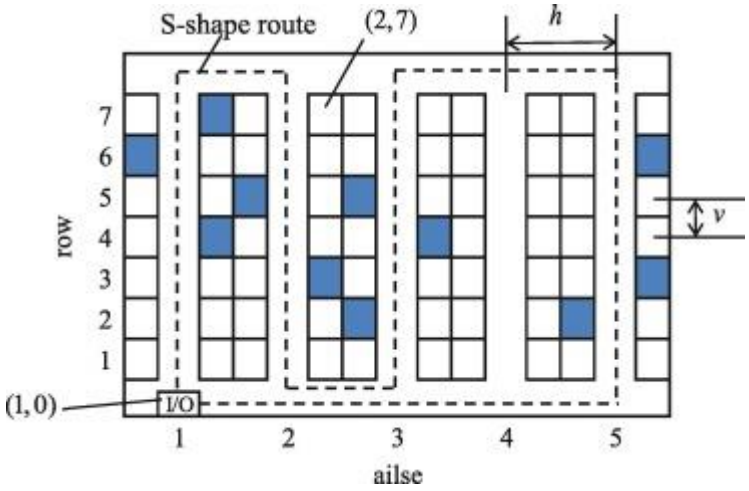
- Genetikus algoritmusok
- Evolúciós stratégiák
- Evolúciós programozás

Ezek közül, a genetikus algoritmusok terjedtek el a legszélesebb körben. A GA-k működése a darwini evolúciós elméleten és a genetika alapjain nyugszik. [11] A genetikus algoritmusok alapjait először John Holland publikálta 1975-ben, utána pedig sorra jelentek meg a tanulmányok a GA-k tulajdonságairól és felhasználási lehetőségeiről

[12]. Konkrét példákat a tárhelykijelölési problémával kapcsolatban a 2.3-mas fejezetben mutatok be. Ugyanakkor például a B. Molnár PhD dolgozatában a tárhelyfelkeresési problémára alkalmazza, de a mérnöki tudományok más területein is használják [12].

2.3.A SLAP probléma és megoldásai

A raktári tárhelyallokációs probléma (storage location assignment problem, ezentúl SLAP) alapja az a kérdés, hogy „hogyan rendezzem el a raktárban a termékeimet úgy, hogy komissiózásnál a lehető legközelebb legyenek egymáshoz a kigyűjtendő termékek. Tételezzük fel, hogy van x_1, x_2, ...x_k termékünk, illetve van egy háromdimenziós raktárunk tárhelyekkel $y_{i,j,z}$ ahol j a sorok száma i az oszlopok száma és z az oszlopok magassága. (Olyan eseteket vizsgálok majd, amelyekben a tárhelyek száma meghaladja az elhelyezendő termékek számát.) Továbbá vannak L listák, melyekben x_i termékek vannak $L=\{ x_a, x_b, ...x_r \}$. L tetszőleges számú terméket tartalmazhat x-ből. A SLAP fő kérdése az, hogy hogyan helyezzem el x_1, x_2, ...x_k termékeket $y_{i,j,z}$ állványrendszerben, hogy az L listák kigyűjtési ideje/távolsága (lehet más paraméter is, de ez a legjellemzőbb) kedvező legyen. A 2. ábra szemléltet egy komissiózást, mely során egy valamilyen meghatározott útvonalbejárási algoritmus alapján a komissiózó ágens felkeresi azokat a tárhelyeket, ahonnan a listán szereplő adott mennyiségű elemet ki kell gyűjtenie.



2. ábra: lista komissiózás raktárból S-shape útvonal alapján, forrás: New model of the storage location assignment problem considering demand correlation pattern [13]

Az általam talált legrégebi forrás, ami a tárhelykiosztási problémával foglalkozik, W. H. Hausmann: *Optimal storage assignment in automatic warehousing system* című 1976-ban közölt cikke, melyben részletesen definiálja a problémát és hatékony metódusokat ír le annak megoldására [14]. Az egyik ilyen megoldásnak az osztályalapú tárolást írja le, melynek egyik továbbfejlesztett fajtáját, a pareto elvre épülő ABC tárhelykiosztást én is modelleztem egy a továbbiakban bemutatott tanulmány által leírt módon [14] .

A SLAP-ot E. H. Frazelle 1989-ben és egy NP nehéz problémának minősítette a termékek száma és a raktár tárolási jellemzői miatt. [15] A probléma tovább bonyolódik, ha a termékek és a raktárhelyek száma megegyezik. Ebben az esetben kvadratikus allokációs problémaként definiálták [16].

Leginkább az osztályalapú, a dedikált és a véletlenszerű tárhelykiosztás elterjedt, [17] ugyanakkor számos modernebb megoldás is született a problémára az elmúlt évtizedben. Az *International Journal of Industrial Engineering Computations* című folyóiratban 2019-ben megjelent egy cikk, amely feldolgozza a téma addig megjelent szakirodalmát, a SLAP problémára alkotott modern algoritmusok tekintetében. [18] A cikk szerzői 2005 és 2017 között megjelent 71 tanulmány eredményét dolgozták fel. Több szempont szerint csoportosították a módszereket, mint például az optimalizálni kívánt paraméter (távolság, idő, infrastruktúraterhelés, emberi tényezők stb.), továbbá az alkalmazott módszer alapján (egzakt, heurisztikus, metaheurisztikus, szimulációalapú, stb…). [18] E tanulmány alapján vizsgáltam meg több módszert, melyekből kiindulva alkottam meg a saját algoritmusomat. Továbbá kiválasztottam pár algoritmust, melyekkel össze is tudtam hasonlítani a módszeremet.

Több kutatást is találtam, melyek használtak a SLAP megoldására genetikus algoritmusokat. Az Atlantis Press folyóiratban 2017-ben megjelent egy tanulmány „Optimization of Automated Warehouse Location Based on Genetic Algorithm" címmel. A tanulmányban sok fontos észrevételt tesz a genetikus algoritmusokkal és azok hatékonyságával kapcsolatban, ugyanakkor mivel az optimalizálandó cél a jobb tárhelykihasználtság és csak speciális automata raktárakat vizsgál a modelljeiben, ezért a kutatása jelentősen eltér az általam vizsgálandó általánosabb esetektől [19]. Egy 2019-es amerikai tanulmány hatékonyan alkalmazta a genetikus algoritmusokat a komissiózás támogatására speciális gördülő állványos automata rendszerekben. A tanulmány tapasztalatai sokat segítettek abban, hogy kialakítsak egy általánosabb (nem feltétlen automata) rendszert, melyben sorosállványos tárolási technológiát alkalmazok. (A sorosállványos tárolási technológia jóval elterjedtebb, mint a gördülőállványos.) [20]

Két, a sajátomtól meglehetősen eltérő módszer keltette fel leginkább érdeklődésemet. Ezeket a módszereket választottam ki arra, hogy összehasonlítsam őket a saját algoritmusommal. Az egyik egy 2011-ben megjelent tanulmány, amely részletesen leírja, hogyan alkalmazzák a pareto elvre épülő ABC-analízist osztályalapú tárhelykiosztásra, amivel jelentős eredményeket érnek el [21]. A másik tanulmány *Data mining based storage assignment heuristics for travel distance reduction* címmel jelent meg 2012-ben. Egy apriori algoritmusra épülő heurisztikus megközelítést mutat be, mellyel szintén a SLAP probléma megoldására nyújt egy módszert [22]. Ez utóbbit azért választottam ki, mert merőben más oldalról próbálja megközelíteni a problémát, így új szemszögből is megvizsgálhatom azt.

3. Technológiák és módszerek

Ebben a fejezetben részletesen kifejtem a raktári komissiózás és SLAP témakörök problémáit, az egyes folyamatokat, az azokra alkalmazható megoldásokat, algoritmusokat, továbbá részletezem saját ötleteimet, rámutatva az azok nyújtotta előnyökre.

A raktári folyamatok szempontjából három részfolyamatot érintünk: betárolás, tárolás és komissiózás.

3.1. A betárolás

A betárolás feladata a beérkezett áruk elhelyezése a raktárban. Érdemes a betárolást részfolyamatai mentén megvizsgálni, melyek az:

- áru tárhelyhez rendelése,
- áru tárhelyre mozgatása,
- áru elhelyezése és adminisztrálása a tárhelyen.

Az első pont az áru tárhelyhez rendelése az, ahol a tárhelykiosztási probléma felmerül, ugyanakkor ennek következménye nem jelenik meg, hiszen a betárolást döntően nem befolyásolja (betárolásnál nagyobb volumeneket mozgatnak, és szabadon lehet összefogni a betárolási tételeket, így a tételre vetített hatékonysága sokszorosa a komissiózásénak). A tárhely kiválasztása sok esetben még mindig a betáras kolléga egyéni döntése, azonban a modern vállalatirányítási rendszerek már kijelölik a kolléga számára a tárhelyet. Ez a SLAP megoldó algoritmusok számára kulcskérdés, hiszen ott mindenképp a rendszernek kell előírnia, hogy az áru milyen tárhelyre kerüljön. [2]

Az áru tárhelyre történő mozgatása a tárolási technológiától függ. Raklapos, állványos technológiánál a targoncák a legelterjedtebbek, de a kézikocsitól az automata felrakógépig számos megoldás létezik. [2]

Az adminisztrálás a vállalatirányítási rendszertől és az alkalmazott azonosítási technikától függ. Jellemzően egy vagy kétdimenziós vonalkódok beolvasásával, vagy RFID technológiával történik. [2]

3.2. A komissiózás és támogató technológiái

A komissiózás folyamata során, adott árucikkek közül előre megadott árukigyűjtési jegyzék szerint adott áruk kigyűjtése zajlik. A folyamat során a raktári tárhelyekről ki kell gyűjtenünk a meghatározott tételeket. A legnagyobb probléma, hogy ezek a tételek gyakran egymástól távoli lokáción helyezkednek el, és a lokációk közötti mozgás jelentős időveszteséget okoz a folyamatban. [4] [17]

A komissiózás alapvetően meghatározza egy raktár hatékonyságát, ugyanis a komissiózás folyamatának gyorsaságától függ, a megrendelések teljesítésének ideje. Ezért azt számos technológiával próbálják támogatni. Érdemese ezen technológiákat szemügyre venni, hogy megvizsgálhassuk, hogyan illeszthető majd bele a SLAP probléma hatékony megoldása ebbe a bonyolult rendszerbe.

Három csoportba soroltam a komissiózást támogató rendszereket. Egyrész léteznek fizikai rendszerek (pick by ligth, automata komissiózógép …), másrészt fontos szerepük van folyamatszervezési megoldásoknak (több lépcsős komissiózás, párhuzamos/soros komissiózás), harmadrészt pedig algoritmusok támogathatják a folyamatot, például a tárhelyfelkeresési probléma hatékony megoldása, ami az utazó ügynök probléma egyik esete. A teljesség igénye nélkül igyekszem bemutatni ezeket a technológiákat. [4] [12]

3.2.1. Komissiózást támogató fizikai eszközök

A komissiózás során számos eszköz segítheti a folyamatot. Érdemes részfolyamatokra bontani és úgy megvizsgálni. A komissiózás részfolyamatai:

- Információt kap a komissiózó ágens, hogy honnan és mennyi terméket kell kigyűjtenie.
- Mozog az adott lokációra.
- Kiveszi a tárhelyről a megfelelő mennyiségű terméket (közben ellenőrzi, hogy biztos az-e a termék és biztos jó-e a mennyiség).
- Adminisztrálja, hogy ki van véve az adott tétel.

Az információ-adás/vétel leginkább digitális eszközök segítségével wifin keresztül valósul meg. Gyakori a tabletek alkalmazása, ugyanakkor ez sok esetben körülményes, hiszen a komissiózás közben plusz mozdulatokat igényel. Hatékony és egyre inkább elterjedő megoldás az un. voice picking, mikor a komissiózó kolléga hang alapú utasítást kap és ad vissza. Ennek előnye, hogy ilyenkor a komissiózó munkatárs mindkét keze szabadon marad. Az un. pick by light rendszereknél az állványba LED-ek vannak beépítve, és egy fényjelzés, vagy az állványon lévő kijelző mutatja, hogy honnan és hány árut kell kigyűjteni. Ennek a rendszernek a kiépítése igencsak költséges, és a rendszer sok szempontból rugalmatlan. [4] [17]

A mozgás tekintetében számos megoldást használnak, amit egyrészt a komissiózás helyén lévő tárolási technológia másrészt az áruelőkészítés módja határoz meg. Dinamikus áruelőkészítés esetén az árut a rendszer odamozgatja a komissiózó kollégához (pl.: görgős pálya, vagy körforgó páternoszter/shuttle rendszerek segítségével), míg statikus esetben a komissiózó kolléga megy az áruért, gyalog, komissió targoncával, homlokvillás targoncával stb. [2] [4] [17]

A mennyiségellenőrzés kritikusan fontos feladat, hiszen, ha a szükségesnél több vagy kevesebb áru lesz kigyűjtve, az számos problémát okoz a továbbiakban. Ugyanakkor a nem megfelelő ellenőrző részfolyamat lassíthatja is a komissiózást, ami hátrányos a folyamat egésze szempontjából. Az ellenőrzést leggyakrabban vagy technológiai támogatás nélkül végzi a komissiózó kolléga (ami gyakori hibához vezet), vagy vonalkódolvasós technikát alkalmaznak, ez azonban jelentősen lassítja a komissiózást. Modern rendszereknél már RFID-s technológiával vagy tömegméréssel támogatják a folyamatot, ezek a technológiák azonban drágák. [4]

3.2.2. Komissiózást támogató folyamatszervezési megoldások

Az egymástól messze lévő tételek kiszedésének problémájára kézenfekvő módszer lehet, a több lépésben történő komissiózás. Ilyenkor több megrendelést összefognak, majd először kigyűjtik az egymáshoz közeli termékeket, utána pedig ezeket szétválogatják külön-külön az egyes listáknak megfelelően. Ez a módszer nagyobb komissiózási teljesítményt jelent, de a két lépés miatt, nagyobb és bonyolultabb a szervezési feladat, továbbá az átfutási időket is növeli. [4]

A folyamat bővíthető a több megrendelés összefogásával, ami még tovább növelheti a komissiózási teljesítményt, de az előbbiekben leírt hátrányok is fokozódnak. [4]

A komissiózás hatékonyságát befolyásolja, hogy honnan történik az áruk kigyűjtése. illetve, hogy a komissiózás, a tárolótér és a betárolás helye elkülönül vagy megegyezik. A komissiózás történhet a tárolótérből, az áruelőkészítő téren és komissiózó raktárban. Ezt minden esetben egyedileg kell meghatározni, a forgalom, a hely és az elvárások függvényében. [4]

3.2.3. Komissiózást támogató algoritmusok

A komissiózást érdemes matematikai problémákra bontani. A komissiózási folyamat hatékonyságát döntően befolyásolja, hogy a komissiózó ágens milyen útvonalon végzi a munkát. Erre egyrészt léteznek heurisztikák (Midpoint, Lagest gap) másrészt algoritmusok, melyek próbálják optimalizálni az útvonalat, például a [12] tanulmány módszerével, amely egy a BME-n készült doktori disszertáció és genetikus algoritmusok használatával oldja meg az utazó ügynök problémához hasonló, táhelyfelkeresési problémát. [23]

Bár a fent említett heurisztikák könnyen automatizálhatók továbbá az útvonaloptimalizáló algoritmusok is régóta léteznek, melyekkel még hatékonyabban lehet kezelni ezt az „utazó ügynök” problémához nagyon hasonló problémát, ennek ellenére a gyakorlatban sok esetben ad-hoc, vagy csak nagyon primitív heurisztikákat alkalmaznak (S-shape). Arra a kérdésre, hogy mi történik, ha a kigyűjtendő tételek nagyon messze helyezkednek el, a modern kigyűjtő algoritmusok sem kínálnak megoldást, és a folyamat továbbra is sok veszteséget fog tartalmazni.

A komissiózást támogató eszközöket végig gondolva látható, hogy nincs olyan eszköz, amely tökéletesen megoldaná a tárhelykiosztási problémát.

3.3. A SLAP probléma és a tárolási rendszer kialakítása

A tárhelykiosztási probléma a termékek egy raktárhelyiségbe történő optimális elhelyezésével foglalkozik. Az optimalizálás általában vagy raktárterület kihasználtságának javítására, vagy a komissiózási folyamat ciklusidő/erőforrásigény csökkentése érdekében történik [18]. A problémát leginkább befolyásoló tényezők:

- A raktárterület kialakítása, mérete
- A tárolási technológiák
- A termékek fizikai jellemzői

- A ki-be szállítási igények

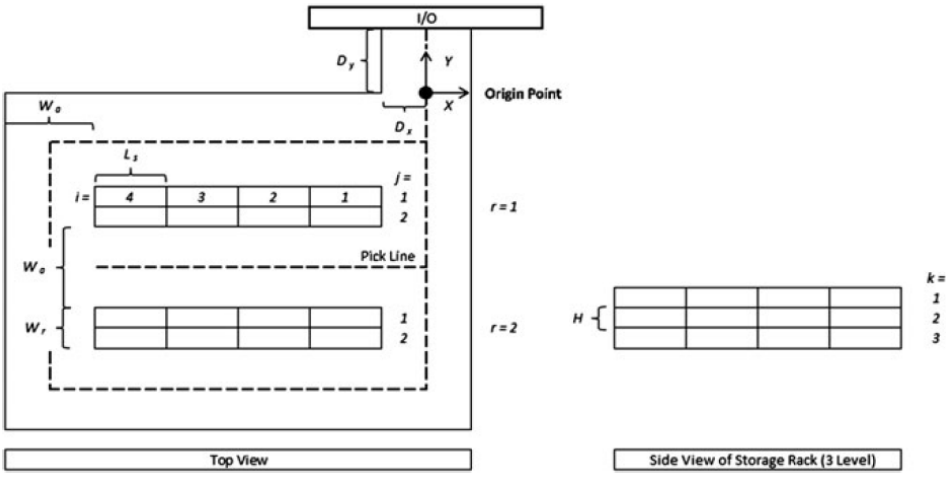
Az általam választott rendszer egy raklapos, sorosállványos tárolási technológiát fog leképezni. A ki-be szállítási igényeket pedig a későbbiekben bemutatandó tranzakciós listák határozzák meg [2]. Komissiózási időoptimalizálásnál, ún. horizontális komissiózást mutatok be, amikor a komissiózás külön komissiózó zónákból/állványnál történik, ezekben az esetekben a tárolási magasság alacsony lesz [4].

3.4. A raktármodell

A probléma és az algoritmusok vizsgálatához modelleznem kellett egy raktárt, a hozzá tartozó termékeket és megrendeléseket.

3.4.1. A paraméterezhető raktár-layout

A raktár elrendezését és kinézetét C. Kasemset és P. Meesuk 2014-es Storage Location Assignment Considering Three-Axis Traveling Distance: A Mathematical Model, című cikkében megjelent struktúra alapján építettem fel [24]. Azért döntöttem emellett a modell mellett, mert a szakirodalomban számos cikk hivatkozik rá, vagy használ ugyanilyen jellegű raktárrendszert.



3. ábra : Raktárlayout, forrás: [24]

Jelölésmagyarázat:

- i = A tárolóblokk pozíciója az egyes tárolóállványokban (sorhossz) (1, 2, 3, ..., i).
- j = A tárolóállvány száma (1, 2, 3, ..., j)
- k = A tárolóblokk szintje (1, 2, 3, ..., k)
- r = A tárolósor száma (1, 2, 3, ..., r)
- Dx = az X tengely mentén az I/O pont és a kiindulási pont közötti távolság.
- Dy = Az Y tengely mentén az I/O pont és az origópont közötti távolság.
- Wa = A folyosó szélessége
- Wr = A tárolósor szélessége (egyenlő a tárolóblokk szélességének kétszeresével)

- Ls = A tárolóblokk hossza
- H = A tárolóblokk magassága

A felsorolt paraméterek tetszőlegesen állíthatók. A tesztelésnél részletezem, hogy milyen paramétereket használtam.

3.4.2. A modell kikötései

A modellben egyszerűsítéseket alkalmazok, melyek az algoritmusok hatékonyságát nem, vagy csak elhanyagolható mértékben befolyásolják. A modellem feltételezi, hogy:

- Minden áru egy tárhelyen van tárolva és egy tárhelyen csak egy áru van tárolva.
- Minden tárhelyen adott árucikkből mindig rendelkezésre áll a szükséges készlet.
- A komissiózó eszköz minden esetben be tudja fogadni a kigyűjtési lista teljes tartalmát.
- A komissiózó eszköz kanyarban lelassít 0-ra megfordul és 0-ról kezd el gyorsítani.
- A raktárban minden komissiózó eszköz ugyanolyan gyorsulási és sebességparaméterekkel rendelkezik.
- Függőleges irányban ugyan van távolságkülönbség az egyes tárhelyek között, de a kigyűjtési időnél nincs időkülönbség.
- A tárhelyen kigyűjtés közben eltöltött időt nullának veszem.

3.4.3. A megrendelésadatok

A megrendelés-adatokat az R-software Groceries adatbázisa szolgáltatja, melyben 169 termék közel 9835 megrendelési listája található. A lista elérhető az R-softwareből, de a hivatkozott honlapon is megtekinthető. [25]

3.5. Megoldó algoritmusok

Ahogy a 2-es fejezetben a szakirodalom bemutatásakor is látható, a problémát számos módon próbálják megoldani. Az iparban Magyarországon az alábbi három módszer a legelterjedtebb:

- osztály alapú tárhelykiosztás,
- a dedikált tárhelykiosztás és
- a véletlenszerű tárhelykiosztás.

Ezek közül az osztályalapú tárhelykiosztásnak egy modern – a 2.3-as fejezetben már bemutatott– 2011-es tanulmányra alapuló pareto elv szerinti un. ABC tárhelykiosztási logikát fogok implementálni, [21] valamint a véletlen kiosztást fogom vizsgálni, mivel az ABC tárhelykiosztás bizonyítottan hatékony módszer, a véletlen tárhelykiosztás pedig jó kontrollmérésnek. A kutatásaim során olvastam több más algoritmusról is, amelyek a SLAP-ra nyújtottak megoldást. Ezek közül David Ming–Huang Chiang, Chia-Ping Lin és Mu-Chen Chen 2012-es apriori datamining algoritmusra épülő heurisztikáján alapuló

metódusát implementálom [22], mivel így egy a többitől alapjaiban eltérő algoritmust és gondolatmenetet is tudok tesztelni.

3.5.1. A véletlen és az ABC tárhelykiosztás

A véletlen tárhelykiosztás során a betároláskor a beérkezett termék (tfh. még nincs a raktárban olyan termékből) véletlenszerűen kap egy tárhelyet, ahova be kell tárolni. A rendszer előnye, hogy ezáltal a termékek egyenletesen lesznek elosztva a raktárban.

Az ABC tárhelykiosztási stratégiánál a termékeket szortimentanalitikai eszközökkel ABC csoportokra osztjuk a Pareto-elv alapján. A Pareto-elv azt mondja ki, hogy általában az áruk 20 %-ka adja a forgalom 80%-át. Fontos, hogy ennél a problémánál felkeresés alapú ABC analízist kell végezni, vagyis azt kell vizsgálnunk, hogy az egyes termékek hány kigyűjtési listán szerepelnek, függetlenül a kigyűjtési volumenektől és az áruk értékétől. A felkeresések 80%át adó termékek lesznek A-kategóriájúak, a 15%-át adó termékek B-k, és az 5%-ot adók pedig C-s termékek. [21] Az algoritmust pszeudokódja:

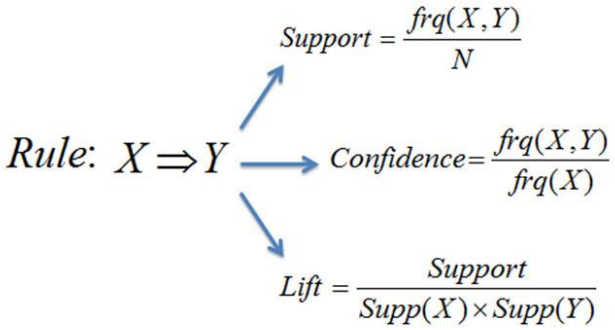
ABC tárhelykiosztó algoritmus

```
Input: tárhelyek_A, tárhelyek_B, tárhelyek_A, termékek
<i> = <0>

ciklus <0-tól termékek számáig>
        ha < termékek[i].kategóriája=A >
            <r> = <random 0-tól tárhelyek_A hosszáig>
                ha<tárhelyek_A[r]= üres>
                   termékek[i]betárolás tárhelyek_A[r] tárhelyre
                elágazás vége
            elágazás vége
        ha < termékek[i].kategóriája=B >
        …
        ha < termékek[i].kategóriája=C >
        …

ciklus vége
eljárás vége

Output: összerendelt termékek-tárhelyek
```

Az áruk mellett a tárhelyeknek is kategóriákat kell adni. A legkönnyebben elérhető tárhelyek lesznek A-s kategóriájúak, a kevésbé a B-sek és a legnehezebben elérhetőek a C-sek. Az áruk a betárolásnál csak a saját kategóriájuknak megfelelő tárhelyre kerülhetnek, azon belül viszont véletlen, hogy pontosan melyikre.

3.5.2. Az apriori datamining technikán alapuló heurisztika

Ennél a technikánál először elvégzem a tanítóhalmazon az apriori algoritmust, melynek segítségével minden termékpárhoz (amely definiálható mértékű) meghatározom az algoritmus un. „confidence” értéket. Ez az érték bemutatja, hogy milyen szoros a kapcsolat a kéttermék között, „mennyire valószínű, hogy szükségem van B-termékre is, ha A-termékre szükségem van”. Ezt az apriori algoritmust minősített internetes kódok és az R software beépített algoritmusai segítségével hoztam létre R-ben. [26]



4. ábra: apriori algoritmus eredményei, forrás: https://www.kaggle.com/code/changjunlee83/apriori-rule-with-grocery-data-for-class [21]

A 4. ábra mutatja az apriori algoritmus által számolt legfontosabb értékeket.

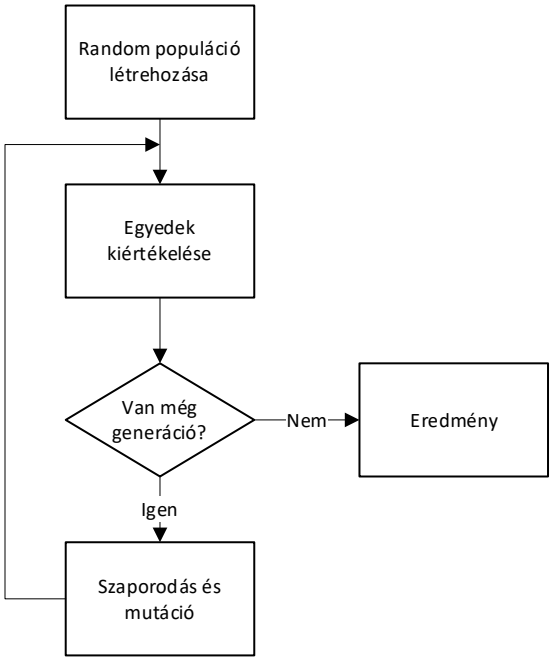
- A „support” egy termék alapértelmezett népszerűségére utal, úgy számítható ki, hogy az adott terméket tartalmazó tranzakciók számát elosztjuk a tranzakciók teljes számával.
- A „confidence” annak valószínűségére utal, hogy egy B terméket is kigyűjtök, ha A terméket is. Ez úgy számítható ki, hogy azon tranzakciók számát, ahol A-t és B-t is kigyűjtötték, elosztjuk azon tranzakciók teljes számával, ahol A-t kigyűjtötték.
- A Lift alapvetően azt mondja, hogy a A és a B együttes megvásárlásának valószínűsége x-szer nagyobb, mint annak, hogy csak a B-t vásároljuk meg.

Miután meghatároztam a termékek „confidence” értékeit, szükséges, hogy a raktárban már legyenek áruk, úgyhogy betárolok pár árucikket. (Érdemes ABC logika alapján betárolni, mert a véletlen betárolásnál gondot okozhat, hogy nagy forgalmú, fontos termékek, amelyek számos más termékkel szoros kapcsolatban vannak előnytelen helyre kerülnek, és mivel szoros kapcsolatban vannak más adott esetben szintén fontos termékkel, azokat is „magukhoz” vonzzák). Utána pedig, amikor egy újabb terméket akarok betárolni, meghatározom, hogy az adott sorban már bent lévő áruk, ehhez a most betárolandó áruhoz kapcsolódva milyen „confidence” értékekkel rendelkeznek. Ezt összegzem soronként, és ha van szabad tárhely abban a sorban, ahol ezen értékek a legmagasabbak, akkor oda teszem a termékemet. Amennyiben nincs szabad tárhely, egy szomszédos sorba igyekszem elhelyezni az árut. Előfordul olyan eset, amikor nincs definiálva „confidence” érték a betárolandó termék és a sorokban lévő termékek között (mert például kimutathatatlanul gyenge a kapcsolat). Ilyen esetekben véletlenül helyezem el a terméket a raktárban. [22]

3.6. Az genetikus tárhelykiosztó algoritmus működési elve

A genetikus algoritmus, amelyet írtam, alapvetően nem különbözik az általános genetikus algoritmusoktól, csupán a problémára szabtam. Az algoritmusnál egy egyed paraméterezhetően sok „kromoszómát tartalmaz”. Minden kromoszóma egy adott

termékhez egy tárhelyet rendel hozzá. Így a kromoszómák száma megegyezik az elhelyezni kívánt árucikkek számával; esetünkben 169-lesz. Az algoritmusnál paraméterezhető az egyedszám, a generációk mértéke, a mutáció mértéke és az algoritmus elitizmusa.



5. ábra: A genetikus algoritmus folyamatábrája

A genetikus algoritmus lefutásának folyamatát az 5. ábra mutatja be. Az első generáció előtt véletlenszerűen létrejön az összes egyed. Egy generáció lefolyása három részből áll: szelekció, szaporodás, mutáció. A szelekció során, minden egyed „minőségét” kiértékelem a fitneszfüggvény segítségével. Kétféle fitnesz-függvénnyel dolgoztam, mivel egyes tesztesetekben a kigyűjtési távolságra optimalizáltam más esetekben a kigyűjtési időre. A fitneszfüggvény mindkét esetben a múltbeli, már ismert megrendelések alapján számolt. A fitneszfüggvény kiértékelése során megrendelések halmazából választott tanítóhalmaz megrendeléseit gyűjtötte ki minden egyednél a program, és meghatározta az azokhoz szükséges utat/időt. Miután a fitneszfüggvény kiértékelte az egyedeket (meghatározta a kigyűjtési utakat/időlet), azok egy része „elpusztul” (hogy mekkora részük azt az elitizmus paramétere határozza meg, ami a modellem esetében 50%ra volt beállítva). Ezzel lezárul a szelekció.

A szaporodás fázisában a populáció „életképesebb” feléből újra létrejön a populáció elpusztult egyedei helyére egy-egy egyed. Miden új egyed miden kromoszómáját véletlenszerűen egy, a populáció életben maradt egyedétől kapja. Miután létrejöttek az új egyedek, a mutáció fázisába lépünk, mely során az új egyedek egy kis százalékánál véletlenszerűen megcserélődnek kromoszómák (hogy melyik termékhez melyik tárhely tartozzon). A mutáció lényege, hogy elkerülje a lokális minimumokban való beragadást. A mutáció után egy újabb generáció következik, mely során újból szelektálják az egyedeket és így fut tovább annyi generáción át amennyit előre megadtunk. A végeredmény az utolsó generáció legjobb egyede lesz. Alább látható egy generáció egyszerűsített pszeudokódja:

GA tárhelykiosztó algoritmus egy generációjának egyszerűsített pszeudokódja

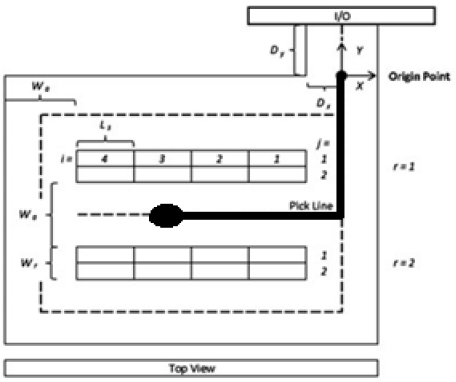
```
Input: egyedek

szelekció
      <i> = <0>
      ciklus <0-tól egyedek számáig>
            egyedek[i] fitneszértéke=kigyűjtési erőforrás
            i++
      ciklus vége
      egyedek növekvő sorba rendezése
eljárás vége;

szaporodás
      <i> = <egyedek száma/elitizmus >
      <r> = random 0-egyedekszáma/elitizmus között
      ciklus <egyedek száma/elitizmus mértéke-től egyedek száma-ig>
            <j> = <0>
            ciklus <kromoszómák száma>
                  egyedek[i] kromoszóma[j]= egyedek[r].kromoszóma[j]
                  j++
            ciklus vége
      ciklus vége
eljárás vége;

mutáció
      ciklus <0-tól mutációs paraméterieg>
            egyedek[rand] kromoszóma [rand2]=egyedek[rand3]
            kromoszóma [rand4]
            i++
      ciklus vége
eljárás vége;
Output: összerendelt termékek-tárhelyek
```

3.7. Az algoritmusok kiértékelése

Az algoritmusok kiértékelésénél, lefuttatjuk ugyanazon paraméterű raktáraknál az algoritmusok által betárolt esetekre ugyanazon teszthalmaz megrendelési listáit, vizsgálva, hogy ez mennyi kigyűjtési távolságot/kigyűjtési időt jelent a rendszer számára. Az az algoritmus lesz a hatékonyabb, melynek kevesebb útra/időre van szüksége a feladat elvégzéséhez. A kigyűjtési távolságok és idők számítási módját a 3.7.1-es és a 3.7.2-es fejezetekben mutatom be.

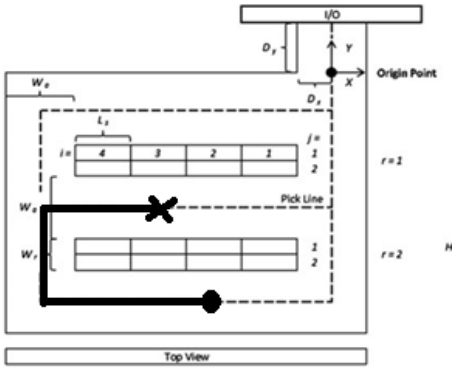
3.7.1. Távolságszámítás

Távolságszámításnál két esetet különböztettem meg, nullpont-tárhely távolságot, és tárhely-tárhely távolságot. A 6. ábra mutatja a nullpont-tárhely esetét, amikor a tárhely és az I/O pont között történik a mozgás. Ilyenkor a távolságszámítás áll egy x irányú egy y irányú és egy z irányú távolságkomponensből.



6. ábra: Nullpont-tárhely távolság forrás: [24]

A tárhely-tárhely távolság két z irányú, két x irányú és egy y irányú távolságkomponensből áll, ahogy azt a 7. ábra is mutatja. Ilyen esetekben mindig a közelebbi folyosó fele indul el a komissiózó ágens, de a modell ennél részletesebben nem foglalkozik a kigyűjtési útvonal optimalizálásával.



7. ábra: Tárhely-tárhely távolság, forrás: [24]

3.7.2. Kigyűjtési idő számítása

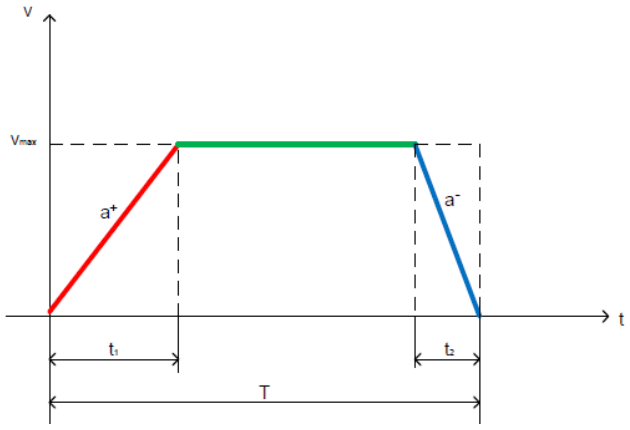
Az időalapú optimalizálás során, nem az egyes listák kigyűjtése folyamán bejárt távolságot, hanem az ahhoz szükséges időt veszem alapul. Ez az idő lesz az egyes lefutások utáni eredmény, továbbá a GA fitnesz függvénye is az időértékek minimalizálására fog törekedni.

A kigyűjtési idő optimalizálásának paraméterei: az x és y távolságadatok az egyes pontok (nullpont-tárhely, tárhely-tárhely) között, továbbá a komissiózó gép gyorsulása és a megengedett maximum sebesség. Fontos paraméter az is, hogy hány mozgásszakasz van, hányszor kell a gépnek kanyarodnia, ugyanis olyankor le kell lassítania nullára. A modellben a gyorsulási és a lassulási tényezőt ugyanannyinak vettem.

A számítás során megkülönböztettem gyorsuló, lassuló és egyenletes sebességgel haladó szakaszokat. A számítást főként az teszi bonyolulttá, hogy bizonytalan tényező az, hogy a mozgás során a gép eléri-e a maximum sebességét. A 8. ábra**Hiba! A hivatkozási forrás nem található.** mutatja ezt az esetet. Látható, hogy egy gyorsuló és egy lassuló szakasz, illetve egy a kettő közötti egyenletes mozgás alkotja ezt az esetet.

Sebesség-idő diagramok esetén a görbe alatti terület adja meg a megtett út hosszát. Ebben az esetben visszafelé számolunk az (1)-es képlet alapján.

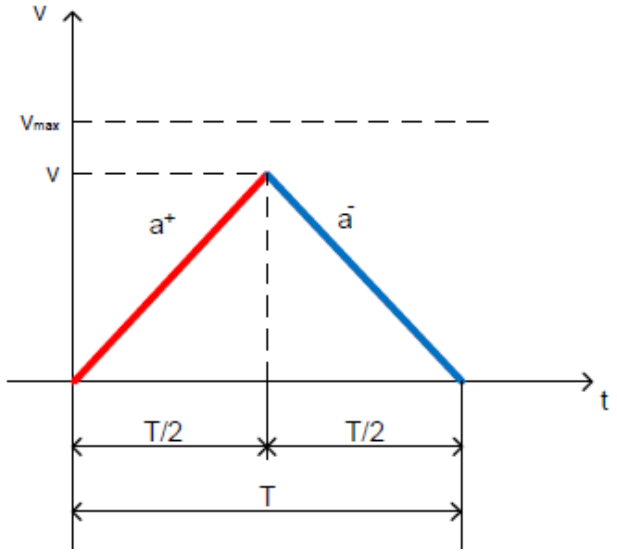
$$(1) \quad T = \frac{s}{v_{max}} + \frac{v_{max}}{a}$$



8. ábra: haladás maximális sebesség elérésével. Forrás: [27]

A9. ábra mutatja azt az esetet, amikor a komissiózó gép nem éri el, vagy csak éppen egy pillanatra éri el a maximális sebességet. Ilyen esetben a mozgás csak egy gyorsul és egy lassúló szakaszból áll. Ilyenkor a (2)-es képlet segítségével számoljuk a szükséges időt.

$$(2) \quad T = \sqrt{\frac{s}{a}}$$



9. ábra: Gyorsuló és lassuló szakasz. Forrás: [27]

A program miután kiszámolja az egyes szakaszok időszükségletét, összegzi annak adatait. Fontos kiemelni, hogy a modellem függőleges irányban nem számol mozgásokat. Látható ugyanakkor, hogy e számítások jóval komplexebbek, mint a távolságoptimalizálás esetében.

4. Tesztek és eredmények

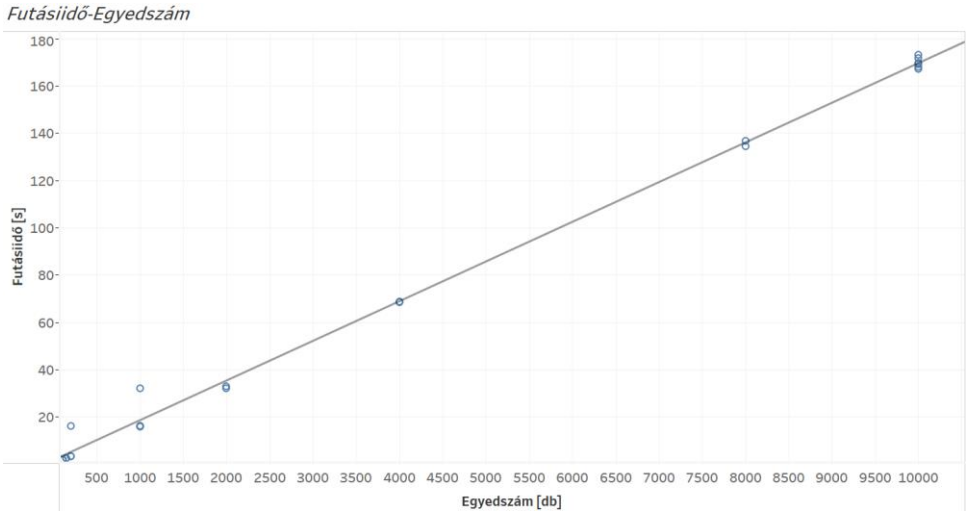
A 3.3-mas és a 3.4es fejezetekben bemutatott algoritmusokat több paraméter függvényében tesztelem és értékelem ki. Ezen tesztek folyamatát és eredményeit mutatom be részletesen ebben a fejezetben.

4.1. A modell implementálása Java környezetben

A modellhez létrehoztam egy raktárosztályt, mely rendelkezik paraméterezhető tárhelyosztályokkal. A raktármodell kap áruosztályú áruegyedeket, melyeket tetszőlegesen be lehet tárolni és ki lehet gyűjteni a raktárból. A kigyűjtésekhez megrendeléseket kap a modell, melyeken lefutva kiszámítja az azok kigyűjtéséhez szükséges távolságigényeket. A modell rendelkezik egy „komissiózó-gép”-osztállyal, mely a gyorsulás és a sebesség paraméterei alapján az egyes listáknál nemcsak a távolság, hanem a kigyűjtési időszükséglet is kiszámítható. A modellnek vannak fizikai méret paraméterei, úgy, mint folyosószélesség, tárhelyszélesség, ahogy ez a 3.4.1-es fejezetben be van mutatva. Ezeket az egyes tesztek során nem változtattam, csupán azt, hogy hány sor oszlop stb van. Az alábbi beállításokat alkalmaztam: Wa=20, Wr=10, Dx=0, Dy=0, L=3, H=1 ahol a paraméterek dimenziója egységnyi távolság. Továbbá a komissiózó gépnek van egy a=1 távolságegység/időegység^2 gyorsulási paramétere és egy v_{max}=10 távolságegység/időegység sebességparamétere.

4.2. Lefutási idők

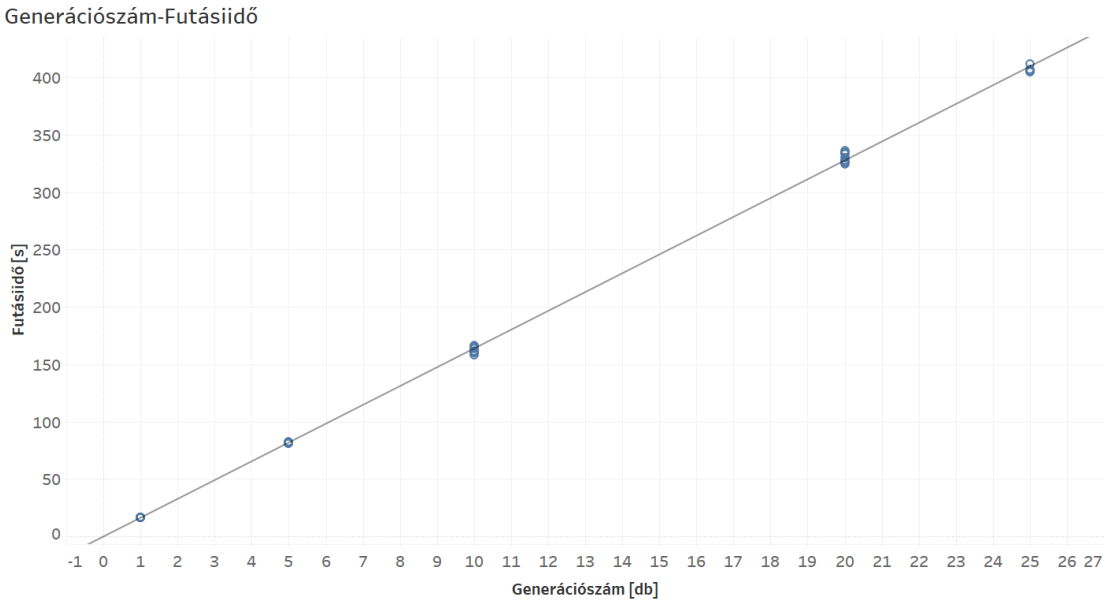
A SLAP egy NP nehéz probléma, ha minden variációt meg akarnék vizsgálni 200 tárhely és 169 termék esetében akkor ~2,246*10^{36}-számú esetet kellene kipróbálni. A véletlen, az ABC és a heurisztika futásideje minden esetben tizedmásodperc–másodperc környékén mozogott ezért azon algoritmusok futási idejét nem vizsgáltam részletesebben. A genetikus algoritmus esetében más nagyságrendű a számítási időszükséglet. A GA esetében azt vizsgáltam, hogy a két legfontosabb változó paraméter, a generációk száma, valamint az egyedek száma, hogyan befolyásolja a futásiidőket.



10. ábra: Futásidő-egyedszám diagram

Az 10. ábra által ábrázolt diagrammon láthatjuk, hogy az egyedszámok tekintetében néhány kisebb kilógó értéket leszámítva egyértelmű a lineáris összefüggés az egyedszám és a futási idő között, melynek egyenlete:

(3) $y = 0{,}0168x + 1{,}0982.$



11. ábra: Generációszám-futásiidő diagram

Az 11. ábra diagramja alapján egyértelműen látható, hogy a generációk számától lineárisan függ a program futásiidejének hossza. Ennek a lineárisnak az egyenlete:

(4) $y = 16{,}391x - 0{,}1716$

Összehasonlítva a (2)-es és az (1)-es egyenletet látható, hogy a második meredeksége nagyságrendekkel magasabb, vagyis a generációszám egységnyi növelése sokkal jobban növeli a futásiidőt mint a populáció egységnyi növelése.

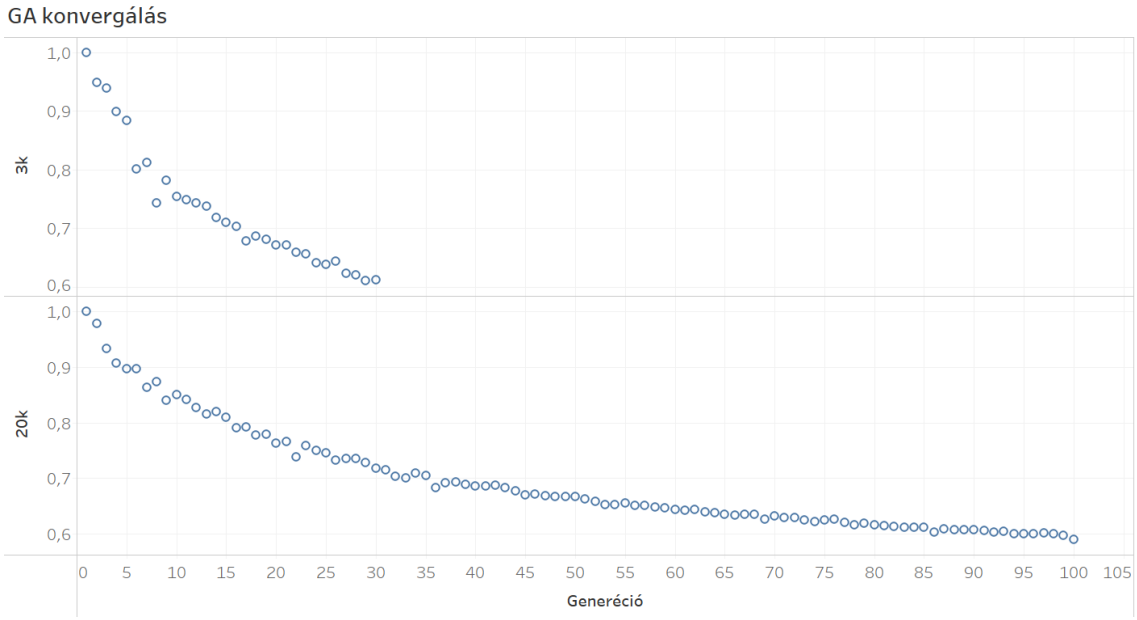
4.3. A genetikus algoritmus beállítása

A 4.2-es fejezetben olvasottak alapján a GA futásideje jelentősen megnő, ha a generációszámot és az egyedszámot magas értékre állítom. Érdemes vizsgálni, hogy hogyan konvergál az algoritmus az optimum felé, továbbá, hogy van-e olyan pont, amikor a teszt halmaz számára már nem javul az eredmény, hiába tanítom tovább a tanító halazon.

4.3.1. Konvergálás

Két tesztesetben vizsgáltam, hogy milyen ütemben konvergál az algoritmus. Egyrész vizsgáltam 3000 egyed tekintetben 30 generáción át. Ebben az esetben a futásii idő ~400 s körül mozgott, ami tesztelés során még kivárható idő sok teszteset elvégzése esetén is. Továbbá vizsgáltam 20000 egyed létrehozása mellett 100 generáción keresztül. Ilyenkor

a futási idő 2,7 óra felett volt, ami miatt ilyen esetekkel nehézkes és nagyon időigényes a tesztelési folyamat.



12. ábra: GA konvergálása a generációk függvényében

Az 12. ábra mutatja, hogyan alakul a generációk folyamán a legjobb egyed fitneszértéke az első legjobb fitneszértékű egyedhez képest, abban az esetben, ha a kezdeti populáció 3000 (felül) illetve abban az esetben ha a kezdeti populáció 20000 (alul). Látható, hogy logaritmikusan konvergál az eloszlás egyre alacsonyabb értéket felvéve.

Megfigyelhető továbbá, hogy a nagyobb egyedszámú populáció lassabban konvergál, de képes eljutni fejlettebb egyedekhez, mint a kisebb populáció. Kézenfekvőnek tűnhet, olyan beállítást alkalmazni, amelyben a kisebb egyedszám nagyobb generációszám mellett fut. Ennek azonban van egy korlátja, ugyanis egy bizonyos generációszám után „belterjes” lesz a halmaz, túlságosan hasonlítanak egymásra az egyedek, ami ahhoz vezet, hogy ugyanarra a tárhelyre több terméket is be akarnak tárolni. Ezt a jelenséget a 4.3.3 fejezetben fejtem ki részletesebben.

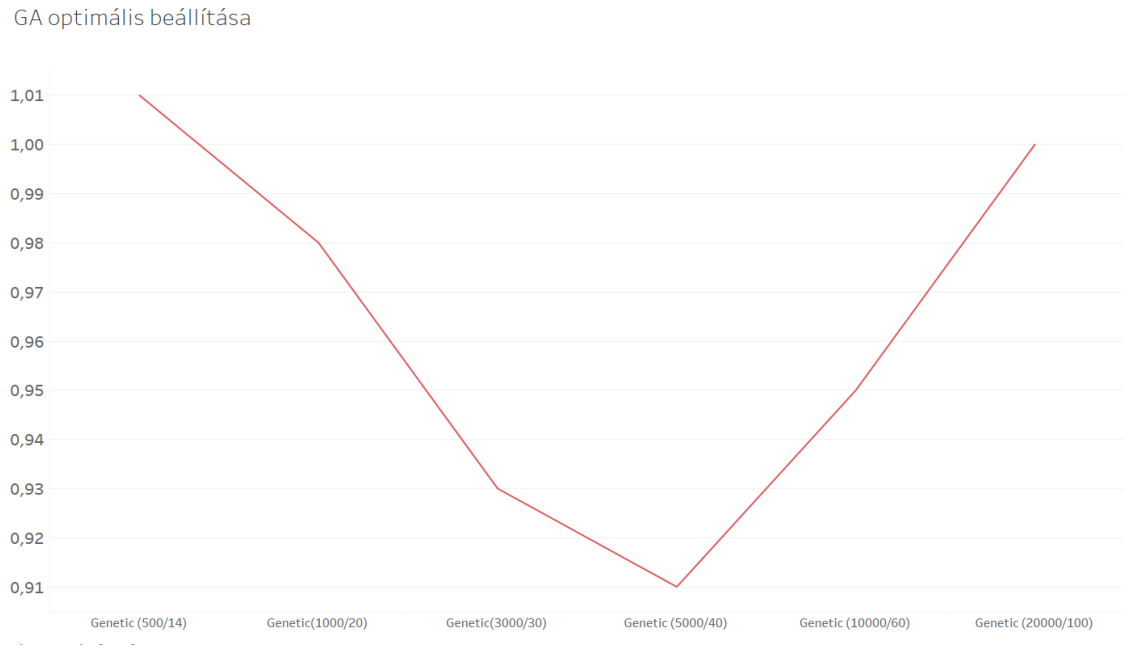
4.3.2. Túltanulás

Ahhoz, hogy a GA-t megfelelően paraméterezhessem, meg kell határoznom azt a tanulási „mélységet”, ahol még nem tanul túl mélyen rá a tanulóhalmazokra, hiszen, ha túl speciális esetre optimalizálunk, akkor az átlagos esetre már nem lesz olyan jó. Ezért, a GA-t több egyedszám/generációszám (a genetic után a zárójelben az első szám a populáció méretét, a második szám a generációk számát jelenti) mellett is lefuttatom egy raktármodellre, vizsgálva, hogy hol adja a legjobb eredményeket.

	Átlag	Szórás	Arány
Véletlen	1342626,7	7%	162%
ABC	876113,1	4%	105%
HEUR	1143724,2	9%	138%
Genetic (500/14)	840148,6	5%	101%
Genetic(1000/20)	812809,3	9%	98%
Genetic(3000/30)	772184,0	8%	93%
Genetic (5000/40)	759550,1	9%	91%
Genetic (10000/60)	790852,9	10%	95%
Genetic (20000/100)	830614,8	11%	100%

1. táblázat: A genetikus algoritmus optimális beállítása

Az 1. táblázat adatai mutatják, hogy milyen eredményeket kaptam az egyes algoritmustípusok kapcsán. A feltűntetett átlagot és szórást minden esetben úgy kaptam, hogy az algoritmust hússzor lefuttattam és kiértékeltem ugyanazon a raktármodellen és paramétereken, utána a kiértékelt legjobb egyedek kigyűjtési út/idő értékeinek az átlagát vettem. A futások során alkalmaztam a 4.4-es fejezedben részletesen leírt keresztvalidációs technikát.

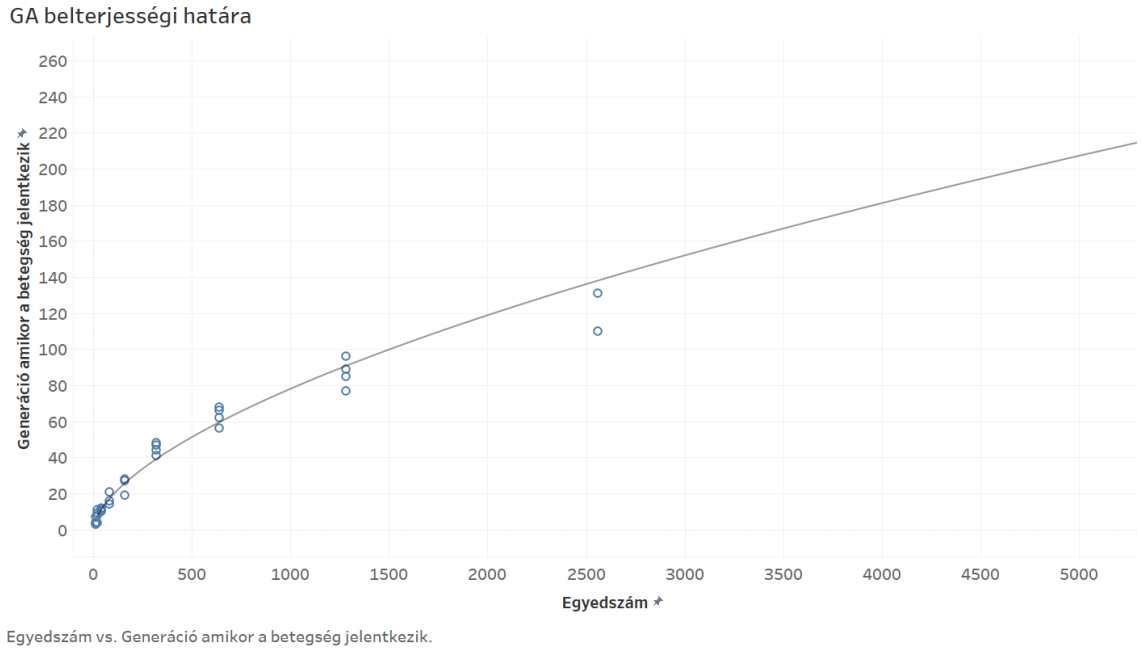


13. ábra: A GA fitnesz értékeinek százalékos alakulása az egyedszám/generációszám függvényében a 20000 egyed 100 generációjához viszonyítva

A 13. ábra által mutatott diagram az 1. táblázat eredményei közül a GA eredményeket mutatja. A diagramról leolvashatjuk, hogy a GA optimális beállítása 5000/40, vagyis 5000 egyed 40 generáción át való futtatása körül van. Ugyanakkor 3000/30as beállításnál is egész jó értékeket kaptunk. Ráadásul annak a tesztelési időigénye lényegesen kevesebb, ezért a tesztek során 3000/30-as beállítást fogok használni, tudva, hogy 5000/40-essel pár százalékpontot még lehet rajta javítani.

4.3.3. Belterjesség

A GA algoritmusok felépítésekor a szaporodás fázisba beleépítettem pár fontos „ellenőrző” metódust, ugyanis a véletlen kereszteződésekből keletkező új egyedeknél előfordulhat, hogy egy egyed két különböző kromoszómájához (vagyis két különböző termékhez) ugyanazon tárhely rendelődik hozzá. Ez nem megengedett, ezért a jelenséget „betegségnek” tituláltam, és ezekkel az ellenőrző metódusokkal próbáltam szűrni. Már a szaporodásnál megnehezítettem, hogy ilyen egyedek létrejöjjenek, de ha mégis bejutott egy-két ilyen egyed a populációba, azokat kiirtottam. Ugyanakkor, minden populációnál megfigyelhető egy generációszám, ahol egy generáció alatt az egész populáció kiirtódik e betegség miatt. Ennek oka pedig – miként a biológiában - a belterjesség. Közel az optimum ponthoz az egyedek már csak nagyon kis mértékben különböznek egymástól, ezért szinte lehetetlen létrehozni olyan egyedeket, melyekben ez a „betegség” nem jelenik meg.



14. ábra: A genetikus algoritmusok belterjességének határvonala

A 14. ábra által mutatott diagrammon megfigyelhető, hogyan alakul az egyedszámok függvényében az a generációszám, ahol ez a betegség jelentkezik. Bár a nagy véletlenfaktor miatt elég jelentős az adatok szórása, egyértelműen kirajzolódik egy hatványtrend, melynek egyenlete: $1{,}1x^{0{,}619}$ -szerű. A tesztelés során figyelemmel kell lennem, hogy ezt a vonalat semmiképpen se lépjem át.

4.4. Keresztvalidáció

A tesztelés során, a 3.4.3 fejezetben megtalálható adatokat „one-leave-out” keresztvalidációs technikával használtam, mely során a megrendelés-adatokat véletlenszerűen négy részre osztottam. A tesztelés során váltogattam, hogy melyik három halmazon tanul, és melyik negyedik halmazon kerülnek kiértékelésre az adott rakármodellen az algoritmusok. Minden teszt esetében mind a négy variáción többször is

lefuttattam az algoritmusokat. A következő fejezetekben az eredmények átlagát tüntetem fel.

Vizsgáltam, hogy az egyes esetekben mekkora mértékű a keresztvalidációs esetek közötti szórás. Erre a szórás értékre 3-5% jött ki (az átlaghoz képest). Az egyes futtatások szórásánál ez az érték 8-12%, (teljesen reális, hiszen a generációk során a véletlennek fontos szerepe van, ami miatt az adatsor valamelyest szórni fog). Láthatjuk, hogy a keresztvalidációs esetek átlagainak szórása kisebb, mint a teljes adatsornál tapasztalható szórás. Így kijelenthető, hogy a túltanulás veszélye ilyen paraméterek között még nem áll fenn, és az is, hogy az eredményeket a megrendelésadatok véletlenszerű kiválasztása nem, vagy elhanyagolható mértékben befolyásolta.

4.5. Távolságoptimalizálás

Először azt az esetet vizsgáltam meg, amelyben a kigyűjtési távolságokat optimalizálom. Az egyes algoritmusok minőségét és a GA fitnesz értékeit is az határozta meg, hogy a kapott elrendezésen mekkora távolságot kell bejárni az egyes listák kigyűjtéséhez, ahogy azt a 3.7.1-es fejezetben részleteztem.

JEL.	MEGNEVEZÉS	TÁRHELY/ÁRUCIKK ARÁNY	R	J	I	K
A	közepes tárhelykapacitás, normál arány	1,8	10	2	5	3
B	nagy tárhelykapacitás, hosszú sorokkal	3	13	2	8	3
C	nagy tárhelykapacitás, rövid sorok	4,5	25	2	5	3

2. táblázat: A távolságoptimalizálás tesztraktár-esetei

A vizsgált esetek közül hármat mutatok be, melyeknek paramétereit a 2. táblázat mutatja be. A feltűntetett átlagot és szórást minden esetben úgy kaptam, hogy az algoritmust hússzor lefuttattam és kiértékeltem ugyanazon a raktármodellen és paramétereken, utána a kiértékelt legjobb egyedek kigyűjtési út/idő értékeinek az átlagát vettem.

.A, raktáreset: közepes tárhelykapacitás, normál arány				
	Véletlen	*ABC*	*HEUR*	*Genetic*
Átlag	818169,2	443426	776497	585230
Szórás	1%	2%	4%	8%
Arány	140%	76%	133%	100%

3. táblázat: A, raktáreset eredményei, távolság optimalizálásnál

Az 3. táblázaton láthatjuk, hogy ilyen paraméterek mellett az ABC elemzés 24%-ponttal jobb eredményt ért el, a genetikus algoritmusnál. A többi viszont még a GA-nál is sokkal kevésbé hatékony.

B, raktáreset: nagy tárhelykapacitás, hosszú sorokkal				
	Véletlen	*ABC*	*HEUR*	*Genetic*
Átlag	809172,6	394247	766757	496607
Szórás	3%	1%	5%	4%
Arány	163%	79%	154%	100%

4. táblázat: B, raktáreset eredményei, távolság optimalizálásnál

A 4. táblázat által bemutatott mérésnél lényegesen több tárhely van, mint ahány kiosztandó árucikk. A tárhelykapacitást a sorok hosszának megnyújtásával értem el. Ebben az esetben kisebb a GA és az ABC hatékonysága közti különbség, de még mindig az ABC módszer eredménye a jobb.

C, raktáreset: nagy tárhelykapacitás rövid sorok				
	Véletlen	*ABC*	*HEUR*	*Genetic*
Átlag	1863478	918136	1631472,4	727826
Szórás	0,07952	0,01596	0,06825093	6%
Arány	256%	126%	224%	100%

5. táblázat: C, raktáreset eredményei, távolság optimalizálásnál

A 5. táblázatban látható C-raktáresetnél jelentősen több tárhely van, mint ahány betárolandó termék, és a tárhelykapacitás bővítését az állványsorok számának növelésével értem el. Ebben az esetben a GA produkálja magasan a legjobb eredményt.

Már a távolságoptimalizálás során is hasznos következtetéseket vonhatunk le, de érdemesebb előbb megvizsgálni az időoptimalizálás kérdéskörét.

4.6. Időoptimalizálás

Az időoptimalizálás egy jóval bonyolultabb feladat, mint az előző 4.5-es fejezetben bemutatott távolság alapú, ezt az állítást a **Hiba! A hivatkozási forrás nem található.**-es fejezetben támasztom alá

Először elvégzem a teszteket az előző 4.5-es fejezetben bemutatott raktáresetekre, és megvizsgálom milyen eredményeket adnak ki az egyes algoritmusok.

A, raktár típus				
	Véletlen	*ABC*	*HEUR*	*Genetic(3000/30)*
Átlag	810307	522127	792605	511023,5833
Szórás	6%	7%	9%	10%
Arány	159%	102%	155%	100%

6. táblázat: A, raktáreset eredményei, idő optimalizálásnál

Megfigyelhető a 6. táblázaton, hogy ebben az esetben a GA produkálja a legjobb eredményt, két százalékponttal jobbat, mint az ABC algoritmus.

B, raktár típus				
	Véletlen	*ABC*	*HEUR*	*Genetic(3000/30)*
Átlag	909797	595698	872728	465706,55
Szórás	9%	5%	8%	8%
Arány	195%	128%	187%	100%

7. táblázat: B, raktáreset eredményei, idő optimalizálásnál

Az 7. táblázatban bemutatott raktártípusnál már egyértelmű a GA dominanciája a többi algoritmussal szemben.

C, raktár típus				
	Véletlen	*ABC*	*HEUR*	*Genetic(3000/30)*
Átlag	1595120	968558	1480371	670666,5
Szórás	10%	8%	16%	11%
Arány	238%	144%	221%	100%

8. táblázat: C, raktáreset eredményei, idő optimalizálásnál

A C, raktáresetre kapott eredményeket tartalmazza a 8. táblázat. A tárhelyszám növekedésével nőtt a GA előnye is.

Az előző tesztek tapasztalatai alapján érdemes további raktáresetekben is megvizsgálni az egyes algoritmusok eredményeit. A következő raktáraknál két fontos szempontot vettem figyelembe. Egyrészt igyekeztem alacsonyabb tárhely/árucikk aránnyal rendelkező raktárakat tesztelni, hiszen ez a jellemzőbb az iparban. Másrészt a az állvány magasságát is csökkentettem, hiszen, a valóságban egy ilyen vertikális komissiózási folyamat során általában nincs sok szint, ahonnan kiszedik az árut. Mivel a szintek között nem definiáltam időbeli különbséget (hiszen vertikális komissiózásnál lényegében mindegy, hogy a kolléga, a polcon melyik termékért nyúl be), a sok szint nagyban befolyásolná az eredményeket. Az alábbi 9. táblázat tartalmazza a továbbiakban vizsgált raktárparamétereket.

SSZ.	MEGNEVEZÉS	TÁRHELY/ÁRUCIKK ARÁNY	R	J	I	K
1	Rövid sorokkal rendelkező	1,2	20	2	5	1
2	Hosszú sorokkal rendelkező	1,3	13	2	8	1
3	Kétszintesállványú egyenletes	1,2	7	2	7	2
4	Háromszintesállványú egyenletes	1,3	6	2	6	3
5	Kétszeres méretű raktár	2	18	2	9	1
6	Ötszörös méretű raktár	5	23	2	9	2

9. táblázat: Az időoptimalizálás további raktáresetei

Az eredményeket táblázatos formában mutatom be. Az átlag, mindenhol a teszthalmazok lefuttatása után azon időszükségeltek összege, mely a teszt halmazok kigyűjtéséhez kell Ezt az összeget akarom minimalizálni..

1. raktáreset, rövid sorokkal rendelkező raktár				
	Véletlen	*ABC*	*HEUR*	*Genetic(3000/30)*
Átlag	1344594	879157	1139266	771162,1
Szórás	7%	4%	9%	8%
Arány	174%	114%	148%	100%

10. táblázat: 1. raktáreset, rövid sorokkal rendelkező raktár

Az 10. táblázat esetében rövidebb sorokkal rendelkező raktárt látunk. A tárhely/árucikk aránya 1,2. Ebben az esetben kiemelkedő a GA teljesítménye.

2. raktáreset, hosszú sorokkal rendelkező raktár				
	Véletlen	*ABC*	*HEUR*	*Genetic (3000/30)*
Átlag	903354	588407	849742	558099,3684
Szórás	7%	5%	11%	11%
Arány	162%	105%	152%	100%

11. táblázat: 2. raktáreset, hosszú sorokkal rendelkező raktár

Az 11. táblázat esetében hosszú sorokat alkalmaztunk. A tárhely/árucikk aránya 1,3. Ebben az esetben a kisebb a GA előnye az előzőhöz képest. Ennek oka egyrészt az, hogy a hosszabb állványsorokban a komissiózó gép alapból jobban ki tudta használni a maximális sebességét (volt helye felgyorsulni), másrészt a tárhelyek közelebb helyezkedtek el egymáshoz így mindegyik módszer jobban teljesített, és a különbségek aránya kisebb lett.

3. raktáreset, kétszintes állvány				
	Véletlen	*ABC*	*HEUR*	*Genetic (3000/30)*
Átlag	655055	499368	587456	447918,5789
Szórás	6%	6%	7%	8%
Arány	146%	111%	131%	100%

12. táblázat: 3. raktáreset, kétszintes állvány

A 12. táblázatban kétszintű állványok vannak a raktármodellben, ugyanakkor a vertikális iránynál nem számolunk utazási időszükséglettel, hiszen a horizontális komissiózási modellről beszélünk, és így ez a szintkülönbség nem jelent időbeli különbséget.

4. raktáreset, háromszintes állvány				
	Véletlen	*ABC*	*HEUR*	*Genetic (3000/30)*
Átlag	543932	307137	502946	398076,1579
Szórás	8%	7%	9%	11%
Arány	137%	77%	126%	100%

13. táblázat: 4. raktáreset, háromszintes állvány

A 13. táblázatban háromszintű állványok vannak a modellben. Ez az egyetlen eset, ahol az ABC módszer optimálisabb eredményt ért el, mint a genetikus modell. Ennek oka, hogy mivel vertikális irányban nem történik mozgás, a modell nagyon leegyszerűsödik, a raktár mérete lecsökken. Ezek a paraméterek pedig kedveznek az egyszerűbb ABC algoritmusnak

5. raktáreset, kétszeres méretű raktár				
	Véletlen	***ABC***	***HEUR***	***Genetic (3000/30)***
Átlag	1232754	784868	1171275	637332,1579
Szórás	8%	6%	9%	6%
Arány	193%	123%	184%	100%

14. táblázat: 5. raktáreset, kétszeres méretű raktár

Az 14. táblázatban olyan raktármodellt láthatunk, melyben kétszer annyi tárhely található, mint ahány betárolandó termék. Ilyen esetben, jelentősen megnő a genetikus algoritmus hatékonysága.

6. raktáreset, ötszörös méretű raktár				
	Véletlen	***ABC***	***HEUR***	***Genetic (3000/30)***
Átlag	1479200	929996	1399367	648170,5789
Szórás	9%	7%	8%	12%
Arány	228%	143%	216%	100%

15. táblázat: 6. raktáreset, ötszörös méretű raktár

Az 15. táblázatban is látszik, ami az 14. táblázatban, miszerint minél nagyobb a tárhelyek/árucikkek arányszáma, minél több tárhely van az árucikkekhez képest, annál jobb a genetikus algoritmus. Ennek oka: az ABC számára túl „bonyolult”, azt figyelembe venni, hogy a sok üres tárhely miatt túl sok tér van az egyes kategóriáknak, és sok üres tárhely lesz, ami miatt távol kerülnek egymástól a termékek.

5. Az eredmények kiértékelése

A 4. fejezet eredményeit érdemes összevetni és elemezni, hogy pontosabb konklúziókat tudjunk levonni. A fejezetben a kiértékelés során az egyes raktáreseteknél csupán az ABC és a GA eredményeit fogom összevetni, mivel ezek bizonyultak hatékonynak. Ugyanakkor a fejezet végén kitérnék a heurisztikával kapcsolatos tapasztalatokra és megfigyelésekre is.

5.1. Távolságalapú és időalapú optimalizálás összevetése

Először megvizsgálom, hogy milyen különbségek léptek fel a távolság és az időoptimalizálás során, továbbá próbálok választ adni a változások okára.

	Táv opt.		**Idő opt.**		**A változás mértéke**	**Paraméterek**			
	ABC	**Genetikus**	**ABC**	**Genetikus**	**(százalékpontban)**	**r**	**i**	**k**	**tárhely/áru**
A, raktáreset	76%	100%	102%	100%	26%	10	5	3	1,8
B, raktáreset	79%	100%	128%	100%	49%	13	8	3	3
C, raktáreset	126%	100%	144%	100%	18%	25	5	3	4,5

16. táblázat: A kigyűjtési távolság- és időopimalizálás összevezése

A 16. táblázat összefoglalja a távolság- és az időoptimalizálásnál modellezett raktáreseteket és a hozzájuk tartozó eredményeket. Egyértelműen látszik, hogy a genetikus algoritmus az időoptimalizálás esetén sokkal jobb eredményeket ért el, mint az ABC, hiszen minden esetben hatékonyabb megoldást talált. Ennek fő oka az, hogy az ABC nem tudott számolni az időoptimalizáció által okozott bonyolultabb tényezővel, míg a genetikus algoritmusnak ez nem okozott különösebb gondot.

Az A raktáresetben a GA csupán 2 százalékponttal ért el jobb eredményt, mint az ABC. Feltételezésem szerint ennek az oka az, hogy a „k” paraméter értéke három, és az időoptimalizálásnál nem számolok a vertikális paraméter jelentette különbséggel. E feltételezésem helyességét az **Hiba! A hivatkozási forrás nem található.** fejezetben fogom vizsgálni.

A legnagyobb mértékű változás a B raktáresetnél figyelhető meg, ahol a GA 49 százalékpontot javított az eredményén. Már a távolságoptimalizálás adatai alapján látszott, hogy a nagyobb áru/tárhely arány kedvezett a GA-nak. Ezt az időoptimalizálásnál úgy tűnik még inkább ki tudja használni az algoritmus.

A GA a legjobb eredményét a C-raktáresetnél érte el, mind távolság, mind időoptimalizálás esetén. Ennek oka egyrészt az áru/tárhely arányszám magas értéke továbbá azt feltételezem, hogy a sorok hosszának mértéke is a GA-t hozza előnyösebb helyzetbe. Ezt, valamint a „k” paraméter értékére vonatkozó feltételezésemet is az **Hiba! A hivatkozási forrás nem található.**-es fejezetben vizsgálom részletesebben.

5.2. Az időalapú optimalizálás eseteinek elemzése

Hat raktáreseten vizsgáltam, hogy az időoptimalizálás során az egyes paraméterek hogy befolyásolják a kapott eredményt. Az előző 5.1-es fejezetben két feltételezést is tettem, e paraméterekre vonatkozóan:

- A k paraméter növelése az ABC algoritmusnak kedvez a GA-val szemben (mivel időoptimalizálásnál nem számoltam időkülönbséget vertikális irányban),
- másrészt, ha hosszúak a sorok az (i paraméter növelése) is kedvez az ABC algoritmusnak a GA-val szemben.

	Idő opt.		Paraméterek			
	ABC	**Genetikus**	**r**	**i**	**k**	**tárhely/áru**
1. raktáreset	114%	100%	20	5	1	1,2
2. raktáreset	105%	100%	13	8	1	1,3
3. raktáreset	111%	100%	7	7	2	1,2
4. raktáreset	77%	100%	6	6	3	1,3
5. raktáreset	123%	100%	18	9	1	2,0
6. raktáreset	143%	100%	23	9	2	4,5

17. táblázat: Tárhelytípusok összevetése a kigyűjtési idő optimalizálásánál

Az 17. táblázatban láthatjuk a hat raktáresetet, a hozzájuk tartozó paramétereket, illetve az ABC és GA egymáshoz viszonyított eredményét. Az adatokat összeségében figyelve látható, hogy a GA hatékonyabb eredményeket tudott adni, mint az ABC algortimus.

Az 1. raktáreset és a 2. raktáreset között a különbség a sorok hosszában és a sorok számában van. A tárhely/áru arány nagyságrendileg ugyanaz. A 2. raktáresetben a GA előnye jelentősen kisebb az ABC-vel szemben. Ez a két teszteset igazolja a második feltevésemet, hogy a hosszabb sorok valamennyire kedveznek az ABC-nek. Ennek hátterében az állhat, hogy a hosszabb sorok esetében kevesebbszer kell kanyarodnia a komissiózó gépnek, kevesebbszer kell gyorsítania, többször el tudja érni a maximum sebességét, többször adódik olyan eset, hogy két termék egy sorba kerül. Ezen okok miatt, az ilyen esetek jobban hasonlítanak a távolságoptimalizáló tesztesetekre, hiszen a döntő különbséget a gyorsulási paraméterek jelentik.

A 3. és a 4. raktáreset közötti jelentős eltérés oka a k értékének változtatása. Szemmel látható, hogy milyen jelentős különbséget okozott a két algoritmus teljesítményében ez a változás. Ez is igazolja a feltételezést, miszerint az, hogy vertikálisan nem definiálunk mozgási időt, befolyásolja a teszteket az ABC javára. Ugyanakkor fontos megjegyezni, hogy az a modell, ahol nem számolunk függőleges mozgásokkal, csak a horizontális komissiózást tudja modellezni, ezért nem érdemes háromszintű raktárakat is figyelembe venni. Amennyiben olyan raktárrendszereket is modellezni akarunk, ahonnan több szintről is zajlik a komissiózás, definiálni kell vertikális mozgásokat. Fontos kiemelni, hogy ilyen esetben a vertikális és a horizontális sebességi és gyorsulási értékek merőben eltérőek lesznek, hiszen teljesen más mértékben gyorsul egy targonca vízszintesen, mint ahogy emeli vagy süllyeszti a villáját.

5.3. A heurisztika eredményei

A 4.5-es és a 4.6-es fejezetben bemutatott eredményeket vizsgálva látható, hogy a 3.5.2-es fejezetben bemutatott heurisztika, nem adott megfelelő eredményeket. Ennek ellenére fontosnak tartom a módszer további vizsgálatát, ugyanis elképzelhetőnek tartom, hogy nagyobb raktárméret, termékszám és megrendelési adathalmaz mellett a hatékonysága jelentősen javul. A dolgozat keretein belül ezt nem tudtam megtenni.

5.4. Az algoritmusok alkalmazhatósága

Mivel az ABC-elemzést és hozzá hasonló módszereket sikerrel alkalmaznak a raktározásnál feltételezhető, hogy egy olyan algoritmus, amely hatékonyabb eredményeket produkált, mint az ABC, jól alkalmazható a gyakorlatban is.

A tesztek alapján a genetikus algoritmus a leghatékonyabb általam vizsgált módszer a bonyolult, a valós rendszerekhez legközelebb álló modelljeim esetében, különösen, ahol horizontális komissiózás zajlik, továbbá ahol az állványrendszer paraméterei (hossz/állványszám/komissiózóutak) - épületi vagy tervezői hiányosságok miatt- nincsenek optimalizálva. Minél komplexebb egy feladat annál jobban teljesít az algoritmus a többi modellhez képest.

Az állításaim további alátámasztására a közeljövőben szeretném tesztelni, valós, jóval nagyobb elemszámú mintákon is azt algoritmusokat, ahol az árucikkek aránya ezres-tízezres nagyságrendben van, a tranzakciók száma pedig millió fölötti. Már kaptam ígéreteket, hogy a rendelkezésemre bocsájtanak ilyen, valós céges adatokat, és így hamarosan el tudom végezni ezeket a teszteket. Remélem sikerül egy olyan alkalmazható eljárást létrehozni, mellyel jelentősen csökkenthető a raktárak komissiózási időszükséglete, ezáltal hatékonyabb logisztikai rendszereket hozva létre.

6. Összefoglaló és további fejlesztési lehetőségek

Dolgozatomban az NP nehéz tárhelykiosztási problémára [15] (SLAP) alkotott genetikus algoritmusomat teszteltem, és azt vizsgáltam, hogy milyen a hatékonysága, más algoritmushoz képest.

A kutatás során először létrehoztam egy szabadon paraméterehető raktármodellezési környezetet, amelyben a méréseket el tudtam végezni. Számos raktármodellt alkottam a kialakított modellezési környezet segítségével, melyekben elvégeztem a kísérleteket. A modellezéshez továbbá szükségem volt árucikkekre és megrendelési adatokra. Ehhez az R-szoftver Groceries adatbázisát használtam, melyben 169 árucikk és hozzájuk tartozó több mint 9 ezer megrendelési lista tartozott. Az apriori datamining-számolást leszámítva az egész modellt JAVA környezetben építettem fel, és a teszteket is itt végeztem.

Négy algoritmus hatékonyságát vizsgáltam. A legegyszerűbb véletlen kiosztás alapú tárhelykiosztást, egy osztályalapú a pareto elvre épülő ABC tárhelykiosztást [21], egy apriori datamining technikára épülő heurisztikát [22], valamint a saját genetikus algoritmusomat.

Több méretparaméterrel rendelkező raktáron vizsgáltam távolság- és időoptimalizálás szempontjából az algoritmusokat. Ipari környezethez legközelebb álló esetekben, amikor időre optimalizáltam, kis tárhely/árucikk arány mellett, alacsony állvánnyal (horizontális komissiózás esete) a genetikus algoritmusom érte el magasan a leghatékonyabb eredményt, holott a számos elvégzendő kísérlet és a hosszabb futási idők miatt az optimálisnál gyöngébb (egyedszám, generációszám) beállításokkal futtattam a genetikus algoritmusomat.

A tesztekből két markáns konklúziót vonok le. Egyrészt az ABC elvre épülő algoritmus, hatékony megoldást nyújt a problémára. Minél egyszerűbb és általánosabb esetről van szó, annál jobban tud teljesíteni, ilyen esetekben képes jobb eredményt elérni, mint bármelyik általam vizsgált algoritmus Másrészt a genetikus algoritmus a leghatékonyabb általam vizsgált módszer a bonyolult, a valós rendszerekhez legközelebb álló modelljeim esetében. Minél komplexebb egy feladat, annál jobban teljesít az algoritmus a többi modellhez képest.

6.1. Továbbfejlesztési lehetőség

A területen végzett kutatást három ütemben tervezem folytatni. A közeljövőben tesztelném az algoritmusokat nagyobb termékpalettán, mely termékek és a hozzájuk tartozó megrendelések valós céges adatbázisokból származnak és több ezer termék több millió megrendelését tartalmazzák. A második fázisban szeretném a modellt kiterjeszteni, és vizsgálni olyan eseteket is, ahol kevesebb a kikötés. Például van vertikális komissiózás, tárolható egy termék több tárhelyen és több termék egy tárhelyen. Ezzel együtt szeretném az általánosan alkalmazható modell hatékonyságát valós rendszereken tesztelni. Remélem ehhez sikerül találnom egyetemi és/vagy ipari partnereket, akik ebben szívesen részt vállalnának. A harmadik fejlesztési lépésben vizsgálnám a neurális hálók használhatóságát.

Hivatkozások

[1] Dr. K. Bóna, „Ellátási-elosztási rendszerek, előadásjegyzet," 2020.

[2] D. K. Bóna, „Az anyagmozgatási és raktározási rendszerek, jegyzet," [Online].Available: https://archive.edu.kozlek.bme.hu/pluginfile.php/33206/mod_resource/content/4/EA01_Anyagmozgato_rendszerek_bevezetes.pdf.

[3] N. F. ,. J. K. K. T. N. Z. Dévai, Lean szolgáltatásfejlesztés 1., Budapest: Kaizen Pro Kft., 2020.

[4] D. K. Bóna, Szerző, *Anyagmozgatási és Raktározási Folyamatok, Komissiózás.* [Performance]. 2020.

[5] D. K. Bóna, „Az RST folyamatok vizsgálata," [Online]. Available: https://archive.edu.kozlek.bme.hu/pluginfile.php/33209/mod_resource/content/4/EA04_Az_RST_folyamatok_vizsgalata.pdf.

[6] G. T., „Grundlagen der Kommissioniertechnik".

[7] Dr. J. Prezenszki, Logisztika I., Budapest, 2003.

[8] Dr. J. Prezenszki, Logisztika II., Budapest, 1999.

[9] B. M. K. Bóna, „Simulation and Optimization of Logistic Systems with Genetic Algorithms".

[10] Á. A., G. S., H. G. és V. K. A, „Genetikus Algoritmusok," 2003.

[11] D. Quammen, „Was Darwin Wrong?".

[12] B. Molnár, Raktári komissiózási folyamatok tervezése és irányítása PhD értekezés, Budapest, 2005.

[13] M. X. P. Ren-Qian Zhanga, „New model of the storage location assignment problem considering demand correlation pattern," *Science Direct,* 1019.

[14] S. L. a. G. S. W. H. Hausmann, „ Optimal storage assignment in automatic warehousing system".

[15] F. E. H., „Stock location assignment and order picking productivity".

[16] K. M., „Optimising the storage location assignment problem under dynamic conditions".

[17] Dr. J. Prezenszki, Rakzározás-Logisztika, Budapest: Ameropa Kiadó, 2010.

[18] E. L. S.-C. a. J. R. M.-T. J. J. R. Reyesa, „The storage location assignment problem: A literature review," *www.GrowingScience.com/ijiec,* 2019.

[19] J. G. T. G. a. H. Z. W. Wang, „Optimization of Automated Warehouse Location Based on Genetic Algorithm," *Atlantis Press,* 2017.

[20] Y. T. L. Y. J. Q. ,. D. D. Zhang, „A Genetic-Algorithm Based Method For Storage Location Assignments In Mobile Rack Warehouses".

[21] F. G. a. M. S. C. Battista, „Storage Location Assignment Problem: implementation in a warehouse design optimization tool,” 2011.

[22] C.-P. L. a. C. M.-C. D. Ming-Huang Chiang, „Data mining based storage assignment heuristics for travel,” *Expert Systems,* 2012.

[23] M. Bertalan, Szerző, *Anyagmozgatási és Raktározási Folyamatok, Komissiózási útvonalak.* [Performance]. 2021.

[24] P. M. a. C. Kasemset, „Storage Location Assignment Considering Three-Axis Traveling Distance: A Mathematical Model”.

[25] „Machune-learning-with-R-dataset/groceries.csv,” [Online]. Available: https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/groceries.csv. [Hozzáférés dátuma: 01 09 2022].

[26] „apriori-rule-with-grocery-data,” 09 11 2020. [Online]. Available: https://www.kaggle.com/code/changjunlee83/apriori-rule-with-grocery-data-for-class. [Hozzáférés dátuma: 12 08 2022].

[27] D. K. Bóna, „Az anyagmozgatás időszükségletének meghatározása,” [Online].Available: https://archive.edu.kozlek.bme.hu/pluginfile.php/33208/mod_resource/content/4/EA03_Anyagmozgatas_idoszuksegletenek_meghatarozasa.pdf.

[28] M. G. Zoltán, „Metaheurisztikus algoritmusok hatékonyságvizsgálata saját szoftver által létrehozott komplex tesztfüggvények segítségével, az eredmények felhasználása futódaru főtartó optimálásához,” Miskolc, 2014.

[29] A. O. E. Atmaca, „Defining order picking policy: A storage assignment model and a simulated annealing solution in AS/RS systems,” *Applied Mathematical Modelling,* 2013.

[30] D. K. Bóna, „Anyagmozgatási időszükséglet meghatározása, előadásjegyzet,” 2020.

[31] V. G. D. M. N. Tsamis, „Adaptive Storage Location Assignment for Warehouses Using Intelligent Products,” 2015.

[32] W. H. D. H. L. X. Wang Haibin, „Research on location optimization of automated warehouse under the background of intelligent manufacturing,” *Academic journal of manufacturing engineering,* 2020.