Intelligens elosztott rendszerek
BMEVIAUAC04
2022

Gál Botond - HUQ0EJ
Kalmár László Dániel - I7BEID
Pete Dávid Dorián - GVJ529

Kézbesítő drónok

Tartalomjegyzék

Feladatleírás

Bizonyos termékek kiszállítására alkalmas drón flotta megtervezése.
A termékek különböző tulajdonságokkal rendelkeznek, ami miatt nem minden drón alkalmas minden termék kiszállítására. A kiszállítási pontok különböző távolságokra vannak a raktáraktól. Ezen felül a drónok különböző távot tudnak egy töltéssel megtenni, és különböző terhet bírnak el.

Környezet:

● Egy központi raktár, az összes termék ott megtalálható, a raktár rendszerezésével nem, a drónok töltésének csak az idő igényével foglalkozunk.
● Pár (~ 2 db) regionális lerakat, csak olyan termékek találhatóak meg ott, melyeket előzőleg, egy drón a központi raktárból szállított oda
● Kiszállítási pont (falu), a rendelések célállomása, majd GUI-ról választható. kb 6-8 db.

Ágensek:

- Drónok, különböző paraméterekkel (elbírt áru mennyisége, sebesség)
    - pl. (1 kg, 2 blokk/időegység)
    - Változás az eredeti specifikáció óta: A drónoknak nincsen konkrét hatótáv paramétere. A hatótáv úgy keletkezik, hogy minden időegységben az alábbi módon csökken a töltöttségi szint, amennyiben a drón éppen nem tartózkodik valamilyen lerakaton: *szállított áru tömege/drón kapacitása + konstans*
- Árvereztető ágens

Megoldás összefoglalása

A megoldásunkban két ágens típus van megvalósítva. Az egyik egy árverező ágens, ami szétosztja a feladatokat a drónok között, a másik a drón, ami végrehajtja a kiosztott feladatokat. A drónnak viszont vannak tulajdonságai, mint kezdeti hiedelem, ami nem változik. Ez a sebesség és kapacitás. A viselkedést, hogy milyen költséggel licitál azt nagyban meghatározzák ezek a tulajdonságok. Ezeket a tulajdonságokat tetszőlegesen lehet változtatni, sok különböző kombinációval rendelkező drón ágens-t.

A drón ágens feladatai:

- Licitál az aukciós ágens által árverésre bocsátott termék kiszállítására. Ennek része az is, hogy költséget számít, amivel licitálni fog.
- A költség nem más, mint a csomag célba juttatásához szükséges idő. Figyelembe veszi azt is, hogyha még éppen aktívan dolgozik valamilyen rendelésen akkor mennyi idő után szabadul fel. Illetve ha nem elegendő a töltöttségi szintje, akkor a töltődéshez szükséges időt is beleszámolja a költségekbe.
- Megtervezi saját maga számára az útvonalat, amin el tudja vinni a csomagot a céljába, és a céljából vagy a fő raktárba, vagy egy töltőállomásra tér vissza, aszerint, hogy mire van elég energiája

Az Aukciós (auctioneer) ágens feladatai:

- A beérkező feladatokat meghirdeti a drónok számára, és az általuk érkező licit alapján eldönti kinek adja oda a feladatot.

Ágensek viselkedése

Amint egy új rendelés érkezik, a drón kiszámítja annak a költségét, hogy a rendelés beérkezése után hány időegység múlva tudja a csomagot leghamarabb célba juttatni. A költség számításakor figyelembe veszi az esetlegesen éppen teljesítés alatt álló, valamint az összes elnyert, de még sorra nem került rendelés teljesítéséhez szükséges időt. A költség számító eljárás előállít egy útvonaltervet is, amelyet a licit megnyerése esetén követnie kell, hogy az ígért költségek megfelelően véghez vigye a rendelést. A költségfüggvény, és annak lehetséges viselkedéseit részletesen kifejtettem, a '***Költségfüggvény számítása***' résznél, lejjebb.

Ezzel a költséggel licitál. Ha a licitet megnyeri, vagyis az adott drón ágensnek volt a költség ajánlata a legkevesebb, akkor az elvégzendő rendeléseket tartalmazó FIFO jellegű adatstruktúrájába menti ezt a rendelést is.
A költség számítási függvény mellékhatása, hogy előállítja az útvonaltervet is, melyet nyerés esetén végre fog hajtani.

Minden rendelést a fő raktárból lehet csak felvenni. Ezért célszerű minden esetben (amikor lehetséges) visszatérni oda, hogy a lehető leghamarabb lehessen a soron következő rendelést elszállítani.

Ágensek kapcsolata

Jelenleg csak az árvereztető és drón közötti kommunikáció szerepel a programban. Drón és egyéb drón közötti kommunikációra nem kerül sor. Ez azért van így, mert az elkészült megoldásban nincs egyetlen falu sem olyan távolságra, hogy egy drón magában ne lenne képes egy rendelést teljesíteni, és így nincs szükség arra, hogy a drónok egymással kommunikáljanak illetve szövetségeket alakítsanak ki.

Egy alternatív megoldás az lenne, ha a drónok magukban nem feltétlenül lennének képesek egy rendelést teljesíteni, hogy egymással kommunikálnak és szövetségeket alakítanak ki annak érdekében, hogy közösen teljesíteni tudják a rendelést. Mi a tervezés folyamán többek között például arra gondoltunk, hogy a drónok ilyen esetben maguk is egyfajta árverést indítanának, és a nyertessel alkotnának csapatot, ami a rendelést teljesíti.

A megoldásunkban valójában az árvereztető sem ismeri a drónokat, azokkal pusztán broadcast üzenetekkel kommunikál, melyre a drónok az üzenet feladóját ismerve képesek válaszolni, azaz licitálni.

Az árverés, MiniSum

A drónok képesek költséget számítani akkor is, ha éppen rendeléseket szállítanak ki, és még van vissza nem teljesített, beütemezett rendelésük. Ez esetben azok időköltségét is hozzáadják az újonnan beérkezett rendelések számított időköltségéhez.

Erre azért van szükség, mivel nem feltétlen az az optimális, ha az a drón nyer, akinek éppen nincsen feladata, és rendelésre vár a raktárnál.
Hiszen, ha az a drón lassú, akkor lehet, hogy jobban járunk abban az esetben, hogyha megvárjuk amíg egy gyors drón befejezi az előző rendelésének kiszállítását, visszatér a raktárba, tölt ha kell és úgy viszi ki az új rendelést.

Mivel mindegyik ágens úgy licitál, hogy a lehetőségeihez mérten a lehető legkisebb teljesíthető idő követelményt adja meg, így a rendelések összesített idő követelményét is minimalizáljuk, így a MiniSum-ot követi a megoldásunk.

Alternatív (optimálisabb) megoldás lehetőségek

Alapvetően arra is gondoltunk, hogy ha a töltőállomások lerakatok, akkor a drónok tudnak ott csomagot cserélni. Ebben az esetben MiniSum esetében lehetséges egy optimálisabb megoldás (aminek az implementálásáig nem jutottunk el, de elméletben gondolkodtunk rajta.)

Ha egy drón számításba venné azt, hogy mi történne, ha elvinné a raktárból a csomagot az egyik lerakatba, majd onnan egy másik drón tovább vinné a célállomásra, és az így keletkezett időköltség kedvezőbb mint az, hogyha csak saját maga szállítja akkor az így keletkezett költséggel licitál, és majd a rendelést a két drón összefogva teljesíti.

Ehhez az kellene, hogy mikor egy ágens költség függvényt számít futtasson egy ‘körbekérdező’ protokollt, amivel megtudja, hogy a lerakattól ki milyen költségért tudná tovább vinni a csomagot. Ez a protokoll lehet akár vállalkozási hálók, vagy a fentebb használt

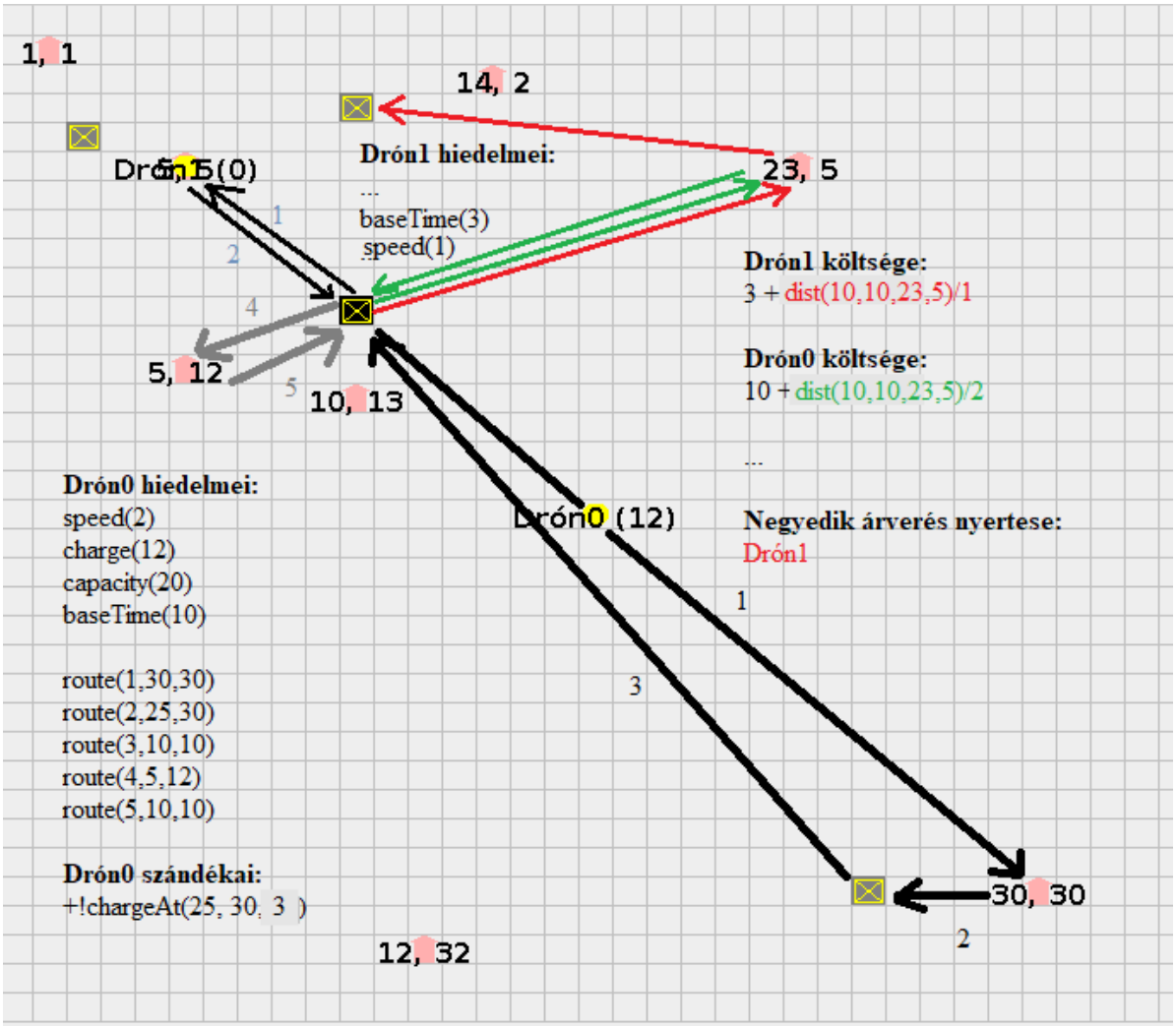
egyszerű árveréses módszer is, úgy, hogy az árvereztető szerepét felvenné a (drón) ágens. Ilyenkor, a megnyert licit költségét hozzáadná a saját, a lerakatig tartó útjának időköltségéhez, és ezzel az összeggel licitálna. És ha így nyerne, akkor szövetséget kötne azzal a drónnal akinek az ajánlatát használta, és együtt teljesítenék a rendelést.
Valószínűleg ezzel a módszerrel, MiniSum szempontjából jobb eredményt érnénk el, mivel sokszor lehetséges az, hogy egy lassú drón pont akkorra tudná a lerakathoz szállítani a csomagot, mikorra egy gyors drón éppen visszatér oda, és tudná tovább szállítani.

Ennek a megoldásnak a felfelé skálázhatósága azonban kérdéses, mivel ha minden drón minen másikat körbekérdez, akár több lerakat szempontjából az már minden árverésnél $d*n^2$-es futási idő (ahol a d a lerakatok száma). Azonban, ha ez a körbekérdezés ráadásul rekurzívan menne tovább (valamilyen max mélységig), tehát nem csak 2 ágensből álló kiszállító csapatok lennének, akkor ebben az esetben a futási idő akár $d*n^{1+m}$-is lehetne (ahol az m a maximális rekurziós mélység, legalább 1), amely nem fenntartható, nagyobb környezet és jóval több ágens esetén.

Erre egy optimálisabb lehetőség, heurisztika az, hogy az ágensek tudásbázisát bővítjük azzal, hogy mi a többi ágens pozíciója (és esetleg azzal, hogy mennyi idő után szabadul fel az elvállalt rendelésekből) és ezen információk alapján lehetséges lenne meghatározott méretű klasztereket alkotni, és az ágens csak azon ágenseket kérné meg, hogy mondjanak ajánlatot, akik kellően közel vannak (és kellően rövid idő után szabadulnak). Így, akár a konstans felső határral megadott klaszter méret miatt a futási idő így csökken jelentősen.

Belegondolva, ez a működés nagy részben ekvivalens azzal a valós feladattal, hogy hogyan lehet minimalizálni az utazási időt A és B között, ha országos, regionális és helyi tömegközlekedést, átszállást is lehet használni, és a különböző távolságot megtevő buszok sebessége különbözik, hasonlóan mint a drónoknál, és a lerakatok az állomásoknak, vagyis átszállási helyeknek felelnének meg.

Összefoglaló ábra



1. Ábra: Egy példa szituáció. Negyedik árverés költség számítása

Az ábrán látható, hogy a 4. árverés költség számítása, és licitálása hogyan történik. Az alapfelállás a következő:

A *Drón0* aktuálisan teljesít egy rendelést, a (30,30)-ba, majd utána töltődni tervez a közeli állomáson, 3 időegységig (Mivel a 12 töltöttségi szint nem lett volna elég visszajutni a raktárba.)
Továbbá, be van tervezve (mivel korábban elnyerte) az (5,12)-be is van egy rendelése (szürke nyilak, a 4-es és az 5-ös ***route*** hiedelemek között). A ***baseTime*** azt az időköltséget mutatja, ami ahhoz kell, hogy az összes betervezett **route-**ot véghezvigye, a töltődés 3 időegységét is beleszámítva.

A *Drón1* éppen most kézbesítette a rendelést az (5,5)-be, és indul vissza a raktárba. 3 időegység alatt tér vissza oda (mivel a ***baseTime*** annyi) és nincs további betervezett rendelése.

Tehát, amint a 4. árverés elkezdődött, a (23,5)-be kell csomagot szállítani (súlyát az egyszerűség kedvéért nem jelenítettük az ábrán meg).
A fent részletezett költségfüggvény alapján a *Drón1* esetében a piros nyílakat követné, a kézbesítés után a töltőállomáson töltődne, mivel az közelebb van, és csak annyi energiájába kerül eljutni. Viszont licitálni csak azzal az idő költséggel fog, ami a (23,5)-be eljutáshoz kell (+***baseTime***). A *Drón1* lassabb (***speed(1)***) mint a *Drón0 (**speed(2)**)*, de meg fogja nyerni az árverést, mivel a *Drón0* hiába gyorsabb, a ***baseTime*** miatt a *Drón1*-nek a költsége kevesebb lesz.

Fejlesztés összefoglalása

A feladat elkészítéséhez a Jason 3.1-es verzióját használtuk. Mind a kettő ágensünkhöz tartozik ASL-ben és Java-ban írt rész is.

Az árvereztető esetében például az ASL-es részben az ágens céljai (például árverés indítása), és a licitek fogadására szolgáló esemény találhatóak. A Java-s részben ezzel szemben az árverés indításához megírt grafikus interfész található.

A drón esetében az információ túlnyomó részét az ASL-ben tároljuk (*drone.asl*) hiedelmekként. Így például, amikor az árvereztető árverést indít, akkor az arra kiszámított ideális útvonalat, illetve költséget a drón egy hiedelmeként kezeljük. Ha az adott drón megnyeri az aktuális árverést, akkor az útvonaltervben szereplő utak is hiedelemként kerülnek mentésre oly formában, hogy a drón hanyadik lépésben melyik pontba kell, hogy eljusson. Ezen kívül a drón ASL-jében még a mozgásra, illetve töltésre megírt célok szerepelnek. Ezek alapvetően rekurzív célok, melyek önmagukat hívják meg egészen addig, amíg valamilyen feltétel nem teljesül. De a különböző adatok számítása, mint például a drón által licitált ár számítása Java-ban megírt függvények segítségével történik.

Az *.mas2j*-ben a drón ágenseknek egyedi *beliefBaseClass*-ot adtunk meg, amiben csupán azt állítjuk be, hogy mik legyenek az ágenseknek a tulajdonságai (sebesség és kapacitás) mint *initialBelief*-ek. Az aukcionárusnak az egyedi *agentArchClass*-a arra szolgál, hogy a rendelést hozzáadó UI-t egy JFrame-ben implementáljuk vele.

A *calc/calc_cost.java* fájlban, ami egy *DefaultInternalAction*-ből leszármazó osztály implementáljuk a lejjebb részletezett költségfüggvény működését.
Valamint a töltésnek és merülésnek a mértékének a számítását is egy-egy *DefaultInternalAction*-ban implementáltuk, szimplán azért, hogy csak egy közös függvény feleljen a töltés/merülés működéséért, és az ezzel kapcsolatos paraméterek egy helyen legyenek elérhetőek.

Ezeken felül a feladat környezete, modellje, illetve a környezet megjelenítéséért felelős view is Java-ban van implementálva a Jason beépített osztályainak leszármaztatásával. Környezetnek a *GridModell* nevezetű Jason környezetet használtuk egyszerűsége miatt.

Ezt a következő fájlokkal valósítottuk meg:

- world/DeliveryEnvironment.java
    - A jason.Environment-ból leszármazó osztály. Az egyes akciók utáni, a vizuális megjelenésre hatással lévő események trigger-elésének

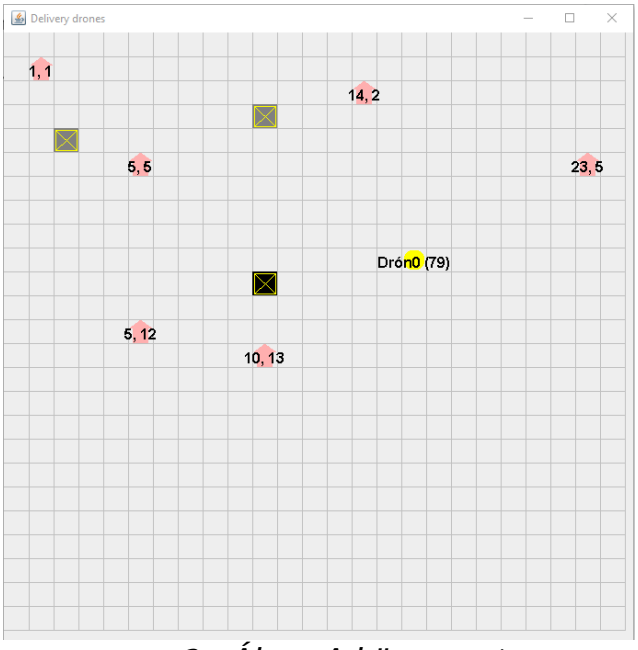
megvalósítása az *executeAction* metúdusban történik. A ‘*move_towards*’ nevű action a drónnak egységnyi mozgását írja le (és mozgás közben dinamikusan frissíti az ágens tudásbázisában a pozíció hiedelmet. A mozgás egy időegységben a célállomás irányába mutató egységvektor sebességgel vett szorzatatának és a drón pozíciójának összege)

- world/WorldModel.java
    - *GridWorldModel* leszármazott. A környezet objektumainak leírása. A kiszállítási pontok (faluk, Village-ek), valamint a töltőállomások és lerakatok koordinátáit és a drónok aktuális töltöttségi szintjei (ez csak megjelenítés céljából) listákban vannak tárolva.
- world/WorldView.java
    - *GridWorldView* leszármazott. Az objektumok kirazolásáért felel.

Kifejlesztett program ismertetése

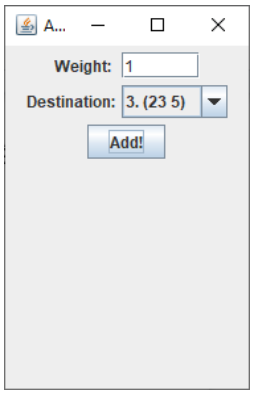
A felhasználói interface

A kifejlesztett program indításakor megjelenik 3 ablak. Az egyik a tér megjelenítése, amiben a drónok szállítják a csomagokat.



2. Ábra. A környezet

Ez az ablak így néz ki. Ezen látszik melyik drón merre van, mi a neve, és mennyi a töltöttségi szintje. Látszanak még rajta a lerakat és a töltőállomások. A fekete-sárga a lerakat, a szürke-sárgák pedig a töltőállomások. A rózsaszín házacskák a célpontok (falvak).



3. Ábra. Rendelés hozzáadó felület

A másik ablak az a kezelőfelület, amin feladatokat lehet feladni.

Ez az ablak így néz ki. A Weight mezőben lehet megadni a csomag súlyát.

A Destination utáni lenyíló mezőben ki lehet választani az egyik célpontot, ahová a csomagot szeretnénk szállítani.
Az Add! gomb megnyomásával lehet feladni a csomagot.

A harmadik ablak a MAS Console, amiben a debuggoláshoz használt kiírások, és üzenetek jelennek meg.

A töltés csökkenése látszik a drón neve mellett. Amennyiben egy töltőhelyen vagy a raktárban áll ez folyamatosan növekszik. Azonban, mikor megrendel egy rendelést nem folymatosan csökken, hanem egyből felveszi azt az értéket, amely a rendelés teljesítése után, és a töltőállomásra/raktárba visszaérkezés után lenne, ha folyamatosan csökkenne. Többek között ezt is a költségfüggvény számolja ki előre.

Költségfüggvény számítása

Alapvetően a töltöttségi szintet figyelembe véve, az a cél, hogy a lehető legkisebb időegységet, mint költséget számítsa ki a függvény, a következő paraméterek, mint hiedelmek alapján:

- Drón **pozíciója** a jövőben, amint rendelkezésre áll (akkor, amikor az összes elnyert, de még sorra nem került rendeléssel végzett. Amennyiben a drón éppen az utolsó rendelését teljesítette, és úton van akkor ez az aktuális pozíciója)
- Drón **töltöttségi szintje** a jövőben, mikor rendelkezésre áll leghamarabb
- **Környezet** ismerete (fő raktár, valamint töltőállomások koordinátái)
- Saját paraméterek (**sebesség, súlykorlát**)
- Rendelés paraméterei (**súly, célállomás**)
- Alap **időköltség (*base*)** amely azt írja le, hogy mennyi idő után végez az elnyert, de még nem teljesített rendelésekkel. (Ehhez lesz hozzáadva a kiszámolt költség)

Ezek alapján a következő számított értékeket, kimeneteket állítja elő, amelyek alapján ezek a végrehajtandó célok keletkeznek:
A töltőállomásokra és a fő raktárra a későbbiekben úgy hivatkozok, mint **állomás.**

- Annak az állomásnak a koordinátái, melyre a rendelés teljesítése után a drón **visszatér**
- **Költség**, mellyel majd licitálhat
- Az a **töltöttségi szint**, amely a rendelés végrehajtása **után** előáll
- Annak az **állomásnak** a koordinátája ahol a drón **töltődik** (ha szükséges) a rendelés teljesítése előtt (ez az esetek nagy többségében megegyezik a fő raktárral.)

Abban az esetben, amennyiben nem bírja el a csomagot, vagyis a csomag súlya nagyobb mint a kapacitása a drónnak, vagyis nem tudja teljesíteni a rendelést akkor a maximális licittel licitál. Az árvereztető ha csak maximális értékű licitet kap akkor eldobja a rendelést, mivel senki sem tudja teljesíteni.

Útvonaltervezés 1, ideális eset, raktár-raktár

Az útvonalat először az alábbi részekre bontom:

- A drón pozíciójából a fő raktárba
- A fő raktárból a célállomásra
- A célállomásból vissza a fő raktárba vezető út

A teljes energiaszükséglet ezeknek az utaknak az időköltségéből adódik. Figyelembe veszük, hogy csak a raktár-célállomás útvonalon szállítja a csomagot a drón.
Ez azért fontos, mivel az energia költség számítás aszerint történik, hogy mennyi a drón terheltsége (szállított tömeg/kapacitás) és mennyi időt tölt a levegőben (ha gyorsabb kevesebb időt).
Amennyiben az így előálló energiaszükséglet kevesebb, mint a drón töltöttségi szintje, akkor eszerint fogja az utat tervezni, mivel ez a legoptimálisabb.
Licitálni a következővel költséggel fog:
timeCost(distance(position,mainDepot))+timeCost(distance(mainDepot,goal))+base

Amennyiben azonban erre nincs elég töltöttség, a következő, alternatív lehetőségek közül fog az egyik megvalósulni.

Útvonaltervezés, 2. lehetőség, raktár-lerakat

Az első lehetőséghez képest annyi az eltérés, hogy megnézi, hogy milyen messze van a **célállomástól a legközelebbi állomás**. **Ha ez közelebb van, mint a fő raktár** akkor lehetséges, hogy odáig viszont elég a töltöttségi szint, így szintén, előtöltés nélkül tudja teljesíteni a rendelést.
Így az **energiaköltség** a következőképpen alakul:
*chargeCost(timeCost(distance(position,mainDepot)) + timeCost(distance(mainDepot,goal)) + timeCost(distance(goal,**closestStationToGoal**)))*

A költség (amellyel licitálni fog) megegyezik az előző lehetőségnél leírtakkal, mivel az nincsen beleszámolva hogy a drón a kiszállítás teljesítése után hova mozog.

Útvonaltervezés, 3. lehetőség, raktár-állomás töltéssel

Erre a lehetőségre akkor kerül sor, ha nincs elég töltöttségi szint se az 1., se a 2. lehetőségre. Ezért tehát kötelező előtöltést végezni az útra indulás előtt.
Ez az eset akkor jut érvényre, ha van elég töltöttség a raktárhoz eljutni, azonban a raktárból a célba, majd onnan valamelyik állomásra már nincs elég töltöttség.
Ebben az esetben addig fog töltődni a drón amíg nem lesz elég töltöttségi szint arra, hogy a 2. lehetőség szerinti útvonalat bejárja, tehát a cél után a legközelebbi állomáson leszálljon. (Ami nem feltétlen a raktár, de lehet hogy az)

Így a költség, amellyel licitál a következőképp alakul:
main_goal_closest = timeCost(distance(mainDepot,goal)) +
*timeCost(distance(goal,**closestStationToGoal**) //Hasonlóan, mint a 2. lehetőségnél*

full_cost = timeCost(distance(position,mainDepot)) + chargeTime(chargeLevel - chargeCost(main_goal_closest)) + timeCost(distance(mainDepot,goal)) + base

Így ‘ki lesz centizve’ tehát pont annyit töltődik a drón, hogy miután visszatért egy állomásra akkor a töltöttségi szintje 0 lesz.
Erre példa lehet az, hogyha az *2. ábrán* lévő környezetet vesszük figyelembe, és rendelés érkezik az (1,1)-es házhoz. Ekkor elég csak annyira feltöltődnie, hogy az (1,1)-hez közeli töltőállomásig eljusson a drón, nem kell tudnia visszajutni a raktárig.

Útvonaltervezés, 4. lehetőség állomás-állomás töltéssel

Erre a lehetőségre akkor kerül sor, ha a raktárba sincs az aktuális pozícióból elég energia eljutni, azonban egy, a raktárnál közelebbi töltőállomásig viszont igen. Erre két lehetőség van, hogy létrejöjjön:

- A drón épp úton van
- A drón épp egy töltőállomáson van, tehát 0 költségből oda tud ‘jutni’. És még nincs eléggé feltöltődve, hogy eljusson a raktárig.

Ebben az esetben először tehát el kell jutnia a töltőállomásig, majd ott fel kell töltődnie, majd onnan a fő raktárig, a raktárból a célállomásig majd a célállomástól a legközelebbi állomásig kell eljutnia.
Ha erre sincs elég töltöttségi szintje (*az csak valamilyen programhiba kapcsán jöhetett csak létre.*) akkor a drón lezuhan, mivel nem tud eljutni semelyik állomásra.

A kifejlesztett programról készült videó:

https://youtu.be/wH1CdH2HVWA