



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

TDK DOLGOZAT

Szerző neve:

Kelemen Dávid



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Hallgató neve: Kelemen Dávid
Intézmény: Óbudai Egyetem
Kar: Neumann János Informatikai Kar
Tanszék: Szoftverfejlesztés és mesterséges intelligencia intézeti
tanszék
Szak, évfolyam: mérnökinformatikus BSC. szak, IV. évfolyam

A dolgozat címe:

**Ételfutár drónok hálózatának optimalizálása és ábrázolása
szimulációban**
**Optimization and representation of a network of food delivery
drones in simulation**

Témavezető neve: **Prof. Dr. Szénási Sándor**
egyetemi tanár

Tartalomjegyzék

1	BEVEZETÉS	5
1.1	A PROBLÉMA	5
1.2	KUTATÁSI MÓDSZER	6
1.3	A KUTATÁS CÉLJA	6
1.4	JELLENLEG LÉTEZŐ MEGOLDÁSOK	8
1.5	VALÓS FEJLESZTÉSEK	11
2	A PROBLÉMA	14
2.1	A PROBLÉMA ANALÍZISE	14
2.2	A SZIMULÁCIÓS TÉR	14
2.3	A DRÓN	15
2.3.1	A FELHASZNÁLT MODELL	15
2.3.2	A MODELL VISELKEDÉSE	16
2.3.3	A DRÓN REPÜLÉSI SZABÁLYAI	18
2.3.4	A DRÓN SZÁLLÍTÁSI SZABÁLYAI	18
2.3.5	A MODELL SZIMULÁLÁSÁNAK ÁLLAPOTAI	19
2.3.6	A DRÓN HELYZETE A SZIMULÁCIÓBAN	20
2.4	AZ ÉTTEREM	20
2.5	A RENDELÉSEK	21
2.5.1	A RENDELÉSEK SZIMULÁLÁSÁNAK ÁLLAPOTAI	23
2.6	A TÖLTŐÁLLOMÁS	24
3	AZ ESZKÖZÖK KIVÁLASZTÁSA	26
3.1	SZIMULÁCIÓT FUTTATÓ MOTOROK	26
3.2	UNITY	26
3.2.1	MONOBHAVIOUR OSZTÁLY	27
3.2.2	COROUTINE OSZTÁLY	28
3.2.3	VECTOR2 STRUCT ÉS AZOK METÓDUSAI	28
3.3	PROGRAMOZÁSI NYELV	28
4	TERVEZÉS	29
4.1	ALAP-OBJEKTUMOK	29
4.1.1	BUFFER	29
4.1.2	SOURCE	29
4.1.3	SINK	30

5	SZÁMÍTÁSOK ÉS OPTIMALIZÁCIÓ	31
5.1	VÁLTOZÓLISTA	31
5.2	MATEMATIKAI FORMULÁK	33
5.2.1	A RENDELÉSKIOSZTÁS ELŐKÉSZÍTÉSE	33
5.3	OPTIMALIZÁCIÓ	46
5.3.1	OPTIMUM KERESŐ ALGORITMUSOK	46
5.3.2	GENETIKUS ALGORITMUS	46
5.3.3	FUTTATÁSI MINIMUM	54
5.4	TOVÁBBI MATEMATIKAI FORMULÁK	55
6	TESZTELÉS	58
6.1	A GENETIKUS ALGORITMUS TESZTELÉSI FUTTATÁSA	58
6.2	ALAP FUTTÁS	61
6.3	FUTTATÁSOK KÜLÖNBÖZŐ FLOTTAMENNYISÉGGEL	66
6.4	FUTTATÁSI LIMIT VÁLTOZÁSA	68
7	ÉRTÉKELÉS	69
7.1	ELÉRT EREDMÉNYEK	69
7.2	A KUTATÁS HATÁSA	70
7.3	FEJLESZTÉSI IRÁNY	70
7.3.1	AZ ENERGIA- ÉS IDŐKALKULÁCIÓ SZÉTVÁLASZTÁSA	70
7.4	EGYÉB TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK	71
8	IRODALOMJEGYZÉK	72
9	ÁBRAJEGYZÉK	75

1 BEVEZETÉS

1.1 A probléma

A minket körülvevő étel- és csomagkiszállító szolgáltatásokat nap, mint nap használjuk. Az ezeket működtető cégek már egy ideje jelentős szerepet töltenek be a nagyvárosok környezeteiben azáltal, hogy mind a belvárosokban, mind azok vonzáskörzetében egyre inkább összehangolva, gyorsan és hatékonyan igyekszenek csomagot vagy ételt eljuttatni a lakosságnak. Itthon talán a *Wolt* és a *Food Panda* nevei csenghetnek legismerősebben az ételkiszállítást illetően a legnagyobbak közül, a *Wolt* körülbelül 4-5000 futárral rendelkezik csak Magyarországon. Ezek a cégek az utóbbi években azon dolgoztak, hogy a nagyvárosi utakon közlekedni képes járművek (autó, motorkerékpár, kerékpár) sebesség- és költség-beli specifikusságait kihasználva alkossanak egy flexibilis koordinációs működést.

A kiszállítást végző rengeteg futár összehangolásának problémáját nem lehet egyszerűen kezelni. Nem csupán azért, mert a rendszer jellegéből fakadóan véletlenszerű viselkedéssel bír, de a mai valós megoldásoknak azért is meglehetősen nagy dinamikussággal kell dolgozniuk, mivel a futárok gyakran spontán ki- és bejelentkezhetnek munkavégzésre és a forgalom változásának hatása a rendszer terheltségét jelentősen befolyásolhatja. Továbbá, egy nagyvárosnyi rendelést kiszolgáló munkavállaló-erőt fenntartani alapvetően költséges és nehezen kezelhető megoldás. A nagy külföldi cégek (*Uber*, *Google*, *Amazon*) tesztjei és indulóban lévő működései azt mutatják, hogy van értelme az olcsóbb alternatíva keresésének, melyhez a földtől el kell rugaszkodnunk és a magasban kezdeni kutatni.

Az önvezető járművek a közeljövőben nagy eséllyel könnyen fogják tudni csökkenteni a kiszállítást végző futár cégek logisztikai költségeit. Ezt a témát manapság sok szervezet és kutatás tanulmányozza, ezen belül is a drónokkal végzett feladatvégzésben látnak különleges, alternatív megoldásokat. Ennek oka az, hogy ugyan a drónok nem feltétlenül képesek nagy tömegeket szállítani, de lég általi fölényük könnyen tud túllépni az eddigi ember alkotta nagyvárosok infrastruktúrai határain, mely gyors kiszállítást eredményezhet, ez pedig az ételszállítás egyik kulcsa.

Ugyan kijelenthetjük, hogy a drónos ételszállításnak hatalmas előnye, hogy nem függ a kialakult úthálózattól, és feltételezhetjük, hogy emiatt gyorsabb lehet a többi járműhálózatnál, a valóságban a megvalósítás elég sok akadályba ütközik még a mai napig is. Ide tartozik például, hogy a nagyvárosi környezeteknek rengeteg teljesen

vagy részben korlátozott 'no-fly' zónája van, melybe a belépés vagy egyáltalán nem lehetséges, vagy csak előzetes engedélykérés útján engedélyezett.[6]

További problémát okoz a kerttel nem rendelkező házak, lakások kérdése, melynél a rendelésleadás jelent problémát, ugyanis a legtöbb megoldás a mai napig csak a rendelő előre lerakott leszállópályájára való landolással együtt lehetséges. A hátráltató tényezők közé tartozik még az időjárás, ugyanis sem bizonyos szélerősség, sem adott csapadékmennyiség mellett a drónok nem tudnak megbízhatóan, biztonságosan üzemelni. Ugyan ezeknek a kérdéseknek a megválaszolása érdekes feladatnak bizonyulhat, ezen dolgozat nem foglalkozik velük, feltételezi azt, hogy az optimális feltételek már adottak, kifejlesztettek.

Amire a dolgozat koncentrálni akar, az arra a kérdésre való válaszadás, hogy egy többcsomópontos drónhálózatot hogyan lehet központilag úgy koordinálni, hogy azok vagy a legkevesebb idő alatt szállítsák ki rendeléseiket, a vevők elégedettségének elérését minél inkább megcélözva, és/vagy a legenergiatakarékosabban működjenek együtt. Mivel a problémában felvázolt egyedek és számítások száma jelentősen nagyra mutatkozik, a számítások igénylik az optimalizációs megoldások egyikének segítségét.

1.2 Kutatási módszer

Az általam használt kutatási módszer heurisztikus útkeresésnek mondható, azaz a dolgozat különböző realitásszerű feltételeket használ bemenetként, majd azok adott aránya mellett igyekszik számszerű adatokat szolgáltatni. A módszer lényege, hogy a sztochasztikus feladatra egy használható és minél hatékonyabb választ ad. E hatékonyság alapszik gyorsaságon és pontosságon, tehát a programban implementált algoritmusok által tartalmazott optimalizációs módszereknek a problémára szabva kell megjelenniük.

1.3 A kutatás célja

A szimulációk olyan szituációkban nyújtanak segítséget, mikor szeretnénk eredményt vagy becslést kapni egy olyan problémára, melyet, ha a valóságban kifejlesztett eszközzel, rendszerrel szeretnénk megoldani vagy megválaszolni, az korlátozott keretekbe ütközne, akár túl drága, túl bonyolult, túl veszélyes, jogilag korlátozott, vagy éppen elképesztően időigényes lehetne.

Ilyen esetekben a szimuláció sokféle problémára lehet megoldás, amihez először a problémát kell meghatározni. Analízisként meghatározhatunk egy eseményt, egy lezajló folyamatot vagy egy fizikai rendszer viselkedését. Tisztában kell lennünk a

szimuláció tárgyának minden elemével. A problémát részekre kell osztanunk és egy olyan modellt kell kreálnunk, amely az előbbieket alapján képes a működést matematikailag reprezentálni. A szimulációnak további előnye, hogy többször tudjuk futtatni, így akár végtelen különböző eredményt is kaphatunk általa.

Minél pontosabb az általunk kreált szimulációs modell, annál megfelelőbben tud valós választ adni a kérdéses problémára. A szimulációknak kétféle csoportját különböztetjük meg, lehetnek determinisztikusak és sztochasztikusak, más néven véletlenszerűek. A sztochasztikus szimulációk, melyek közé az általam kidolgozott dolgozat is tartozik, időben lezajló folyamat. Ezek a modellek valószínűségeket és véletlen számokat tartalmaznak. Véletlenszerű számként egy olyan egyenletesen elosztott sorozat elemét határozhatjuk meg, mely egy pszeudo random generáló által lett létrehozva.[1]

Ennek a kérdésnek a megvizsgálására kíván ez a dolgozat egy olyan szimulációt alkotni, mely felhasználja az általam implementált heurisztikus optimalizációs algoritmust. Cél az, hogy a szimuláció minél több tényezőt tudjon figyelembe venni, ami alapján generálja, számolja és ábrázolja a környezetet. A szimulált, központi irányítású drónoknak determinisztikus és sztochasztikus körülmények között minél stabilabban munkát végezni. A kifejlesztett algoritmus működése saját rendszerfeltételek alapján fog működni, kikötések által.

A szimuláció által szolgáltatott adatok becslőértékkel szolgálhatnak arra nézve, egy ilyen, realitásszerű vagy feltételezett erőforrásokkal bíró rendszer hogyan és milyen hatásokkal lenne képes funkcionálni a valóságban. Ezek az adatok tudnak segíteni megbecsülni azt, hogy a rendszert hogyan lenne érdemes kiépíteni, egyáltalán milyen hatással lenne a környezetre és milyen költségekkel járna kiépítése. Választ adhat arra, milyen drónokat érdemes használni egy ilyen hálózatban, milyen környezetben képzelhető el egyáltalán egy ilyen rendszer és például arra is, hová érdemes olyan állomást telepíteni, amely szervizeli, tölti, begyűjti a drónokat, hogy a legköltséghatékonyabban lehessen üzemeltetni a hálózatot.

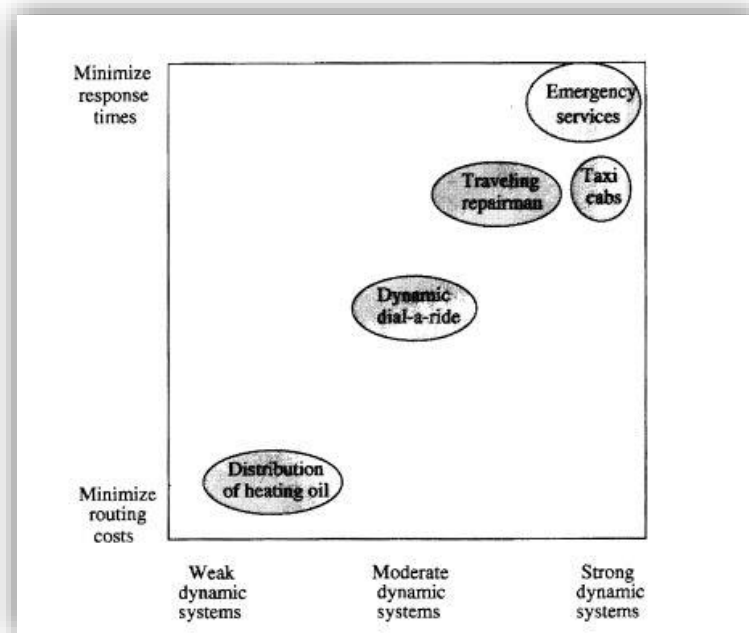
Az általam létrehozott szimuláció alapvetően determinisztikus lenne, de ez alól kivételt képez a rendelések megjelenésének generálása, mellyel igyekeztem a valóságban is történő, kiszámíthatatlan viselkedést utánozni.

Az itt bemutatott kutatás a *VRP*, azaz '*Vehicle Routing Problem*' dinamikus csomagfelvételes altémáját vizsgáló *DVRP*, azaz '*Dynamic Vehicle Routing Problem*' kategóriájába tartozik. A *VRP* -re elég sok vizsgálat és megoldás létezik, mely a statikus, előre meghatározott útvonalhálózaton, előre meghatározott, ismert lokációjú rendelések kiszállítására képes választ adni, de a dinamikus problémára is jószerével létezik válaszkészlet algoritmusokból. Jelen dolgozat az euklidészi térben

igyekszik eligazodni gráfhálózat nélkül, magas dinamikussággal rendelkező rendszert vázol fel, megoldása folyamatosságra törekszik.

1.4 Jelenleg létező megoldások

A dinamikusság különböző szintjeit különbözteti meg az a tanulmány, ami bemutat több VRP algoritmust / módszert az általa vázolt *Partially Dynamic Travelling Repairman Problem*-re, melynek alapötlete az ismert *Travelling Salesman* problémából származik.



1.1 ábra. Példák VRP rendszerekre a dinamikusság és az idő/költség függvényében

A tanulmány szintén tartalmazza azt a két alapvető tényezőt, mely egy dimenzió két oldalát képviselhetik egy megoldásban; az lehet vagy válaszidő alapján minimalizált, avagy útiköltség-hatékony. *Weak*, azaz gyenge dinamikusságúnak nevezik például azt a rendszert, ahol a kiszállítási célpontok nagy része előre ismert, kalkulált. A moderáltan dinamikus rendszerek közé sorolódnak azok, amik már igényelhetnek jelentős mennyiségű azonnali válaszreakciót. Az erősen dinamikus rendszerekre hoznak példát a vészhelyzeti szolgáltatásokkal, melyeket folytonos, magas prioritású sürgősség jellemez. Egy rendelésre való kérés vonatkozhat valamilyen ellátmány, tárgy kiszállítására, vagy szolgáltatás igénylésére. [8]

A DVRP dinamikusságának effektív szintjének meghatározására is találhatunk matematikai formulákat kutatásokban. Ehhez először egy '*planning horizon*'-t, azaz tervezési időablakot kell meghatároznunk, melynek mérete T . Legyen n_s a statikus igények (előre ismert rendelések/kérések), n_d a dinamikus igények (a kiszállítás

alatt megjelenő igények) száma. t_i legyen az az idő, mikor egy rendelés megjelenik a rendszer számára. t_i tehát a statikus rendelések esetében 0 lesz, míg a dinamikus rendelések esetében egy T -nél kisebb szám. Így születik meg az első képlet a dinamikusság szintjére:[9]

$$\delta = \frac{n_d}{n_s + n_d} \quad (1.1)$$

, ahol δ egy 0 és 1 közé eső szám. *Larsen* kutatásában a képletet kibővíti azzal, hogy kiterjeszti azt, hogy tartalmazza az igények megjelenési idejét és az esetleges tervezési időablakot:[10]

$$\delta = \frac{\sum_{i=1}^{n_s+n_d} \frac{t_i}{T}}{n_s + n_d} \quad (1.2)$$

Larsen ezt a képletet úgy is kibővíti, hogy tartalmazza a lehetséges igénylő által meghatározott határidőt is, ahol b_i jelöli ezt.[10]

$$\delta = \frac{\sum_{i=1}^{n_s+n_d} \frac{[T - (b_i - t_i)]}{T}}{n_s + n_d} \quad (1.3)$$

A dinamikus *VRP*-t kezelő szekvenciális stratégiák közé sorolhatjuk a következőket:

- *First Come First Served*: Az algoritmus egy *FIFO* (*First In First Out*) sort használ, melynél minden egyes új elkészült, vagy elkészülendő rendelés a sor végéhez adódik hozzá. Az algoritmus megnézi mely drónok érhetőek el, majd a *FIFO* sor első, egyetlen levehető elemét, azaz a 'head'-et osztja ki rendelésként a kiválasztott drónnak. Ezután a fej-elem a sorban utána következő elem lesz. Mivel ez az idő függvényében történik, ezért láthatjuk majd, hogy van a *Priority Scheduling*-gel összefüggésük. Felmerülhet problémaként ennél az algoritmusnál, hogy a *head* feltarthat olyan rendeléseket a sorban, melyeket nála messzebb kell kiszállítani. [12]
- *Stochastic Queue Median*: Ennél a módszernél az adott jármű a számára előírt szerviz helyszín mediánközeppontján várakozik arra, hogy a rendszer igénybe vegye. Munkavégzésre az adott helyszínre siet, majd a medián ponthoz tér vissza várakozni.[11]
- *Nearest Neighbor*: A megoldás lényege, hogy amikor a munkavégző egység végez valahol a feladatával, a hozzá legközelebb eső feladathoz siet további munkavégzésre. [11]
- *Shortest Job First*: A *Shortest Job First* algoritmus azt nézi meg, hogy az elkészült rendelések közül melyiket kell a legközelebbre kiszállítani az étteremből, majd azt osztja ki először a futárok közül. Így kiküszöbölhető az előbb leírt FCFS algoritmus problémája, miszerint a hosszabb rendelések

feltartják a közelebb, gyorsabb rendelések kiosztását. Ennek használata azonban olyan problémát okozhat, hogy folyamatosan kreálódnak rövid rendelések, egy-egy hosszabb rendelés pedig "beragad" ezek mögé, mert a futárok sosem kapják meg ezeket. [12]

- *Least Lost Value*: Az irányelv lényege, hogy a feladathoz értéket rendel hozzá, mely érték egy monoton csökkenő függvénnyel határozható meg. Az a feladat lesz kiválasztva elvégzésre, mely a legnagyobb értéket veszti, ha nem találunk számára munkavégzőt a vizsgálat pillanatában. [12][13]
- *TSP policy*: A *Travelling Salesman Problem* alapján az igények egy adott méretű halmazba gyűjtődnek. Ha ez megtörtént és fixálva van, megkeressük azt a megoldást, mely optimális megoldást ad az igények összességére. [11]

Egyszerűnek mondható stratégiákon kívül '*classical insertion procedures*'-ként beszélhetünk azokról a megoldásokról, melyek az új adat, a beérkezett új igény után újraoptimalizálják a bejárandó, kiszorgálandó útvonalat. Az ilyen megoldások felhasználják a '*rolling horizon*' elvet, mely lényege, hogy bizonyos időközönként vizsgálatot indítványoz, figyelembe vételt az új igényekre, kérésekre. [11]

MIP, azaz *mixed integer programming* alapú heurisztikus elgondolással készült a *Dynamic Dispatch* algoritmus. Az MIP az '*integer programming*' olyan kategóriája, amelyben a változók nem csak diszkrét változók lehetnek. A közlekedési ütemezésben előfordul alkalmazása, ahol a bináris döntési változók jelezhetik a hozzárendeléseket járművekhez, útvonalakhoz. A *Dynamic Dispatch*-et megalkotó dolgozat olyan időablakkal dolgozik, mely minden lefutáskor optimalizálja a rendszer állapotát az igények alapján. Az időablak hosszát a T paraméter jelzi, L azt az időbeli lépést jelzi, amennyire a rendszernek érdemes előre terveznie, M pedig a lefutási iterációkkal követendő időben való előrelépési konstans. [14]

A metaheurisztikák szintén megoldást tudnak nyújtani, hiszen képesek nagy kalkulációs igényű kombinatorikai problémára legjobb, vagy ahhoz közelítő értéket nyújtani. Hasonlóan a jelen dolgozat optimalizációs megoldásához, a *Decision Support for Vehicle Dispatching Using Genetic Programming* kutatás is az evolúciós algoritmusokból merít, de nem az itt bemutatott dolgozatban felvázolt módon. Különbség, hogy genetikusan programozással dolgoztak az alkotók, illetve a rendeléskiosztás szekvenciális módon működik. Egy új szállítás igényekor a rendszer megvizsgálja az adott rendszer feltételeit és a kiosztandó rendelés hatását a generált függvényekkel a futárokra, melyek különböző potenciális útattribútumok alapján vannak befolyásolva (felvételi idő, leadási idő, szállítási idő, stb.). Egyszerűen mondva ezek az adatok, melyek leírják a futárok útvonalába beilleszteni kívánt új tényezőket. [15]

Az egyik szerző, *Potvin* egyik másik munkája a neurális hálók *backpropagation* algoritmusát használja fel arra, hogy jó döntésre tanítsa a szimulált diszpécser. A neurális háló bemeneti vektorai egy-egy futárra való leírása a kiosztási helyzetnek, ezek a vektorok a másik munkában is megjelenő útattribútumokkal megegyező paraméterértékekből származnak. [16]

Genetikus algoritmust használ jelen dolgozathoz hasonlóan futár és rendelés fel-/leadási pontok közötti kalkulációkhoz az a tanulmány, mely a statikus és dinamikus rendelési igényeket is így ossza szét a közlekedő futárok között. A tanulmányban vázolt megoldás azonban, különbözve a jelen kifejtendő implementációtól, szekvenciálisan ossza ki a dinamikus szakaszban a mozgó pontok között a megjelenő rendeléseket. [17]

Érdemes a megemlíteni, hogy járműtípusokat ötvöző DVRP megoldások is léteznek, melyekben együtműködik egy földön közlekedő jármű, és egy vagy több drón egység. Az elképzelést bemutató munka során két új probléma is vázolódni látszik; a '*flying sidekick TSP*' (egy drón és egy teherautó együtt kiszolgál egy rendelőhalmazt) és a '*párhuzamos drón-ütemezés TSP*' (egy teherautó és egy drónflotta kiszolgál egy rendelőhalmazt).[18]

Szimulációk terén az itt felvázolthoz hasonlatosat készített el a *Profit Driven Drone Scheduling in Last Mile Food Delivery Networks* munka, mely önvezető drónok több csomópontos hálózatában mutatta meg több ütemező eljárás (FCFS, SJF, PS, LLV) működését különböző adottságú helyzetekben. A dolgozatban szimulálták a 'no-fly zone' területeket, így a drónok egyik teljesítendő feladata lett ezeknek a kikerülése is. A kutatás rendelkezik egy olyan időfüggvényvel, mely egy értéket rendel hozzá a rendeléshez aszerint, hogy mennyi ideje (nem) lett kiszállítva. A szimuláció szintén két dimenzióban lett ábrázolva, *Unity* felhasználásával, de felhasználtak egy olyan eszközt is, mely a szimulációhoz generált egy 3D-s térképet is. A dolgozat szimulációjában nem szerepel a szél hatásának figyelembe vétele, illetve az akkumulátor fogyasztása sem játszik központi szerepet benne.[12]

1.5 Valós fejlesztések

A következőkben olyan nemrégiben fejlesztendő, létező technológiai megoldásokat fogok bemutatni, melyek az önvezető járművek csomagszállításával kapcsolatban folynak. Ezen belül is olyan fejlesztésekre fókuszálva, mely a szakirodalom által "last mile delivery"-nek, azaz olyan kiszállításnak van nevezve, mely a szállítási folyamat végén a közeli raktárból/logisztikai állomásról közvetlenül a címzetthez szállít csomagot. Ezeknek a megoldásoknak az áttekintése hozzájárulhat ahhoz, hogy

egyfajta viszonyítási és kiindulási alapot nyújtson az általam elkészített szimulációhoz.

Amazon Prime Air

Talán a leghíresebb e technológiai fejlesztések közül az *Amazon Prime Air* nevű szolgáltatása, mely 2013-ban indult útjára. 2016-ban sikeresen elvégezték az első teszt-kiszállítást Cambridge-ben, ahol előnyükre szolgált az ott létesült, átlagosnál enyhébben korlátozó légi szabályrendszer. A DoD (*Department of Defense*) meghatározása alapján az *Amazon* drónjai "small" kategóriába sorolhatók. [19][20]

Repülési idejük körülbelül 30 perc, maximális repülési sebességük 80 km/h. 2.3 kg súlyt képesek szállítani. Ezek egyelőre úgy tudnak leszállni, hogy a megrendelő egy landoló szőnyeget terít le saját kertjében, melyet a drón biztonságos, leszállásra alkalmas területnek érzékel. [21]

A hatósági engedélyek megszerzésének nehézségei miatt egészen 2019-ig kellett várni arra, hogy a FAA (*Federal Aviation Administration*) engedélyezze működését, mellyel most az alacsony népsűrűségű területek felett és csak meghatározott 2.3 kg, vagy annál könnyebb csomagokkal szeretnék elkezdni a kiszállításokat.

DHL Parcelkopter

A DHL is a rövidtávú drónok által végzett kiszállításra koncentrált, melyet városi környezetben máshogy fejlesztettek, mint a vidéki, gyengébb infrastruktúrával rendelkező területeken. A cégnek 2013-ban lett partnere a *Microdrones* nevű gyártó, mellyel a gyógyszerészállítás tesztelését tűzték ki célul az utóbb említett, vidéki környezetben, ahol nem voltak zavaró tényezőt jelentő épületek. A cég a Parcelkopter projekt keretén belül az első kereskedelmi, ember nélkül irányított kiszállító szolgáltatását igyekezett elindítani Észak-Németországban.[22]

A *Parcelkopter* 4. generációjának drónjai már összesen 2200 km-t teljesítettek és képesek voltak 60 km-es szállítás teljesítésére. Céljuk az volt, hogy a kifejlesztett technológiát később városi vagy nagyvárosi környezetben is tudják alkalmazni. A cég 2021-ben jelentette be, hogy leáll a Parcelkopter fejlesztésével.

Google Wing

A Google leányvállalata is egy automatizált, drónok által végzett csomagszállító rendszert fejleszt, melyben a szállítóegységek hibrid drónok, melyek rendelkeznek vertikálisan és horizontálisan elhelyezkedő propellerekkel is. A járművek 30-40 méter magasan repülnek és a 110 km/órás sebesség elérésére képesek. Leszálláskor nem feltétlenül pontosan a cél GPS-koordinátáin landolnak, hanem a megfelelő területen. A csomag kiszállítása után a drón visszatér a Wing töltőállomására, ahol

egy töltőpályára leszállva azonnal tölteni is kezdi magát. A cég már több, mint 250.000 kereskedelmi szállítást vitt véghez drónjaival.

2 A probléma

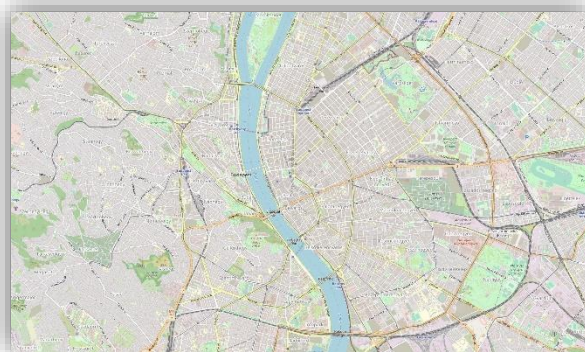
2.1 A probléma analízise

A dolgozat célja tehát a fennálló ütemezési és rendeléskiosztási problémára való dinamikus és folyamatos válaszadás. A dolgozat legfőbb szereplői tehát a mobilis drón-entitások, akik a rendeléseket az éttermekből a rendelés által meghatározott helyre szállítják egy szimulált nagyvárosi környezetben. Ők dolgozatban megjelennek még futár és kiszállítódrón néven is. A rendelések véletlenszerűen generálódnak. A rendelések szintén mozgó objektumok, melyek mozgását nem önmaguk végzik. Az éttermek statikus egységek, a program futása közben nem változtatható helyük, a program futása előtt megadható lokációjuk. A dolgozatban elsődleges fontosságú, hogy a drónok valódi viselkedését szimulálva, szerepet kapjon a szimulációban a drónok akkumulátorainak energiaváltozása és az erre való érzékenység a címkiosztó algoritmus részéről. Töltőállomásokként szerepelnek a dolgozatban azok az entitások, ahol a drónok az akkumulátorcseréjüket elvégzik. Statikus objektumként adhatók meg a program futása előtt.

A dolgozat nem kívánt kitérni a fennálló téma minden egyes valóságalapú részletére, mely a problémában nem közvetlen szerepet tölt be, ilyen esetben kikötések lettek hozva, melyek a rendszert felépítő modelleknél vannak tárgyalva.

2.2 A szimulációs tér

A szimulációs teret a futtató környezet fogja kialakítani, mely rendelkezik a kezdetben meghatározott objektumszámmal és az objektumok tulajdonságaival. Célszerű úgy kialakítani a programot, hogy indításkor meg lehessen adni a szimulációban létrehozandó modellek entitásainak számát. Így többféle esettel is tanulmányozni tudjuk a szimuláció által kialakított környezet viselkedését. A dolgozat szimulációja 2 dimenziós nézetben mutatja az ábrázolni kívánt történéseket, de a valós számítások természetesen 3 dimenzió alapszanak, a drónok végeznek vertikális mozgást. A vertikális mozgások a drónmodellek maximális emelkedési, illetve



2.1 ábra. A felhasznált felület

süllyedési sebességével történnek. A szimuláció sebessége megadható a program indítása előtt.

A környezet szimulálására egy budapesti térképet használ fel a szimuláció, melynek jelentősége egyedül a kerületi rendélesozslásban jelenik meg, melyről később ejtek szót. A szimuláció által feldolgozott terület, mely a térkép valóságbbéli mérete:

- Szélesség: 11890 méter
- Magasság: 6920 méter
- Terület: $\sim 83 \text{ km}^2$

Ezen adatok és az éttermek elhelyezkedése alapján a várható maximális kiszállítási távolság kb. 5,3 kilométer.

2.3 A drón

A drónok olyan légi járművek, melyeknek a fedélzetén nem tartózkodik pilóta. Hivatalos megnevezésük UAV, ami az angol *Unmanned Aerial Vehicle* szavakból származik. Lehetnek távolról irányítottak, illetve vagy előre beprogramozott repülési tervet követő, vagy dinamikusan automatizált működésűek. Eredetileg katonai felhasználásuk volt legjobban elterjedt, manapság már használják tűz-felügyeletre, az időjárás monitorozására, de akár a levegő szennyezettségének vizsgálatánál is tudnak segítséget nyújtani. Méretük változó, de általánosan kijelenthető, hogy minél kisebb egy drón, annál könnyebb üzemben tartani és megfizetni. A megvalósult vagy megvalósulóban lévő projekteknél látható, a drónok képesek kis csomagok szállítására, ezen a kategórián belül, talán legfőképp, az ételszállítást is értve.

2.3.1 A felhasznált modell

Mindenekelőtt egy olyan drónmodellt kellett dolgozatomnak megalkotnia, mely reális és általános képet ad az aktuális drónok feltételeiről. Emiatt ahelyett, hogy egy nagyvállalat által, a feladatra specifikáltan fejlesztett kiszállítást végző drón adatait használná fel a projekt, egy jelenleg kereskedelmi forgalomban lévő, megbízható gyártótól származó, a feladatra potenciálisan alkalmas, létező drónmodell tulajdonságait használja fel a szimulációs modell. Cél volt továbbá, hogy a drónnak minden szükséges tulajdonságához hozzá lehessen férni, melyet a gyártó publikusan közzétett.

A választás így a DJI Matrice 300 RTK drónra esett, melynek a szimuláció szempontjából érdekes adatai a következők: [2]

Tömeg:	6.3 kg (2 db akkumulátorral)
Maximális hasznos teher:	2.7 kg
Maximális felszálló tömeg:	9 kg
Maximális emelkedési sebesség:	6 m/s vagy 22 km/h
Maximális süllyedési sebesség (függőleges):	5 m/s vagy 18 km/h
Maximális sebesség:	23 m/s vagy 83 km/h
A drón által használt akkumulátor (TB60) adatai pedig a következők: [2] [3]	
Kapacitás:	5935mAh
Feszültség	52.8 V
Energia:	274 Wh
Típus:	LiPo (12S)

2.3.2 A modell viselkedése

A szimulációm alapja tehát az lesz, hogy egy általa használt drón 2.7 kg tömeg maximális hasznos terhet tud vinni. Ez ételszállításhoz megfelelő lehet, számolva azzal, hogy átlagosan 0.5-1 kg élelmet és folyadékot fogyasztunk el egy étkezésnél, még úgy is, hogy plusz csomagolás is fogja terhelni a drónt.

Egy drón energiafogyasztására meglehetősen sok tényező befolyásolhatja. Ezek közül én a következőket vettem bele a szimulációmba:

- Tömeg (mely tartalmazza az akkumulátor tömegét)
- Szállított tömeg / Szállított rendelések mennyisége
- Akkumulátor energia kapacitása
- Sebesség

Ezen kikötések mellett az energiafogyasztásra a következő képlet lesz alkalmazva: [4] [5]

$$E_{pm} = \frac{g \sum_{k=1}^3 m_k}{r\eta} + \frac{P_a}{v_{rep}} \quad (2.1)$$

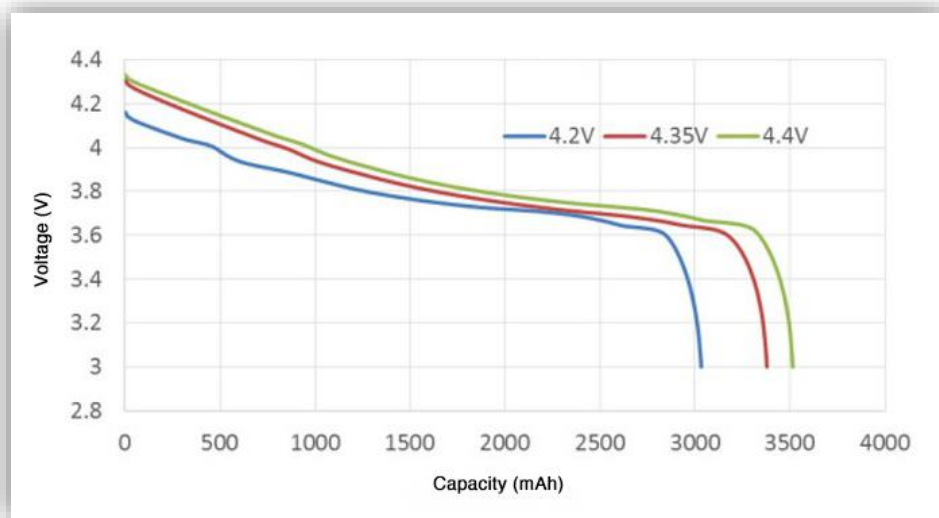
Jelölésekben továbbá g a gravitációs gyorsulás, k a drón komponenseinek indexe (ahol 1 = dróntest, 2 = drón akkumulátor, 3 = szállított csomag), m_k az adott komponens tömege.

Továbbá, r a repülő járműveknél használt siklószám (kutatásában a képlet megalkotója 3-mal számolt általánosan), η a propeller és a motor közötti áramszállítás hatásfoka, P_a a szükséges teljesítménye a repülőelektronikának és v_{rep} pedig a repülési sebesség. E_{pm} pedig egy Joule/méterben megadott arány az energiafogyasztásra.

A felépített szimulációs tér nem tartalmazza a szélviszonyokat, mint tényezőt a drón utazása közben, de a fent bemutatott képlet szélfaktoros verziójával könnyen lehetne implementálni ezt a viselkedést. [4] [5]

$$E_{pm} = \frac{1}{1-\varphi} \left(\frac{g \sum_{k=1}^3 m_k}{r\eta} + \frac{P_a}{v_{rep}} \right), \text{ ahol } \varphi = \frac{\text{szél sebessége}}{\text{drón repülési sebessége}}. \quad (2.2)$$

A biztonság érdekében a drón akkumulátora repülés közben nem eshet 20-30%-os kapacitás alá, mivel előre nem látott tényezők is felléphetnek repülés közben. Ilyen például a valóságban az erős szél megjelenése, ami a drónt igyekszik röppályájáról letéríteni, amely tényező erős hatással lehet az energiafogyasztásra, de egy



2.2 ábra. Lítium polimer akkumulátor merülési görbéje, forrás: lipobattery.com

félrekalkulált útvonal vagy a repülőelektronika meghibásodása is vezethet nagyobb energiafogyasztáshoz. A gyártó hivatalos karbantartási útmutatója is 20%-os energiaszintnél javasolja az azonnali repülés megszakítását.

A szimulált energiát adó lítium-polimer akkumulátorok a drónokban 12 cellásak, amely cellák kezdetben 4.4 V feszültségűek. 2 ilyen akkumulátor lehet egy drónban egyszerre. A lítium-polimer akkumulátorok fontos tudnivalója, hogy kb. 3V feszültség felett működnek, ha feszültségük ez alá csökken, akkor akár kapacitásukból veszíthetnek, sérülhetnek. A gyártó 20%-os javaslatát figyelembe véve a szimulációban 3.3V-os cellatöltöttségnél lesz a biztonsági limit, mely után a drónok vészhelyzeti módba kapcsolnak és a legközelebbi töltőállomás felé indulnak. Ezáltal meghatározható Joule-ban a drón energiaszintje, mely a következő képlet szerint számolódik:

$$U * Q * 3600 * 2$$

$$\text{kezdeti állapotban tehát: } (4.4 \text{ V} * 12) * 5935 \text{ C} * 3600 * 2 = 2.256.249,6 \text{ J}$$

$$\text{vészhelyzeti állapotra váltáskor: } (3.3 \text{ V} * 12) * 5935 \text{ C} * 3600 * 2 = 1.692.187,2 \text{ J}$$

2.3.3 A drón repülési szabályai

A drón a szimulált nagyvárosi környezetben mindig 100 méter magasan közlekedik. A szimuláció nem veszi figyelembe az olyan területeket, melyek légtere a civil légi járművektől el van zárva (no fly zones). Emellett fontos gondolni a tényre, hogy a szimuláció belvárosi környezetű térképpel fog rendelkezni, melynek a valós életben vett potenciális kivitelezésénél sok gondot kellene fordítani az épületek elkerülésére. A szimulációban a fel- és leszállás abszolút vertikális útként fog megtörténni, ahol az esetleges épületek modellezése és kikerülése nem lesz feladat. A drón más drónokkal való ütközésének lehetőségét a szimuláció figyelmen kívül hagyja, feltételezve saját szenzorokat a drónon, mely képes a két közel kerülő drónnál elindítani egy elkerülő manővert. A drón a lehető legnagyobb sebességgel fog közlekedni, ami a felvázolt modellnél 23 m/s, illetve vertikális mozgásait is a leggyorsabban igyekszik elvégezni.

A drón elsődleges biztonsági szabálya közé tartozik, hogy egy bizonyos szint alá nem eshet az energiája. Ha a megállapított érték alá esik mégis, akkor a drón veszélyhelyzeti állapotba vált, majd a legközelebbi töltőállomás felé veszi az irányt azonnal.

2.3.4 A drón szállítási szabályai

A fent említett specifikációk alapján a drón 2.7 kg tömeget képes szállítani. Ha ennél súlyosabb rendelést kínál fel neki az étterem, azt a rendszer el fogja utasítani. Ezen felül a szimulációban kikötés van arra, hogy a maximálisan számított rendelések

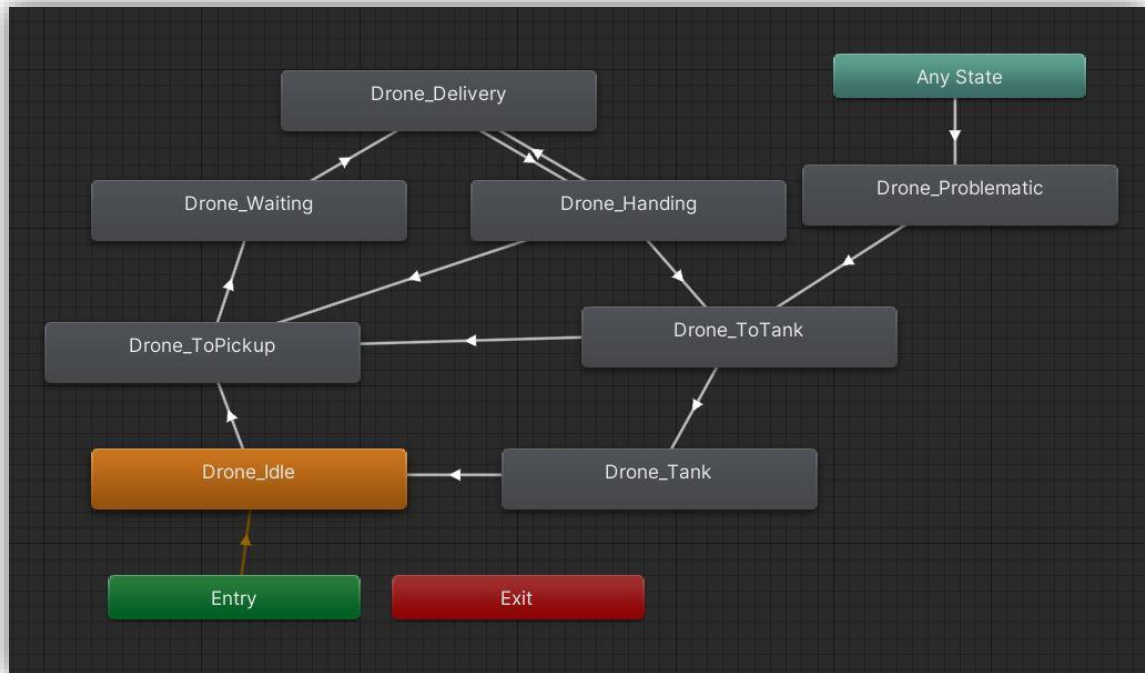
száma 2, amely a modell specifikációja alapján reális és biztonságos. Ennél több maximálisan szállítható rendelésre egyébként pontosan ugyanolyan számításokat kellene végezni, mint amiket a program végez, a rendeléseket hosszabb listában tárolva. Ha az étteremnél a drónba bepakolódott egy rendelés, onnantól kezdve annak a rendelésnek a kiszállítása nem ruházható át más drónra, pont úgy, mintha nem lehetne kivenni a drón dobozából az ételt, ami egyszer oda bekerült.

2.3.5 A modell szimulálásának állapotai

Az előzőekben felvázolt különféle behatások és meghatározások alapján látható, hogy szükségszerű megállapítanunk a szimulációban használandó drón által felvehető állapotokat. Ezek a következők lehetnek:

- Feltöltött, pihenő állapot: Ebben az állapotban a drón arra vár, hogy a rendszer ráosszon egy rendelést. Ebben az állapotban csakis a töltőállomáson tartózkodhat az egység.
- Légi mozgást végző állapot, csomagfelvétel céljából: Felemelkedés, majd utazás a drónra osztott rendelés elkészülési lokációjára.
- Várakozási és átvételi állapot: Az étteremnél a drón landol, majd ha a rendelés még nincs elkészülve, várakozik. A rendelés vagy rendelések készenlétét egy bepakolási fázis követi, szimulálva az állapotot, mintha egy kezelőszemély vagy automata robot a drón szállítóegységébe pakolná a csomagot. Ez a szimulációban 30 másodpercet ölel fel. Ezután a drón felemelkedik a repülési magasságába.
- Légi mozgást végző állapot, csomagleadás céljából: Utazás a drónra osztott rendelés kiszállítási helyére.
- Csomagátadási állapot: A célnál a drón landol, majd egy csomaglerakást szimulál. A valóságban ez a lerakási fázis még mindig gondot okoz a mérnököknek, melyet különféle módszerekkel próbálnak orvosolni. Az itt megvalósított megoldásban a drón a beállított magasságból fogja, leeresztést szimulálva, a földre engedni az ételt. A rendelés átvétele szintén 30 másodpercig tart, mely után a drón felemelkedik a repülési magasságába.
- Légi mozgást végző állapot, töltőállomás felé haladva: Kiszállítás után a drón megkeresi a legközelebbi töltőállomást és odautazik, ha nincs nála több kiszállítandó rendelés. Miután odaért, landol.

- Akkumulátorcsere állapota: A szimulációban az cselekvés van utánozva, mintha egy kezelőszemély vagy automata robot a drón akkumulátorait teljesen új akkumulátorokra cserélné le. A töltőállomásokon a felhasználható akkumulátorok száma végtelen. A szimulációban 40 másodperces csereidővel van számolva.
- Veszélyhelyzeti állapot: A drón azonnal a legközelebbi töltőállomás felé repül, majd landol.



2.3 ábra. A drónok állapotváltozásainak lehetséges irányai

2.3.6 A drón helyzete a szimulációban

A drón helyzete x , y koordinátákkal meghatározható, illetve lekérhető aktuális magassága is. A drón mozgása valós időben látható, viselkedése nyomon követhető a modellre implementált animációk alapján, melyből megállapítható, hogy milyen munkát végez éppen.

2.4 Az étterem

Az étterem modelljére sokkal kevesebb szerep hárul, mint a drón modelljére. Ettől függetlenül pár kikötés meg van szabva az étteremmel kapcsolatban is, hiszen a

drónok mindig innen fogják hordani az elkészített ételt a megrendelőhöz, az entitások egy valódi étteremhez hasonló működést szimulálnak.

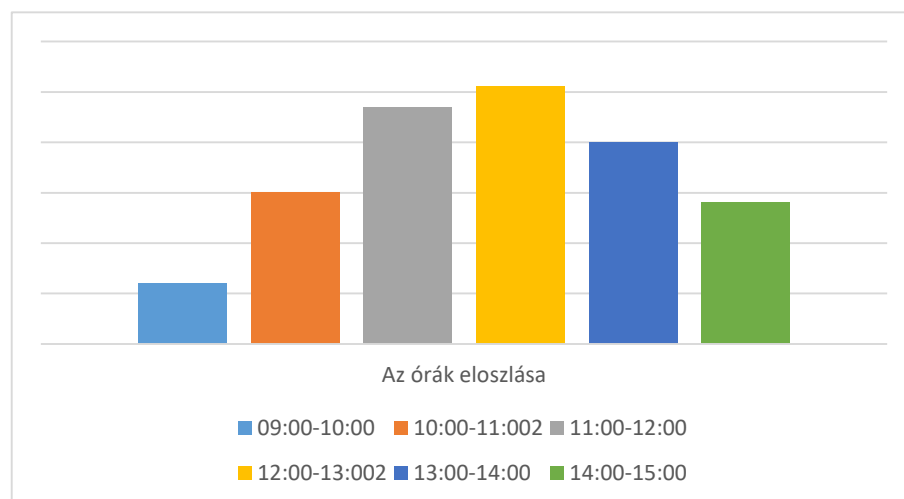
Az étterem a nála készülő ételek elkészülését lineárisan végzi az időben, minden rendelésnél, bármennyi is készül éppen ott.

Az étteremnél elképzelt leszállópálya végtelen számú férőhellyel bír, azaz a rendszernek nem kell kalkulálnia azt, hogy a drónnak lesz-e helye, ha megközelít egy éttermet. Az étteremnél az ételek bepakolását végtelen számú egység tudja végezni a drónok szállítóegységeibe, tehát egy drónnak nem kell az étteremnél semmilyen módon figyelembe vennie más drónok mozgását vagy állapotát. Az étterem helyzete konstans x és y koordináták alapján kérhető le, melyet a program futása közben nem lehet megváltoztatni. Az étterem és a nála készülő, vagy készenlévő rendelések koordinátái megegyeznek.

2.5 A rendelések

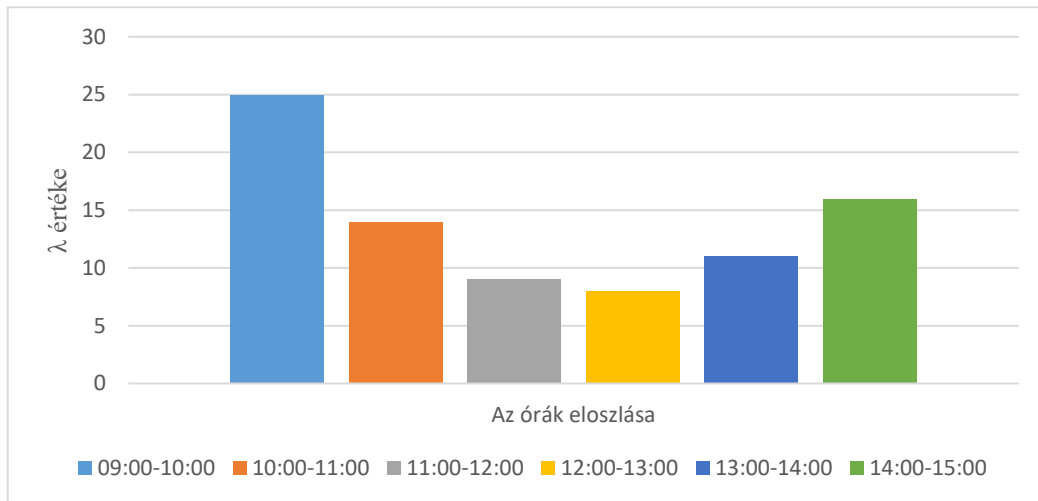
Az étterem több rendelést is fog kezelni, melyeknek saját megrendelési ID-ja és saját súlya lesz, mely adatok szükségesek a program működéséhez. A rendelések súlya a rendelt ételek összsúlyát fogja mutatni. A program a rendelésekhez randomfüggvénnyel fog hozzárendelni ételt, italt és desszertet.

A rendelések generálása *Poisson random értékek* alapján fog működni, ahol a szimulációban az λ érték óránként fog változni. A rendelések sűrűsége, követve egy átlagos 6 órás budapesti étterem hétköznapi sűrűségét, a következőképpen fog változni:



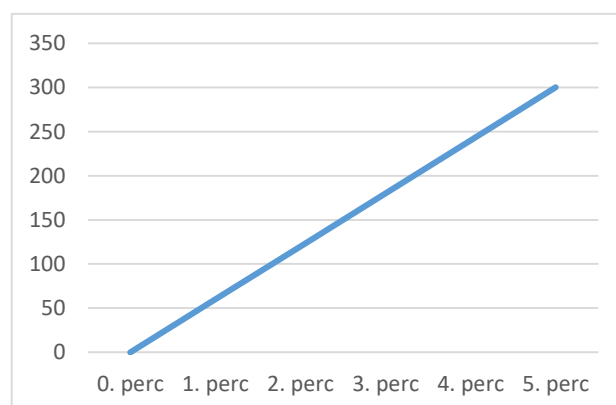
2.4 ábra. Az éttermek rendelési sűrűségének változása

A rendelés kiszállítási helyének generálásához felhasználja a szimuláció az adott budapesti kerületek környezetének jellemzőit, azaz a rendelések a kerületek méretük és lakottságuk arányában vannak kiosztva, melyeket statikus adatként adhatunk meg a szimuláció indítása előtt. A szimuláció ezzel azt a hatást igyekszik keltetni, hogy bizonyos lokációs zónákra nagyobb valószínűséggel rendelnek az emberek, mondhatni ezek „forróbb” zónák.

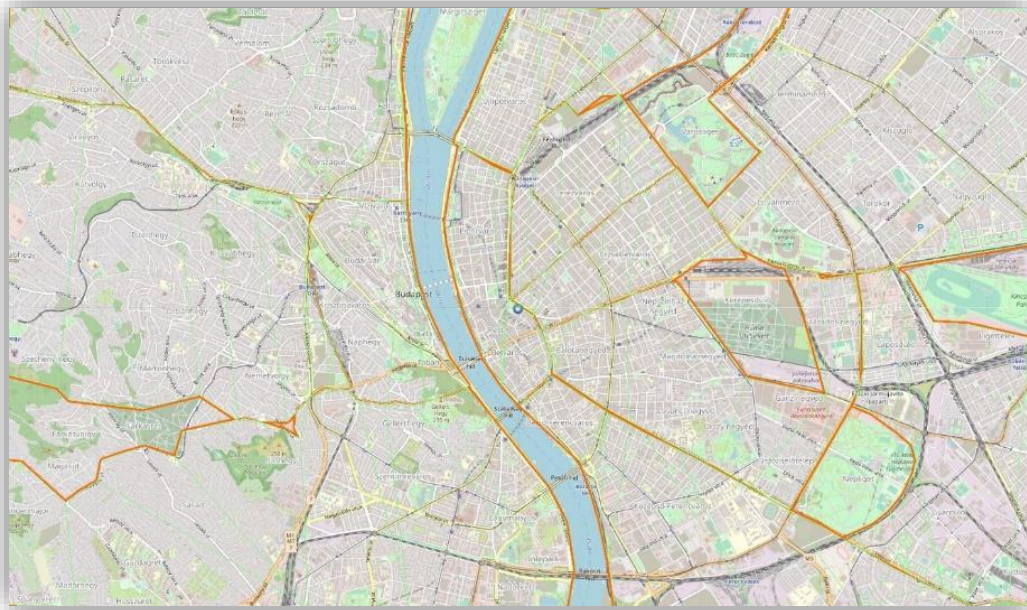


2.5 ábra. λ változása a szimuláció ideje alatt

A rendelések elkészülési ideje 10 és 20 perc közé eshet, mely véletlenszerűen dől el. A rendelésnél két időérték fog számlálódni; az egyik egy változó, mely visszaszámol a rendelés elkészültéig, amelyet a rendszer mindig meg fog vizsgálni. A másik egy olyan változó, mely azt méri, hogy mennyi idő telt el azóta, hogy a rendelés készen lett. A rendelés készenlététől kezdve addig emelkedik értéke, míg a rendelést ki nem szállítja valamelyik drón. Ez a változó fontos szereppel bír, hiszen a rendszer hatékonyságát gyakorlatilag ennek mértékével tudjuk a legjobban megmérni. A változó értéke, mely a „késést” kívánja szemléltetni, a konkrét eltelt idő alapján kalkulálódik, másodpercben megadva:



2.6 ábra. A felhasznált késés-függvény



2.7 ábra. Letiltott rendelési területek a térképen ábrázolva

Ahogy a 2.7-es ábra szemlélteti, bizonyos, nagyrészt lakatlannak ítélt területekre a program nem enged rendelést generálni, ezek az ábrán a narancssárga jelölő vonalak közé eső territóriumok.

A rendelés helyzete egészen odáig megegyezik az őt készítő étterem statikus koordinátái által megadott hellyel, amíg egy drón el nem kezdi őt szállítani. Onnantól célba érésig a drón koordinátái meg fognak egyezni a rendelés koordinátaival.

A rendelés élettartama a generálástól a célhoz való megérkezéséig tart, ezután a rendszer eltárolja a megállapított adatokat, majd megsemmisíti az objektumot.

A rendelésnél szimuláljuk a konkrét szállítmányt, mely szállítmánytípust meghatározza, hogy egy pizzakészítő franchise működését szimuláljuk. Így a bepakolt rendelés minimum egy pizzából áll (tömeg: 370-430 gramm), 50% eséllyel a szimulált vásárló kért hozzá üdítőt is (tömeg: 350-1025 gramm), illetve rendelhetett hozzá, 25%-os valószínűséggel, desszertet is (tömeg: 110-130 gramm).

2.5.1 A rendelések szimulálásának állapotai

Ahogy a drónoknál, a rendelések is felvesznek állapotokat, melyek közötti váltás egyirányú:

- Preparáció: A rendelés az étteremben tartózkodik, és még nem tudja felvenni semelyik futár.

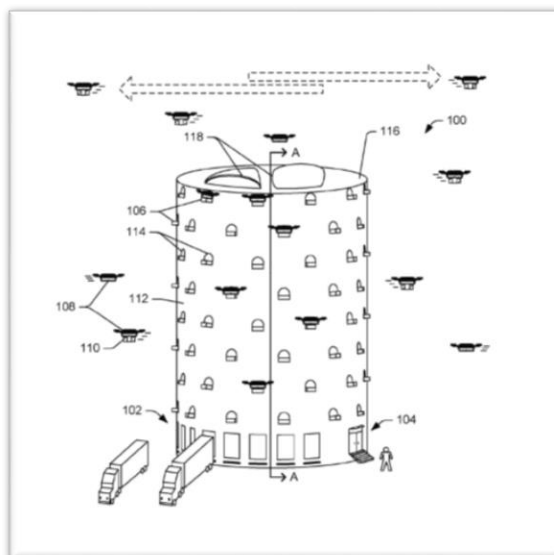
- Kész: A rendelés készen van, innentől kezdve felveheti az érte érkező vagy eddig rá váró futár.
- Kiszállítás alatt: A rendelést felvette egy futár és helyzetét innentől kezdve ő változtatja.
- Feldolgozva: A rendelés kiszállítása sikeres volt, a rendszerből nemsokára törlődni fog a rendelés.



2.8 ábra. A rendelés állapotai

2.6 A töltőállomás

Töltőállomásként egy olyan leszállási zóna lett alkotva, ahonnan a drónok elindulnak munkavégzésre, illetve ahová bármikor betérhetnek feltöltődni vagy az



2.8 ábra. Az Amazon drón-méhkas koncepciója [7]

akkumulátoraikat lecserélni. Ennek mérete képes befogadni, illetve feltölteni egyszerre az össze drónt, tehát itt nem alakulhat ki várakoztatás. Töltőállomásnak látványos elképzelése lehet az, amit az a tanulmány mutat be [7], mely méhkas-szerűen konstruál meg tervként egy drónokat és teherautókat egyaránt befogadó logisztikai központot. Jelen dolgozatban ugyan nem a töltőállomásról hordják ki a csomagokat a drónok, de maga a koncepció érdekes lehet abból a szempontból, hogyan lehet drónokat tárolni, szervizelni és kezelni kisméretű területen, városi környezetben.

Ugyan érdemes lenne elgondolni olyan problémákat és megoldásokat, melyek az akkumulátorok véges számának optimalizációjával is foglalkozik, jelen dolgozatban, ha a drónok betérnek egy töltőállomásra, akkor automatikusan 100%-os töltöttségű, "új" akkumulátort fognak kapni ahelyett, hogy elkezdenék tölteni magukat. Ez a csere a szimulációban 30 másodpercig tart.

Töltőállomásból többet is lehet létrehozni és érdemes több tesztet végezni a szimulációban, akár a töltőállomások más helyekre való helyezésével.

A kiszállítás szimulálásának lépései drón szemszögből

1. A szimuláció első lépéseként egy rendelést generálunk véletlenszerű címmel és súllyal.
2. Az étteremnél megjelenik a rendelés.
3. A címkiosztó algoritmus megkeresi azt a drónt, mellyel, az adott feltételek alapján, a leghatékányabban lehet kiszállítani a megadott rendelést.
4. A drón jelzést kap, hogy egy új rendelést kell kiszállítania és amint felszabadul az aktuális feladatából, elindul felvenni a megrendelést az étteremtől.
5. A drón az étteremhez érkezik, leszáll, felveszi az ételt vagy ételeket. Adott esetben várakozik, ha a rendelés vagy rendelések még nem készültek el.
6. A drón kiszállítja a rendelést vagy rendeléseket a rendelésben szereplő címre.
- 7a. Ha nincs több feladata, a legközelebbi töltőállomásra igyekszik. Ha új rendelést kap, megszakítja a hazatérési folyamatot és elindul újból csomagot kiszállítani.
- 7b. Ha további szállítási feladatai vannak, visszatér a 4-es ponthoz.

3 Az eszközök kiválasztása

3.1 Szimulációt futtató motorok

Ahhoz, hogy a szimulációt meg lehessen valósítani és ábrázolhatóvá tenni, szükség van egy olyan fizika-alapú játékmotorra, mely a szimuláció indításáért, futtatásáért és a folyamatos megjelenítéséért felel. Így egy olyan játékmotort, fizikát létesítő rendszer lett a fejlesztéskor alkalmazva, mely megfelelően képes megjeleníteni a projekt által felvázolt modelleket három dimenzióban és mely egy biztos környezetet biztosít a dolgozat szimulációjának számára. A talált felhasznált program rendelkezik elegendő mennyiségű és megfelelő minőségű oktatási anyaggal, mely megkönnyíti a tervezést és fejlesztést a kezdők számára is.

Az alábbi grafikus motorok kerültek mérlegelésre a választott mellett:

- **Unreal Engine 4:** Az *Unreal* szintén egy erős grafikus motor, mely magas minőségű megjelenítést biztosít. Teljesen *open-source* 2015 óta és sok olyan anyag biztosított egy kezdő számára, mellyel jól meg tudja tanulni a program használatát. Ellenben, ennek a programnak a használata nem titulálható olyan kezdőbarátnak, mint az általam választott. C++ alapú fejlesztést támogat.
- **Godot:** A *Godot* egy ingyenesen elérhető, open-source játékmotor, mely az utóbbi időben egyre népszerűbb lett a 2D-s játékokat fejlesztők körében. A *Godot* a saját, GDScript nevű nyelvét használja, de támogatja a C# és a C++ alapú fejlesztést is. A programot használó közösség viszont még nem elég nagy, illetve a *Godot* dokumentációja is hiányol dolgokat.

3.2 Unity

A *Unity* egy olyan játékmotor, amely segítségével háromdimenziós és kétdimenziós játékokat, szimulációkat lehet készíteni. Rendelkezik a megjelenítőeszközökkel és a fejlesztői felülettel, melyre szükségünk van a szimuláció megalkotásához. A *Unity* erős fejlesztői közösséggel rendelkezik, mely segíthet abban, hogy a program kezdő felhasználója is sok segítséget kaphasson a feladat közben felmerülő problémák megoldásában. A program maga is kezdő-barátnak van titulálva, emellett ingyenes is. A *Unity* C#-alapú fejlesztést támogat. Hátrányának felhozható, hogy elsősorban nem 2D játékok fejlesztésére van kitalálva.

A választás a ráesett, mely a legkiforrottabb, legnagyobb közösséggel és oktatási anyaggal rendelkező játékmotor. Érv számomra a *Unity* mellett, hogy volt

szerencsém már dolgozni vele, még ha csak keveset is, illetve, hogy C# alapú programozást biztosít, mely számomra a legkézenfekvőbb.

A Unity 2020.3.40-es verziójában van implementálva a szimuláció, mely elégnek bizonyult a számításokra és az animációk megjelenítésére egyidőben.

Továbbá rendelkezik a beépített osztályokkal és metódusokkal, mely a szimuláció megalkotásához elengedhetetlenek. Ezeket ismertetem.

3.2.1 MonoBehaviour osztály

A *MonoBehaviour* az a bázisosztály, melyből minden Unity szkript van leszármaztatva. Az osztály rendelkezik azokkal a függvényekkel, melyek előre vannak definiálva és akkor vannak végrehajtva, mikor egy esemény bekövetkezik. Ez gyorsítja és megkönnyíti a szimulációs elemek fejlesztését.

Az általam használt MonoBehaviour függvények, metódusok a következők:

- *Start()*: A függvény egyszer hívódik meg az objektum életciklusa során, annak inicializálásakor.
- *Update()*: Ez a metódus minden képalkotáskor, azaz *frame*-enként periodikusan hívódik meg. Minden dinamizmus ezzel a metódussal van reprezentálva a szimulációban, amikor változást szeretnénk előidézni az idő alapján.
- *FixedUpdate()*: A metódus minden fix időintervallumban fut le. A szimulációm fizikai számításokat tartalmaz, melyeknél fontos volt, hogy *frame*-sebesség mentes működés legyen implementálva. Továbbá, olyan esetben lett használva a programban a *FixedUpdate*, mikor egy objektum frissítése olyan mértékben volt szükséges, mint a programban megadott *frame*-enkénti 50 darabszám.
- *LateUpdate()*: Ez a metódus is minden képalkotáskor lefut, de csak az *Update* metódusok után. Jó kalkulációkra, melyeket egy *frame* végén érdemes kiszámolni. A szimuláció sebességének megváltoztatásakor van használva a programban.
- *Destroy()*: A MonoBehaviour statikus metódusai közé tartozó *Destroy* képes arra, hogy közvetlenül az *Update* futása után megszünteti egy objektum létezését. A programban a folyamatosan generálódó rendelések közül azoknak a törlésére használom, amiket már kiszállított valamelyik drón.

3.2.2 Coroutine osztály

A legtöbb esetben metódushíváskor bármilyen művelet a képkockaalkotás idején belül „kell”, hogy lefusson. Hosszú ideig tartó, bonyolult számítások esetében viszont nem előnyös számunkra, ha a metódus futása miatt a szimuláció futása szünetel. Így egy háttérben stabilan futó, arra szolgáló osztályra van szükségünk, ami egy feladatot tud elvégezni több képkockaalkotás ideje alatt és ezt nyújtja nekünk a *Coroutine*. A *'yield'* parancs szünetelteti az adott frame idejére a *Coroutine* futását, majd a program a következő frame-nél újraindítja azt.

A fejlesztett programban fontos szerep hárult ennek az osztálynak a futására, hiszen az optimalizációs kalkulációk több másodperc alatt tudnak csak eredményt szolgáltatni. A programban a metódust a *StartCoroutine()* metódus meghívásával tudom elindítani.

3.2.3 Vector2 struct és azok metódusai

Két objektum távolságát számolni és számon tartani szükséges a programban, ahhoz, hogy útvonalat tudjunk tervezni és mozgatni tudjuk a dinamikus objektumokat. A Unity beépített, 2D vektort reprezentáló *struct*-ja olyan metódusokkal segítette a fejlesztést, mint például a programban rengeteget használt *Distance()*, ami két pont távolságát tudja visszaadni, illetve a *MoveTowards()*, mely önmagában képes arra, hogy objektumot mozgasson, a megadott sebességlépés függvényében.

3.3 Programozási nyelv

A programozási nyelv kiválasztásakor döntő szerepet játszott az, hogy milyen szimulációs környezetet építő játékmotor lett kiválasztva. A Unity esetében megfelelőnek az bizonyult, hogy a kiválasztott programozási nyelv a C# legyen.

A C# az egyik legnépszerűbb programozási nyelv, amit fejlesztők és játékfejlesztők használnak. A *.NET* keretrendszer használata mellé lett kifejlesztve, mely sok fejlesztői lehetőséget nyújt. Jól használható, imperatív, általános célú, objektumorientált nyelv. Professzionális, nagy programok írására alkalmas.

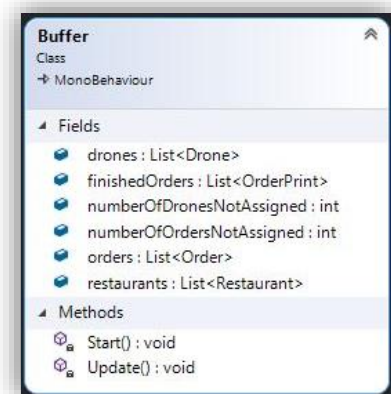
4 Tervezés

4.1 Alap-objektumok

A szimuláció helyes felépítéséhez és futásához szükségünk van egy olyan rendszerre, melyben szerepelnek a modellek mellett olyan alap-objektumok, melyek a modelleket generálják, tárolják, törlik és módosítják.

4.1.1 Buffer

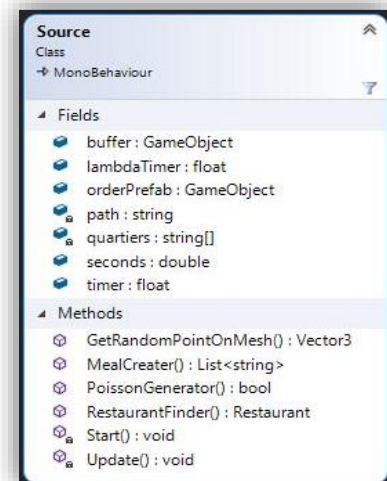
Az objektumokat fogadja és tárolja a *Buffer* objektum a programban, mely vissza tud adni a tárolt objektumok számára vonatkozó adatokat valós időben, a *numberOfOrdersNotAssigned* jelzi, hogy hány rendelés nem lett még kiosztva drónra, a *numberOfDronesNotAssigned* mutatja, hogy hány drónra nem osztott még ki a rendszer rendelést. A rendelések, drónok és éttermek példányai listában vannak tárolva.



4.1 ábra. Buffer objektum

4.1.2 Source

Az alap-objektum, mely a rendelés objektumok kibocsátását végzi. A *Source* a gyökérmappában található fájlból olvassa ki az ételek neveit és súlyait, melyeket a rendelésekhez rendel a *MealCreator()* metódus segítségével. Az osztályon belül az *Update* metóduson belül *Poisson randomszám-generátor* segítségével van döntés hozva, hogy adott szimulációbeli másodpercben jelenjen-e meg rendelés, vagy nem. A *Source* az az objektum, mely megkeresi azt az étterem objektumot is, melyben a rendelés meg fog jelenni. A *GetRandomPointOnMesh()* segítségével lehet egy adott két- vagy háromdimenziós területen belül, a program esetében a kerületek területein belül, találni egy random pontot, ahová a rendelés címezve lesz.



4.2 ábra. Source objektum

4.1.3 Sink

A Sink osztály egyetlen feladata, hogy az optimalizációs számítások után eltávolítsa azokat a rendeléseket, amiknek már aktív szerepe nincs a szimuláció futása közben.

5 Számítások és optimalizáció

A következőkben a dolgozat azokról az algoritmusokról ejt szót, amellyel a program a felvázolt problémára próbál megoldást szolgáltatni. Ismertetem a megoldásom alapját, a Genetikus Algoritmust, majd a további pszeudokódokat, melyek hozzá kapcsolódva, őt kiszolgálva, vagy az általa szolgáltatott eredményt feldolgozva működnek.

A rendeléskiosztás és ütemezés filozófiája a következő:

- egyszerre egy futár egy rendelést kaphat plusz feladatként
- a kiosztás nem időablak szerint történik, hanem rendelésmennyiség szerint
- futároktól nem veszünk el már rájuk osztott rendelést
- bizonyos futár és rendelésmennyiség alatt nem indul el a rendeléskiosztás

A rendszer ilyen működése mellett mondhatni viselkedhet ignoránsan. Gyakori tehát, hogy más megoldásokkal ellentétben itt egy rendelésre a rendszer reakciója már sok másik rendelés beérkezésével együtt fog csak megtörténni. Ez szándékolt, hiszen a megoldás célja nem az, hogy egy adott rendelésigényt azonnal a legjobb futáropcióval kiszolgáljuk, hanem egy meghatározott mennyiségű, új rendelések halmazával tervezzük tovább az egyszerre minél gyorsabb és egyszerre minél optimalizáltabb működést. A gyakorlatban ez azt jelenti, hogy „tiszteljük” a futárokat, a már szállító drónok nem fogják az eredeti, aktuális röppályájukat megváltoztatni a jelenlegi céljuk felé, de kaphatnak új feladatot az utazásukba beiktatva. A mennyiséget, mely meghatározza, hogy milyen rendelésigény esetén kezdjük kiosztani a rendeléseket a futárokra, heurisztikusan célozza meg a rendszer és az optimalizáció alrészben szó esik róla.

5.1 Változólista

A következőkben a változólista tekinthető meg, mely jelzések szerepelnek az algoritmusokban, azok leírásaiban:

DRÓNOKRA VONATKOZÓ VÁLTOZÓJELÖLÉSEK:

Állapotjelölések:

$drón^{Pihen} = 1$, ha a drón *Feltöltött, pihenő* állapotban van.

$drón^{Felvesz} = 1$, ha a drón *Légi mozgás csomagfelvétel céljából* állapotban van.

$drón^{Átvész} = 1$, ha a drón *Várakozó, átvételi* állapotban van.

$drón^{Kiszállít} = 1$, ha a drón *Légi mozgás kiszállítás céljából* állapotban van.

$drón^{Hazamegy} = 1$, ha a drón *Légi mozgás töltés céljából* állapotban van.

$drón^{Tölt} = 1$, ha a drón *Akkumulátorcsere* állapotban van.

$drón^{VeszélytJelez} = 1$, ha a drón *Veszélyhelyzeti* állapotban van.

Szállítással kapcsolatos jelölések:

$drón_{r1}$: A drón elsődlegesen kiszállítandó rendelése.

$drón_{r2}$: A drón másodlagosan kiszállítandó rendelése.

$drón_{r1\ r2}^{feladat}$: A drónnak két aktív rendelésfeladata van.

$drón_{r1\ 0}^{feladat}$: A drónnak csak egy aktív rendelésfeladata van (az elsődleges).

$drón_{0\ 0}^{feladat}$: A drónnak nincs aktív kiszállítási feladata.

$útvonaltípus_{drón,\ rendelés}^{szóló}$: A drón olyan rendeléskiosztása, melyben csak egy rendeltetést kell kiszállítania.

$útvonaltípus_{drón,\ rendelés}^{együtt}$: A drón olyan rendeléskiosztása, melyben két rendeltetést visz ki, melyeket különböző címeken, különböző időben rak le a rendelőknél. Ezeket megegyező étteremnél veszi fel, az egyik készenléte után, ha az még nincs kész, megvárja a másikat.

$útvonaltípus_{drón,\ rendelés}^{visszatérő}$: A drón olyan rendeléskiosztása, melyben két rendeltetést visz ki, de nem egyidőben veszi fel őket az étterem-/éttermeknél.

$útvonaltípus_{drón,\ rendelés}^{lehetetlen}$: A drón olyan rendeléskiosztása, melyre nincs elég energiája.

Energiafogyasztásra vonatkozó jelölések:

$v_{drón}^{MinLimit}$: Az az energiaszint-mennyiség, mely alá a drón nem engedheti csökkenni az energiáját.

$v_{drón}$: Egy drón adott pillanatban vizsgált energiaszintje

RENDELÉSEKRE VONATKOZÓ VÁLTOZÓJELÖLÉSEK:

Állapotjelölések:

$rendelés^{Preparáció} = 1$, ha a rendelés *Preparáció* állapotban van.

$rendelés^{Kész} = 1$, ha a rendelés *Kész* állapotban van.

$rendelés^{Kiszállítás} = 1$, ha a rendelés *Kiszállítás alatt* állapotban van.

$rendelés^{Feldolgozva} = 1$, ha a rendelés *Feldolgozva* állapotban van.

Szállítással kapcsolatos jelölések:

$rendelés_{drón}$: A rendelést kiszállító drón. Ha a rendelés még nem rendelkezik futárral, akkor ez értelemszerűen nullérték.

$rendelés_{étterem}$: Az étterem, mely a rendelést generálja, készítésének lokációja.

Rendeléssel kapcsolatos időjelölések:

$t_{rendelés}^{Inicializáció}$: A rendelés preparációjától számított időt mutatja az adott vizsgált pillanatban.

$t_{rendelés}^{Preparáció}$: Az a statikus időintervallum, mely a rendelés kreációja és a rendelés elkészülte között telik el.

$t_{rendelés}^{Várakozás}$: Az időmérő, mely a rendelés készenlététől eltelt időt adja meg.

$t_{rendelés}^{Kiszállítás}$: Az idő, mely alatt az étterem és a vásárló lokációja között a kiszállítás megtörténik.

$t_{rendelés}$: Az időmérő, mely a rendelés elkészülte és a vásárlóhoz való megérkezése között telt el. Legideálisabb esetben ez egyenlő $t_{rendelés}^{Kiszállítás}$ értékével.

HALMAZOKRA VONATKOZÓ JELÖLÉSEK:

D: Az összes drónt tartalmazó halmaz.

R: Az összes rendelést tartalmazó halmaz.

Minden itt nem található jelölés a következő részben lesz ismertetve.

5.2 Matematikai formulák

5.2.1 A rendeléskiosztás előkészítése

Ahhoz, hogy meghatározzuk, milyen útvonalkiosztás lehetséges egyáltalán a drónok és az aktív rendelések között, meg kell keresnünk azokat az objektumokat, melyek

az útvonalkiosztásra alkalmasak. Ezt a felmérést a *Dispatcher* objektum végzi, mely ismerettel rendelkezik D és R halmazok elemeinek állapotáról. Az aktív rendelések felmérését és listába gyűjtését végzi el a Szabad rendelések felmérése algoritmus:

1. Algoritmus: Szabad rendelések felmérése

Bemenet: R – rendelések listája, n -egész;

Kimenet: R_{kiv} – kiválogatott rendelések listája

1: függvény RendelésekKiválasztása(R – rendelések listája, n – egész)

2: $R_{kiv} \leftarrow \{\}$

3: $R_{temp} \leftarrow \{\}$

4: ciklus mind $rendelés \in R$

5: ha $rendelés_{drón} \neq \emptyset \wedge rendelés^{Feldolgozva} \neq 1$

6: ha $t_{rendelés}^{Várakozás} > 0$ akkor

7: $R_{kiv} \leftarrow rendelés$

8: különben ha

9: $R_{temp} \leftarrow rendelés$

10: elágazás vége

11: elágazás vége

12: ciklus vége

13: SORBARENDEZ($R_{kiv}, t_{rendelés}^{Várakozás}$)

14: SORBARENDEZ($R_{temp}, t_{rendelés}, t_{rendelés}^{Preparáció}$)

15: ciklus mind $rendelés \in R_{temp}$

16: $R_{kiv} \leftarrow rendelés$

17: ciklus vége

18: vissza RÉSZLISTA(R_{kiv}, n)

R: az összes rendelést tartalmazó lista

R_{kiv} : kiválogatott rendelések listája

R_{temp} : Rendeléseket tartalmazó átmeneti lista.

n: D halmaz elemeinek száma

SORBARENDEZ($R, t_{rendelés}, t_{rendelés}^{Preparáció}$): Egy olyan metódus, mely sorba rendezi a kiválasztott rendelések listáját, aszerint, hogy mennyi idő múlva lesznek készen.

SORBARENDEZ($R, t_{rendelés}^{várakozás}$): Egy olyan metódus, mely sorbarendezi a kiválasztott rendelések listáját, aszerint, hogy mennyi ideje vannak készen.

RÉSZLISTA(R_{kiv}, n): A bemenetlista n -edik elemén túli elemeket nélkülözőlistát adja vissza

Az algoritmus lényege az, hogy sorrendet állít fel a kész vagy készülő rendelések között, idő által adott fontosságuk szerint. A lista első elemei, sorba rendezve, azok a rendelések lesznek, amik már készen vannak és kiszállításuk azonnal elkezdhető lenne. Ezek után a rendelések után kerülnek be azok az elemek növekvő sorban rendezve, amik még nincsenek készen. Így egy lineáris fontosságot tudunk meghatározni a rendelések között az idő alapján. Ennek azért van jelentősége, mert az egyszerre címkiosztónak adható rendeléslista maximális hossza megegyezik a D tömb elemeinek számával:

$$|R_{kiv}| \leq |D|$$

Ennek indoklása a későbbiekben lesz kifejtve.

Adott tehát R_{kiv} , mely az általunk kiválasztott rendelések listája, amik kiszállításra várnak. A következő lépés az, hogy minden $rendelés \in R_{kiv}$ -ről megállapítsuk, hogy a számítás időpontjában mi a viszonyuk a futárokkal. Erről szükséges több információt monitoroznunk, melyeket az *ÚtInformáció* objektumok tárolnak. Egy ilyen objektum gyakorlatilag azt mondja meg pontosan, hogy ha egy adott drónra egy adott szabad, aktív rendelés kiszállítását bízánk rá, az

- milyen energiaveszteséggel járna (v^u),
- milyen időveszteséggel járna a rendelés szempontjából (t^u),
- egyáltalán kivitelezhető-e a drón állapota alapján
- ha kivitelezhető, milyen útvonaltípus alapján (*útvonaltípus*)
- a drón mennyi idő alatt képes a rendelésfelvételre ($t^{felvétel}$),

Legyen a következőkben az *ÚtInformáció* objektumok jelölése:

$$u_{drón, rendelés}, \text{ vagy egyszerűbben } u$$

Továbbá t^u jelölje azt az időt, mely aközött eltelik, míg a rendelés elkészül és a potenciális kiszállítás megtörténik.

v^u jelölje azt az energiaveszteséget, amit az út megtétele alatt fogyaszt a drón.

2. Algoritmus: Futárok kapacitásának felmérése

Bemenet: R_{kiv} – kiválogatott rendelések listája, D – futárok listája

Kimenet: RD – U objektumok listája

```
1: függvény FutárokKapacitása( $R_{kiv}$ , D)
2:   RD  $\leftarrow \emptyset$ 
3:   ciklus mind  $rendelés \in R_{kiv}$ 
4:     U  $\leftarrow \emptyset$ 
5:     ciklus mind  $drón \in D$ 
6:       u  $\leftarrow \emptyset$ 
7:       ha  $drón_{r_1 r_2}^{feladat}$ 
8:          $t^u \leftarrow \infty$ 
9:          $v^u \leftarrow \infty$ 
10:         $t^{felvétel} \leftarrow \infty$ 
11:         $útvonaltípus \leftarrow útvonaltípus_{drón, rendelés}^{lehetetlen}$ 
12:        különben
13:          u  $\leftarrow$  Útvonalszámítás( $rendelés, drón$ )
14:        elágazás vége
15:        ha  $v^u < (v_{drón} - v_{drón}^{MinLimit}) * biztonságiLimit$ 
16:          U  $\Leftarrow$  u
17:        különben
18:           $t^u \leftarrow \infty$ 
19:           $t^{felvétel} \leftarrow \infty$ 
20:           $útvonaltípus \leftarrow útvonaltípus_{drón, rendelés}^{lehetetlen}$ 
21:          U  $\Leftarrow$  u
22:        elágazás vége
23:      ciklus vége
24:      RD  $\Leftarrow$  U
25:    ciklus vége
26:  vissza RD
```

R_{kiv} : kiválogatott rendelések listája

D: futárok listája

RD: u objektumok listájának listája, melyben tároljuk minden kiosztani kívánt rendeléshez minden drón viszonyát / lehetőségeit.

u: olyan objektum, mely információt tárol egy rendelés és egy drón viszonyáról.

U: u objektumok listája

Útvonalszámítás(drón, rendelés): metódus, mely kiszámolja u objektum paramétereit.

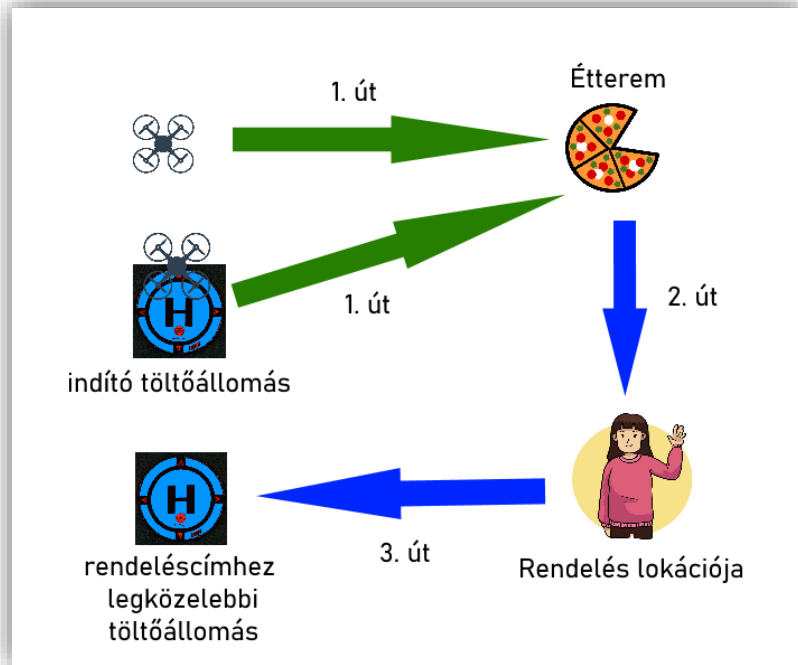
biztonságiLimit: statikus numerikus érték 0 és 1 között

Az algoritmus arra szolgál, hogy felmérje, mely drón-rendelés párosítások kivitelezhetetlenek, majd a kivitelezhetőket „megméri” az *Útvonalszámítás* algoritmussal. A ∞ érték egy olyan robusztus maximum értéket jelöl, mely az öt értékül megkapó változó átlagértékéhez képest jóval nagyobb, annak többszöröse. Ezek után, a 14. sorban láthatjuk, a drón a visszakapott adatok alapján még lemérheti egy biztonságiLimit értékkel, hogy melyik lehetséges adatokat érdemes még ezek után is elutasítani. Erre azért lehet szükség, mert az optimalizációt végző algoritmus futási ideje akár másodpercek hosszú is lehet, ezalatt pedig a drónok további energiát vesztenek. Lehetséges azt elképzelni, hogy bizonyos értékek itt még valószínűnek mutatnak egy drón-rendelés párosítást, majd az algoritmusok kiértékelése után már a drón nem képes elvinni egy rendelést, mert nincs elég energiája. Ezért dönthetünk itt bizonyos mértékű elutasítás mellett. Az algoritmus a végén egy U objektumok listát ad vissza, mely tartalmazza az összes u objektumot, így az összes drón-rendelés párosítás viszony információit.

A harmadik bemutatni kívánt algoritmus azt a feladatot végzi, hogy kiszámolja, ha az adott információk alapján elindítanánk egy drónt egy rendelésért, az milyen energiaveszteséggel járna a drón számára és mennyi idő alatt sikerülne a rendelést kiszállítani. Ez az algoritmus először megnézi, hogy a drónra hány feladat van bízva, ha két rendelés is rá van bízva, akkor nem oszt rá többet. A formulában három esetet különböztetünk meg. Ezek az esetek tudják meghatározni, hogy a drón útvonaltípusa milyen legyen. A fel- és leszállási időkkel és energiaveszteségükkel az algoritmuselemzés nem foglalkozik, ezek konstans idő- és energiaértéket képviselnek, melyek természetesen a programban kalkulálva vannak.

3. Algoritmus: Útvonalszámítás(*rendelés, drón*):

1. eset: ha $drón_0^{feladat}$, azaz a futárnak nincsen aktív feladata



5.1 ábra. 1. eset útjai és alternatívái

Ebben az esetben, mivel a címkiosztó algoritmus egyszerre egy futárra csak egy rendelést tud kiosztani, az adott drón-rendelés párosítás kimenetele csak *szóló* vagy *lehetetlen* lehet. A bemenetelnél feltételezzük, hogy a kiszámítás eshetősége valós. Az algoritmus a következőképp számol ebben az esetben:

1. út: Az első útnál két alternatívát láthatunk, melyek zölddel vannak jelölve. Egyik esetben a drón egy töltőállomáson várja, hogy a rendszer rendelést osszon rá, töltés alatt van vagy pihen. Másik esetben éppen mozgásban van, egy töltőállomás felé igyekszik, tehát a három állapot, melyben a drónt vizsgáljuk:

$$drón^{Pihen} = 1 \text{ vagy } drón^{Tölt} = 1 \text{ vagy } drón^{Hazamegy} = 1$$

Legyen tehát az első út megtett távolságának jelölése állapottól függetlenül:

$$út_1^{szóló}$$

2. út: A második út a rendelés felvétele után történik, mely közben a drón állapota:

$$drón^{Kiszállít} = 1$$

A megtett távolság:

$$út_2^{szóló}$$

3. út: A harmadik út a rendelés leadása után történik, mely közben a drón állapota:

$$drón^{Hazamegy} = 1$$

A megtett távolság:

$$út_3^{szóló}$$

Ahhoz, hogy a vizsgált drón kiszállíthassa a rendelést, a következő feltételnek kell teljesülnie:

$$v_{(út_1^{szóló} + út_2^{szóló} + út_3^{szóló})} \leq (v_{drón} - v_{drón}^{MinLimit}),$$

különben a drón-rendelés párosítás megvalósíthatatlan, azaz:

$$útvonaltípus = útvonaltípus_{drón, rendelés}^{lehetetlen}$$

A drón sebessége alapján kiszámolható, hogy az étteremhez való érkezése $t_{(út_1^{szóló})}$ alatt következhet be. Ez alapján kapjuk $t_{rendelés}^{Várakozás}$ értékét:

$$\text{ha } t_{(út_1^{szóló})} + t_{rendelés}^{Inicializáció} < t_{rendelés}^{Preparáció}, \text{ akkor}$$

$$t_{rendelés}^{Várakozás} = 0, \text{ máskülönben}$$

$$t_{rendelés}^{Várakozás} = t_{(út_1^{szóló})} + t_{rendelés}^{Inicializáció} - t_{rendelés}^{Preparáció}.$$

Innen pedig már adódik a következő:

$$t_{rendelés} = t_{rendelés}^{Várakozás} + t_{rendelés}^{Kiszállítás},$$

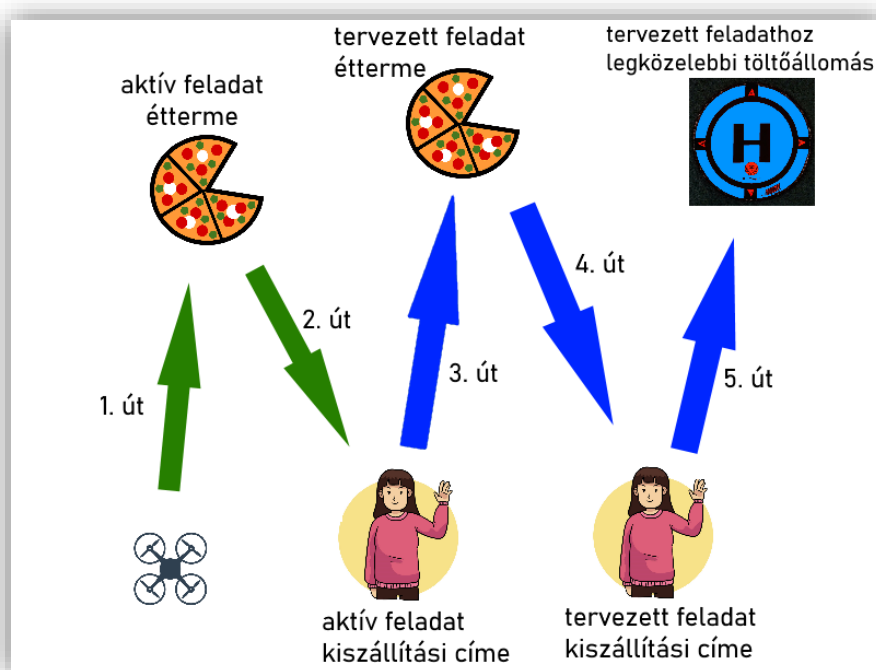
ahol $t_{rendelés}^{Kiszállítás}$ szintén a drón sebessége alapján adott.

$$\text{Továbbá, } t^{felvétel} = t_{(út_1^{szóló})}.$$

2. eset: ha $drón_{r10}^{feladat}$, azaz a futárnak egy aktív feladata van

Ez esetben a vizsgált drónnak van egy aktív feladata, melynek kiszállítását elsődlegesen el kell végeznie. Az adott drón-rendelés párosítás ebből kiindulva lehet *visszatérő* vagy *lehetetlen*. A visszatérést feltételezve itt az algoritmus azt határozza meg, hogy mennyi idő múlva lenne lehetséges a már drónra ruházott aktív rendelés

után a tervezett, bemeneti adatként megadott rendelést kiszállítani, illetve, hogy van-e a drónnak elég energiája hozzá.



5.2 ábra. 2. eset útjai és alternatívái

1./2. út: Az, hogy a drónra milyen állapotában oszt a rendszer így rendelést, az adott pillanattól függ. Erről az állapotról biztosan tudjuk, hogy az elsődleges, aktív rendelés kiszállításával függ össze, mivel $drón_{r10}^{feladat}$ ilyen állapotra utal. Ezek a következő állapotok lehetnek:

$$drón^{Felvesz} = 1 \text{ vagy } drón^{Átvesz} = 1 \text{ vagy } drón^{Kiszállít} = 1$$

A drón a potenciális feladat ráosztása esetén megteheti az 1., illetve az 1. és 2., utakat, vagy ezeknek valamely hátralevő részeit. A függvény érdeke a hátralevő távolság, és az az alatt fogyasztott energia kalkulációja, legyen az út fent említett bármelyik állapot alapján egységesen:

$$út_{1/2}^{visszatérő}$$

3. út: A feltételezett, program által tervezett rendelés kiosztása esetén az aktív rendelés kiszállítása után a futár elindul felvenni a tervezett rendelést. Jelölése:

$$út_3^{visszatérő}$$

4. út: A tervezett rendelésvétel után a rendelés kiszállítása következik. Jelölése:

$$út_4^{visszatérő}$$

5. út: A rendelés vásárlónál való leadása után a drónnak még el kell jutnia a legközelebbi detektált töltőállomáshoz is.

$$út_5^{visszatérő}$$

Csakúgy, mint a *szóló* típusú útvonalnál, ebben az esetben is feltételeknek kell teljesülnie. Ezek a következők:

$$v_{(út_{1/2}^{visszatérő} + út_3^{visszatérő} + út_4^{visszatérő} + út_5^{visszatérő})} \leq (v_{drón} - v_{drón}^{MinLimit}),$$

más esetben a drón-rendelés párosítás megvalósíthatatlan, azaz:

$$útvonaltípus = útvonaltípus_{drón, rendelés}^{lehetetlen}$$

Csakúgy, mint a *szóló* típusú útvonalnál, kiszámolható, hogy a drónnak az étteremhez való érkezése $t_{(út_{1/2}^{visszatérő} + út_3^{visszatérő})}$ alatt következhet be. Ez alapján kapjuk $t_{rendelés}^{Várakozás}$ értékét:

$$\text{ha } t_{(út_{1/2}^{visszatérő} + út_3^{visszatérő})} + t_{rendelés}^{Inicializáció} < t_{rendelés}^{Preparáció}, \text{ akkor}$$

$$t_{rendelés}^{Várakozás} = 0, \text{ máskülönben}$$

$$t_{rendelés}^{Várakozás} = t_{(út_{1/2}^{visszatérő} + út_3^{visszatérő})} + t_{rendelés}^{Inicializáció} - t_{rendelés}^{Preparáció}.$$

Továbbra is adott:

$$t_{rendelés} = t_{rendelés}^{Várakozás} + t_{rendelés}^{Kiszállítás},$$

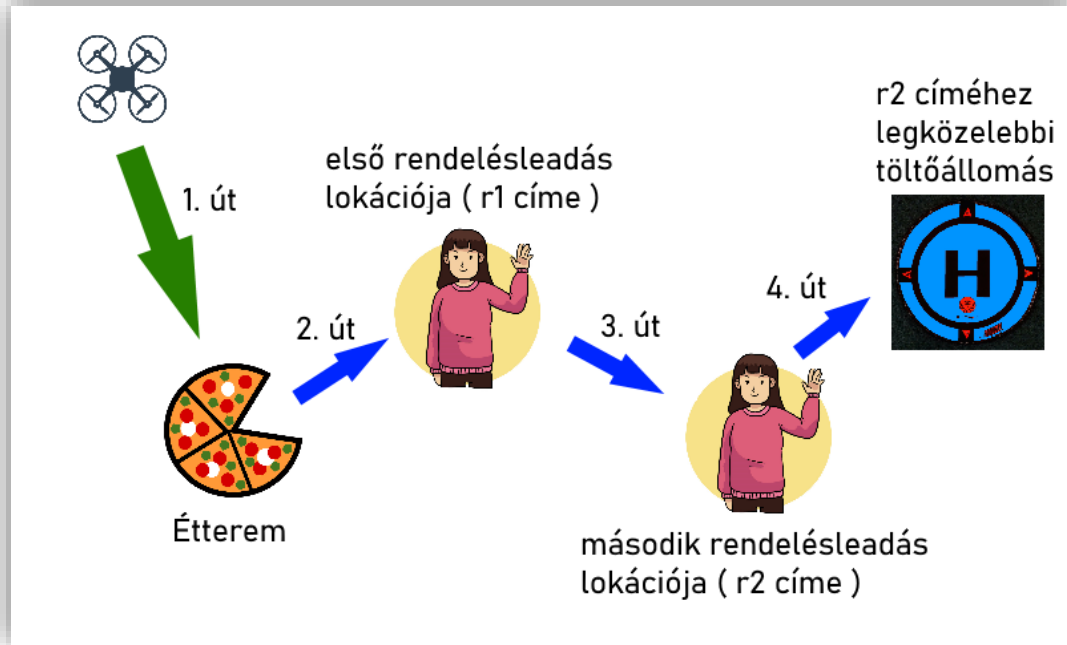
ahol $t_{rendelés}^{Kiszállítás}$ szintén a drón sebessége alapján adott.

$$\text{Továbbá, } t^{felvétel} = t_{(út_{1/2}^{visszatérő} + út_3^{visszatérő})}.$$

3. eset: ha $drón_{r10}^{feladat}$ és $rendelés_{étterem} = r1_{étterem}$, azaz a futárnak egy aktív feladata van, melyet elkészítő étterem megegyezik a függvény bemeneti értékeként megadott rendelést elkészítő étteremmel. A helyzet lényege, hogy a program azt szeretné kiszámolni, képes-e a drón egyszerre felvenni két rendelést, majd utána azokat idő- és/vagy energiahatékonyan kiszállítani. Továbbá, hogy

megtörténhessen az egyszerre való átvétel, feltételezzük, hogy $r1$ rendelés még nincs a drónnál, tehát további feltételei az esetnek, hogy a drón aktuális állapota:

$$drón^{Felvesz} = 1 \text{ vagy } drón^{Átvesz} = 1$$



5.3 ábra. 3. eset útjai és alternatívái

1. út: Ha $drón^{Felvesz} = 1$ esete áll fenn, akkor

$$út_1^{együtt} > 0,$$

ellenkező esetben az igaz, hogy $drón^{Átvesz} = 1$, vagyis

$$út_1^{együtt} = 0 .$$

2./3./4. út: Onnantól kezdve, hogy az étteremnél a drón magához veszi mindkét rendelést, döntésre kényszerül az algoritmus. El kell döntenie, hogy melyik rendelést viszi ki először és melyiket később, eszerint dől el, hogy $r1$ az eddigi aktív rendelés marad-e, vagy $r1$ $r2$ -vé válik, és helyét a bemeneti rendelés veszi át.

Döntés születhetne aszerint, hogy melyik rendelés készült el először. Emellett, a döntés alapja számítható lenne egy megadott időlimit segítségével, miszerint a limit felé esve lehetne az először elkészült rendelés kiszállítása mellett dönteni. Mivel az egész problémára való megoldás a dolgozatban globális optimum kereséseken alapszik, ezért itt most azt veszem alapul, hogy bármelyik rendelést is venné elő a kiszállításban a futár, a mindekét esetben „vesztett” időnek titulálható időtartam megegyezne:

ha $r_1 \leftarrow r_1$ és $r_2 \leftarrow \text{rendelés}$, akkor

$$t_1 = t_{r_1} + t_{r_2},$$

ha $r_2 \leftarrow r_1$ és $r_1 \leftarrow \text{rendelés}$, akkor

$$t_2 = t_{r_1} + t_{r_2}$$

$$t_1 = t_2 .$$

Így a megoldás itt arra koncentrál, hogy a drón által megtett út minél rövidebb, ezáltal minél energiatakarékosabb legyen. Ez a feltételezett két eset, r_1 és r_2 értékét illetően, külön-külön való kiszámításával kapható meg.

ha $r_1 \leftarrow r_1$ és $r_2 \leftarrow \text{rendelés}$, akkor

$$v1 = v_{(út_2^{együtt} + út_3^{együtt} + út_4^{együtt})},$$

ha $r_1 \leftarrow \text{rendelés}$ és $r_2 \leftarrow r_1$, akkor

$$v2 = v_{(út_2^{együtt} + út_3^{együtt} + út_4^{együtt})}$$

ha $v1 \leq v2$, akkor $r_1 \leftarrow r_1$ és $r_2 \leftarrow \text{rendelés}$ kiosztás mellett döntünk,

ha $v2 < v1$, akkor $r_1 \leftarrow \text{rendelés}$ és $r_2 \leftarrow r_1$ mellett.

Továbbá, itt is érvényes, hogy

$$v_{(út_1^{együtt} + út_2^{együtt} + út_3^{együtt} + út_4^{együtt})} \leq (v_{drón} - v_{drón}^{MinLimit}),$$

más esetben a drón-rendelés párosítás megvalósíthatatlan, azaz:

$$\text{útvonaltípus} = \text{útvonaltípus}_{drón, \text{rendelés}}^{\text{lehetetlen}}$$

$t_{rendelés}^{Várakozás}$ értéke, hacsak nem pont ugyanabban a pillanatban készülnek el, a két rendelésre két különböző értéket fog mutatni. Ha r_1 később készül el, az ő $t_{r_1}^{Várakozás}$ értéke kisebb lesz, mint r_2 -é és ez fordítva is igaz. Legyen a drón érkezési időpontja most $t_{érkezés}$, a lehetőségek áttekintése végett:

1. lehetőség, ha mindkét rendelés elkészül, mire a drón az étteremhez ér:

$$\text{ha } t_{(út_1^{együtt})} + t_{r_1}^{Inicializáció} > t_{r_1}^{Preparáció} \text{ és } t_{(út_1^{együtt})} + t_{r_2}^{Inicializáció} > t_{r_2}^{Preparáció},$$

akkor r_1 -re igaz: $t_{r_1}^{Várakozás} = t_{(út_1^{együtt})} + t_{r_1}^{Inicializáció} - t_{r_1}^{Preparáció}$, mint ahogy r_2 -re:

$$t_{r_2}^{Várakozás} = t_{(út_1^{együtt})} + t_{r_2}^{Inicializáció} - t_{r_2}^{Preparáció}.$$

2. lehetőség, ha csak r_1 kész mire a drón az étteremhez ér:

$$\text{ha } t_{(\text{út}_1^{\text{együtt}})} + t_{r1}^{\text{Inicializáció}} > t_{r1}^{\text{Preparáció}} \text{ és } t_{(\text{út}_1^{\text{együtt}})} + t_{r2}^{\text{Inicializáció}} < t_{r2}^{\text{Preparáció}},$$

akkor $t_{r2}^{\text{Várakozás}} = 0$, továbbá r1-re igaz:

$$t_{r1}^{\text{Várakozás}} = t_{(\text{út}_1^{\text{együtt}})} + t_{r1}^{\text{Inicializáció}} - t_{r1}^{\text{Preparáció}} + (t_{r2}^{\text{Preparáció}} - (t_{(\text{út}_1^{\text{együtt}})} + t_{r2}^{\text{Inicializáció}})).$$

3. lehetőség, az előző fordítottja, ha r_2 kész mire a drón az étteremhez ér:

$$\text{ha } t_{(\text{út}_1^{\text{együtt}})} + t_{r1}^{\text{Inicializáció}} < t_{r1}^{\text{Preparáció}} \text{ és } t_{(\text{út}_1^{\text{együtt}})} + t_{r2}^{\text{Inicializáció}} > t_{r2}^{\text{Preparáció}},$$

akkor $t_{r1}^{\text{Várakozás}} = 0$, továbbá r2-re igaz:

$$t_{r2}^{\text{Várakozás}} = t_{(\text{út}_1^{\text{együtt}})} + t_{r2}^{\text{Inicializáció}} - t_{r2}^{\text{Preparáció}} + (t_{r1}^{\text{Preparáció}} - (t_{(\text{út}_1^{\text{együtt}})} + t_{r1}^{\text{Inicializáció}})).$$

4. lehetőség, ha a drón étteremhez való érkezésekor még nincs kész egyik rendelés sem:

$$\text{ha } t_{(\text{út}_1^{\text{együtt}})} + t_{r1}^{\text{Inicializáció}} < t_{r1}^{\text{Preparáció}} \text{ és } t_{(\text{út}_1^{\text{együtt}})} + t_{r2}^{\text{Inicializáció}} < t_{r2}^{\text{Preparáció}},$$

akkor $t_{r1}^{\text{Várakozás}}$ és $t_{r2}^{\text{Várakozás}}$ értékeit az dönti el, hogy melyik készül el előbb.

4.a lehetőség:

$$\text{Ha } t_{r1}^{\text{Preparáció}} - t_{r1}^{\text{Inicializáció}} < t_{r2}^{\text{Preparáció}} - t_{r2}^{\text{Inicializáció}}, \text{ akkor}$$

akkor $t_{r2}^{\text{Várakozás}} = 0$, továbbá r1-re igaz:

$$t_{r1}^{\text{Várakozás}} = (t_{r2}^{\text{Preparáció}} - t_{r2}^{\text{Inicializáció}}) - (t_{r1}^{\text{Preparáció}} - t_{r1}^{\text{Inicializáció}}).$$

4.b lehetőség:

$$\text{Ha } t_{r1}^{\text{Preparáció}} - t_{r1}^{\text{Inicializáció}} > t_{r2}^{\text{Preparáció}} - t_{r2}^{\text{Inicializáció}}, \text{ akkor}$$

akkor $t_{r1}^{\text{Várakozás}} = 0$, továbbá r2-re igaz:

$$t_{r2}^{\text{Várakozás}} = (t_{r1}^{\text{Preparáció}} - t_{r1}^{\text{Inicializáció}}) - (t_{r2}^{\text{Preparáció}} - t_{r2}^{\text{Inicializáció}}).$$

Innen kiszámítható $t_{r1} = t_{r1}^{\text{Várakozás}} + t_{r1}^{\text{Kiszállítás}}$, és $t_{r2} = t_{r2}^{\text{Várakozás}} + t_{r2}^{\text{Kiszállítás}}$,

ahol $t_{r2}^{\text{Kiszállítás}} = t_{r1}^{\text{Kiszállítás}} + \text{út}_3^{\text{együtt}}$,

tekintve, hogy ebben az esetben két rendelést szállítunk ki, $t_{\text{rendelés}} = (t_{r1} + t_{r2}) / 2$.

$$\text{Továbbá, } t^{\text{felvétel}} = t_{(\text{út}_1^{\text{együtt}})}.$$

Azoknál az algoritmus futásoknál, ahol létrejön a 3. eset összes feltétele, ott a 2. eset feltételei is adóttak. Az algoritmus elsősorban megvizsgálja, hogy a két esetből melyik teljesíthető a drón energiaszintje alapján:

$$\begin{aligned} &\text{ha } v_{(\text{út}_1^{\text{együtt}} + \text{út}_2^{\text{együtt}} + \text{út}_3^{\text{együtt}} + \text{út}_4^{\text{együtt}})} \leq (v_{\text{drón}} - v_{\text{drón}}^{\text{MinLimit}}) \text{ és} \\ &v_{(\text{út}_{1/2}^{\text{visszatérő}} + \text{út}_3^{\text{visszatérő}} + \text{út}_4^{\text{visszatérő}} + \text{út}_5^{\text{visszatérő}})} > (v_{\text{drón}} - v_{\text{drón}}^{\text{MinLimit}}), \\ &\text{akkor útvonaltípus} = \text{útvonaltípus}_{\text{drón, rendelés}}^{\text{együtt}}. \end{aligned}$$

$$\begin{aligned} &\text{Ugyanígy, ha } v_{(\text{út}_1^{\text{együtt}} + \text{út}_2^{\text{együtt}} + \text{út}_3^{\text{együtt}} + \text{út}_4^{\text{együtt}})} > (v_{\text{drón}} - v_{\text{drón}}^{\text{MinLimit}}) \text{ és} \\ &v_{(\text{út}_{1/2}^{\text{visszatérő}} + \text{út}_3^{\text{visszatérő}} + \text{út}_4^{\text{visszatérő}} + \text{út}_5^{\text{visszatérő}})} \leq (v_{\text{drón}} - v_{\text{drón}}^{\text{MinLimit}}), \\ &\text{akkor útvonaltípus} = \text{útvonaltípus}_{\text{drón, rendelés}}^{\text{visszatérő}}. \end{aligned}$$

Abban az esetben, ha mindkét útvonaltípus lehetséges a drón energiaszintje alapján, az algoritmus megvizsgálja, hogy a 2. eset, vagy a 3. eset nyújt a két rendelés alapján adott átlagra kisebb értéket időben:

$$\begin{aligned} &\text{ha } v_{(\text{út}_1^{\text{együtt}} + \text{út}_2^{\text{együtt}} + \text{út}_3^{\text{együtt}} + \text{út}_4^{\text{együtt}})} \leq (v_{\text{drón}} - v_{\text{drón}}^{\text{MinLimit}}) \text{ és} \\ &v_{(\text{út}_{1/2}^{\text{visszatérő}} + \text{út}_3^{\text{visszatérő}} + \text{út}_4^{\text{visszatérő}} + \text{út}_5^{\text{visszatérő}})} \leq (v_{\text{drón}} - v_{\text{drón}}^{\text{MinLimit}}), \\ &\text{ha } t_{\text{rendelés}}^{\text{együtt}} \leq t_{\text{rendelés}}^{\text{visszatérő}}, \text{ akkor útvonaltípus} = \text{útvonaltípus}_{\text{drón, rendelés}}^{\text{együtt}}, \\ &\text{ha } t_{\text{rendelés}}^{\text{együtt}} > t_{\text{rendelés}}^{\text{visszatérő}}, \text{ akkor útvonaltípus} = \text{útvonaltípus}_{\text{drón, rendelés}}^{\text{visszatérő}}. \end{aligned}$$

Természetesen ez a rendszer filozófiája alapján megadott irány, energiaszint-alapú döntések esetén választhatnánk a minél kevesebb energiát fogyasztó lehetőség mellett, de az itt vázolt program minden esetben a szimulált rendelő javára dönt, azaz a legrövidebb kiszállítási időt preferálja.

Mindezek alapján az algoritmus $u_{\text{drón, rendelés}}$ ÚtInformáció objektummal tér vissza, aminek paraméterei a következők lesznek a fenti elágazások szerint:

$$v^u = v_{(\text{út}_1 + \text{út}_2 + \dots + \text{út}_n)}$$

$$t^u = t_{\text{rendelés}}$$

A fenti algoritmus elemzésére azért volt szükség, mert a programban implementált genetikus algoritmus fitness-értékeit az ÚtInformáció objektumok listája alapján tudja megkapni.

5.3 Optimalizáció

5.3.1 Optimum kereső algoritmusok

Ha egynél több lehetséges megoldás közül kell kiválasztanunk egy megoldást, előre meghatározott szempont alapján, akkor optimalizálási feladat áll előttünk. Célfüggvénynek hívjuk azt a mennyiségi változót, mely segítségével a lehetőségek közül választani vagyunk képesek. Az optimalizálás miatt lesz a rendszerünk a célfüggvény által megszabott irány szerint funkcionáló. Az a pont vagy terület, ahol a célfüggvény minimuma vagy maximuma található, a paraméterek összessége által szolgáltatott optimális megoldás. Az optimalizáló eljárásokat több módon is rendszerezhetjük. Lehetnek determinisztikus vagy sztochasztikus problémát megoldó eljárások, kísérletező vagy matematikai módszerek, statikus vagy dinamikus optimalizálások.

A dolgozatban felvázolt probléma sztochasztikus jellegű, az azt megoldó módszer igényli a dinamikusságot. Az eljárásunk nem feltétlenül szigorúan korlátozott, a szélsőértékeket a paraméterek felállítása során statikus maximum/minimum értékekre tudjuk állítani, gyakorlatilag egy büntetőfüggvénnyel. A probléma nem folyamatos értékkel leírható. Az evolúciós algoritmusok a kombinatorikus problémákra robosztus jellegükkel képesek nagy hatékonysággal válaszolni. A hegymászó algoritmusokkal szemben a genetikai algoritmusok mindig tartanak a globális szélsőértékhez, még olyan célfüggvények esetében is, ahol lokális szélsőérték található. További szempont volt az optimumkereső algoritmus kiválasztásánál, hogy egyszerűen hibridizálható legyen, aminek csak a célfüggvény értékét elég legyen megadni. A jelen felvázolt implementáció ugyan teljesen egykritériumos optimalizálást alkalmaz, csak a legrövidebb kiszállítási időt veszi figyelembe, de a genetikai algoritmusok speciális területei közé tartalmazó multikritériumos módszerekkel vegyíteni lehet a legrövidebb kiszállítási idő figyelembe vételét például a drónok fogyasztásának figyelembe vételével. A genetikai algoritmus olyan sebességgel bír, mely a feladathoz megfelelőnek bizonyult, az akár többmásodperces számítási idő a szimuláció futása közben egy ételrendelő szolgáltatásnál elfogadható.

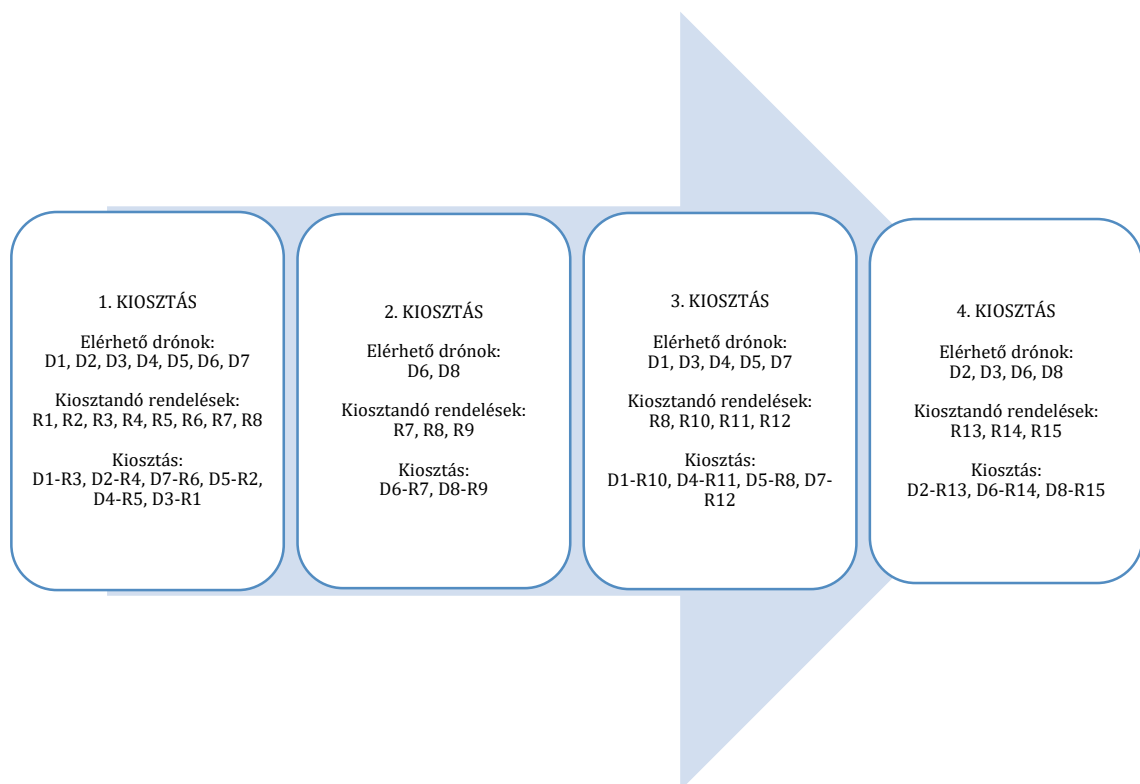
5.3.2 Genetikai algoritmus

A Genetikai Algoritmus egy olyan nemlineáris módszeren alapuló programírási- és feladatmegoldási metaheurisztika, mely az evolúcióbiológia által ihletődött és a speciális evolúciós algoritmusok kategóriájába tartozik. A módszer, mely általános

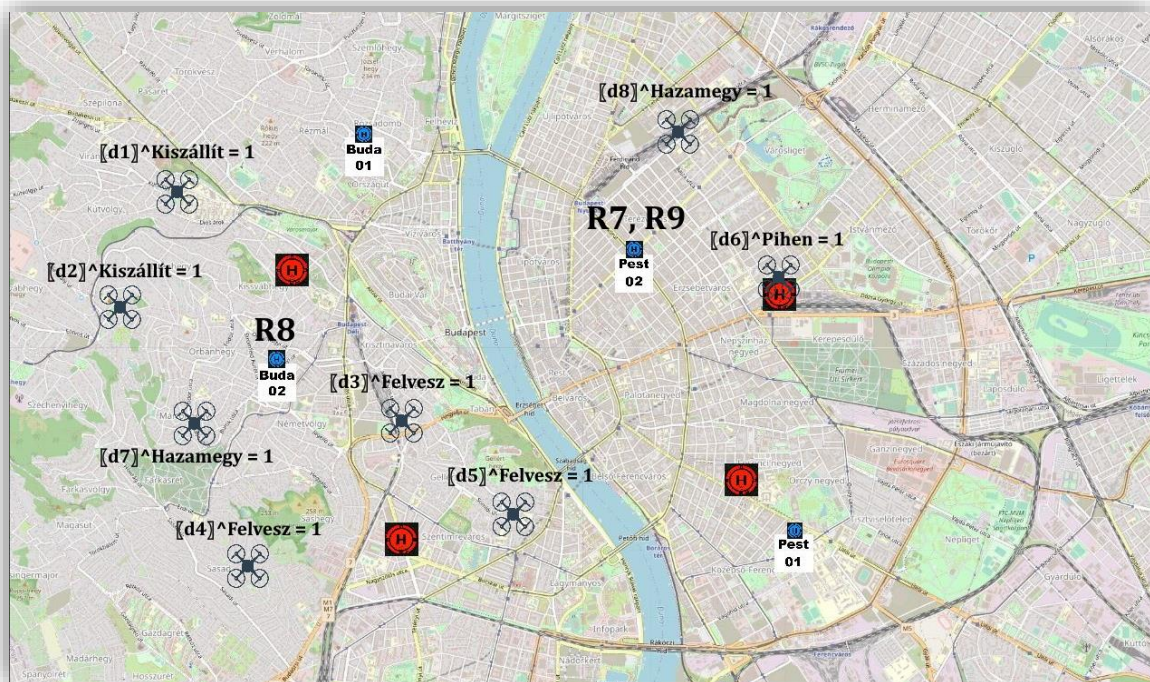
keresési terekben funkcionál, egy globális optimumot keres, célja, hogy a nagyszámú megoldás közül a legjobbat találja meg. Az algoritmus változói az úgynevezett egyedek. Ezek az egyedek alkotják az algoritmus halmazát, melyet populációnak hívunk. Ebben a halmazban az egyedek a túlélésért küzdenek. Ezt a fennmaradást, amit a valóságban egy sejt gyorsasága, erőssége jelent, az implementációban az adott feladat megoldásában való eredményessége biztosítja. A probléma variációját a megoldásban minden egyed saját kromoszómagyűjteménye, szekvenciája jelenti. Ezekkel a kromoszómákkal írható le az egyedek állapota, „jósága”.

Az algoritmus általában véletlenszerűen generálja a kezdeti populációt a rendszerünk alapelemei alapján. Ennek a populációnak a mérete a problémától függ. A módszer további részében az alapelv a természetes kiválasztódás, melynek lényege, hogy a jóság-, azaz fitnessfüggvény által meghatározott „sikeresebb” egyedek nagyobb valószínűséggel élnek tovább, vagy nagyobb valószínűséggel örökök tovább kromoszómakészletüket a következő generációnak, míg a kevésbé megfelelő egyedek kihalnak. A folyamat lényege, hogy a „jobb” egyedek aránya folyamatosan nagyobb és nagyobb lesz a populációban, ezáltal egyre jelentősebb esélyt találva a legszerencsésebb kromoszómaösszetétel megtalálására. A reprodukciós szakasz során az egyedeket véletlenszerűen párosíthatjuk egymással, mely a gének keresztezésével történhet. Ez fontos rész az algoritmusban, hiszen populációnk diverzitását tudjuk így biztosítani. Csakúgy, mint a kereszteződést, a gének mutációját is egy megadott arányértékkel tudjuk szabályozni. A mutáció lényege, hogy a kiválasztott egyedek kromoszómái változáson esnek át, ez történhet cserével, randomizációval. A populációból való tovább élésre a szelekciós algoritmus választja ki az arra érdemes elemeket. A szelekciós megoldásban szerepelhet elitlista, melynek lényege, hogy a populáció legjobb fitnessszéggel rendelkező elemeit egyenesen a következő populációba viszi át. A szelekciós algoritmus rendelkezik egy limitálási paraméterrel, mely a nem kiválasztott elemeket eltávolítja. Ez a limitálási paraméter függ a szelekciós algoritmus implementációjától, melynek egyik formája a rulettkerék-megoldás. Ennek lényege, hogy a jobb fitnessszéggel rendelkező egyedek nagyobb valószínűséggel szaporodnak és öröklődnek be a következő populációba. A nagyobb paraméterszámú optimalizációs problémák megoldására a rulettkerék-módszer alkalmas, így ezt a megoldást alkalmazza az itt bemutatott program is.

A genetikus algoritmus működési terve az, hogy minden adott aktív rendelést megpróbál egy drónhoz társítani. Ezeket a társításokat hívjuk egyben kiosztásnak. Az algoritmus azután tud lefutni, hogy az eddig említett algoritmusokkal elemezzük és felmérjük a rendszert. Ezután a genetikus algoritmus képes lehet arra, hogy egy olyan optimális kiosztást találjon, amely akár minden rendeléshez rendel futárt. Olyan kiosztás is előfordulhat, amelyben némelyik rendelésnek kiosztódik futár, de valamelyikre nem talál a rendszer. Továbbá, a rendszer kellő terheltsége mellett az is megeshet, például ha egyszerre minden futárunk pont a rendszernek nem optimális helyen tartózkodik, hogy egyik rendelésnek sem osztódik ki futár. Ez egyébként a rendszer tervezéséhez tartozik, ugyanis ha egy rendelés kiosztása nem az adott kiosztási körben történik meg, hanem körökkel később, az sok esetben nem is jelent problémát, hiszen a rendelések megjelenésük után még sokáig az elkészülési fázisban lesznek, csak a rendszernek már tudomása van róla. A genetikus algoritmus abban segít, hogy minden szituációhoz, minden adott pillanatban melyben vizsgálódunk, a sztochasztikus rendszerünkhöz képest legoptimálisabb megoldást nyújtsa nekünk. Az algoritmus egy elképzelt működése így ábrázolható az időben:



5.4 ábra. A rendeléskiosztó rendszer működése az időben



5.5 ábra. Az idővonalon szereplő 2. kiosztás előtti példaállapot

Az adott probléma egyébként egy olyan kombinatorikai feladat megoldása, melyben az ismétlés nélküli variációra ismerhetünk rá. Ennek képlete ismert:

$$\frac{n!}{(n-k)!}, \text{ ahol } k \leq n. \quad (5.1)$$

A felvázolt probléma esetén ha $|R| \leq |D|$, akkor a megfeleltetésben k jelenti $|R|$ rendelések listájának számosságát és n jelenti $|D|$ drónok listájának számosságát, de ugyanúgy igaz ha fordítva feleltetünk meg, azaz $|D| \leq |R|$, csak akkor a rendelésekből válogatunk a drónokhoz.

Az optimalizációra azért van szükség, mert például 10 drón és 20 rendelés összes lehetséges kiosztásának száma már egy elég nagy számot, $6,7 \cdot 10^{11}$ -at ad, és egy nagyobb rendszernél ennél még hosszabb számok is előfordulhatnak, melyek számítási ideje meglehetősen jelentős.

Megvalósításban a következőképp történik meg; az implementált genetikusan algoritmus egyedei maguk a kiosztások, melyeknek fitness értéke az egyed drón-rendelés párosítások, vagyis az u objektumok t^u értéke hozzáadva az adott párosítás $t^{\text{felvétel}}$ értékéhez. Előbbi minél kevesebb, annál gyorsabban kerül kiszállításra, miután kész lesz. $t^{\text{felvétel}}$ értéke hiába időintervallumot jelöl, valójában azért adódik hozzá t^u -hoz, hogy megkülönböztessen két drónt, egy messzebből és egy közelebből érkezőt, így gyakorlatilag az energia béli igények között képes viszonyítani. Egy kiosztás ábrázolása táblázatban:

	Drón Id	Rendelés Id	t^u	$t^{felvétel}$	fitness-összeg
1. párosítás	1	2	300	100	400
2. párosítás	2	3	200	120	320
3. párosítás	3	4	240	230	470
4. párosítás	4	6	310	400	710
5. párosítás	5	7	212	100	312
					összesen: 2212

Kezdetben a kiosztásokhoz véletlenszerűpárosítások generálódnak. Ezután megtörténik a fitness kiértékelése, mely alapján alkalmazódik az elitizmus, vagyis a legjobb értékű kiosztások egy része ezen a ponton elmentődik egy tömbbe.

A program ezután a fitness-értékek alapján minden kiosztáshoz valószínűség-értéket és kummulatív valószínűséget számol. Ezt úgy tudja megtenni, hogy az összes fitness értéket összeadja, majd minden kiosztás fitness értékét ezzel az összeggel elosztja és elmenti. A kummulatív valószínűséghez pedig végig iterál a kiosztások listáján és mindegyiknél hozzáadja a saját valószínűséghez az előzőleg számolt kiosztás kummulatív valószínűségét; ez lesz saját kummulatív valószínűség értéke, a lista ezen érték szerint rendezett ezen a ponton. Ezek után annyi randomszámot generálunk 0 és 1 között, ahány kiosztással dolgozunk. Minden egyes randomszámhoz megkeressük azt a kummulatív valószínűség értéket a kiosztásaink listájában, aminél kisebb értéket mutat és ahol a sorban előtte lévő kiosztás kummulatív valószínűsége a randomértékkel egyenlő vagy nála kisebb, és ezt a kiosztást örökítjük tovább a következő generációba.

A szelekció után a keresztezés következik. A keresztezés alapvetően úgy történik, hogy a kromoszómakészletet egy adott ponton elmetsszük két egyednél, majd az egyik elem megkapja a másik elem elmetszéstől való utolsó elemeit a sajátjai helyére, a másik elem pedig ugyanígy új első elemeket kap a metszéspontig. Jelen implementációnál ügyelni kell arra, hogy minden drón és minden rendelés egy kiosztásban csak egyszer szerepelhet. Ehhez minden egyes keresztezés esetén megvizsgáljuk, hogy lehetséges-e az adott elemet a neki szánt új helyre betenni. Egy keresztezéssel vizsgálva, ahol a drónok azonosítóit cserélne az algoritmus, nem lehet például végrehajtani például a következőt egyszerűen:

1. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	2	3	4	5
Rendelés ID	2	3	4	5	7

2. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	4	5	2	8
Rendelés ID	2	3	4	5	7

Adott, generált metszési pont: 3

Azaz, a kiosztásaink így néznének ki a keresztezés után:

1. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	2	3	2	8
Rendelés ID	2	3	4	5	7

2. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	4	5	4	5
Rendelés ID	2	3	4	5	7

Láthatjuk, hogy mindkét kiosztásnál van olyan drón, akire két rendelés osztódna. Ehelyett a duplikált elemek nem lesznek cserélve, és a keresztezés eredménye ez lesz:

1. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	2	3	4	8
Rendelés ID	2	3	4	5	7

2. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	4	5	2	8
Rendelés ID	2	3	4	5	7

Látható, hogy a kiosztásaink alig módosultak, ezért van lehetőségünk behozni olyan elemeket a keresztezés folyamatába, amik alapvetően nem szerepelnek egyik kiosztásban sem. Például, tegyük fel, hogy létezik még olyan drón, melyre oszthatunk rendelést. Legyen jelölve 6-os ID-val. Az első nem cserélt elemek helyére beírhatjuk minden kiosztás esetén:

1. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	2	3	6	8
Rendelés ID	2	3	4	5	7

2. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	4	5	6	8
Rendelés ID	2	3	4	5	7

Ezt megtehetjük annyiszor és annyi alapvetően nem szereplő elemmel, ahány sikertelen beillesztési próbálkozásunk volt. Továbbá, alkalmazhatunk több mutációt, ha jelentős mennyiségű sikertelen beillesztési próbálkozásunk van.

Mutáció a következőképp működik az algoritmusban; a populáció bizonyos egyedeit kijelöljük véletlenszerűen mutációra, majd ezeknek az egyedeknek egy párosításuk egyik tényezőjét cseréljük meg közöttük. A párosítás kijelölése itt is véletlenszerűen történik. Szintén ügyelnünk kell arra, hogy nem szerepelhet kétszer egy ilyen párosítás-elem kétszer egy kiosztásban. Egy mutációesetet vizsgálva, ahol a drónok azonosítóit cserélne az algoritmus, nem lehet például végrehajtani például a következőt egyszerűen:

1. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	2	3	4	5
Rendelés ID	2	3	4	5	7

2. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	4	5	2	8
Rendelés ID	2	3	4	5	7

Adott, generált mutációs hely: 3 (3. párosítások)

Azaz, a kiosztásaink így néznének ki a mutáció után:

1. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	2	5	4	5
Rendelés ID	2	3	4	5	7

2. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	4	3	2	8
Rendelés ID	2	3	4	5	7

Látható, hogy míg a 2. kiosztásban nincs drón ID ismétlődés, addig az 1. kiosztásban az 5-ös ID kétszer is szerepel. Az előzőleg, a kereszteződésnél látott korrigációt követve itt is alkalmazhatjuk más drón ID-k alkalmazását, ha a feltételek adottak. Például, tegyük fel, hogy létezik még olyan drón, melyre oszthatunk rendelést.

Legyen jelölve 6-os ID-val. Az első nem cserélt elemek helyére beírhatjuk minden kiosztás esetén:

1. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	2	6	4	5
Rendelés ID	2	3	4	5	7

2. kiosztás

	1. párosítás	2. párosítás	3. párosítás	4. párosítás	5. párosítás
Drón ID	1	4	3	2	8
Rendelés ID	2	3	4	5	7

Itt megint kiértékeljük az elemek jóságát. Még azelőtt, hogy a generációnk végére érünk, a generáció közben kivett legjobb elemeket a mutáció után tesszük vissza azoknak az elemeknek a helyére, amik ennél a kiértékelésnél a legrosszabbaknak mutatkoznak. Majd annyi generációt futtatunk még le, amennyit megadtunk a programban. A generációink futása után előáll az a kiosztás, melyet a program megpróbál felhasználni.

5.3.3 Futtatási minimum

Előfordulhat olyan eset is, hogy a körülményekhez képest viszonylag kevés rendelés generálódik, viszonylag kevés drónra, ezek ismétlés nélküli variáció verziókészletét tehát érdemes lehet brute force módszerrel kiszámolni a genetikus algoritmusos megoldással szemben. Az itt implementált alapötlet számít arra a valóságalapú bekövetkezésre, hogy a rendszer bizonyos időn belül el fogja érni azt a rendelés- és drónmennyiséget, aminél már valószínűleg érdemes genetikus algoritmust alkalmazni. Az itt bemutatott algoritmus tehát „várakozik”, nem fut le addig, amíg bizonyos számú rendelés- és drón nem várakozik kiosztásra. Mindez egy konstanssal van limitálva, melyet futtatási minimumként nevezhetünk. Ennek a számnak a megtalálása függhet a rendszer összes paraméterétől és sajátosságától. Egy kisebb érték abban segíthet, hogy a rendelések minél előbb futárt találjanak magukra, a genetikus algoritmus többször fusson le és egy esetleges szünettől minél inkább megóvja a rendszert. Viszont egy kis érték nem feltétlenül tud elég rendelőt

összehangolni; minél kisebb, annál lokálisabb optimumot talál az időben vizsgálva. Egy nagyobb limittel több rendelést hangolhatunk össze, viszont a rendszer lehet, hogy sokáig várakozik, mire elkezd a kalkulációkat futtatni. Ez akár egy rendelés idővesztésébe is kerülhet. A rendszert vizsgáló tapasztalat alapján kijelenthető, hogy csúcsideőben, mikor nagyon sok rendelés generálódik kis idő alatt, érdemes nagy számot választani, hiszen biztosak lehetünk abban, hogy rövid időn belül sok rendelés érkezik be. Ugyanígy gondolkozva, egy kevésbé zsúfolt időszakban nem érdemes nagy számot választanunk, egy kis számmal érdemesebb „kiszórni” a megjelenő rendeléseket. Ezért a rendszert értékelő teszteseteknél külön futtatásokban van bemutatva ennek a számnak a változtatásával a hálózat minősége.

Természetesen érdekes lehet egy hibrid rendszert letesztelni, melyben egyszerre működik egy genetikusan és brute force algoritmus, ahol a genetikusan a nagyszámú kiosztásokat vizsgálja, a brute force algoritmus pedig a gyorsan kiszámítható esetekben használja. Továbbá, érdekes lehet egy olyan verziót implementálni, ahol a futtatási limit a rendszer viselkedése alapján intelligensen változik.

5.4 További matematikai formulák

Jelöljük a Kiosztás objektumunk egy példányát k -val a továbbiakban. A genetikusan algoritmusunk után tehát egy k objektumot kap a Dispatcher munkavégző objektumunk, melynek párosítás elemei kapjanak $p_{rendelés, drón}$ jelölést, ahol jelölje $p_{rendelés}$ a párosítás rendelés objektumát, $p_{drón}$ pedig a párosítás drón objektumát. A Kiosztás objektumon még utoljára vizsgálni kell a rendelések és drónok párosításának feltételeit, mert azok változhattak közben, míg a genetikusan algoritmus lefutott, például egy drón lehet, hogy már nem bír annyi energiával, mint amivel a kiosztásban szereplő rendelést el tudná szállítani és még tankolni sikerülne visszaérnie.

4. Algoritmus: Végrehajthatóság Ellenőrzés

Bemenet: k – Kiosztás objektum

- 1: függvény VégrehajthatóságEllenőrzés(k)
 - 2: ciklus mind $p_{rendelés, drón} \in P$
 - 3: $u \leftarrow \text{Útvonalszámítás}(p_{rendelés}, p_{drón})$
 - 4: ha $v^u < (v_{drón} - v_{drón}^{MinLimit}) * \text{futtatásiLimit}$
 - 5: VÉGREHAJTÁS($p_{rendelés}, p_{drón}$)
-

6: elágazás vége
7: ciklus vége

k : A megvizsgálni kívánt Kiosztás objektum, mely drón-rendelés párosításokat tartalmaz

P : k objektum $p_{rendelés, drón}$ elemeit tároló tömb

VÉGREHAJTÁS($p_{rendelés}, p_{drón}$): a függvény, mely elindítja a paraméterként megadott drónt a paraméterként szintén megadott rendelés kiszállítására.

Mivel a genetikus algoritmus implementálásával programunk egy globális optimumot próbál megkeresni, ezért nem garantálható végérvényesen, hogy egy rendeléshez mindenképpen egy bizonyos időn belül futár rendelődik hozzá. Ennek kiküszöbölésére szolgálhat az 5. algoritmus, melynek futtatása bizonyos időközönként történik meg, az 1. algoritmus futása előtt.

5. Algoritmus: Régi Rendelések Kiosztása

Bemenet: R – rendelések listája

1: függvény RégiRendelésekKiosztása(R – rendelések listája)
2: $R_{kiv} \leftarrow \{\}$
3: $RD \leftarrow \{\}$
4: ciklus mind $rendelés \in R$
5: ha $rendelés_{drón} \neq \emptyset \wedge t_{rendelés}^{Várakozás} > t_{max}^{Várakozás} \wedge rendelés^{Kész} = 1$
6: $R_{kiv} \leftarrow rendelés$
7: elágazás vége
8: ciklus vége
9: SORBARENDEZCSÖKKENŐ($R_{kiv}, t_{rendelés}^{Várakozás}$)
10: $RD \leftarrow FutárokKapacitása(R_{kiv}, D)$
11: ciklus mind $U \in RD$
12: SORBARENDEZ(U, t^u)
13: $u \leftarrow U[1]$
14: ha $v^u < (v_{drón} - v_{drón}^{MinLimit}) * biztonságiLimit$
15: VÉGREHAJTÁS($u_{rendelés}, u_{drón}$)

16: elágazás vége

17: ciklus vége

R: az összes rendelést tartalmazó lista

R_{kiv} : kiválogatott rendelések listája

RD: u objektumok listájának listája, melyben tároljuk minden kiosztani kívánt rendeléshez minden drón viszonyát / lehetőségeit. (U objektumok listája)

u: olyan objektum, mely információt tárol egy rendelés és egy drón viszonyáról.

U: u objektumok listája

D: futárok listája

$t_{max}^{Várakozás}$: Az a konstans időmennyiség, melynél ha régebb óta várakozik készen a rendelés, akkor automatikusan rendelődik hozzá egy futár.

SORBARENDEZCSÖKKENŐ($R_{kiv}, t_{rendelés}^{Várakozás}$): a megadott listát rendezi csökkenő sorba a paraméter alapján.

SORBARENDEZ(U, t^u): a megadott listát rendezi sorba a paraméter alapján.

Az algoritmus gyakorlatilag egy prioritáson alapuló, azon belül pedig a leggyorsabb fuvart kiválasztó folyamat, gyakorlatilag egy First Come First Serve típusú implementáció. Először sorba rendezi azoknak a rendeléseknek a lehetőségeit szállításra, akik már $t_{max}^{Várakozás}$ időnél régebb óta várakoznak, majd a legrégebb óta várakozótól kezdve megkeresi hozzájuk a legrövidebb idő alatt lezajló fuvart.

6 Tesztelés



6.1 ábra. Képernyőkép a szimulációról annak futása közben

6.1 A genetikus algoritmus tesztelési futtatása

Az implementált genetikus algoritmus és az az előtt feladatot végző algoritmusok (1., 2. és 5. algoritmusok) futási ideje különböző beállításokkal lett tesztelve.

1. beállítás

Első beállításként a később bemutatandó alap futtatási értékeknél is szereplő számokkal:

Generációk száma: 500

Egyedek száma: 300

Keresztezési ráta: 40%

Mutációs ráta: 15%

Legjobb értékű egyedek elmentése következő generációba: 5 darab

A genetikus algoritmus előtt lefutó, annak működéséhez szükséges előkészítő algoritmusok (összesített) futási időit tartalmazó táblázat látható a következőkben, melyben az adatok másodpercben megadva szerepelnek:

		Rendelések száma						
		5	7	10	15	30	50	100
Drónok száma	10	0.00002	0.00003	0.00004	0.00004	0.00004	0.00005	0.00007
	20	0.00004	0.00004	0.00007	0.00010	0.00020	0.00020	0.00020
	40	0.00006	0.00009	0.00015	0.00020	0.00036	0.00064	0.00064
	80	0.00012	0.00017	0.00025	0.00038	0.00097	0.00121	0.00238
	120	0.00018	0.00024	0.00040	0.00054	0.00120	0.00213	0.00349
	150	0.00023	0.00035	0.00066	0.00082	0.00178	0.00277	0.00577
	200	0.00029	0.00062	0.00073	0.00088	0.00185	0.00367	0.00806

A genetikus algoritmus futási időit tartalmazó táblázatban nem minden történik olyan lineárisan, mint az első táblázatban:

		Rendelések száma						
		5	7	10	15	30	50	100
Drónok száma	10	0.112	0.138	0.300	0.300	0.300	0.310	0.300
	20	0.105	0.127	0.152	0.247	0.847	0.830	0.840
	40	0.106	0.120	0.145	0.188	0.437	2.950	2.950
	80	0.109	0.118	0.146	0.176	0.306	0.590	12.866
	120	0.106	0.122	0.140	0.179	0.299	0.515	2.688
	150	0.110	0.122	0.140	0.179	0.292	0.490	1.582
	200	0.107	0.122	0.146	0.176	0.295	0.497	1.282

Ez két okból fakad; az egyik az, hogy a rendszer maximálisan csak annyi rendeléssel kezd el kalkulálni az optimalizációs fázisban, amennyi a drónok száma annál a futásnál, tehát az első sorban ahol 15, 30, 50, 100 rendelés szerepel, ott igazából csak 10 rendeltet engedett maximálisan számolni a rendszer, így az adatok ezért stagnálhatnak. A másik ok a szabálytalanságra a szürkén jelölt adatoknál látható megugrás az előző szabályossághoz képest; ez abból fakad, hogy minél inkább a drónok számához közeli mennyiségű rendeltet próbálunk optimalizálni, a speciális keresztezési és mutációs folyamatoknál annál többet kell keresnie a rendszernek a végrehajtandó cserékhez. Ez onnan is látszik, hogy alacsonyabbra vett keresztezési (10%) és mutációs rátánál (5%) a következő adatok mutatkoztak ezeknél a pontoknál:

Rendelések száma	Drónok száma	Futási idő (mp)
10	10	0.163
30	20	0.322
50	40	0.909
100	80	1.178

Tehát akár ~90%-os javulás is elérhető ha nem fontos az algoritmust szigorú rátákkal futtatni, de az is kijelenthető, hogy a szimulációs futtatások során a drónok száma alapvetően többszöröse volt a számolni kívánt rendelések számánál.

2. beállítás

Második beállításként tehát érdekes volt megvizsgálni, hogy megnövelt generációs számnál és egyedszámnál milyen eredményt kapunk átlagosnak mondható rátaszámokkal:

Generációk száma: 1000

Egyedek száma: 500

Keresztezési ráta: 15%

Mutációs ráta: 10%

Legjobb értékű egyedek elmentése következő generációba: 5 darab

Előkészítő algoritmusok összesített futási idői másodpercben:

		Rendelések száma						
		5	7	10	15	30	50	100
Drónok száma	10	0.00002	0.00002	0.00004	0.00004	0.00004	0.00005	0.00006
	20	0.00003	0.00005	0.00007	0.00011	0.00014	0.00017	0.00016
	40	0.00006	0.00011	0.00013	0.00020	0.00038	0.00050	0.00051
	80	0.00011	0.00019	0.00026	0.00040	0.00084	0.00136	0.00226
	120	0.00021	0.00029	0.00036	0.00063	0.00109	0.00180	0.00382
	150	0.00022	0.00034	0.00044	0.00069	0.00162	0.00287	0.00494
	200	0.00035	0.00048	0.00073	0.00097	0.00212	0.00308	0.00602

A genetikus algoritmus futási idői másodpercben:

		Rendelések száma						
		5	7	10	15	30	50	100
Drónok száma	10	0.338	0.393	0.629	0.643	0.638	0.645	0.634
	20	0.332	0.376	0.444	0.615	1.356	1.411	1.395
	40	0.333	0.375	0.431	0.545	1.022	4.215	4.164
	80	0.333	0.369	0.437	0.534	0.837	1.044	16.586
	120	0.331	0.371	0.434	0.529	0.859	1.403	4.699
	150	0.336	0.376	0.438	0.529	0.845	1.320	3.320
	200	0.332	0.374	0.432	0.546	0.856	1.350	2.961

Látható, hogy az előbb indokolt szabálytalanság itt is megjelenik, és alapvetően nem sokkal több időt emésztett fel az algoritmus futása.

6.2 Alap futttás

Egy VRP megoldásokat bemutató kutatás szavait idézve: „Meglepő, de nincsen szabvány probléma definíció vagy formula a dinamikus VRP-t tekintve...továbbá, nincsen olyan tesztfeladat, amivel lemérni és összehasonlítani lehetne a bemutatott megoldások eljárásait objektív módon”.[23]

Szimulációról beszélve, a programmal készített folyamatok csak részben lesznek képesek a valóságot ábrázolni. Ebben szerepe van a véletlen eseményeknek is, melyek a valóságban is sokféle módon jelenhetnek meg. Az itt bemutatott modellt így működésében lehet megvizsgálni, és arra a kérdésre lehet választ adni, sejtésünk szerint az az elvártak szerint működik-e.

A szimuláció így különböző beállításokkal lett futtatva, a program működésének minél alaposabb vizsgálatát célba véve. Egy futtatás 6 órát vett igénybe, szimulálva egy 6 órás városi étteremhálózat kiszolgálásának működését. Az adatokat a rendszer szöveges formátumban mentette a program futása közben vagy a program futása vége után közvetlenül, az adat típusától függően.

[illegible]

62

A genetikus algoritmus beállításai a következők voltak:

egyedek száma	300
generációk száma	500
keresztelési ráta	40%
mutációs ráta	15%
elitlista bekerülési ráta	5%

A futtatásban szerepet kapó konstans értékek:

- biztonsági limit: 0.95
- futtatási limit: 0.9
- maximum várakozási limit: 15 perc
- rendelések átadási ideje (étteremnél és rendelésleadásnál): 30 másodperc
(ez az idő nem foglalja magában a süllyedést az átadás előtt és az felemelkedést az átadás után)
- drónok akkumulátorcseréjének ideje: 40 másodperc
(szintén nem tartalmazza a süllyedési és emelkedési időintervallumokat)

A Poisson-véletlenekkel generált rendelések száma: ~1600 db

A szimulációban megjelenő ikonok:



6.4 ábra. Rendelés célját
jelző rendelő



6.5 ábra. Drón objektum

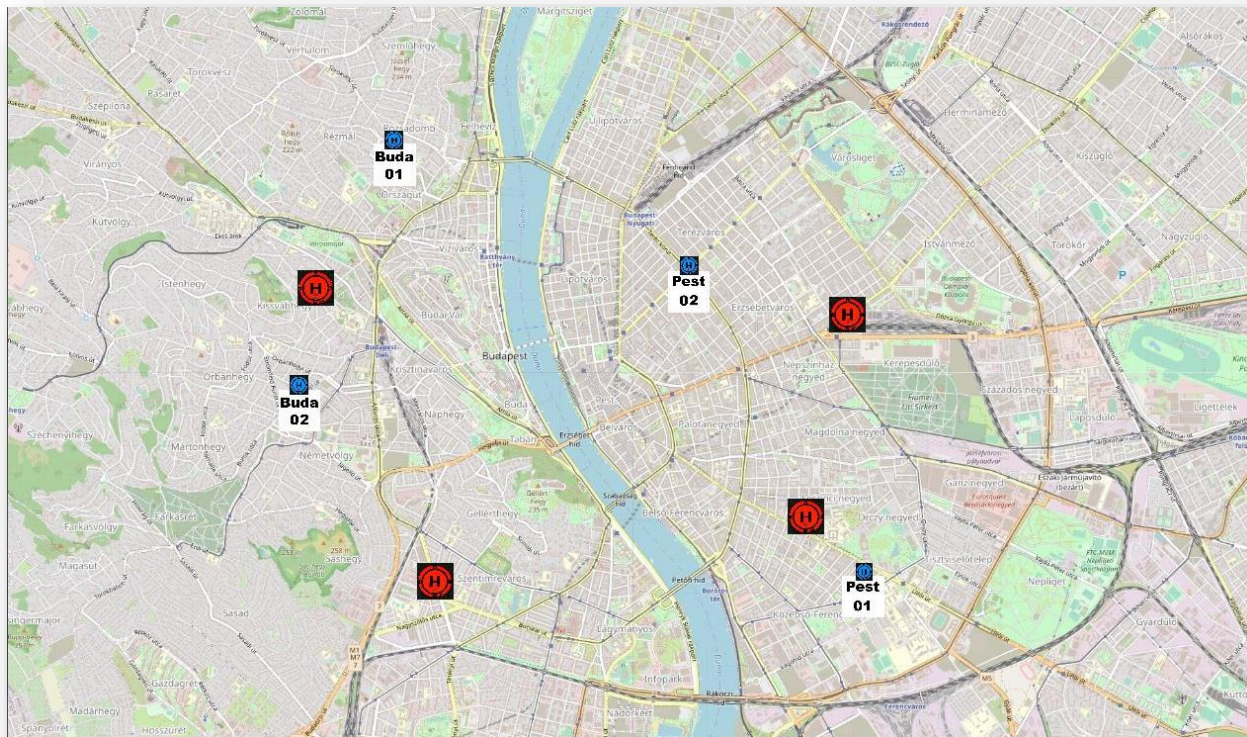


6.6 ábra. Étterem
objektum



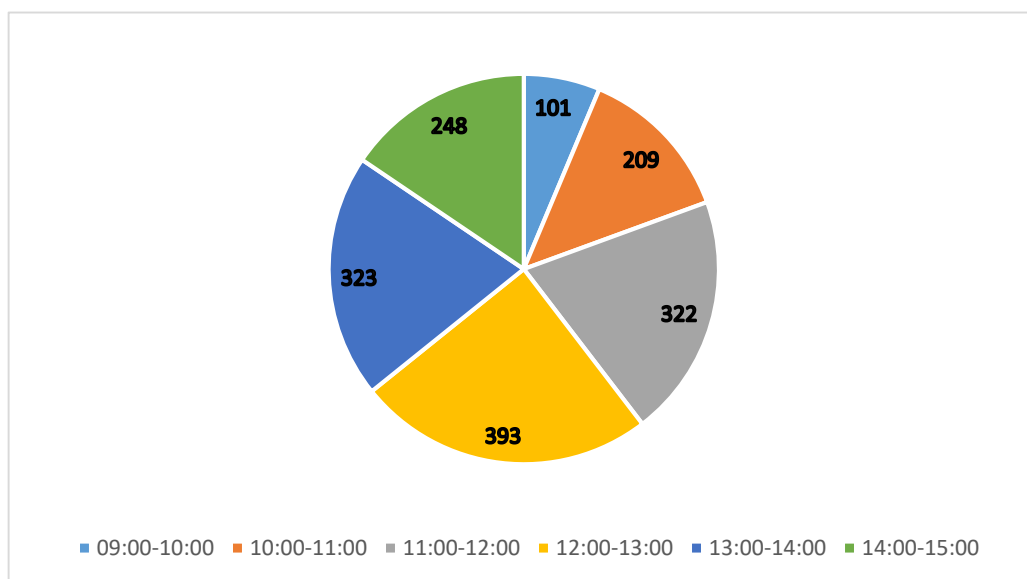
6.7 ábra. Töltőállomás
objektum

Az éttermek és töltőállomások elhelyezkedései a következő ábrán láthatók:



6.8 ábra. Éttermek és töltőállomások elhelyezkedései a térképen

Az alapvető beállításokkal prezentált futtatással bemutatható a rendelések kiszállításának eloszlása:



6.9 ábra. Rendelések kiszállítási mennyiségének eloszlása az időben

A kiszállítási mennyiséget az időben az is befolyásolja, hogy a futárok a szimuláció elején alapállapotból indulnak, illetve a rendelések is a szimuláció elejétől kezdve generálódnak, ezért például az első órában nem kell az előző órából származó rendeléseket a futároknak szállítaniuk. Ugyanezen okból látható az is, hogy hiába generálódik 10 és 11 óra között több rendelés a 14 és 15 közötti időszakhoz képest, utóbbi órában több lesz kiszállítva, hiszen a 13:00-14:00 közötti időszakból sok rendelés társul hozzá.

Az átlagos kiszállítási idő körülbelül 200 másodpercnél, vagyis alig több, mint 3 percnél van, ami kifejezetten gyorsnak mondható. Ám itt érdemes kihangsúlyozni újra, hogy ez a szimuláció nem tartalmaz rengeteg hátráltató tényezőt, sőt, nagyon optimális átvételi, akkumulátortöltési és repülési feltételekkel dolgozik. A valóságban a 3 percnél vélhetően a többszöröse lenne a megvalósult kiszállítási idő.

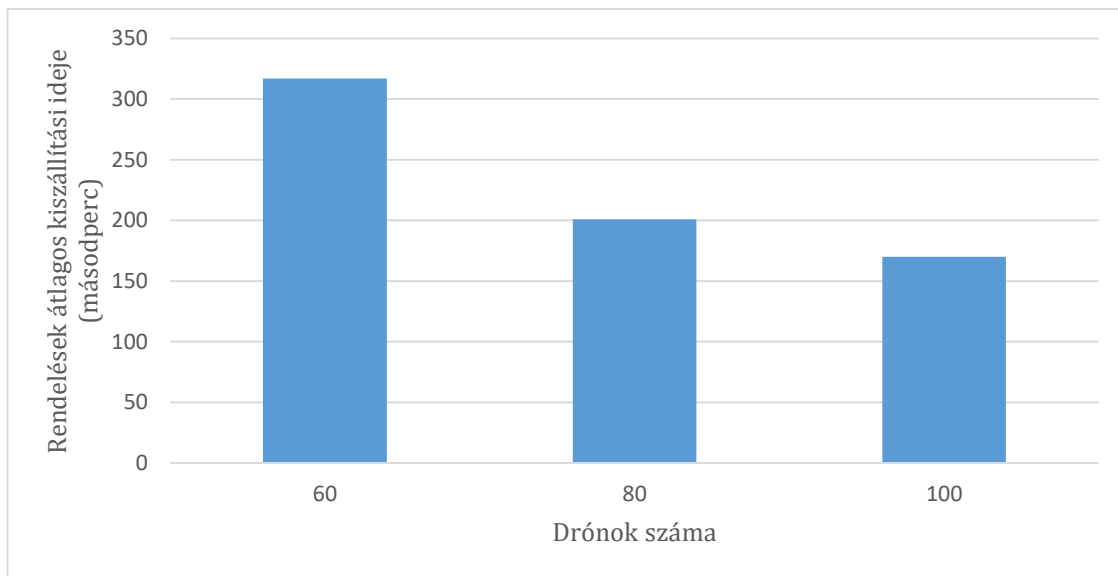
Megfigyelhető tovább a futárok elfoglaltsága az alábbi táblázatban:

Drón Id	Összes szállítás	Átlagos megmaradt energia
1	11	1792744 J
2	10	1810861 J
3	12	1862091 J
4	12	1856181 J
5	11	1809305 J
6	12	1834814 J
7	10	1758443 J
8	12	1825706 J
9	12	1800954 J
10	10	1834947 J

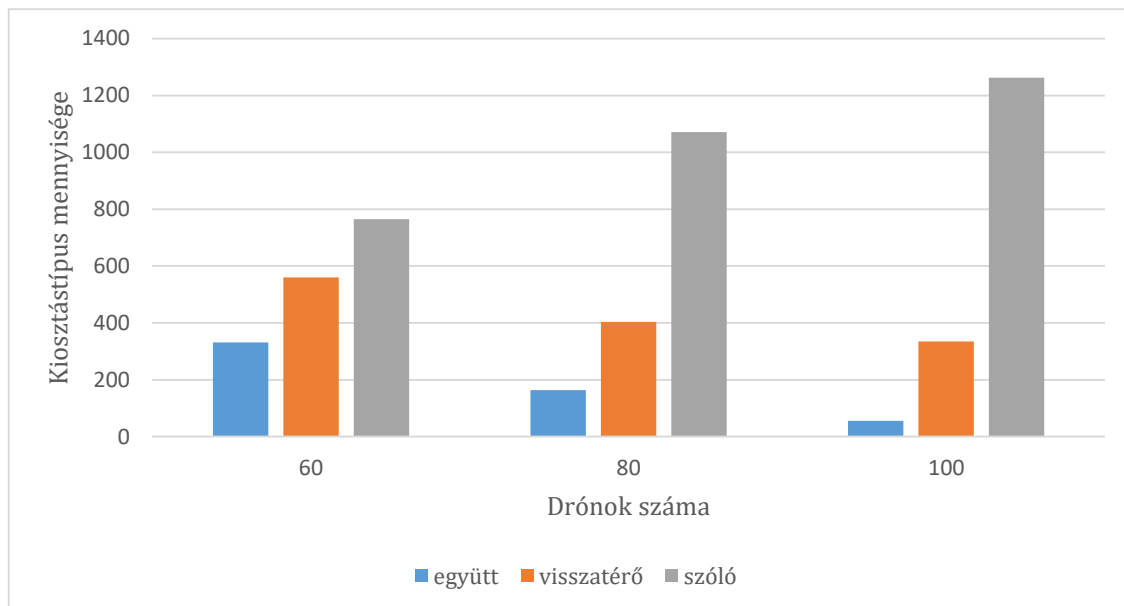
Látható, hogy a futárok közel azonos mértékben voltak elfoglalva, átlagos, akkumulátorcsere előtt mért megmaradt energiaszintjük pedig a 3.4 V ($1.743 \cdot 10^6$ J) – 3.6 V ($1.846 \cdot 10^6$ J) tartományba esik.

6.3 Futtatások különböző flottamennyiséggel

Az itt bemutatott futtatások alapján a várt eredmények megfelelnek annak, hogy a rendszer több futárral rövidebben lezajló átlagos kiszállítási időre képes, könnyebben talál egy rendelésre minél alkalmasabb futárt. A **6.10-es** ábrán látható diagramról leolvasható, hogy ilyen mértékű rendelésmegjelenés mellett a futárok képesek voltak kb. 5 perc alatt tartani a rendelések kiszállítását úgy, hogy előfordulhatott egészen távoli (5 km feletti) rendelésigény is.



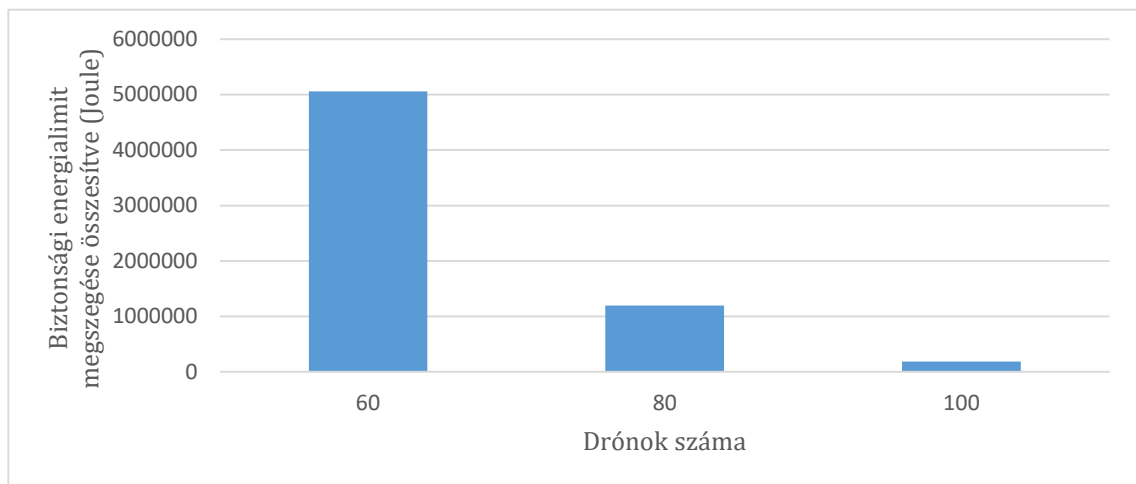
6.10 ábra. *Rendelések átlagos kiszállítási ideje különböző mennyiségű flottákkal*



6.11 ábra. *Kiosztások mennyiségei típus szerint*

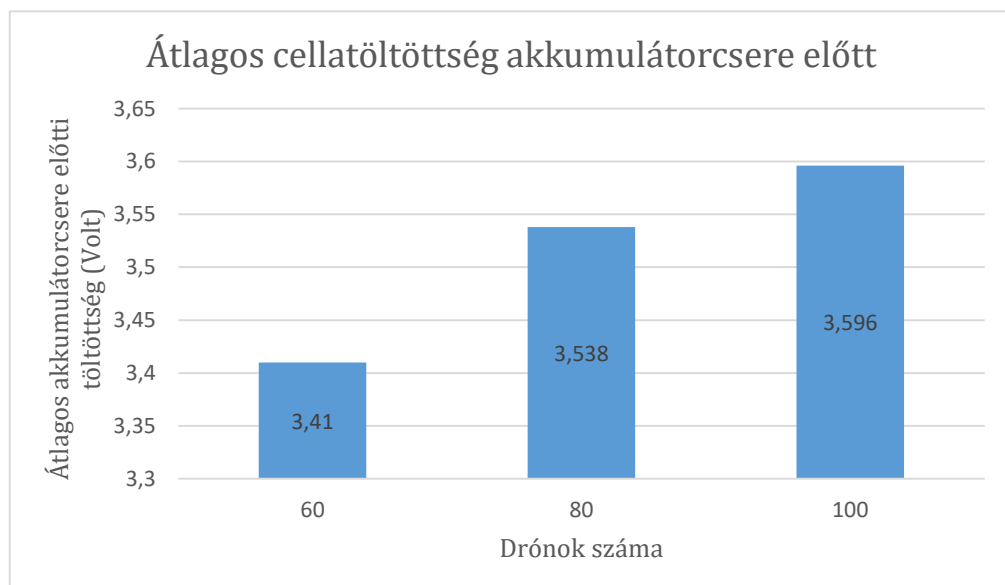
Érdekes adat lehet továbbá a rendeléskiosztások típusainak mennyiségváltozása a változások függvényében, mely a **6.11** ábrán látható.

A felfedezni vélt viselkedés abból fakadhat, hogy kisebb flottával a rendszer rákényszerül egy drónt minél tovább, minél többször használni. Egyrendeléses, szülő utak helyett sokkal több esetben fog kiosztani olyan útvonalat, mely több rendeltést érint. Eszerint várható a drónok kihasználtságának aránya is, mely a következő diagramon azt mutatja be, milyen cellatöltöttséggel tértek be a töltőállomásra a drónok akkumulátorcserére:



6.12 ábra. Biztonsági energialimit összesített megszegései

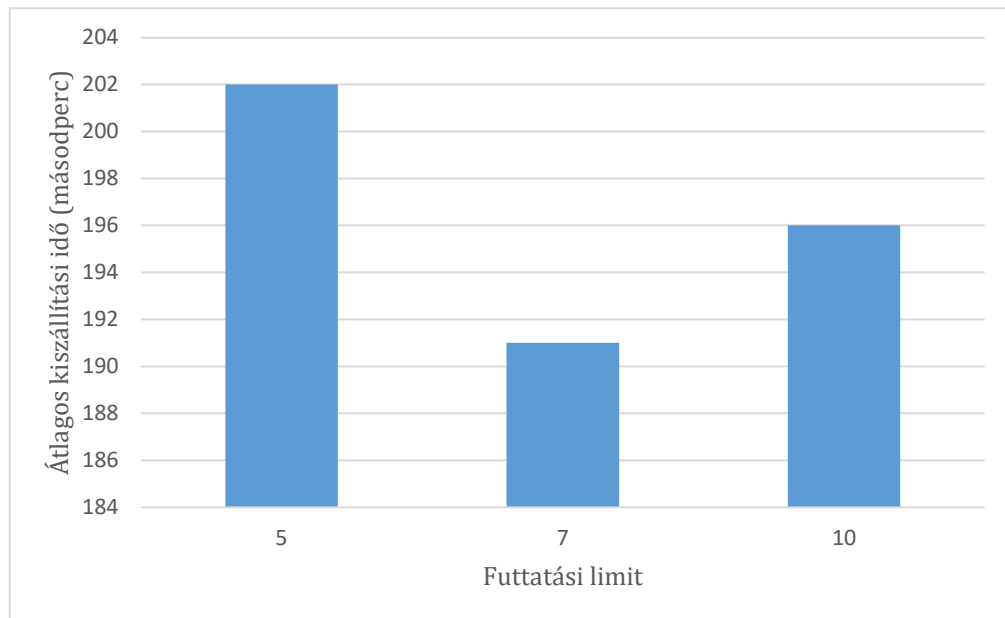
Hasonlóan, a rendszer több olyan rendelésbe „kényszeríti” bele a futárokat, amiknél akár a biztonsági limit megszegését követhetik el:



6.13 ábra. Átlagos cellatöltöttség akkumulátorcsere előtt

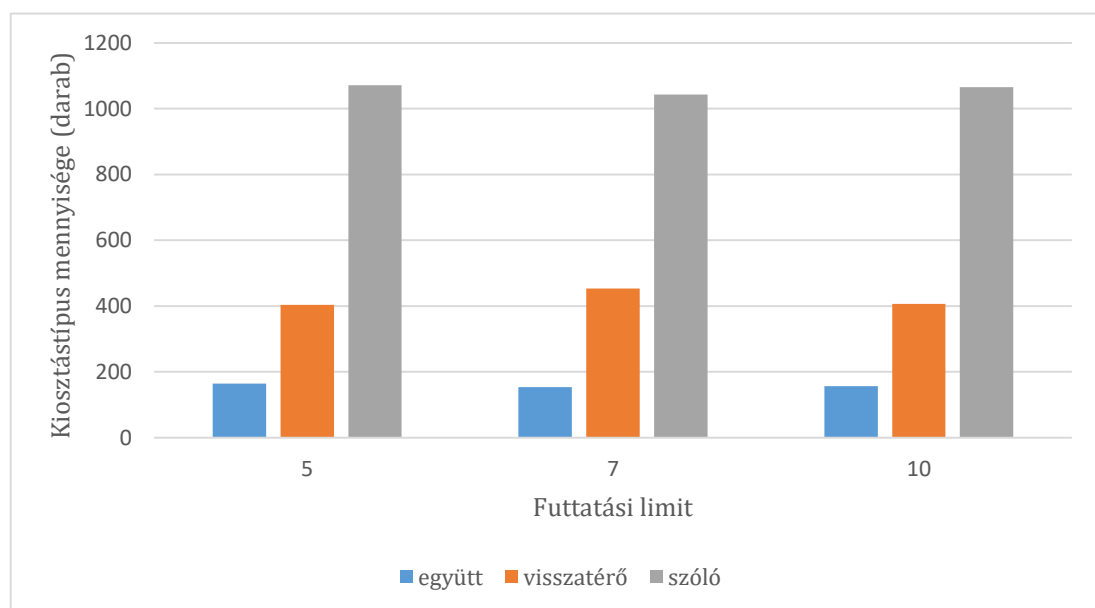
6.4 Futtatási limit változása

A már bemutatott futtatási limit felel azért, hogy az új rendelések bizonyos mértékű halmazokban optimalizálódjanak. Különböző értékeket felvéve a rendszer viselkedése a legfontosabb eredmény-érték, a kiszállítási idő függvényében így változott:



6.14 ábra. Átlagos kiszállítási idő a futtatási limit különböző értékeinél

A rendelések kiosztásának típusai között nagyjából ugyanolyan mértékű eloszlás figyelhető meg:



6.15 ábra. Kiosztások mennyiségei típus szerint

7 Értékelés

7.1 Elért eredmények

A kutatás célja kettős volt; alkotni egy szimulációt, mely felhasználható arra, hogy légi járművekkel modellezhessük ételrendelések kiszállítását, illetve egy ehhez felhasználható megfelelő algoritmus kifejlesztése, mely optimalizálja a hálózat működését. A két kitűzött célt sikerült elérni, a program viselkedése a tesztek során teljesen a vártak szerint történt. A szimulációval képes voltam olyan futtatásra, melyben a rendelések folyamatosan generálódtak, amelyeket aztán a drónok a céllokációkra vittek, így egyszerre megvalósítva akár 100 drón együtműködését és több, mint 1500 rendelés kiszállítását.

Ehhez először arra volt szükségem, hogy feltérképezzem, milyen megoldást lehet érdemes még a VRP vagy DVRP témakörben kifejleszteni. A rendelések generálását lokálisan súlyozva állítottam elő, mely reprezentálja a nagyvárosi környezet viselkedését. A szimuláció modelljeinek működését is igyekeztem minél valóságoszerűbben ábrázolni, a szimuláció sebességének megfelelő funkcionálásra bírni. Nagy figyelmet fordítottam arra, hogy a drónok energiafogyasztása kellőképpen főszerepet kapjon a modellben, így felállítva a modell fizikai részét. A rendelések kiosztásának előkészítését a saját definiált útvonaltípusokkal előállított matematikai modellel számoltam. A genetikus algoritmussal halmazokban kiosztott optimalizált rendelések egy olyan megoldás, ami önálló ötletként jelenik meg a dolgozatban. Ez egy futtatási limittel szabályozható. A tesztek alapján a kiosztások a vártak megfelelően minimalizálták a kiszállítási időt, amely elért sebesség a valóságban idealisztikusnak tűnhet.

Mindehhez egy szimulációt készítettem, mely az eredmények előállítása mellett arra is szolgál, hogy vizuálisan ábrázolja a valós idejű rendelések megjelenését, a modellek működését. A szimuláció indítási beállítása a felhasználó képzeletére van bízva; a rendszer akár többszörös sebességen is képes futni és természetesen a vázoltnál több vagy kevesebb rendelés generálására is képes.

A rendszer felállítása során saját szabályrendszert alkottam meg, melyben olyan kikötéseket tettem, amelyek nem nyújták azt az elérhető szituációs helyzetet, amit egy ilyen ételszállító hálózat teljességgel megkíván; a dolgozatban kiszállító drónok egyszerre maximum két rendelést tudnak csak szállítani, illetve egy drónhoz egyszerre csak egy rendelést képes a rendszer egy kiosztási fázisban hozzárendelni. Így kijelenthető, hogy az optimalizációs megoldásom működik, de mindenképpen

indokolt lehet a szabályrendszerének kiszélesítése, működési spektrumának bővítése.

7.2 A kutatás hatása

A szimuláció felhasználható arra, hogy az itt bemutatott optimalizációs implementációval mérjük le különböző futtatási beállításokkal egy potenciális drónflotta hatásfokát, vizsgáljuk mozgását. A szimulációs modelltér önmagában is felhasználható, melyhez hozzáilleszthető bármilyen optimalizációs vagy ütemezési megoldás. Egy esetleges ételszállító hálózat kialakítása előtt megfelelő lehet használata, hogy megállapítsuk, hogyan érdemes kialakítani reális modelljeinket, hová érdemes töltőállomást telepíteni, bizonyos rendelésigény esetén milyen terhelésre számíthat egy ilyen rendszer. A programmal könnyen kipróbálható több drónmodell működése is, így megállapítható, melyik dróntípus lehet a leghatékonyabb futár.

Továbbá, kijelenthető, hogy a rendeléskiosztó algoritmus felhasználható lehetne egy valós ételkiszállító rendszer működésében.

7.3 Fejlesztési irány

A dolgozat megírása óta felmerült az igény, hogy a benne foglalt implementáció úgy legyen továbbfejlesztve, hogy egyrészt hatékonyabban, gyorsabban működjön, másrészt a drónok bonyolultabb mozgatására is több lehetőségünk legyen, ezáltal teret adva több szituációs helyzetnek. Ezért két folyamatban lévő fejlesztési irány mutatható be, mint elképzelés a továbblépésre; az energia- és időkalkuláció szétválasztása, illetve egy újfajta útvonaltervezés, mely elveti a korábban dolgozatban vázolt rendeléskiosztási szabályokat.

7.3.1 Az energia- és időkalkuláció szétválasztása

Azt számítani, hogy egy drón mennyi idő alatt ér oda egy rendelésért és mennyi idő alatt szállítja azt ki, miközben nincs elég energiája, hogy az ehhez szükséges utakat megtegye, felesleges. Éppen ezért szükséges, hogy időbeli sorrendet állítsunk fel a két kalkulációs folyamat részére. Először azt szükséges megállapítani, hogy a tervezett kiszállítás alatt fogyasztott energia a drón rendelkezésére áll-e, és csak ezután terhelni a rendszert plusz időre vonatkozó kalkulációkkal. Amint egy tervezett kiszállításról megállapítható, hogy az általa érintett drón már „fáradt” hozzá, az egész időkalkulációs részt ki lehet hagyni.

7.4 Egyéb továbbfejlesztési lehetőségek

Az itt bemutatott program mind a szimulációs részében, mind a matematikai modelljében további potenciálokkal bír, melyek közül néhány:

- A szimuláció 3D-ben való megjelenítése, 'no-fly' zónák implementálása
- Fizikai környezet bővítése: szél szerepének hatása a drónok mozgására,
- Rendelésekiosztó algoritmus továbbfejlesztése: drónok helyezkedését előre kalkuláló komponens, mesterséges intelligencia általi megállapítása és változtatása a statikus értékekre

8 Irodalomjegyzék

1. Harry Perros. Computer Simulation Techniques. NC State University, 2009.
2. DJI Matrice 300 RTK Specification, 2021, URL:
<https://www.dji.com/hu/matrice-300/specs> , utoljára megtekintve: 2022. 12. 05.
3. Matrice 300 RTK – User Manual v1.0 – DJI, 2020.05. , URL:
<https://www.dji.com/hu/matrice-300/downloads> , utoljára megtekintve: 2022. 12. 05.
4. Zhang, J., Campbell, J. F., Sweeney, D. C., Hupman, A. C., Energy Consumption Models for Delivery Drones: A Comparison and Assessment, Transportation Research Part D: Transport and Environment, Vol. 90., 2020
5. D'Andrea, R., Guest editorial can drones deliver?. IEEE Transactions on Automation Science and Engineering, Vol. 11. Issue 3, 2014, pp. 647-648
6. Magyarország légterének térképe, URL: <https://terkep.lgter.hu/> , utoljára megtekintve: 2022. 12. 05.
7. Curlander et al., Multi level fulfillment center for unmanned aerial vehicles, 2017, URL:
<https://patentimages.storage.googleapis.com/f0/c4/d5/05500e31ddaf38/US20170175413A1.pdf> , utoljára megtekintve: 2022. 12. 05.
8. A. Larsen, O. Madsen and M. Solomon, Partially Dynamic Vehicle Routing-Models and Algorithms, The Journal of the Operational Research Society, Vol. 53, 2002, pp. 637-646
9. Lund, K., Madsen, O.B.G. ,Rygaard, J.M., Vehicle routing problems with varying degrees of dynamism, Technical report, IMM Institute of Mathematical Modelling, 1996
10. Allan Larsen, The Dynamic Vehicle Routing Problem, PhD thesis, Technical University of Denmark (DTU), 2000, ISSN 0909-3192

11. Ghiani, G., Guerriero, F., Laporte, G., Musmanno, R., Real-time vehicle routing: solution concepts, algorithms and parallel computing strategies, European Journal of Operational Research, Vol. 151, 2003, pp. 1-11
12. Andrew Percy, Profit Driven Scheduling in Last Mile Food Delivery Networks, Imperial College London, 2019 URL: <https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1819-ug-projects/PearcyA-Proft-Driven-Drone-Scheduling-in-Last-Mile-Food-Delivery-Networks.pdf>, utoljára megtekintve: 2022. 12. 05.
13. Shireen Seakhwa-King, Paul Balaji, Nicolas Trama Alvarez, and William J. Knottenbelt. Revenue-Driven Scheduling in Drone Delivery Networks with Timesensitive Service Level Agreements. VALUETOOLS 2019: Proceedings of the 12th EAI International Conference on Performance Evaluation Methodologies and Tools, 2019, pp. 183-186
14. Yanchao Liu, An optimization-driven dynamic vehicle routing algorithm for on-demand meal delivery using drones, Computers and Operations Research, Vol 111., 2019, pp. 1-20
15. Ilham Benyahia and Jean-Yves Potvin, Decision Support for Vehicle Dispatching Using Genetic Programming, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, Vol. 28., 1998, pp. 306-314
16. Shen, Y., Potvin, J.-Y., Rousseau, J.-M., Roy, S., A computer assistant for vehicle dispatching with learning capacities, Annals of Operations Research, Vol. 61, 1995, pp. 189-211
17. Bernard K.-S. Cheung, K.L. Choy, Chung-Lun Li, Wenzhong Shi, Jian Tang, Dynamic routing model and solution methods for fleet management with mobile technologies, International Journal of Production Economics, Vol 113, 2008, pp. 694-705

18. Murray, C.C., Chu, A.G., The flying sidekick traveling salesman problem: optimization of drone-assisted parcel delivery, Transportation Research Part C: Emerging Technologies, Vol. 54, 2015, pp. 86-109
19. CBSNews. Amazon Drones: Amazon Unveils Futuristic Delivery Plan - CBS News - CBS News, 2013. URL: <https://www.cbsnews.com/news/amazon-unveils-futuristic-plan-delivery-by-drone/>, utoljára megnyitva: 2022. 12. 05.
20. M. E. Dempsey, „Eyes of the Army” U.S. Army Roadmap for Unmanned Aircraft Systems, United States. Army. Unmanned Aircraft Systems. Center of Excellence, 2010
21. Sunghun Jung and Kim Hyunsu. Analysis of Amazon Prime Air UAV Delivery Service, Journal of Knowledge Information Technology and Systems , Vol. 12, 2017, pp. 253-266
22. Matthias Heutger, Dr. Markus Kückelhaus, Unmanned Aerial Vehicles in Logistics: A DHL Perspective on Implications and Use Cases for the Logistics Industry, DHL Customer Solutions & Innovations, 2014
23. Braekers, K., Ramaekers, K., Nieuwenhuyse, The vehicle routing problem: state of the art classification and review, Computers & Industrial Engineering, Vol 99., 2016, pp. 300-313

9 Ábrajegyzék

1.1 Példák VRP rendszerekre a dinamikusság és az idő/költség függvényében	8
2.1 A felhasznált terület.....	14
2.2 Lítium polimer akkumulátor merülési görbéje.....	17
2.3 A drónok állapotváltozásainak lehetséges irányai	20
2.4 Az éttermek rendelési sűrűségének változása	21
2.5 λ változása a szimuláció ideje alatt	22
2.6 A felhasznált késés-függvény	22
2.7 Letiltott rendelési területek a térképen ábrázolva.....	23
2.8 A rendelés állapotai	24
2.9 Az Amazon drón-méhkas koncepciója	24
4.1 Buffer objektum	29
4.2 Source objektum	29
5.1 1. eset útjai és alternatívái.....	38
5.2 2. eset útjai és alternatívái.....	40
5.3 3. eset útjai és alternatívái.....	42
5.4 A rendeléskiosztó rendszer működése az időben.....	48
5.5 Az idővonalon szereplő 2. kiosztás előtti példaállapot.....	49
6.1 Képernyőkép a szimulációról annak futása közben.....	58
6.2 & 6.3 A rendelések megérkezéseinek pillanatában mentett rendszeradatok	62
6.4 Rendelés célját jelző rendelő.....	63
6.5 Drón objektum.....	63
6.6 Étterem objektum.....	63
6.7 Töltőállomás objektum.....	63
6.8 Éttermek és töltőállomások elhelyezkedése a térképen	64
6.9 Rendelések kiszállítási mennyiségének eloszlása az időben.....	64
6.10 Rendelések átlagos kiszállítási ideje különböző mennyiségű flottákkal....	66

6.11 Kiosztások mennyiségei típus szerint.....	66
6.12 Biztonsági energialimit összesített megszegései.....	67
6.13 Átlagos cellatöltöttség akkumulátorcsere előtt	67
6.14 Átlagos kiszállítási idő a futtatási limit különböző értékeinél.....	68
6.15 Kiosztások mennyiségei típus szerint (futtatási limit vizsgálat).....	68