

BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
GÉPÉSZMÉRNÖKI KAR
ENERGETIKAI MÉRNÖKI MESTERKÉPZÉSI SZAK

KOCSIS KENDE ÉS KÖVENDI JÓZSEF

BUDAPEST MÁSODIK
HULLADÉKHASZNOSÍTÓ MŰVÉNEK
GAZDASÁGI ÉS LOGISZTIKAI ÉRTÉKELÉSE

TDK dolgozat

Konzulensek:

Dr. Bokor Balázs

adjunktus, BME ÉPGET

Szücs Botond

tanszéki mérnök, BME EGR

Budapest, 2022 november 7.

TARTALOMJEGYZÉK

1. Bevezetés	2
1.1. Előzetes információk.....	3
2. Összehasonlítási értékek	6
2.1. Szállítási költségek	6
2.2. Szennyezőanyag kibocsátás.....	7
3. Keletkező hulladékmennyiség meghatározása	9
3.1. Elméleti háttér.....	9
3.2. Területmeghatározás pixelelemzéssel.....	11
4. Zónák súlyozása TSZH tartalom alapján	19
5. A TSZH begyűjtés Útvonalainak meghatározása	22
5.1. Gráfelméleti modell létrehozása	22
5.2. Útvonalak meghatározása és lekérése a Google Maps rendszeréből.....	24
5.3. Az optimalizálás logikája	27
5.3.1. Classok, azaz osztályok meghatározása	27
5.3.2. Ellátott zónák meghatározása	30
5.3.3. Útvonaloptimalizáló algoritmus.....	33
6. Eredmények kiértékelése	46
6.1. Kibocsátási értékek.....	53
6.2. A kutatómunka folytatása.....	57
7. Felhasznált források.....	59

Külön köszönettel tartozom Orbán Tibornak, a FŐTÁV Zrt. műszaki vezérigazgató-helyettesének, aki segítségével elősegítette a tanulmány megvalósulását.

1. BEVEZETÉS

Világszerte éves szinten 1,3 trillió tonna települési szilárd hulladék termelődik, amely fejenként 180 kg/évet jelent. Feltételezhetően 2025-ig ez 2,2 trillió tonnára fog emelkedni. A fejlett országokban, mint az Európai Unió országaiban és az USA-ban ez az érték jelentősen magasabb. Az EU szabályozási rendszerének megfelelően a leghatékonyabb hulladékkezelési módszerek a megelőzés, az újrahasznosítás és az energetikai hasznosítás. A legrosszabb megoldás a települési szilárd hulladék depókban történő lerakása helyigény, az esztétikai és környezetkárosító hatások, illetve a bomlás során keletkező, nem hasznosított metán atmoszférába történő kiengedése miatt. A metán GWP (global warming potential) értéke huszonötszöröse a CO₂ értékének, amely így a globális felmelegedés szempontjából is jelentősen nagyobb problémát jelent. Ebből adódóan egy második hulladékhasznosító mű telepítése Budapest városgazdálkodásában kiemelt jelentőségű lehet. A Budapesten termelődő 270 000 tonna/év települési szilárd hulladék jelenleg nagyrészt a pusztazámori hulladéklerakóba kerül. A válogatás nélküli lerakás a legnagyobb környezetterhelést jelenti mind az elfoglalt terület, mind pedig a talajszennyezés szempontjából, ezért ellentétes az EU hulladékkezelési stratégiájával. Megfelelő füstgáztisztítás mellett a települési szilárd hulladék (TSZH) újból felhasználható alapanyagokká (hamu és salak), valamint az eredeti térfogat 0,1%-át képző, szűrőben visszamaradt szennyeződéssé alakítható, ez utóbbi kerül depóniára. Mindezek mellett a környező térség energiaellátásában hasznosítható olcsó hő- és villamosenergia termelhető.

Jelen tanulmányban az előzetes tervek alapján az Ócsai út és az M0-ás autópálya kereszteződéséhez telepítendő HUHA 2 hulladékhasznosító erőmű hulladékbeszállításával foglalkozunk. Jelen tanulmány része egy általunk

készített - a HUHA 2 telepítésének teljeskörű értékelésével foglalkozó – tanulmánynak, amely azonban a Tudományos Diákköri Konferencia terjedelmi korlátain túlnyúlna, így ebben a tanulmányban csak a szállítási útvonaloptimalizációval foglalkozunk. Ezen tanulmány eredményeit felhasználva kívánjuk a hulladékhasznosító erőmű versenyképességét növelni a földgáztüzelésű energiatermeléssel szemben.

A kutatás eredményeként meg fogjuk kapni az egységnyi Települési Szilárd Hulladék (TSZH) beszállításához szükséges anyagi költségeket és a hozzá rendelhető kibocsátási adatokat. Ezen érték csökkentésével versenyképesebbé válhat a hulladékhasznosítás, mint energiatermelő erőforrás és lecsökkenthetjük Budapest távhőrendszerének és elektromos ellátásának földgáz függését, valamint nem utolsósorban redukálhatjuk a beszállítási tevékenység során keletkező környezeti szennyezést.

1.1. Előzetes információk

A tanulmányban a vizsgálandó szempont a hulladékszállítási útvonalak optimalizálása, ami egy kiemelt fontosságú kérdés több szempontból is, mint CO₂ és NO_x kibocsátás, szállítási költségek és nem utolsó sorban a városi közlekedésre rótt közlekedési teher szempontjából is.

„A kommunális hulladék beszállítás az FKF hulladékgyűjtő és szállító járműveivel történik, melyek a kocsni megtelezése után a város területéről a telephelyre hozzák a kommunális hulladékot, és itt a HUHA 2 ledobó aknájába ürítik.

A telephely jellemzője, hogy Dél-Pesten, a várostól távol az M0 körgyűrű és az 5. út kereszteződésénél helyezkedik el, így a városban gyűjtött szemét megosztható a két, a város ellentétes pontjain elhelyezkedő hulladékhasznosító között. Ez a szállítási távolságokat csökkenti. Hátrány ugyanakkor, hogy a

telephely a várostól távol van, így a szállítási távolságok nagyobbak, mint egy város közeli létesítménynél.” [1.]

A beszállítandó, FKF által kezelt éves hulladékmennyiség 230 000 tonna, ez a mennyiség növelhető lenne az előválogató által kezelt SRF (Solid Recovered Fuel) égetése esetén, mivel annak előállítása a TSZH-ból lényegesen lecsökkenti a mennyiségét, ám jelen tanulmányban a 230 000 tonnás éves szemétfelfogadó képességgel számoltunk. SRF-nek nevezzük azt a másodlagos tüzelőanyagot, amire meghatározott szabvány szerinti előírások vonatkoznak.

A további – maximális mennyiségű – hulladék hasznosítása az erőműben a tanulmány folytatásának potenciális alternatívája.

Az FKF hulladékszállító járműállománya legfőképpen Mercedes márkájú járművekből tevődik össze, így a számításainkat ezekre alapoztuk. Kétféle járműmodellt tudunk megkülönböztetni: Rotopress-t és Variopress-t. A két modellt a begyűjtött hulladékkezelés módja különbözteti meg egymástól. A járművek leírása az FKF honlapján találhatóak meg:

„A Rotopress-es járművet külsőleg a felépítményről úgy ismerhetjük meg, hogy egy fél hordó formájához hasonlít. Nem véletlen ez a forma, hiszen belül egy forgó henger belsejében, csigavonalban felhegesztett lemez van, amely a szemetet bizonyos mértékig aprítja, valamint – mondjuk úgy - csigaforgással a hátul beöntött hulladékot előre tömöríti.”

„A Variopress típusú hulladékgyűjtő jármű a Rotopress-szel ellentétben nem „darál”, hanem ún. tömörítő lapáttal végzi az optimális helykihasználást. Leegyszerűsítve, a szemét tömörítés úgy történik, hogy a vezetőfülke mögötti hátsófal mozgatható, vagyis, ahogy a tömörítő lapát nyomja előre a bekerült szemetet, fokozatosan telik a tartály, a fal egyre inkább közelít a vezetőfülke felé.” [3.]

A járműállomány túlnyomó része a második modellből áll, 90 – 278 darab arányban, így – mivel sokkal nagyobb arányban vannak jelen a variopresszek és

a modellek fogyasztási értékei alig különböznek fajta szerint – az elemzéshez az összes modellt Variopressnek vettük és ezen modell adatait használtuk fel.

2. ÖSSZEHASONLÍTÁSI ÉRTÉKEK

2.1. Szállítási költségek

Az optimalizálási eredmények összehasonlításához alapul vettük a 2017-es HUHA 2 Új Iszapégető és Hulladék-hasznosító Erőmű Műszaki Döntés Előkészítő Tanulmányt, amely tartalmazott beszállítási költségeket. [1.]

A járművek szállítókapaacitása 8 tonna, amelyet figyelembe véve ez évente 28 750 fordulót jelent, feltételezve, hogy minden kocsi teljesen megtelve érkezik meg a HUHA 2-be. Az elemzés során meg fogjuk tudni, hogy természetesen ez a feltételezés nem minden esetben teljesül, de egy kezdeti értéknek megfelel.

A fenti beszállítás fajlagos költsége erre a telephelyre a jármű, személyzeti és üzemanyag költségek figyelembevételével a 2017-es elemzés alapján 8.446 Ft/t volt. Természetesen ez az érték a mai helyzetre vetítve hamis értéket fog adni az üzemanyagárak drasztikus megemelkedése miatt, emiatt át kell számolni jelenlegi üzemanyagárakra. A 2017-es dízelolaj árak átlaga ~345 Ft/l volt [2.]

	Üzemanyag árak [Ft/l]	Fajlagos szállítási költség [Ft/tonna]	Éves szállítási költség [Ft]
2017	345	8446	1 942 580 000
2022	820	20 075	4 617 146 667

1. tábla: Szállítási költségek 2017-ben és 2022-ben az üzemanyagárak alapján arányosítva

Ha a jelenlegi világpiaci árazást vesszük alapul, akkor jelentősen megemelkednek a költségek. A NAV által közölt adatok alapján 2022 októberében regisztrált adatok alapján 820 Ft/l lenne a dízelolaj eladási ára a stop nélkül. Ezzel az értékkel számolva a fajlagos TSZH szállítási költségére 20 074 Ft/t-t kapunk, ami több, mint kétszeres költségnövekedést jelent az eredeti árhoz képest.

Ugyanilyen megfontolások alapján megkapjuk, hogy az éves szállítási költségek az üzemanyagárakkal normálva több, mint 4 617 millió Ft lesz. Ezt az értéket a HUHA 2 előkészítő dokumentáció által megadott, 2017-es szállítási költségek alapján számoltam ki 2022-es árakra vonatkoztatva. Tartalmazza a beszállítási, salakanyag elszállítási és a veszélyes hulladék elszállítási költségeket is.

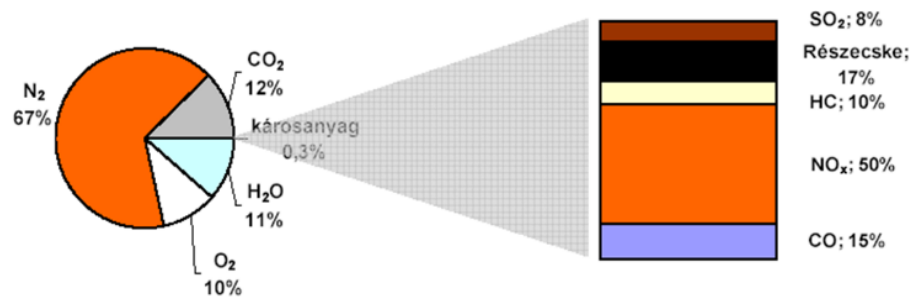
Az itt látható előzetes adatok alapján tisztán látszik, hogy miért kulcsfontosságú az útvonalak optimalizálása és ezzel együtt a fogyasztási értékek csökkentése a hulladékhasznosítás gazdaságos üzemelésének fenntartásának érdekében.

2.2. Szennyezőanyag kibocsátás

A hulladékszállítás jelenleg Budapest területén dízel üzemanyaggal működő szemétszállító járművekkel történik, amely a korábban említett – csupán a HUHA 2-t érintő - 28 750 fuvarkör fényében megterheli a városi frisslevegő minőségét és a közlekedési útvonalak járhatóságát egyaránt.

A dízel üzemanyag elégetéskor nagy mennyiségű nitrogén-oxid vegyületeket szabadít fel, amelyek nagy szerepet játszanak a fotokémiai szmog kialakulásában, meghatározó szerepük van a savas ülepedés, vagy közismertebb nevén a savas esők kialakulásában és károsítják az emberi egészséget is. A nitrogén-oxid légzőszervi megbetegedéseket okoz, a légutak nyálkahártyájának és a szem kötőhártyájának gyulladását, a vérerek kitágulását eredményezheti. 200 mg/m³ fölött roncsolja a tüdő szöveteit. [4.]

Dízelmotor



1. ábra: Dízelmotorok kibocsátási arányai [6.]

Ezen anyagok környezetbe való kerülésének csökkentése szintúgy a megtett táv minimalizálását szorgalmazza.

3. KELETKEZŐ HULLADÉKMENNYISÉG MEGHATÁROZÁSA

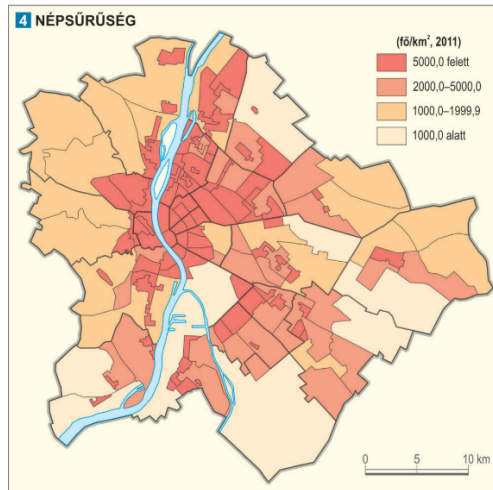
3.1. Elméleti háttér

A szállítási útvonalak optimalizálásához azt az egyszerűsítést alkalmaztuk, hogy Budapest egészét zónákra bontottuk, majd minden egyes zónának meghatároztunk egy csúcspontot, ahonnan a szemetet be fogjuk gyűjteni. A modell pontosságának növelésének érdekében Budapestet minél több részre próbáltuk felbontani, a végeredmény összesen 182 zóna lett. A modell szerint a hulladékszállító járművek a HUHA 2-ből indulva a lehető legrövidebb úton elmennek a célzóna kijelölt csúcspontjába, ahol begyűjtik a kapacitásuknak megfelelő mennyiségű szemetet, majd a lehető legrövidebb úton visszatérnek a HUHA 2-be, ahol lerakják terhüket. Ezt a tevékenységet addig folytatják, amíg a hulladékhasznosító erőmű befogadó kapacitását el nem érik.

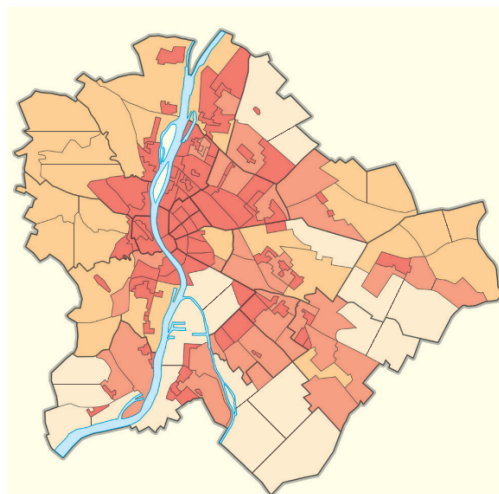
Azonban észrevehető, hogy ha csupán a csúcspontokig néznénk az útvonalat és a fogyasztási értékeket, akkor hamis eredményeket kapnánk. Ennek kiküszöbölésére kidolgoztunk egy arányosítási rendszert, amivel minden zónához hozzárendeltem egy négyzetkilométerre fajlagosított bejárando úthosszt, amelyet arányosítottunk a kukásautó szállítási kapacitásával, és így kiadódik, hogy a zónán belül mekkora fogyasztással rendelkezett a kukásautó, mire begyűjtötte a hátralevő kapacitásának megfelelő hulladékmennyiséget.

A szállítási útvonalak meghatározásához szükség van hulladék keletkezési adatokra, amelyek segítségével meg lehet határozni egy adott régióban keletkező – és egyben elszállítandó – hulladék mennyiségét. Egy hulladéksűrűségi adatbázissal kapcsolatban először az FKF-hez fordultunk, ám sajnos nem rendelkeztek ilyen adatbázissal, vagy kizárólag belső használatra engedélyezett, emiatt más megközelítést kellett alkalmaznunk.

A **2. ábra** a Magyar Nemzeti Atlasz által készített budapesti népsűrűségtérkép, amelyet alapjául vettem a számításainknak.



2. ábra: Budapest népsűrűségi adatai [7.]



3. ábra: Tovább felosztott zónákat ábrázoló térkép

A térkép zónákra osztott területek szerint határozza meg a népsűrűséget, amelyeket bizonyos zónák esetén a felbontás sűrűségének növelésének érdekében tovább osztottunk alrégiókra, így kisebb kiterjedésű területegységeket kaptunk.

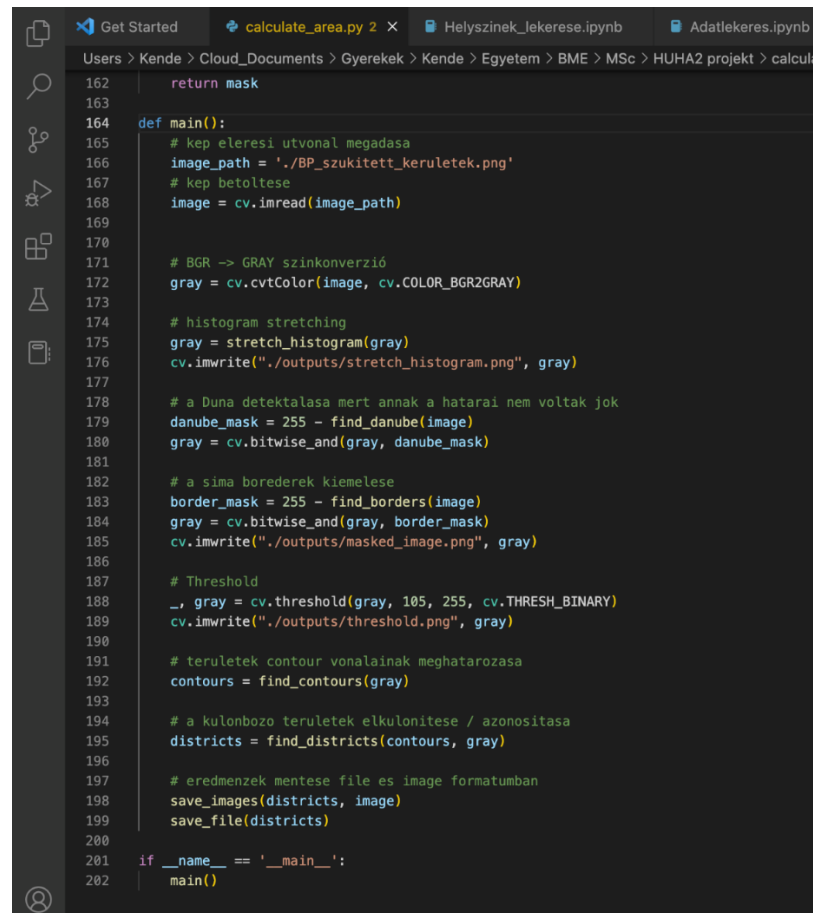
Ezen ábrákon található szín- és területi adatokat használtuk fel a jelzett zónák területének és népsűrűségének, illetve lakosságának meghatározására. Magyarországon az Európai Parlament által közölt adatok szerint fejenként 381 kg hulladék termelődik évente. [5.] A népsűrűség és a kiszámolt zónaterület adatok alapján meg tudjuk határozni az adott zónában élő népesség számát, amely alapján meg tudjuk határozni a zónákban keletkező TSZH mennyiségét és egy hulladéktermelési sűrűségtértépet tudunk előállítani.

3.2. Területmeghatározás pixelelemzéssel

A térképben tárolt adatok kinyeréséhez kidolgoztunk egy módszert, amivel csupán a képen látható információk alapján meg tudjuk határozni az összes zóna egyéni adatát. Manuális eszközökkel rendkívül időigényes lett volna és csak hatalmas hibaszázalékkal lett volna megoldható feladat, így más módszerhez folyamodtunk.

Ennek megoldására egy pixelelemző programot írtunk a Python programozási nyelven. Ezt a programnyelvet az egész tanulmány során használtuk, az összes eredményünket ebben dolgoztuk fel. A program alapfeladata a képen látható körvonalak felismerése, elkülönítése és a közrefogott területük megmérése volt, ezen területek elkülönítése egyedi grafikus állományokba, majd minden egyes terület geometriai középpontjának meghatározása volt.

A program első pár lépésének kódja a következő képen látható:



```
162     return mask
163
164 def main():
165     # kép eleresi utvonal megadása
166     image_path = './BP_szukitett_kerulet.png'
167     # kép betöltése
168     image = cv.imread(image_path)
169
170
171     # BGR -> GRAY szinkonverzió
172     gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
173
174     # histogram stretching
175     gray = stretch_histogram(gray)
176     cv.imwrite("./outputs/stretch_histogram.png", gray)
177
178     # a Duna detektálása mert annak a hatara nem voltak jók
179     danube_mask = 255 - find_danube(image)
180     gray = cv.bitwise_and(gray, danube_mask)
181
182     # a síma borederek kiemeleése
183     border_mask = 255 - find_borders(image)
184     gray = cv.bitwise_and(gray, border_mask)
185     cv.imwrite("./outputs/masked_image.png", gray)
186
187     # Threshold
188     _, gray = cv.threshold(gray, 105, 255, cv.THRESH_BINARY)
189     cv.imwrite("./outputs/threshold.png", gray)
190
191     # teruletak contour vonalainak meghatarozasa
192     contours = find_contours(gray)
193
194     # a kulonbozo teruletak elkulonitese / azonositasa
195     districts = find_districts(contours, gray)
196
197     # eredmenzek mentese file es image formatumban
198     save_images(districts, image)
199     save_file(districts)
200
201 if __name__ == '__main__':
202     main()
```

4. ábra: A pixelelemző program „main”, azaz fő része

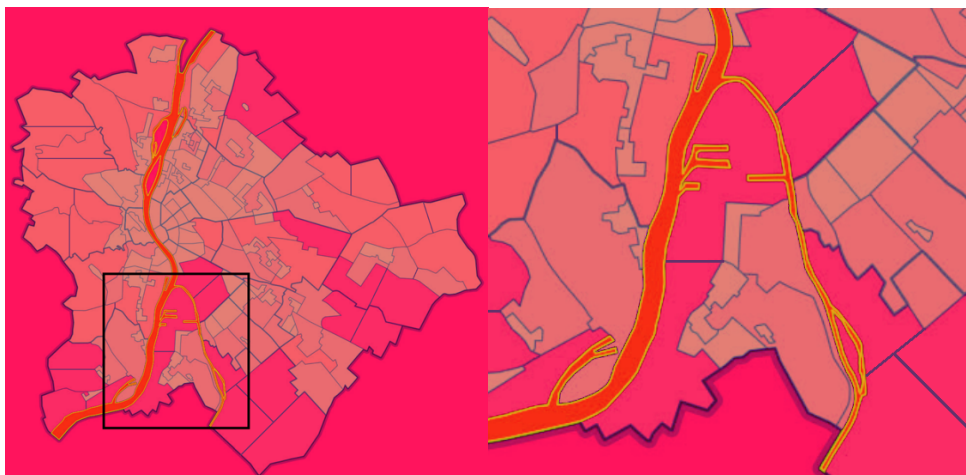
Első lépésnek betápláljuk a programnak az elemzendő – szűkített zónákkal rendelkező – képet (3. ábra). Ez a kép azonban még tartalmaz sok, számunkra zajnak, avagy zavaró tényezőnek számító információt, ami megakadályozza a körvonalak pontos meghatározását. Ennek kiküszöbölésére az alábbi módszert alkalmaztuk: elsőként a kép „colouring scheme”-ét, azaz a színösszetételét átkonvertáljuk először is RGB-ből HSV-be. Ezzel a módszerrel az R (piros), G (zöld) és B (kék) értékek helyett a színcsatornák H (árnyalat), S (telítettség) és V (érték) értékeket fognak felvenni.

A konvertálás nagyban megkönnyíti a szűrési folyamatot, mert HSV színösszetételben a színre vonatkozó adatokat három csatorna helyett csak egy csatorna tartalmazza, a V.

A HSV (hue, saturation, value; más néven HSB, hue, saturation, brightness) az RGB színmodell alternatív ábrázolásai, amelyeket az 1970-es években a számítógépes grafikusok terveztek, hogy jobban megfeleljenek az emberi látás színérzékelési tulajdonságainak. Ennek segítségével jobban el tudjuk különíteni intenzitás szempontjából a képen található pixeleket. A konvertáláshoz használt programrészlet az **5. ábrán** látható.

```
136 def find_danube(image):
137     # RGB -> HSV szinkonverzio (konnyeb szegmentalni)
138     hsv = cv.cvtColor(image, cv.COLOR_RGB2HSV)
139
140     # boundary value -k meghatarozasa
141     lower_range = np.array([0,120,190])
142     upper_range = np.array([30,180,270])
143
144     # mask meghatarozasa
145     mask = cv.inRange(hsv, lower_range, upper_range)
146
147     return mask
148
149 def find_borders(image):
150
151     # RGB -> HSV szinkonverzio (konnyeb szegmentalni)
152     hsv = cv.cvtColor(image, cv.COLOR_RGB2HSV)
153     cv.imwrite("./outputs/hsv.png", hsv)
154
155     # boundary value -k meghatarozasa
156     lower_range = np.array([70,20,100])
157     upper_range = np.array([150,120,200])
158
159     # mask meghatarozasa
160     mask = cv.inRange(hsv, lower_range, upper_range)
161
162     return mask
```

5. ábra: A Duna és a zónák megtalálásáért felelős programrészlet



6. ábra: A kép HSV fájl formátumba konvertálva (bal), részlet a Dunával (jobb)

Az intenzitásbeli különbségek alapján a „boundary value”-k, azaz határértékek megfelelő megválasztásával el tudjuk különíteni a képen található Duna szakaszait, amelyek nem jelenhetnek meg a térkép zónafelismerésében. A jobb oldali kinagyított ábrán megfigyelhető az éles kontúr a Duna és a környező területek között. A kijelölt Duna képét egy maszk létrehozásával elkülönítjük. Ezt a maszkot később felhasználjuk a folyó területének elkülönítésére.

Az input képet átkonvertáljuk „Gray scale” formátumú, azaz színes képből fekete-fehér képbe, hogy ne zavarják az algoritmust színek, majd erre a képre ráillesztjük az imént elkészített – Duna területét tartalmazó – maszkot, szintén fekete fehér színben. A keletkező kép alább található 6. **ábrán** látható.



7. ábra: Fekete – fehér kép a hisztogram széthúzás előtt és után

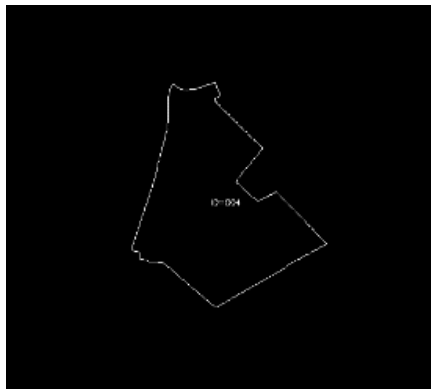
Ezzel a formátummal egy lépéssel közelebb kerültünk a megoldáshoz, ám jelen állapotában a képünk nem tartalmaz a színskála két végén található, azaz teljesen fekete és teljesen fehér pixeleket, ami megnehezíti az elemzést. Ennek kijavítására a fent látható kép hisztogramját széthúzzuk, amivel a sötét és a világos pixelek fényesség értéke módosul a szélső értékeik felé. Gyakorlatban alkalmazva a sötét egyre feketébb lesz, a világos pedig egyre fehérebb, így megkönnyítve a pixelek közötti különbségek detektálását.

Az így megkapott ábrát „threshold”-oltuk, – azaz egy küszöbértéket határozzunk meg neki - azaz a teljesen fekete tulajdonságú pixeleken kívül minden pixel fényesség értékét fehérre változtatjuk, így kizárólag a zónák kontúrjai maradnak meg az ábrán.



8. ábra: A térkép a küszöbölés után

Ezek után a kontúrok meghatározásával el tudjuk különíteni az általuk körbe határolt zónákat, amelyek területét és középpontját meg kívántuk határozni. Ezeket a zónákat külön – külön elmentjük és egy ID azonosítóval látjuk el területszerinti csökkenő sorrendben. A **8. ábrán** fekete körrel jelöltük az a **9. ábrán** bemutatott zóna lokációját.



9. ábra: Az ID = 4 sorszámú mentett zóna képe

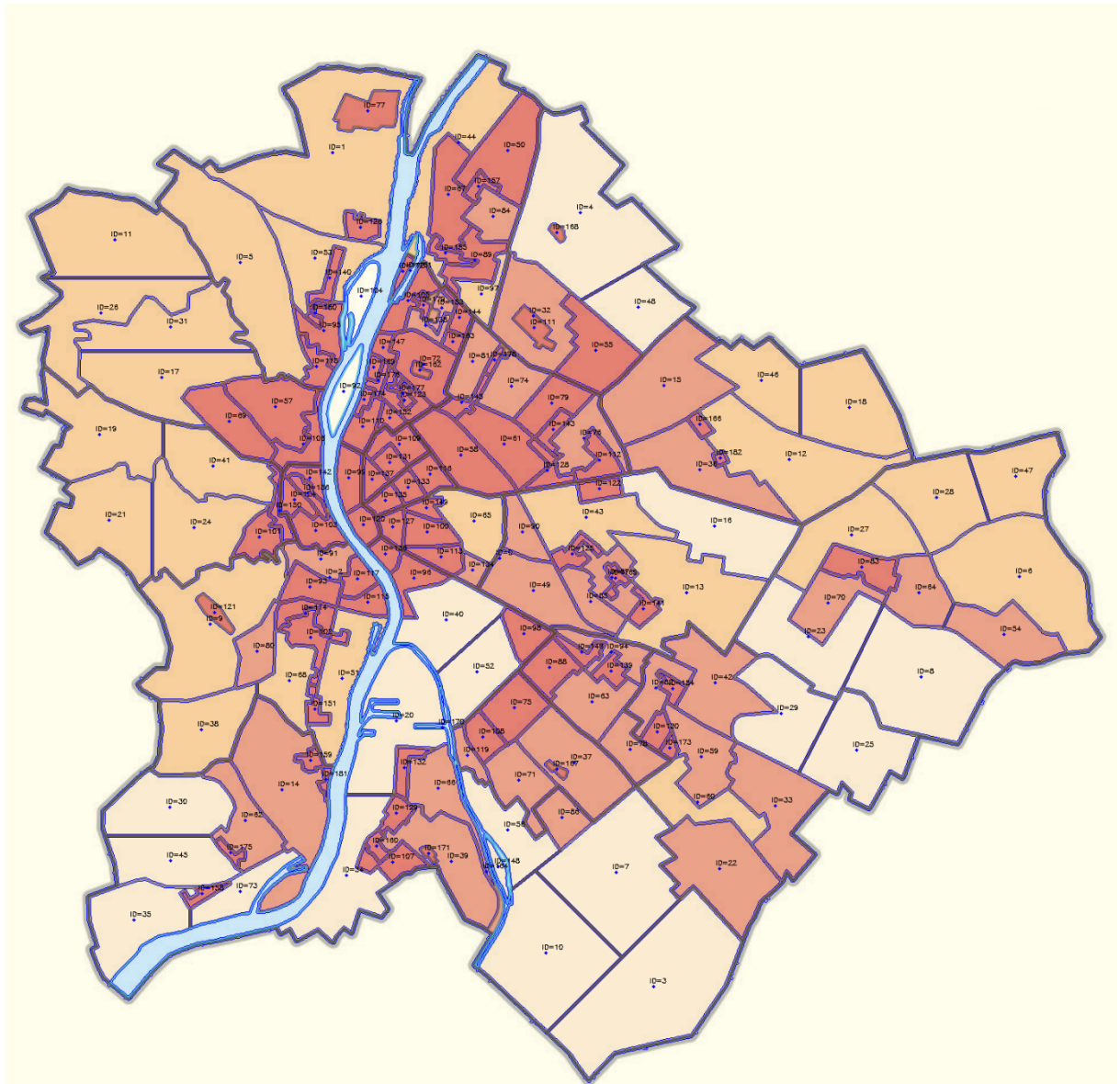
Összesen így zónaként 182 területet számoltunk össze és azonosítottunk be. A területek nagyságát egy arányosítással adjuk meg. Az egész város területén található pixelek számát meghatározzuk, valamint mivel ismert Budapest területe, így meg tudunk határozni egy pixel/km² arányossági tényezőt, ami kapcsolatot teremt a kép és a valóság méretarányai között. Ezek után pedig csak arányosítani kell a kiszámolni kívánt zónában található pixelek számát az egész városban található pixelek számához a következő képlettel:

$$A_{zóna} = \frac{A_{zóna,pixel} + A_{Bp}}{A_{Bp,pixel}}$$

Ahol:

- $A_{zóna}$ = a keresett zóna területe [km²]
- $A_{kerület,pixel}$ = a keresett terület pixelszáma [-]
- A_{Bp} = Budapest területe [km²]
- $A_{Bp,pixel}$ = Budapest területén lévő pixelek száma [-]

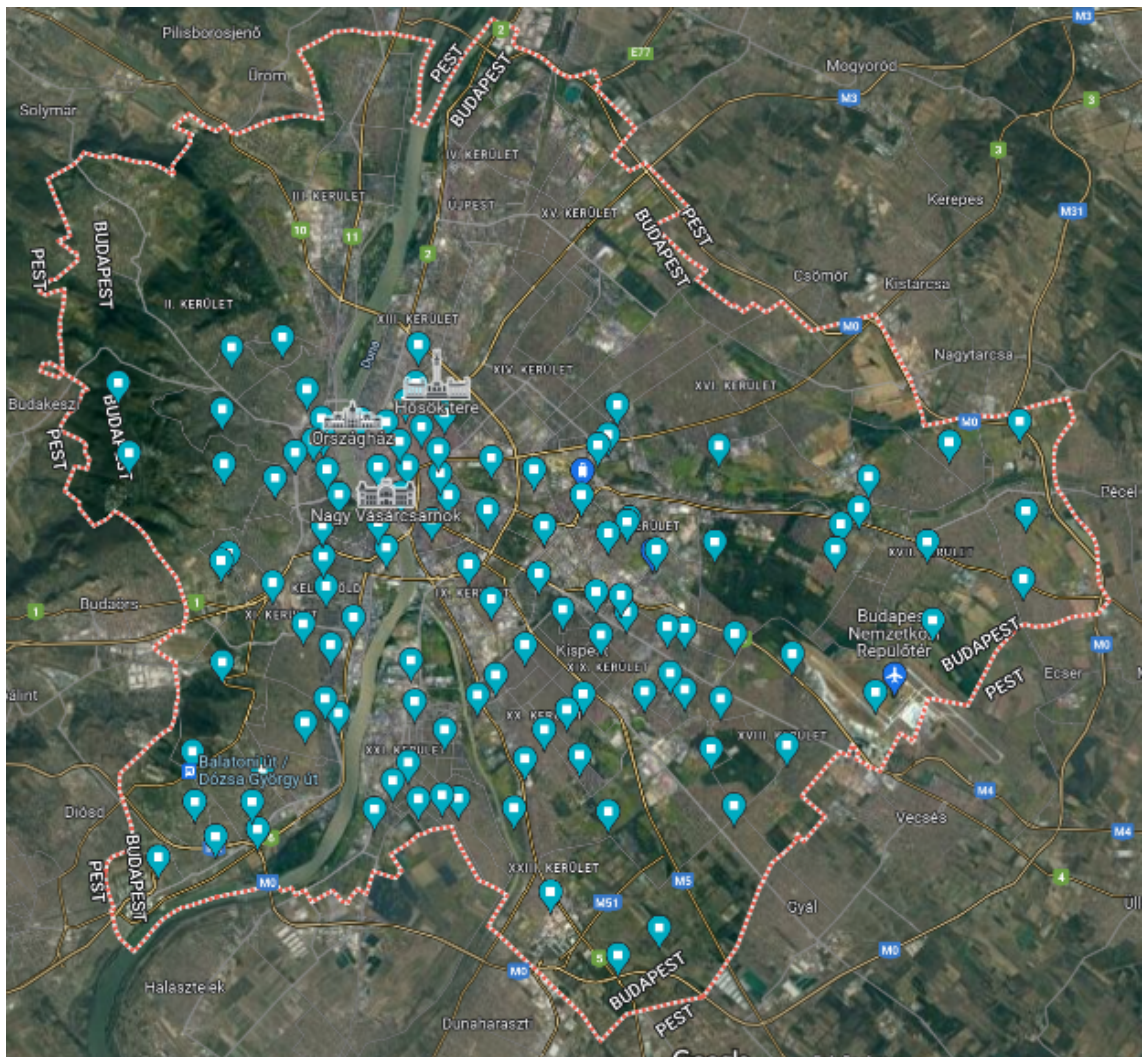
Végző lépésként pedig a 182 azonosított zónát ráhelyezzük az eredeti képre jelöléssel és azonosító („ID”) számmal együtt, amely a következőképp néz ki:



10. ábra: A pixelemelés eredménye: Budapest az összes azonosított zónával és csúcsponttal

A meghatározott, ID értékhez rendelt terület adatokat kiíratjuk egy külön adatállományba, amelyet egy táblázatkezelőbe, jelen esetben Microsoft Excelbe importálva az adatokat számítási műveletekhez tudjuk felhasználni.

A korábban meghatározott geometriai zóna középpontok a Google Maps program térképére való felvitel után a középpontok szélességi és hosszúsági koordinátaként fognak szolgálni a belőlük alkotott gráfrendszer csúcspontjaiként.



11. ábra: A zónák csúcspontjai Google Mapsen jelölve

Észrevehető, hogy nincs az **10. ábrán** bemutatott összes csúcspont megjelölve a Google Mapsen (**11. ábra**). Ennek oka az, hogy az északi területeken keletkező hulladékmennyiség nagy részét a XV. kerület északi csúcsán található HUHA 1 hasznosítja, így felesleges ezen területek hulladékának a város déli felére való szállítását vizsgálni. Valószínűsíthetőleg a jelölt pontok is bőven meghaladják a HUHA 2 hulladékhasznosítási kapacitását, ennek vizsgálatát elvégezzük a tanulmány folyamán lejjebb.

4. ZÓNÁK SÚLYOZÁSA TSZH TARTALOM ALAPJÁN

Az elméleti összegzésben már említettekkel összhangban az útvonalak hosszának meghatározásakor kizárólag a bejelölt csúcspontok figyelembevétele nagy hibaértéket eredményezne számításaikban, mivel nem számolna az adott zónán belül megtett úthosszal és így a hozzá tartozó fogyasztással sem.

Ennek a problémának a megoldására egy arányosítást hoztunk létre, aminek segítségével számszerűsíteni lehet a zónán belül megtett úthosszt kukásautó fennmaradó üres kapacitásának és a zóna hulladéksűrűségének függvényében.

Első lépésként a népsűrűségi és a területi adatok alapján meghatároztuk a zónákra vonatkozó heti TSZH termelés mértékét.

$$m_{TSZH} = f_{nép} * A_{kerület}$$

Ahol:

- m_{TSZH} = heti TSZH termelés [kg/hét]
- $f_{nép}$ = népsűrűség [fő/km²]
- $A_{kerület}$ = zóna területe [km²]

A zónánként megtett úthossz meghatározásához létrehoztuk a fajlagos úthossz fogalmát, ami megadja, hogy egy egységnyi területen hány kilométer úthossz helyezkedik el. Ez az érték átlagosan $a = 24,9 \frac{km}{km^2}$ lett Budapest területére nézve. A fajlagos úthossz segítségével ki lehet számolni az egy zónában található úthossz értékét. Ezen érték segítségével fogjuk tudni kiszámolni az egyes zónákban megtett út hosszát a begyűjtött hulladékmennyiség függvényében.

$$l = A_{kerület} * a$$

Ahol:

- l = Úthossz [km]
- a = Fajlagos úthossz [km/km²]

Ezt a számítást azért alkalmazhattuk, mert feltételeztük, hogy az összes keletkező hulladék begyűjtéséhez a járműveknek végig kell haladnia az összes utca teljes hosszán. Ezzel a számítási móddal a sűrűn lakott területek esetén jóval kevesebb úthosszt kell bejárni a szállító járműnek egy egységnyi hulladék felvételéhez, mint egy gyéren lakott területen, ami megfelel a való életben tapasztalt helyzetnek.

Ennek vizualizálására létrehoztuk a lineáris TSZH sűrűséget, ami megadja, hogy mekkora úthosszt kell bejárni a szállító járműnek egy egységnyi hulladék felvételéhez.

$$f_{TSZH} = \frac{m_{TSZH}}{l}$$

Ahol:

- f_{TSZH} = lineáris TSZH sűrűség [kg/km]

Az úthosszak azonosításával meg tudtuk határozni, hogy egy zóna hulladék-kiürítéséhez hány liter üzemanyagra van szükségünk figyelembe véve, hogy a szemétszállító járművek átlagos fogyasztása 73 l/100 km. [8.]

$$v_{zóna} = \dot{v} * \frac{l}{100}$$

Ahol:

- $v_{zóna}$ = Üzemanyag fogyasztás [l]
- $\dot{v}$ = Fajlagos üzemanyag fogyasztás [l/100 km]

Az esetek túlnyomó részében egyetlen jármű azonban nem képes egy zóna kiürítésére, emiatt a 8 tonnányi hulladék szállítókapacitásra kell normálni a fogyasztást.

$$v_{kocsi} = \frac{k}{f_{TSZH} * l} * v_{zóna}$$

Ahol:

- v_{kocsi} = Egy jármű fogyasztása 8 tonna kapacitás esetén [l]
- k = Szállítókapacitás [kg], üres állapotban 8000 kg

A felsorolt számítások és összefüggések jobb szemléltetéséért mellékeljük az első pár csúcsponthoz tartozó adatot a **2. táblában**.

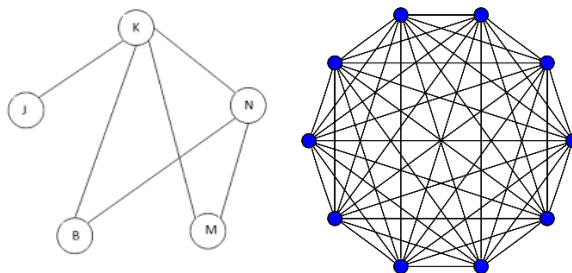
ID	AREA	Szeleségi fok	Hosszusági fok	Népsűrűség	TSZH termelés	TSZH területi sűrűség	Normált útvonalhossz	TSZH lineáris sűrűség	Normált üzemanyag	Normált úa. 8t-ra
	[km2]	[°]	[°]	[fő/km2]	[kg/hét]	[kg/km2*hét]	[km]	[kg/km]	[l]	[l]
3	13,083	47.373245	19.162598	372,77	35732,67	2731,26	129,00	276,99	79,34	17,76
5	10,713	47.452592	19.267040	1500	117743,46	10990,38	105,64	1114,59	64,97	4,41
6	9,890	47.480684	19.302386	1340	97104,74	9818,08	97,52	995,70	59,98	4,94
9	9,557	47.467833	18.995491	1981	138716,19	14514,63	94,24	1472,00	57,96	3,34
10	9,087	47.382384	19.121253	372,77	24818,42	2731,26	89,60	276,99	55,10	17,76

2. tábla: Számítási adatok összefüggése

5. A TSZH BEGYŰJTÉS ÚTVONALAINAK MEGHATÁROZÁSA

5.1. Gráfelméleti modell létrehozása

Gráfnak határozzuk meg objektumok (csomópontok, csúcsok) és rajtuk értelmezett összeköttetések (élek) halmazát. Ezen csúcspontoknak és a közöttük elhelyezkedő éleknek súlyozási értékeket tudunk adni, amely súlyozással különböző kapcsolati értékeket tudunk definiálni és kialakítani. Ilyen súlyozási érték lehet például pszichológiai alkalmazásban a két személy közötti kapcsolat, vagy esetünkben például a két pont közötti távolság és a megtételükhöz szükséges idő. Fontos megjegyezni, hogy a gráfrendszerünk súlyozási értékeinek nem adhatunk meg légvonalbeli távolságértékeket, mert az gyakorlati alkalmazásra vagy valós események szimulálására alkalmatlan lenne.



12. ábra: Egy parciális és egy teljes gráf képe [9.][10.]

A gráfoknak rengeteg fajtája van, az egyik csoportosítási mód a gráfok csúcspontjainak kapcsolata alapján parciális és teljes gráf szerint különíti el a gráfokat. A lényegi különbség az élek teljességében rejlik: a parciális (bal oldali) gráf esetén nincs minden csúcspont között összeköttetés egy másik gráfpont érintése nélkül, míg a teljes gráfok esetében ez a kritérium teljesül.

Jelen modellben egy teljes gráf felépítését tűzzük ki célnak, amellyel lehetővé tesszük a – későbbiekben leírt – algoritmusnak, hogy szabadon tudjon választani a legmegfelelőbb útvonal között a szállítási útvonaltervezéskor.

A zónák középpontjainak meghatározásával kapott 182 pont közül a HUHA 2 lokációja szerint kiválasztott 100 pontot használtuk fel. Ezen pontok mindegyikéhez meghatároztuk a szélességi és hosszúsági koordinátákat és hozzárendeltük őket a fentebb definiált ID értékekhez. A felállított gráf csúcspontjainak ezen pontokat feleltettük meg.

Az útvonalkeresés során azt vizsgáljuk, hogy melyik az a lehető legrövidebb élsorozat, amelyen végighaladva a jármű a HUHA 2 erőműből indulva el tud érni a kiválasztott csúcspontba, ott be tudja gyűjteni a kívánt mennyiségű hulladékot, majd vissza tud érni a kiindulási pontjába: az erőműbe. A vizsgálathoz kapcsolatot kell tudnunk teremteni az összes pont között, hogy az optimalizációs algoritmus szabadon ki tudja választani a legjobb útvonalat a következő csúcspontig.

A vizsgálathoz -a korábbiakban leírtakkal egyezve – egy teljes gráfot hozunk létre, amely a 100 csúcspont figyelembevételével 4550 élt határozunk meg, ami a következő módon írható le matematikailag:

$$K_n = \frac{n(n-1)}{2}$$

Ahol:

- K_n = n csúcspontú teljes gráf éleinek száma
- n = csúcspontok száma

Azonban a mi – valóságot elemző – esetünkben ennél az ideális gráf-megközelítésnél összetettebb a modell: a normális teljes gráfok esetében a 2-es

osztótag azért van jelen, mert „A” csúcspontból „C” csúcspontba eljutni ugyanakkora táv, de ez a valós közlekedési esetekre nem igaz. Teljesen elképzelhető, hogy két lokáció közötti legrövidebb útvonal hossza, vonalvezetése és ezáltal a megtételéhez szükséges idő sem egyezik a közlekedési útvezetés miatt (pl.: útlezárás, egyirányú utak stb.). Ennek okán az általunk alkalmazott rendszer összes elemzendő éle:

$$K_n = n(n - 1)$$

ami esetünkben 9900 élt jelent.

5.2. Útvonalak meghatározása és lekérése a Google Maps rendszeréből

A gráf meghatározása után a kialakult éleket súlyozni kell, a későbbiek során ez alapján tudja majd eldönteni az útvonalkereső algoritmus, hogy melyik útvonalat válassza. Az algoritmus mindig a kisebb „ellenállású”, azaz a kisebb súlyozási értékű útvonalat fogja előnyben részesíteni.

A modellben két súlyozási paramétert határoztunk meg, ezek a paraméterek a távolság és a hozzá tartozó idő volt. Ezen adatok meghatározásához a Google Maps rendszerét használtuk, amelynél be lehet állítani, hogy milyen kritérium szerint folytasson keresést: leggyorsabb, legrövidebb, vagy legjobb megítélése szerint a kettőt vegyítve. A harmadik megoldást alkalmazva egy racionális eredményt fogunk kapni a leggyorsabb útvonalról mindkét paraméter figyelembevételével.

Az élek súlyozása mellett fontos megjegyezni, hogy a későbbiekben súlyozási rendszert fogunk bevezetni a csúcspontokban termelődő TSZH mennyiség rögzítésére és annak elszállítási állapotának követésére. Ez a súlyozás az élekkel ellentétben nem konstans, hanem egy változó formájában jelenik meg, aminek

segítségével számszerűsíteni tudjuk, hogy egy adott csúcson mennyi maradék hulladék található az adott pillanatban és ki tudjuk hagyni az útvonalaink közül, ha már ki lett ürítve.

```
#excel fileből kiveszem a szükséges adatokat

import googlemaps
import pandas as pd
import numpy as np
from tqdm import tqdm # progress bar
import googlemaps

gmaps = googlemaps.Client(key = 'AIzaSyB2CKH1EvRHGai4s9d9fRDr175fPYtx0d4')
data = pd.read_excel(r'/Users/Kende/Cloud_Documents/Gyerekek/Kende/Egyetem/BME/MSc/HUHA2 projekt/Számítások/Szállítási optimalizacio.xlsx')
df = pd.DataFrame(data, columns = ['ID', 'Szelessegi fok', 'Hosszusagi fok', 'TSZH termelés (kg/het)']) #definiálom tszh-t is majd későbbre

df = np.asarray(df)
df = df[~np.isnan(df).any(axis = 1)]
print(df)

Outputs are collapsed ...

+ Code + Markdown

routes = []
for kiindulo_pontok in df:
    kiindulo_ID = kiindulo_pontok[0]
    for vegpontok in df:
        vegpont_ID = vegpontok[0]
        if kiindulo_ID != vegpont_ID:
            routes.append([kiindulo_ID, vegpont_ID, kiindulo_pontok[1], kiindulo_pontok[2], vegpontok[1], vegpontok[2]])

routes = np.asarray(routes)
print(routes)
```

13. ábra: Előzőleg kiszámított adatok bekérése és rendezése

Az itt látható kódrészlet (13. ábra) felelős a korábban kiszámolt, zónánként meghatározott TSZH termelési adatok és a hozzájuk tartozó ID azonosító kód lekéréséért és összekapcsolásáért. Első lépésként egy biztonságos, titkosított kapcsolatot kell létrehozni a Google Maps szerverével, amit a későbbiekben az adatok lekéréséhez fogunk alkalmazni.

Ezt követően megadjuk a forrásként használt Excel fájl elérhetőségét, majd a „df”, azaz data frame változóba eltároljuk a kiválasztott oszlopok adatait. Az így kimentett adatok egy része nem szükséges számunkra, – 182 csúcsból csak 100-at alkalmazunk a HUHA 1 vonzaskörzete miatt - így kiválogatjuk az üres cellákat tartalmazó sorokat, hogy ne zavarjanak bele a számításba.

A következő részlegben látható „routes” elnevezésű listába átrakjuk az imént kimentett adatokat a következő logikai vizsgálat során:

- a *df* változóból azon pontokat keressük ki, amelyek a nulladik helyen helyezkednek el, magyarul az ID adatokat. Ezeket az adatokat elmentjük „*kiindulo_ID*” néven.
- Ugyanezt a folyamatot megismétljük és elmentjük őket „*vegpont_ID*” néven. Ez a folyamat alciklusa a felette lévőnek, tehát az összes iterációjának lefutása után lép ki magából és ugrik vissza a felső ciklusba.
- Logikai vizsgálatot végzünk az adatokon, ami során csak azon sorokat írjuk be a *routes* listába, amelyeknél nem egyezik meg a kezdeti és a végponti ID, így kiszűrjük az önmagába tartó útvonalakat.

A beírás után készen áll a listánk az útvonaltervezési lépésre, amelynek kódrészletét itt láthatjuk a **14. ábrán**.

```
import csv

data_2 = pd.read_csv(r'/Users/Kende/Cloud_Documents/Gyerekek/Kende/Egyetem/BME/MSc/HUHA2 projekt/Python_kodok/Google_utvonalak.csv')
df_2 = pd.DataFrame(data_2, columns = ["Kezdo ID", "Veg ID", "Kezdo szelesseg", "Kezdo hosszusag", "Vegso szelesseg", "Vegso hosszusag", "Tavolsag", "Ido"])
df_2 = np.asarray(df_2)

with open("/Users/Kende/Cloud_Documents/Gyerekek/Kende/Egyetem/BME/MSc/HUHA2 projekt/Python_kodok/Utvonal_adatok.csv", 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(["Kezdo ID", "Veg ID", "Kezdo szelesseg", "Kezdo hosszusag", "Vegso szelesseg", "Vegso hosszusag", "Tavolsag", "Ido"])
    tavok = []

    for route in tqdm(routes):
        benne_van = False
        for sor in df_2:
            if route[0] == sor[0] and route[1] == sor[1]:
                benne_van = True
                break

        if not benne_van:
            adatok = gmaps.distance_matrix((route[2], route[3]), (route[4], route[5]), mode = 'driving', departure_time = 'now', traffic_model = 'best_guess')

            #távolság és idő kiválasztása
            tavolsag = adatok['rows'][0][0]['elements'][0]['distance']['text']
            ido = adatok['rows'][0][0]['elements'][0]['duration']['text']
            tavok.append([route[0], route[1], route[2], route[3], route[4], route[5], tavolsag, ido])
            csvwriter.writerow([route[0], route[1], route[2], route[3], route[4], route[5], tavolsag, ido])
```

14. ábra: Valós útvonalak lekérése

A programrészletben létrehozzuk az „*Utvonal_adatok*” fájlt és meghatározzuk a beleírandó adatok oszlopainak elnevezését, illetve egy „*tavok*” nevű listát, amelyet a későbbiekben használni fogunk az adatok tárolására.

A Google Maps distance matrix függvény használatával lekérjük az útvonaladatokat. A függvény használatához meg kell adni a kezdeti és a végponti koordinátákat és lehetőségünk van megadni az utazási módot, időt, és a használatos forgalom modell fajtáját és egyéb információkat. Eredményként megkapjuk az utazási távolságot és a hozzá tartozó utazási időt az alábbi formátumban:

```
{'destination_addresses': ['Haridwar, Uttarakhand, India'],  
'origin_addresses': ['Dehradun, Uttarakhand, India'], 'rows':  
[{'elements': [{'distance': {'text': '56.3 km', 'value':  
56288},  
'duration': {'text': '1 hour 40 mins', 'value': 5993},  
'status': 'OK'}]]], 'status': 'OK'} [11.]
```

(Az itt látható adatsor csak bemutató jellegű, a tanulmány szempontjából nem tartalmaz releváns adatokat).

Ezt a műveletet elvégeztettük mind a 9900 útvonal esetében, amelyeket eltároltunk az „*adatok*” változóban, majd a számunkra lényeges adatokat elmentettük a korábban létrehozott *tavok* listába és kiírtuk egy külön fájlba a könnyebb kezelhetőség érdekében.

5.3. Az optimalizálás logikája

5.3.1. CLASSOK, AZAZ OSZTÁLYOK MEGHATÁROZÁSA

Az alapadatok rögzítése után megkezdhetjük az optimalizálási logika kidolgozását, amely segítségével megtaláljuk a megfelelő útvonalakat.

A logikai folyamat során több esetben fogunk visszatérő parancsokat és műveleteket adott változókhoz rendelni, ezért létrehoztunk pár osztályt, amelyek segítségünkre lesznek ebben.

Az osztály egy kódsablon objektumok létrehozására. Az objektumok tagváltozókkal rendelkeznek, amelyekhez viselkedés/tulajdonság társul. Pythonban egy osztályt a "class" kulcsszóval hozhatunk létre. Az osztály létrehozása után el tudjuk látni tulajdonságokkal, mint például TSZH tartalom, látogatások száma vagy időtartam. Az osztály „fizikai” megtestesüléseként hozhatók létre objektumok, amelyek mind az adott osztályba tartoznak és hordozzák az osztály összes – előzetesen meghatározott- tulajdonságát. Ezt az objektumot ezután az osztály példányának nevezzük.

A feladat ellátására létrehoztunk két osztályt „Csucs” és „Kukasauto” néven.

```
import pandas as pd
import numpy as np
from tqdm import tqdm
import csv
import sys
import matplotlib.pyplot as plt

# ----- CLASSES -----

class Csucs: # class definiálása
    def __init__(self, ID, lat, long, tszh, tavolsag_huhatol): # osztály konstruktor: egy konkrét elem létrehozásakor preset kezdeti értékeket definiál
        self.ID = ID # a self mindenhol be kell adni első paramétereknek a definiálás alatt
        self.tszh = tszh
        self.lat = lat
        self.long = long
        self.tavolsag_huhatol = tavolsag_huhatol
        self.látogatások = 0 # jelzi, hogy hányszor látogattak meg egy csúcspontot

    def update(self, elvitt_szemet):
        self.tszh = self.tszh - elvitt_szemet
        self.látogatások = self.látogatások + 1

class Kukasauto:
    def __init__(self): #hozzá kéne rakni a defhez egy olyan paramétercsoportot, ami megjegyzi, hogy milyen ID-kat látogatott meg a kocsi
        self.kapacitas = 8000
        self.látogatott_csucskok = []
        self.distance = 0
        self.duration = 0
        self.kerulet_beli_tavolsag = 0
        self.tmp_dist = []
        self.tmp_dur = []

    def update(self, elvitt_szemet, distance, duration, látogatott_csucs, kerulet_beli_tavolsag):
        self.kapacitas = self.kapacitas - elvitt_szemet
        self.distance = self.distance + distance
        self.duration = self.duration + duration
        self.látogatott_csucskok.append(látogatott_csucs)
        self.kerulet_beli_tavolsag = self.kerulet_beli_tavolsag + kerulet_beli_tavolsag
        self.tmp_dist.append(distance)
        self.tmp_dur.append(duration)
```

15. ábra: Osztályok létrehozása

A Csucs osztályt használtuk fel a gráf csúcspontjainak meghatározására, több elemét is alkalmazni fogjuk a későbbiekben. A korábbiakban említésre került, hogy a gráf élein kívül a csúcspontjai is súlyozással lesznek ellátva. Ez a

súlyozás az adott csúcsponthoz tartozó zónában egy hét alatt termelődő hulladék mennyiségét tartalmazza. Ez az adat kulcsfontosságú szerepet fog betölteni a későbbiekben az utazási cél meghatározásánál és az onnan elvitt hulladékmennyiség meghatározásánál.

A *Csucs* osztályt az imént részletezett TSZH mennyiséggel együtt a következő tulajdonságokkal láttuk el:

- *ID*: ezzel rögzítjük az adott csúcspont ID értékét
- *tszh*: ezzel rögzítjük a csúcspontban található TSZH mennyiséget. Ez egy dinamikus tulajdonság, tehát a program futása során folyamatosan változik. A hulladékszállító jármű minden egyes látogatása esetén frissítjük a maradék TSZH mennyiségét és ezt az értéket rögzítjük a tulajdonságban, így pontos információt kapunk a zónában található szemét mennyiségéről.
- *lat és long*: a pont koordinátáinak értéke.
- *tavolsag_huhatol*: a HUHA 2 erőműtől vett távolság. Későbbiekben a látogatási sorrend és a megtett úthossz meghatározásánál lesz szükséges.
- *latogatasok*: megadja, hogy a jármű hányszor látogatta meg az adott csúcspontot.

Az osztály meghatározásán belül található még egy „*update*” függvény, amelynek szerepe a *tszh* érték frissítése minden egyes látogatás alkalmával, kivonva belőle az adott körben elszállított hulladékmennyiség tömegét és a látogatások számának növelése eggyel.

A második – általunk alkalmazott – osztály a *Kukasauto* volt, ami a modellezett jármű tulajdonságait tartalmazza:

- *kapacitas*: a hulladékszállító jármű szállítási kapacitása, konstans érték.
A mi esetünkben 8000 kg.
- *distance*: a megtett távolság.
- *duration*: a táv megtételéhez szükséges időtartam.
- *latogatott_csucsok*: egy lista, amelyet az *update* függvénnyel fogunk feltölteni. Az egy körben meglátogatott csúcsok ID értékét tartalmazza.
- *keruleten_beluli_tavolsag*: számosítjuk, hogy egy csúcspontba eljutván hány km-t kell mennie a járműnek a felvett hulladék alapján.

Ebben az osztályban minden tulajdonság dinamikus, frissítésüket és a *latogatott_csucsok* feltöltését szintén egy *update* függvény végzi.

Az erőmű éves befogadóképessége 230 000 tonna TSZH, ami heti 4 423,07 tonna befogadóképességet jelent, ezt az értéket tároljuk el a „*huha_befogadas*” változóban.

5.3.2. ELLÁTOTT ZÓNÁK MEGHATÁROZÁSA

Az osztályok létrehozása után az első feladat a 100 definiált zóna közül kiválasztani azokat, amelyek ellátására képes az erőmű, mivel a 100 zóna összes hulladéktermelése hetente jóval magasabb az imént kiszámolt befogadóképességnél.

```

# ----- LOAD IN DATA -----

def read_location_data(path):
    data = pd.read_excel(path)
    df = pd.DataFrame(data, columns = ['ID', 'Szeleségi fok', 'Hosszasági fok', 'TSZH termeles [kg/het]', 'TSZH linearis suruseg [kg/km]'])
    df = np.asarray(df)
    df = df[~np.isnan(df).any(axis = 1)]
    return df

def read_route_data(path):
    data = pd.read_excel(path)
    tavok = pd.DataFrame(data, columns = ["Kezdo ID", "Veg ID", "Kezdo szelesseg", "Kezdo hosszasag", "Vegso szelesseg", "Vegso hosszasag", "Tavolsag [km]", "Ido [mins]"])
    tavok = np.asarray(tavok, dtype = object)
    return tavok

# ----- GENERATE CSUCSOK -----

def get_csucsok(df, tavok, huha_befogadas):
    huhabol_indul = []
    for R in tavok:
        if R[0] == 1:
            huhabol_indul.append(R)
    huhabol_indul.sort(key = lambda x: x[6])

    csucsok = []
    i = 0
    while huha_befogadas > 0 and i < len(huhabol_indul):
        tavolsag_huhatol = huhabol_indul[i][6]

        ID = huhabol_indul[i][1] # a celpont ID-ja
        terulet = df[np.where(df[:,0] == ID)][0] # kiválasztjuk a celpontot a df-ből
        tszh = terulet[3]

        i = i + 1
        if huha_befogadas - tszh <= 0:
            if i > len(huhabol_indul):
                print('ez soha nem kene kiiradjon :)')
            else:
                huha_befogadas = huha_befogadas - tszh
                if ID != 1: #kiszűröm, hogy huhaból ne menjen huhába
                    csucs = Csucs(terulet[0], terulet[1], terulet[2], tszh, tavolsag_huhatol) #konkrét példány a Csucs osztálynak. (id, tszh, lat, long, tavolság)
                    csucsok.append(csucs)

    return csucsok

```

16. ábra: Ellátott zónák meghatározása és sorba rendezése

A 16. ábrán látható az ellátott zónák meghatározásáért és sorba rendezéséért felelős kódrészlet. A szükséges adatok betöltése után első lépésként megvizsgáljuk, hogy a *tavok*-ban eltárolt adatsorok közül melyik útvonal indul az 1-es ID-ből, ami a HUHA 2 azonosító kódja, majd ezen útvonalakat behelyezzük a „*huhabol_indul*” listába és a távolság alapján növekvő sorrendbe rendezzük őket.

Mivel az erőmű nem tudja ellátni az összes csúcspontot, emiatt ki kell válogatnunk azokat, amelyek a lehető legjobban illeszkednek a maximális befogadóképességhez és a lehető legközelebb helyezkednek el hozzá. A távolsági kritérium nem szorul sok magyarázatra, minél kisebb sugárban – tehát minél közelebbről - tudjuk kielégíteni az erőmű TSZH igényét, annál kevesebb üzemanyagot kell felhasználnunk a szállításra. A befogadóképességhez való illeszkedés alatt azt értjük, hogy ha:

$M_i = M_{i-1} - m$, ahol $M_i \neq R^-$ és M_i minél közelebb van 0-hoz pozitív oldalról.

Ahol:

- M_i = a HUHA 2 i.-ik maradék befogadóképessége
- M_{i-1} = a HUHA 2 i-1-ik maradék befogadóképessége
- m = az adott zóna TSZH termelése
- R^- = negatív racionális számok halmaza

Tehát azt a lehető legközelebbi zóna-csoportot keressük, amely kiszolgálása esetén nem lépjük túl az erőmű befogadóképességét, de azt minél jobban megközelítjük. Ennek vizsgálatát a *while* ciklusban vizsgáljuk egyéb változók létrehozása mellett:

- *tavolsag_huhatol*: a *huhabol_indul*-ból kivesszük a HUHA 2-től számított távolságokat
- *ID*: a zóna azonosítója
- *terulet*: azon sorok a korábban szemléltetett *df* adatsorból, amelyeknél a nulladik oszlopában lévő érték (azaz a kezdeti pont ID-ja) megegyezik az imént meghatározott *ID*-nkal. Ezzel választjuk ki a következő célpontunkat, ahova menni fogunk.
- *tszh*: a *terulet* sor TSZH adatot tartalmazó eleme

Ezután azt vizsgáljuk, hogy a *huha_befogadas* negatív értéket vesz-e fel, ha kivonjuk belőle az kiválasztott zóna *tszh* értékét. Ha igen, az azt jelenti, hogy túllépünk az erőmű befogadóképességén, tehát a tovább folytatjuk a keresést.

Ha a vizsgálat eredménye „nem” lett, akkor kivonjuk a *huha_befogadas*-ból a behozott *tszh* mértékét, ezzel újradefiniáljuk a *huha_befogadas* értékét. Ez a ciklus

addig fut folyamatosan, amíg el nem fogy az összes vizsgálható pont, vagy be nem telt az erőmű kapacitása.

A ciklus végére beépítettünk egy biztonsági kritériumot, ami megvizsgálja az *ID* értékét és csak akkor engedi rögzíteni az eredményeket a „*csucs*” változóba, majd a *csucs* változót a *csucsok* listába, ha az *ID* értéke nem 1. Ez megakadályozza azt, hogy az algoritmus egy 0 távolságú, az erőműből önmagába tartó útvonalat jelöljön ki.

5.3.3. ÚTVONALOPTIMALIZÁLÓ ALGORITMUS

A következő programrészlet az útvonalkereső algoritmus első részét tartalmazza, amely algoritmus több részletben lesz ismertetve a hossza miatt. A programrészlet elején található egy „*distance*” elnevezésű függvény, aminek feladata a HUHA 2 és az éppen kiválasztott csúcspont között meghatározni egy egyenest, majd a meghatározott egyenesre egy normálirányú (merőleges) vektort állítani a vizsgálandó csúcshoz. Ezt a függvényt a keresési tér definiálására és mozgatására fogjuk használni a későbbiekben, így a részletes leírás is ott lesz elolvasható.

A továbbiakban már csak magáról az útvonalkereső algoritmusról fogunk beszélni, illetve magáról az optimalizálásról, ám a feladat komplexitása miatt több ábrán fogjuk tudni szemléltetni (a **18. ábra** az első a három, ezt követő ábra közül).

Az algoritmus megértéséhez először nagyvonalakban szeretnénk bemutatni az alapját képző logikát és a gondolkodásmenetet a könnyebb megértés elősegítésének érdekében.

A korábbiakban meghatároztuk a HUHA 2 befogadóképességét legjobban kielégítő pontok csoportját, ez lett a keresési terünk. Ebben a csoportban 85 csúcspont foglal helyet a 100-ból és ~550 kg híján teljesen lefedik az erőmű maximális befogadóképességét. Ezeken a pontokon belül azonban bárhogy neki lehet állni begyűjteni a hulladékot, nincs semmilyen kezdeti kritérium.

Felvettük a kapcsolatot az FKF-fel és az egyeztetésünk során arra következtetésre jutottunk, hogy jelenleg még nincs szállítási útvonal optimalizációját elősegítő módszerük vagy programjuk a hulladék elszállítására Budapesten. A szállítóokcsik átlagos 73 liter/km-es fogyasztása és a jelenlegi üzemanyagárak mellett éves szinten jelentős megtakarítás érhető el nem csupán gazdasági területen, de környezetkárosító kibocsátások szintjén is.

Ennek okán a kijelölt csúcspontjaink elérését és hulladék begyűjtését egy kidolgozott logika szerint oldottuk meg. Köztudott információ, hogy két pont között a legrövidebb lehetséges útvonal egy – a két pontot összekötő – egyenes. Minden kitérés, ami a két pontot összekötő egyenes vonalától eltér, vagy az egyik egyenesen túlmutat, az többlet úthosszt fog jelenteni.

A csúcspontokat az erőműtől vett távolságuk alapján sorba rendezzük. A hulladékbegyűjtést a kijelölt csúcspontok HUHA 2-től vett legmesszebbi elemétől kezdjük meg, majd fokozatosan haladunk befelé ciklikusan. Ennek az oka, hogy egy csúcspont kiürítése közben szinte biztos, hogy lesz egy olyan ciklus, mikor is a csúcspontban elfogyott már a felvehető hulladék, de a jármű 8 tonnás kapacitása nem telt még meg. Ilyen esetben vissza is térhetne az erőműbe, de az veszteséget jelentene mind időben, mind üzemanyag felhasználásban.

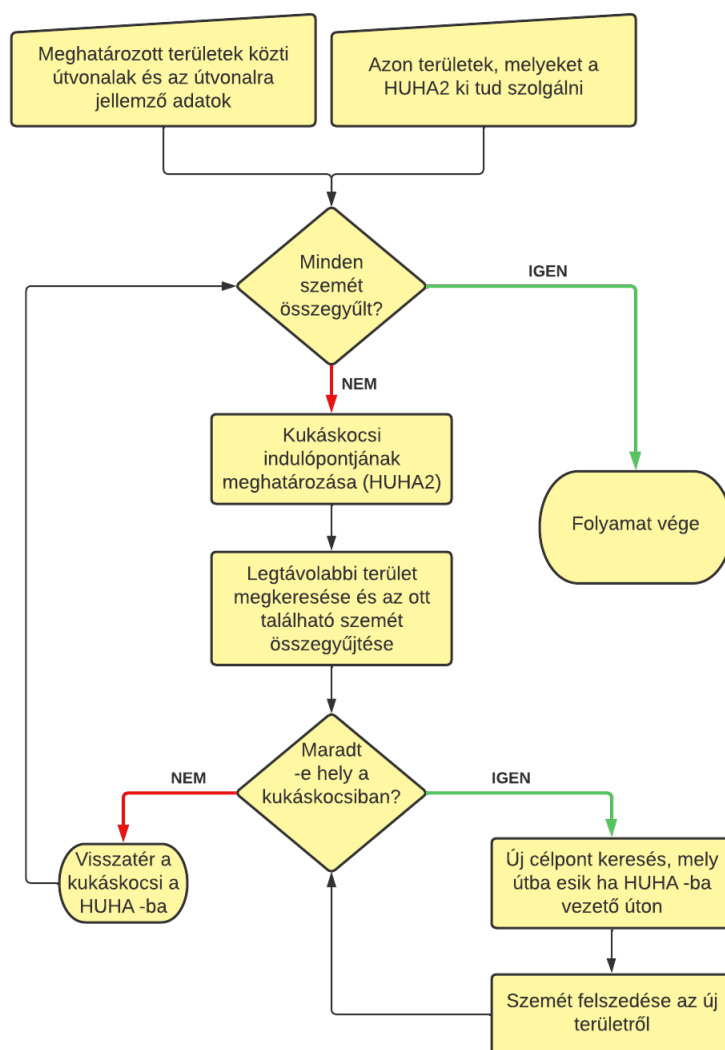
A szabad kapacitással rendelkező jármű az algoritmus által meghatározott kezdő csúcspontjának kiürítése után megkeresi a hozzá legközelebbi – az erőmű irányában elhelyezkedő – csúcspontot, amihez az erőműbe visszafele tartva betér és begyűjti az ott található hulladékot. Ha a járműnek ezt követően

is maradt szabad kapacitása, akkor az algoritmus megkeresi a következő legközelebbi – szintén az erőmű irányában lévő – csúcspontot a jármű számára és oda vezeti. Ezt a procedúrát addig folytatja, amíg meg nem telik a jármű teljes kapacitása, mindezt úgy, hogy folyamatosan a HUHA 2 felé halad a csúcspontok keresése közben. Ezen logika követésével minimalizálható az összes hulladék begyűjtéséhez szükséges utazási távolság.

Azért célszerű a legmesszebb lévő csúcspontból kezdeni a szállítást, mert azokból kiindulva, a visszaúthoz legközelebb eső, a fent említett legrövidebb virtuális egyenes vonalát követve egyéb csúcspontokba eljutni minimális kitérőt jelent. Emellett a következő hulladékbegyűjtési járat során csökkenthetjük annak maximális távolságát, mivel a legtávolabbi ponthalmazokból történő hulladékszállítás során lehetőség nyílik egy - az erőmű irányába eső - sugárirányban elhelyezkedő pont kiürítésére az adott pont direkt megcélzása nélkül.

Ezzel ellentétben, ha a begyűjtést nem a legközelebbi csúcspontból kezdené a jármű, akkor a következő célpont keresése során nagy valószínűséggel elfogunk térni a legrövidebb utat jelentő egyenestől, így nincs lehetőségünk a visszatérési fázisban bizonyos csúcspontokat érinteni, mindegyik csúcsponthoz közvetlen módon kellene eljutni.

Ezen megfontolások alapján az erőműtől legmesszebb található pontokból kezdjük a szállítást és fokozatosan haladunk az erőmű irányába szükség esetén további csúcspontokat meglátogatva. Ez a folyamat addig megy, ameddig mind a 85 zónát teljes mértékben kiürítettük az összes TSZH-t. Az eljárás egyszerűsített verzióját ábráztuk az alábbi folyamatábrán (**17. ábra**).



17. ábra: Útvonaloptimalizálás folyamatábrája

Az eljárás leírását alapján a **18. ábrán** látható kódrészletben logikájának ismertetésével folytatjuk. A többszörösen összetett *while* ciklus első rétege felelős azért, hogy az algoritmus addig fusson, amíg a korábban meghatározott *huha_befogadas* nagyobb nullánál, tehát amíg fel nem szedtük az összes előre meghatározott zóna TSZH tartalmát. Ha ez igaz, akkor beállítjuk a hulladékszállító járművek alapadatait a HUHA 2-ből való indulásra, majd a csúcspontokat csökkenő sorrendbe rendezzük az erőműtől lévő távolságok alapján.

```

# ----- DRIVER CODE -----
#definiálom a distance függvényt, ami meghatározza a HUHA és a célpont közötti egyenest és vizsgálja a második célpont egyenestől vett távolságát (alias csucs)
def distance(HUHA, cel, csucs):
    p1 = np.array([HUHA.lat, HUHA.long])
    p2 = np.array([cel.lat, cel.long])
    p3 = np.array([csucs.lat, csucs.long])
    return np.linalg.norm(np.cross(p2-p1, p1-p3))/np.linalg.norm(p2-p1)

def find_best_routes(HUHA, csucsok, tavok, df, huha_befogadas, r):
    with open("./Eredmenyek/Optimalizalasi_eredmenyek_keruleti_szemettel.csv", 'w') as csvfile:
        csvwriter = csv.writer(csvfile)
        eredmények = []
        csvwriter.writerow(['Latogatott IDk', 'Távolság', 'Időtartam', 'Maradék kapacitás', 'Kerületen belüli távolság'])
        sum_distance = 0
        sum_duration = 0

        while huha_befogadas > 0:
            kukasauto = Kukasauto()
            kukasauto.update(0, 0, 0, HUHA, 0) # a kukásautó változó alapadatainak megadadása (kapacitás, távolság, idő, latogatott csúcsok)
            celpontok = sorted(csucsok, key = lambda x: x.tavolsag_huhatol, reverse = True) # a célpontokat a távolságuk szerint csökkenő sorrendbe rendezem
            i = 0
            while True:
                cel = celpontok[i]
                if cel.tszh > 0:
                    break
                else:
                    i = i + 1

            elvitt_szemet = min(kukasauto.kapacitas, cel.tszh)
            keruleten_beluli_tavolsag = elvitt_szemet / (df[np.where(df[:,0] == cel.ID)][0][4])
            idotartam = tavok[np.where(np.logical_and(tavok[:,0] == HUHA.ID, tavok[:,1] == cel.ID))[0][0]][7]
            kukasauto.update(elvitt_szemet, cel.tavolsag_huhatol, idotartam, cel, keruleten_beluli_tavolsag)
            cel.update(elvitt_szemet)
            huha_befogadas = huha_befogadas - elvitt_szemet

```

18. ábra: Útvonaltervezési algoritmus (1/3)

A következő *while* ciklusban azt vizsgáljuk, hogy az utazási célpontban van-e még tárolt hulladék vagy már elfogyott. Az előbbi esetben a következő célpontot kezdjük vizsgálni. Ezzel biztosítjuk, hogy sose látogassunk meg olyan csúcspontot, ahol már nincs begyűjthető hulladék.

A vizsgálat során néhány változó értékét újra definiáljuk, hogy tartalmazzák a kiszámolt adatokat:

- *elvitt_szemet*: megvizsgáljuk, hogy a kukásautó maradék kapacitása a kisebb (amit folyamatosan frissítünk minden meglátogatott csúcs után) vagy a tartózkodási csúcsban lévő TSZH mennyisége. Ezek közül a kisebbet választjuk ki és rendeljük hozzá az *elvitt_szemet*-hez, mivel logikusan ennél többet nem tudunk elszállítani.
- *keruleten_beluli_tavolsag*: az elvitt szemet mennyiségét elosztjuk a korábban kiszámolt – az adott csúcsponthoz tartozó - lineáris szemétsűrűséggel, ezzel megkapjuk, hogy mekkora távolságot tettünk meg a zónában a hulladék begyűjtése közben.

$$s_{zóna} = \frac{m_{elvitt}}{f_{TSZH}}$$

Ahol:

- $s_{zóna}$ = a zónában megtett úthossz [km]
- m_{elvitt} = elvitt hulladék tömege [kg]
- *idotartam*: az csúcspontokhoz való utazási időt adja meg. Megkeressük azon adatsorokat a *tavok*-ban, ahol a kezdő ID a HUHA 2-é, a vég ID pedig az adott csúcs ID-ja, majd innen kivesszük a köztük lévő utazási időt.

A *kukasauto*-t és a célpontot frissítjük az itt definiált adatokkal és csökkentjük az erőműbe vihető szemét mennyiségét a felvett szemétmennyiséggel.

A fenti ciklusban megvizsgáltuk azon eseteket, amelyekben a járművek az első csúcspontban begyűjtött hulladék a jármű teljes kapacitását igénybe veszi. Következő lépésként azon esetekkel fogunk foglalkozni, amelyekben a jármű az első csúcspontban történt látogatása után rendelkezik felhasználható szállítókapa-
citással.

Amennyiben közvetlenül az erőműbe indulna, jelentős gazdasági és környezeti veszteségeket tapasztalnánk hosszútávon. Ezt az opciót a későbbiekben mutatjuk be.

Érdemes megjegyezni, hogy a későbbiekben található „*cel*” változó továbbiakban a jármű tartózkodási pontját fogja jelenteni, tehát azt a csúcspontot, ahol legutóbb begyűjtési feladatot végzett a jármű. Ez a változó minden egyes meglátogatott csúcspont után újra definiálódik és felveszi a tartózkodási csúcspont értékeit. (látható a **19. ábrán**).

```

if kukasauto.kapacitas > 0 : # addig szeretnénk újabb csúcsokat kiválasztani, amíg van még hely a kukásautóba
    while kukasauto.kapacitas > 0:
        legkozelebbi_csucs = cel #mindig amikor új keresés indul, akkor az lesz a legközelebbi csúcs, ahol vagyunk
        region_of_interest = []

        for csucs in csucsok:
            d = distance(HUHA, cel, csucs)

            if d < r and csucs.tszh > 0:
                region_of_interest.append(csucs)

        if len(region_of_interest) > 0: #ha van olyan csúcs, ami beleesik a keresési tartományba
            minimum = 40 #ennél a számnál kell kisebbet keresni a ciklusban
            for csucs in region_of_interest:
                celtol_valo_tavolsag = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:,1] == csucs.ID))[0][0]][6]
                if celtol_valo_tavolsag < minimum and csucs != cel:
                    minimum = celtol_valo_tavolsag
                    legkozelebbi_csucs = csucs
            if legkozelebbi_csucs == cel: # ha önmagába akarnak menni, mint megközelebbi 2. csúcs, akkor ne menjen oda
                tavolsag_huhatol = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:,1] == HUHA.ID))[0][0]][6]
                idotartam = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:,1] == HUHA.ID))[0][0]][7]
                kukasauto.update(0, tavolsag_huhatol, idotartam, HUHA, 0)
                break
            else: #találtunk egy olyan csúcsot, ahova el tudunk menni
                elvitt_szemet = min(kukasauto.kapacitas, legkozelebbi_csucs.tszh)
                kerulet_beluli_tavolsag = elvitt_szemet / (df[np.where(df[:,0] == cel.ID)][0][4])
                idotartam = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:,1] == legkozelebbi_csucs.ID))[0][0]][7]

                cel = legkozelebbi_csucs #mostmár a második állomás lett a cél, és innen tud keresni további (3.) pontot, ahol fel tud szedni TSZH-t
                kukasauto.update(elvitt_szemet, minimum, idotartam, legkozelebbi_csucs, kerulet_beluli_tavolsag)
                legkozelebbi_csucs.update(elvitt_szemet)
                huha_befogadas = huha_befogadas - elvitt_szemet

        else: #elmegy a legtávolabbi pontba, felszedi majd visszatér
            tavolsag_huhatol = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:,1] == HUHA.ID))[0][0]][6]
            idotartam = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:,1] == HUHA.ID))[0][0]][7]
            kukasauto.update(0, tavolsag_huhatol, idotartam, HUHA, 0) # felvett szemet, távolság a huhatól, időtartam, érkezési hely
            break

```

19. ábra: Útvonaltervezési algoritmus (2/3)

Azon helyzet elkerülésének érdekében, hogy a jármű részlegesen kihasznált kapacitással idő előtt visszatérjen az erőműbe, egy logikai vizsgálatot hajtunk végre. Fennmaradó szabad kapacitás esetén a következőkben leírt algoritmus lép életbe:

Egy *while* ciklust indítunk, ami addig fog futni, ameddig maradt kapacitás a járműben, illetve létrehozuk a következő változókat:

- *legkozelebbi_csucs*: újra definiáljuk a jelenlegi tartózkodási helyünkhöz a keresési paraméterek (később kifejtett) szerint legközelebb található csúcspont adatait, mivel ez lesz a következő célpontunk.
- *region_of_interest*: lista, amely azon csúcsokat fogja tartalmazni, amelyek megfeleltek a keresési feltételnek. Ezekből a pontokból fogja kiválasztani az algoritmus a következő célpontot.

- r : a korábban meghatározott – a tartózkodási csúcsot és a HUHA 2-t légvonalban összekötő egyenesre merőleges vektor hossza. Kezdeti értéke 0, mert fokozatosan növelni fogjuk.

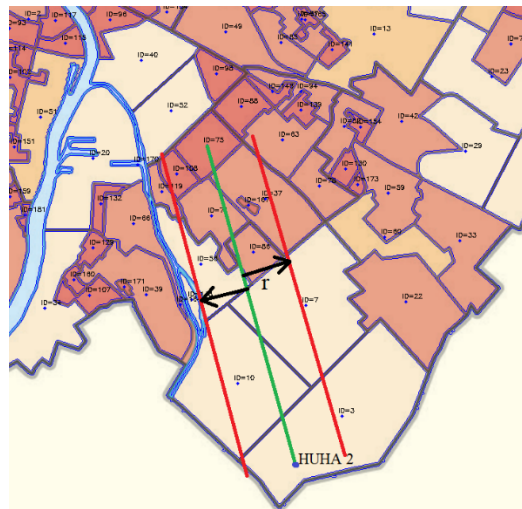
A következő *while* ciklus a korábbi *while* ciklus beágyazott eleme és addig fut, amíg *region_of_interest* hossza megegyezik nullával, azaz nem talált olyan csúcspontot a program, ami megfelelné a keresésnek. Ha nem talált olyan pontot, ami beleesne a keresési tartományba, akkor egy adott értékkel növeljük az r vektor kezdeti hosszát, tehát ezzel együtt a keresési tér méretét. Ezzel a növeléssel a HUHA 2-t és a tartózkodási csúcspontot összekötő lineáris egyenessel párhuzamosan két irányba egy r szélességű keresési sávot határozunk meg, amelyet folyamatosan szélesítünk egy bizonyos méretig, a könnyebb érthetőség kedvéért a **20. ábrán** ábrázoltuk a sáv elméletét.

A fentebb leírt cikluson belül indítunk egy *for* ciklust, amely végigfut az összes csúcson és mindegyikhez létrehoz egy „ d ” változót, amely minden iteráció esetén meghívja a korábban megírt *distance* függvényt. A megírt *distance* függvény segítségével létrehoz egy új – a korábban meghatározott, a tartózkodási csúcspont és a HUHA 2 közé húzott lineáris egyenesre – merőleges vektort, amely a következő csúcspontba mutat.

Ezen belül egy logikai vizsgálatot indítunk, ami azt felügyeli, hogy az imént létrehozott d vektor értéke kisebb-e, mint a keresési tér korábban meghatározott r értéke. Ha igen, akkor az adott csúcspontot behelyezi a *region_of_interest* listába: megtaláltuk az első alkalmas csúcspontunkat.

Ezzel azokat a pontokat válogatjuk ki, amelyek beleesnek a definiált keresési sávba, tehát elég közel helyezkednek el az optimális visszatérési útvonalhoz, minimalizálva a kitérés mértékét.

Az alant látható rajz (20. ábra) csak vizualizáció, valós összeköttetést nem feltétlenül jelent. Zöld vonallal jelöltük a két pontot légvonalban összekötő egyenest, amelyre a merőlegesen állított r vektor fekete színben látható. Piros színnel jelöltük a keresési sáv külső határát, amit nem léphet túl. A zöld és a piros vonalak között minden egyes iterációval halad kifelé az r vektor, így növelve a keresési teret.



20. ábra: Keresési sáv vizualizálása

A *for* ciklus végeztével egy újabb logikai vizsgálatba kezdünk (szintén a 19. ábra), ami azt kutatja, hogy az r vektor hossza meghaladja-e az előre megadott keresési sugarat. Ha ez megtörténik, akkor kilép a ciklusból és egy újabb logikai vizsgálatba érkezik.

Ezen ciklus vizsgálja, hogy a *region_of_interest* hossza nagyobb-e nullánál, tehát talált-e alkalmas csúcspontot. Ha talált, akkor létrehozuk a *minimum* nevű változót:

- *minimum*: egy tetszőleges kezdeti értékkel megadott változó, összehasonlításhoz szükséges.

A *region_of_interest* listát érintő logikai vizsgálaton belül egy *for* ciklust indítunk, amely az összes csúcson végig fut a keresési zónában

- *celtol_valo_tavolsag*: változó, amiben eltároljuk a tartózkodási csúcspont és a hozzá tartozó sávban található csúcspontok közötti távolságot.

Újabb vizsgálatot végzünk azzal kapcsolatban, hogy a *celtol_valo_tavolsag* kisebb-e a *minimum* értéknél és a cél, illetve a csúcs nem egyezik meg (tehát nem önmagát találja meg). A *minimum* értéke az első iterációnál még tetszőlegesen választott, úgyis újra lesz definiálva az iteráció során.

Ha a fenti feltételek teljesülnek, akkor a *minimum* értékét újra definiáljuk a *celtol_valo_tavolsag* értékével, tehát ettől fogva ez lesz az a paraméter, aminél kisebbnek kell lennie egy távolságnak és a vizsgált csúcspont értéke lesz a *legkozelebbi_csucs* új értéke. Ez annyit tesz, hogy kijelöltük a jelenlegi ismeretünk szerinti legközelebbi pontot a sávon belül. Ez természetesen bármikor felülírható, amennyiben a ciklus talál a korábbi csúcspontnál közelebbi csúcspontot. A ciklus addig fut, amíg az összes sávon belüli ponton végig nem megy.

Ezen ciklus segítségével meghatároztuk, azon hogy a keresési sávon belül található csúcspontok közül melyik van a legközelebb a jármű jelenlegi tartózkodási helyéhez valós útvonalak használatával.

A *for* ciklusból kilépve egy újabb logikai ellenőrzést végzünk el, ami azt vizsgálja, hogy a kiválasztott *legkozelebbi_csucs* megegyezik-e a *cel*-l, tehát a jelenlegi tartózkodási pontjával. Itt azon esetet vizsgáljuk, mikor az algoritmus nem talált más pontot az ismert jelenlegin kívül. Ebben az esetben visszairányítjuk közvetlenül az erőműbe, valamint a *kukasauto* elem változói a következő módon módosítjuk:

- *tavolsag_huhatol*: az erőmű és a jelenlegi pont közötti úttávolságot veszi fel
- *idotartam*: a megtételhez szükséges időtartamot veszi fel

Ha az előbbi vizsgálat hamis (azaz új meglátogatandó csúcspont megegyezik a jelenlegi tartózkodási hellyel), akkor az alábbiak módosulnak:

- *elvitt_szemet*: megvizsgáljuk, hogy a kukásautó maradék kapacitása a kisebb vagy a tartózkodási csúcsban lévő TSZH mennyisége. Ezek közül a kisebbet választjuk ki és rendeljük hozzá *elvitt_szemet*-hez.
- *keruleten_beluli_tavolsag*: az elvitt szemét mennyiségét elosztjuk az adott csúcsponthoz tartozó lineáris szemétsűrűséggel, ezzel megkapjuk, hogy mekkora távolságot tettünk meg a zónában a hulladék begyűjtése közben.
- *idotartam*: az csúcspontokhoz való utazási időt adja meg. Megkeressük azon adatsorokat a *tavok*-ban, ahol a kezdő ID a *cel*-é (a jelenlegi tartózkodási helyünk), a vég ID pedig az adott csúcs ID-ja (ahova szeretnénk menni), majd innen kivesszük a köztük lévő utazási időt.
- *cel*: újra definiáljuk a *legkozelebbi_csucs* értékével. A korábbiakban említettük, hogy a *cel* változó mindig a tartózkodási lokáció csúcspontját veszi fel, a következő csúcspont pedig a *legkozelebbi_csucs* lesz. A két változó értékének egyeztetésével azt jelezzük, hogy a jármű eljutott az azonosított csúcspontba és az algoritmus újra kezdheti a keresést, amennyiben továbbra is van szabad kapacitása.

A *kukasauto* és a *legkozelebbi_csucs* változókat (a jármű jellemzőit és a tartózkodási csúcsponton tárolt hulladék mennyiségét) frissítjük az itt definiált adatokkal és csökkentjük az erőműbe vihető hulladék mennyiségét a felvett szemétmennyiséggel.

Az eddig leírt ciklusok és logikai vizsgálatok mind az *if len(region_of_interest) > 0* feltételű logikai ellenőrzés szubrutinjai voltak. Ebből a feladatból kilépve azt az esetet kell meghatározni, amikor a

$len(region_of_interest) = 0$, tehát a keresési sávban tartózkodó, TSZH tartalommal rendelkező csúcspontok száma nulla. Ebben az esetben a járművet közvetlenül az erőműbe irányítjuk vissza, mivel a keresési sávon kívül eső csúcspontok meglátogatása túl nagy távolság megtételét tette volna szükségessé, valamint a *kukasauto* elem változói a alábbi módon változnak:

- *tavolsag_huhatol*: az erőmű és a jelenlegi pont közötti úttávolságot veszi fel
- *idotartam*: a megtételhez szükséges időtartamot veszi fel

A frissítések értelmében az *elvitt_szemet* (újonnan begyűjtött hulladékmennyiség) 0 kg és a *keruleten_beluli_tavolsag* (zónán belül megtett távolság) is 0 km. Ezek után kilépünk az egész ciklusból a következő logikai elemzéshez.

```

else: #elmegy a legtávolabbi pontba, felszedi majd visszatér
    tavolsag_huhatol = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == HUHA.ID))[0][0]][6]
    idotartam = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == HUHA.ID))[0][0]][7]
    kukasauto.update(0, tavolsag_huhatol, idotartam, HUHA, 0) # felvett szemét, távolság a huhától, időtartam, érkezési hely)
    break

if kukasauto.kapasitas == 0:
    tavolsag_huhatol = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == HUHA.ID))[0][0]][6]
    idotartam = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == HUHA.ID))[0][0]][7]
    kukasauto.update(0, tavolsag_huhatol, idotartam, HUHA, 0)

latogatott_csucskok = ""
i = 0
for i, latogatott_csucs in enumerate(kukasauto.latogatott_csucskok):
    if i == len(kukasauto.latogatott_csucskok) - 1:
        latogatott_csucskok = latogatott_csucskok + str(latogatott_csucs.ID)
    else:
        latogatott_csucskok = latogatott_csucskok + str(latogatott_csucs.ID) + " -> "

eredmenyek.append([kukasauto.latogatott_csucskok, kukasauto.distance, kukasauto.duration, kukasauto.kapasitas, kukasauto.keruleten_beluli_tavolsag])
csvwriter.writerow([latogatott_csucskok, kukasauto.distance, kukasauto.duration, kukasauto.kapasitas, kukasauto.keruleten_beluli_tavolsag])
x = 0
for csucsi in csucskok:
    if csucsi.tszh == 0:
        x = x + 1

sum_distance = sum_distance + kukasauto.distance + kukasauto.keruleten_beluli_tavolsag
sum_duration = sum_duration + kukasauto.duration

maradek_szemet_sum = 0
for maradek in csucskok:
    maradek_szemet_sum = maradek_szemet_sum + maradek.tszh

if maradek_szemet_sum == 0:
    break

return sum_distance, sum_duration

```

21. ábra: Útvonaltervezési algoritmus (3/3)

Ez a logikai kapu az utolsó az algoritmusunkban és az eddig leírt ciklusokon kívül helyezkedik el, szerepe vizsgálni azt, hogy a kukásautónak van-e fennmaradó szabad kapacitása. Ha ez az állapot bekövetkezik, akkor a következő változtatásokat eszközöljük:

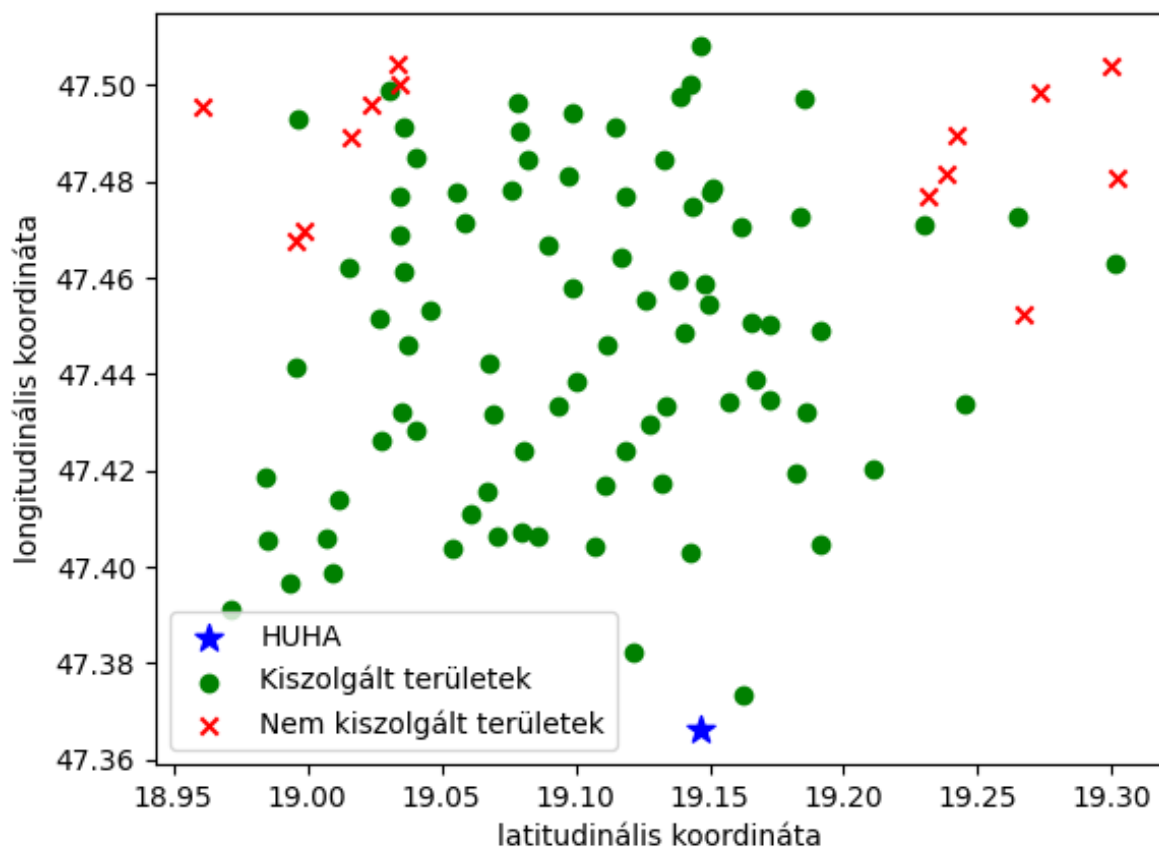
- *tavolsag_huhatol*: az erőmű és a jelenlegi pont közötti úttávolságot veszi fel
- *idotartam*: a megtételhez szükséges időtartamot veszi fel

emellett az algoritmus közvetlenül az erőműbe irányítja a kukásautót, mivel az nem tud több hulladékot begyűjteni.

A fent leírt lépéseket elemző és végrehajtó algoritmussal lehetővé tettük az optimális útvonal megtalálását a program számára. Végző lépésként a kapott adatokat kiíratjuk egy CSV formátumú adatállományba. A továbbiakban nem mutatjuk be a diagramok vizualizációjáért és az egyéb iterációs lépésekért felelős álló programokat, viszont a tanulmány végére csatoljuk a teljes programkódot, hogy teljes képet adhasson a megoldásról.

6. EREDMÉNYEK KIÉRTÉKELÉSE

A korábbiakban említettük, hogy szükségünk van azon zónák listájára, amelyek kiszolgálására képes az erőmű. Ezen zónák csúcspontjait meghatároztuk az algoritmusban leírtak szerint, az elemzés eredménye képpen 85 zónát tudtunk elkülöníteni a kezdeti 100 definiált csúcspontból. Ezen zónákat a szemléletesség kedvéért ábrázoltuk az alábbi diagramon földrajzi koordináták szerint (22. ábra).



22. ábra: A kiválasztott zónák elhelyezkedése

Jól azonosítható az ábrán a keresési paramétereknek megfelelő kiválasztási módszer. Az első feltétel a lehető legközelebbi lokációkat kereste, a másik pedig azt vizsgálta, hogy az adott zóna TSZH termelésének befogadásával nem haladnánk-e meg az erőmű befogadóképességét. Ez az oka annak, hogy némelyik pont nem kiszolgáltként van jelezve annak ellenére, hogy közelebb helyezkedik el az erőműhöz, mint egy másik kiszolgált csúcspont.

Azt is fontos megemlíteni, hogy az algoritmus nem a légvonalbeli távolságot veszi alapul a számításhoz, hanem a valós úthálózati távolságokat, – az időfaktorral súlyozva – így a fenti ábrán található távolságok nem mindig reprezentatívak.

Abban az esetben, ha az első célpontot meglátogatva még maradt szabad kapacitás a kukásautóban, akkor definiáljuk a korábban említett r szélességű keresési sávot. Kezdeti értéknek az $r = 0$ értéket határozzuk meg, majd fokozatosan szélesítjük a keresési sávunkat 0,01-es lépésmérettel ciklusonként. Az r vektor és a léptetés mértékegysége is koordináta fokban értendő. Az eredmények első kiértékelésekor ezzel a léptetéssel számoltunk és meghatároztunk előre egy szélességi maximumot értéket, amelyet elérve a sáv nem növekedik tovább. Ez határérték a jelenlegi tartózkodási csúcspont és a cél csúcspont közötti valós távolság harmadával volt egyenlő fokban kifejezve.

Az első számításnál az imént leírt iterációs módszert alkalmaztuk az r vektor hosszának meghatározására, ám ezt a módszert a későbbiekben meg fogjuk változtatni egy fix r értékre, amely a legrövidebb megtett úthosszt eredményezi.

Kiszámoltuk Budapest vonalában a Föld szélességi körének területét, majd ezt felosztva 360 részre megkaptuk, hogy egy fok hány valóságos kilométert jelent ezen a szélességi körön, így át tudtuk váltani a kilométerben meghatározott távolságértéket koordináta fokba:

$$F_{fok} = \frac{2 * r_{Föld, egyenlítő} * \cos \alpha * \pi}{360^\circ}$$

$$F_{fok} = \frac{2 * 6371 \text{ km} * \cos 47,498970 * \pi}{360^\circ}$$

$$F_{fok} = 75,124 \frac{\text{km}}{1^\circ}$$

Ahol:

- $r_{Föld, egyenlítő}$ = a Föld egyenlítői sugara [km]
- α = szélességi fok Budapest magasságában [°]
- F_{fok} = egy fok kilométerben megadott értéke Budapest ívén [km/1°]

$$r_{max} = \frac{s_{1,2}}{3} * F_{fok}$$

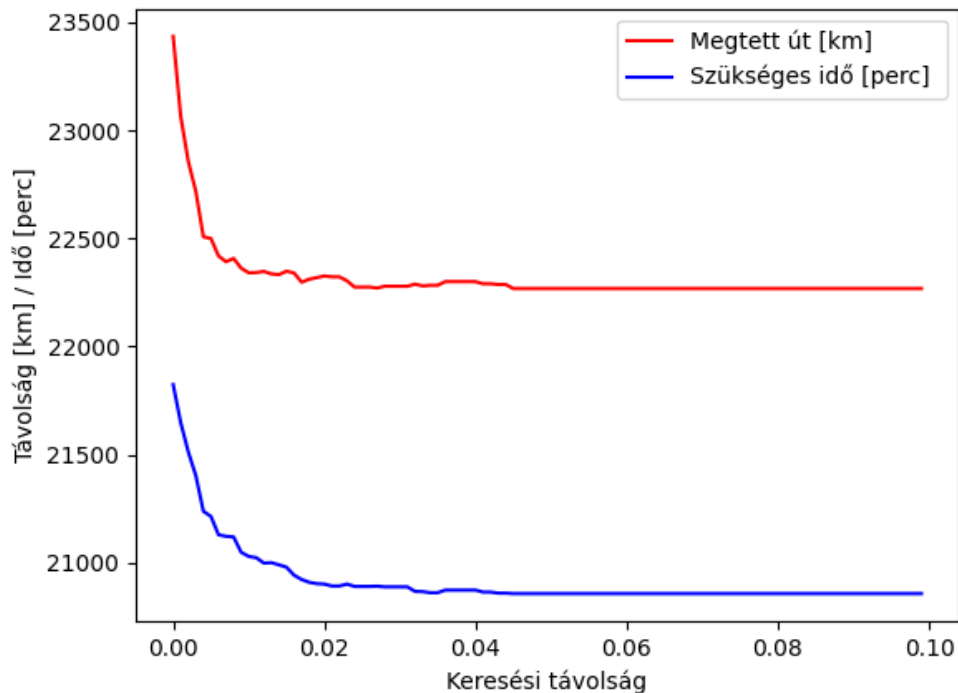
Ahol:

- r_{max} = keresési sáv maximum nagysága [°]
- $s_{1,2}$ = tartózkodási csúcspont és a cél csúcs közötti távolság [km]

Ezzel a módszerrel megkaptuk ugyan az eredményeket, ám a célunk a korábban említett optimális keresési sáv meghatározása volt, amely alkalmazásával a lehető legkevesebb kilométert kell megtennie a járműveinknek összességben.

Ennek a célnak az eléréséhez létrehoztunk egy függvényt, ami 150 alkalommal futtatta végig az egész programot, miközben az összes kilométer adat rögzítésre került. Az iterációk során a korábban meghatározott maximális sáv szélességnél jóval nagyobb méretűre engedték az r vektort futni. A megtett

távolság mellett az időtartamokat is rögzítettük, majd az összes kapott eredményt az alábbi diagramon rögzítettük (23. ábra).



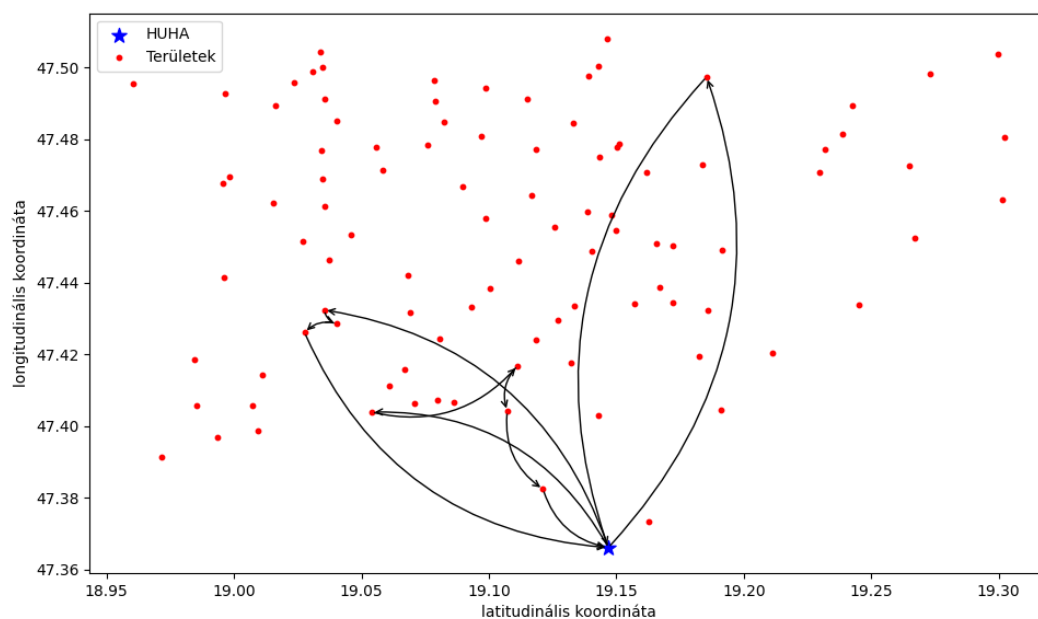
23. ábra: Az összes megtett távolság és a hozzá tartozó idő ábrázolása

A diagramon piros színnel láthatjuk a megtett távolság adatokat és kék színnel láthatjuk az utazási időtartam adatokat. A függőleges tengelyen található jelzés nem osztást szimbolizál, hanem a megosztott tengely két változóját mutatja be. Látható az ábrán, hogy mindkét függvény meredeken csökkenni kezd a keresési távolság növelésével, majd a meredekség fokozatosan mérséklődik és végül a keresési távolság $\sim 0,04$ értéknél a függvények állandó értéken stabilizálódnak. A kezdeti nagy meredekség annak köszönhető, hogy egyre több közeli pontot tud figyelembe venni az újra és újra futtatott algoritmus. A keresési távolság további növelésének jelentősége egyre csökken, mivel csak kiválasztható pontokhoz képest csak távolabbi pontokat tudna alkalmazni a program.

A kirajzoltatás után kiszámoltuk a függvény minimum értékét, amelynek eredménye a keresési távolság $r = 0,045$ ponthoz tartozó függvényérték lett. Ezen pont a statikus szakasz első eleme. Ezt az optimumértéket alkalmazva átalakítottuk az algoritmus keresési sávért felelős részét, a maximális távolságra az itt kiszámolt optimum értéket állítottuk be a keresési sáv stabil szélességeként, ezáltal garantáljuk, hogy a lehető legrövidebb útvonalat követik a járművek.

Természetesen állíthattuk volna nagyobbra is a vektor értékét, ám a diagram végső, statikus szakasza alapján látható, hogy nem hozott volna változást az eredményben, így a program erőforrásigényének és futási sebességének csökkentésének érdekében $r = 0,045$ lett a határérték.

Az útvonaloptimalizálás eredményeként megkaptuk az egyes járművek adott körének látogatási sorrendjét, aminek segítségével ábrázolni tudtunk néhány jármű által megtett útvonalat is a **24. ábrán**.

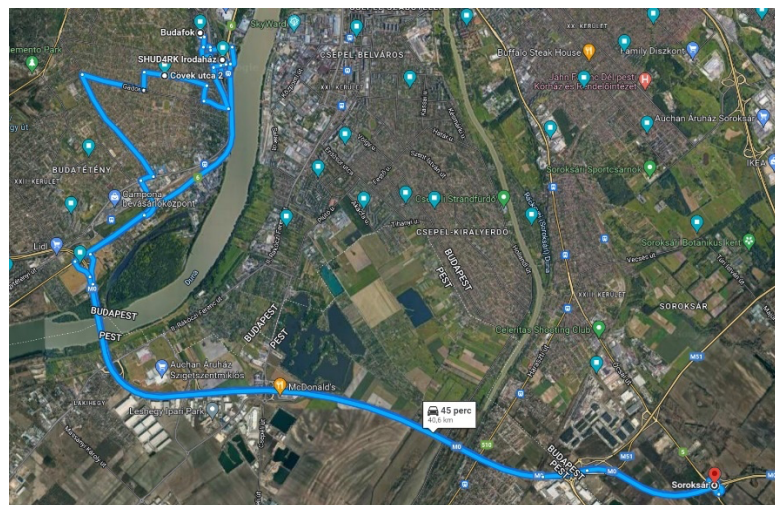


24. ábra: Három hulladékszállítási útvonal vizualizációja

A diagramon a hosszúsági és a szélességi koordináták változása szerint ábrázoltuk a csúcspontok elhelyezkedését, majd kirajzoltattuk a 3 jármű útvonalát. Jól látható, hogy az algoritmus adott esetben egytől akár négyig terjedő csúcspontot is meglátogat hulladék begyűjtés céljából egy körön belül. A három ábrázolt út látogatási sorrendje ID értékek szerint csoportosítva a következő:

- 1.0 -> 16.0 -> 1.0
- 1.0 -> 159.0 -> 181.0 -> 14.0 -> 1.0
- 1.0 -> 34.0 -> 56.0 -> 145.0 -> 10.0 -> 1.0

Az 1. ID a HUHA 2 azonosítója. Ezek az útvonalak a korábbiakban ismertetett okokból sematikusak, a valós számítások során a **25. ábrán** jelölt útvonalhoz hasonló útvonalakat követtek a járművek:



25. ábra: Egy útvonal vizualizálása a valós út alapján [12.]

A kapott eredmények megadták a hulladékszállító járművek látogatási sorrendjét a csúcspontok között, mekkora távolságot tettek meg a csúcspontig való eljutás során, mennyi időbe került eljutni a csúcspontokig, mekkora szabad kapacitás maradt a járművekben az erőműbe való visszatéréskor és a zónán

belül – hulladékfelszedés során - megtett távolságot. Ezek az adatok alapján kiszámoltuk az elfogyasztott üzemanyag mennyiségét és az ezzel járó gazdasági kiadásokat is, majd minden eredményt összegeztünk. Az eredmények közül párat a **3. táblában** szemléltettünk:

Latogatott ldk [-]	Távolság [km]	Időtartam [mins]	Maradék kapacitás [kg]	Kerületen belüli távolság [km]	Összes távolság [km]	Üzemanyag fogyasztás [l]	Költségek [Ft]
1.0 -> 16.0 -> 1.0	83,30	66,00	0,00	25,27	108,57	79,26	64 992
1.0 -> 16.0 -> 1.0	83,30	66,00	0,00	25,27	108,57	79,26	64 992
1.0 -> 16.0 -> 112.0 -> 1.0	70,40	71,00	0,00	25,27	95,67	69,84	57 270
1.0 -> 64.0 -> 1.0	62,00	48,00	0,00	4,71	66,71	48,70	39 932
1.0 -> 64.0 -> 1.0	62,00	48,00	0,00	4,71	66,71	48,70	39 932
1.0 -> 64.0 -> 1.0	62,00	48,00	0,00	4,71	66,71	48,70	39 932
1.0 -> 64.0 -> 1.0	62,00	48,00	0,00	4,71	66,71	48,70	39 932
1.0 -> 64.0 -> 1.0	62,00	48,00	0,00	4,71	66,71	48,70	39 932
1.0 -> 64.0 -> 54.0 -> 1.0	62,70	51,00	0,00	4,71	67,41	49,21	40 351

3. tábla: A kapott eredmények egy részlete

A modellünk eredményeként mind az 553 járat tele kapacitással érkezett meg az erőműbe, miközben minimalizálta az megtett útvonal hosszát. Ez alól kivétel volt az utolsó jármű, amelynek ~1200 kg szabad kapacitása volt - a 8000 kg-os tárolójából - az erőműbe érkezéskor, mivel a kijelölt zónákban nem volt több begyűjthető hulladékmennyiség.

Ezek alapján minden egyes útnál meg tudtuk határozni az üzemanyag fogyasztás mértékét és a hulladék fajlagos beszállítási költségét 8 tonna TSZH-ra vetítve. Az összes költséget összegezve és elosztva az összes járattal megkapjuk, hogy 8 tonna TSZH beszállítása átlagban **24 105 Ft/8 tonna** lett. Ugyanezen költség a 2017-es HUHA 2 előkészítő tanulmány jelenlegi üzemanyag árakra arányosított ár-bebecslése szerint 20 074 Ft/tonna, azaz 160 592 Ft/8 tonna szállítási költséget jelent. Összehasonlításképp ez több, mint hatszoros különbséget jelent.

Természetesen megjegyzendő, hogy az általunk számolt értékek nem tartalmazzák az FKF Kft. üzemeltetési, karbantartási és személyzeti költségeit, amelyek megnövelik a teljes költségalmazt. Ezt is figyelembe véve észrevehető különbséget tapasztalhatunk a két összeg között.

Számszerűsíteni kívántuk az optimalizáló algoritmus segítségével elért üzemanyag és pénzmennyiség megtakarítás mértékét, így felállítottunk egy olyan modellt is, ahol az összes jármű kizárólag direkt útvonalak mentén közlekedik és nem gyűjt be több csúcspontból hulladékot. Ez azt jelenti, hogy az első meglátogatott csúcspontban történt hulladék begyűjtés a jármű minden esetben visszatér az erőműbe attól függetlenül, hogy a begyűjtött hulladék mennyisége megtelítette-e a kapacitását. Ezen modellt lefuttatása és eredményeinek kiértékelését követően a kapott eredményeket az alább található **4. táblában** összegeztük.

Optimalizálás hatása a költségekre		
	Direkt útvonalak	Optimalizált útvonalak
Távolság [km]	23 477,11	22 268,31
Időtartam [min]	21 824,00	20 857,00
Üzemanyag fogyasztás [l]	17 138,29	16 255,87
Költségek [Ft]	14 053 398,59	13 329 810,91
Havi megtakarítás [Ft]	2 894 351	

4. tábla: Havi költségmegtakarítás mértéke

A táblázatban összevetettük a két modell heti operációs költségeit, majd ezek alapján meghatároztuk egy hónapos időtartam alatt elért megtakarítási potenciálunkat. Eredményképp kijelenthető, hogy az útvonal optimalizálásunkkal rendszerszinten az FKF Kft. havonta **2,894 millió Forintot** tudna megtakarítani.

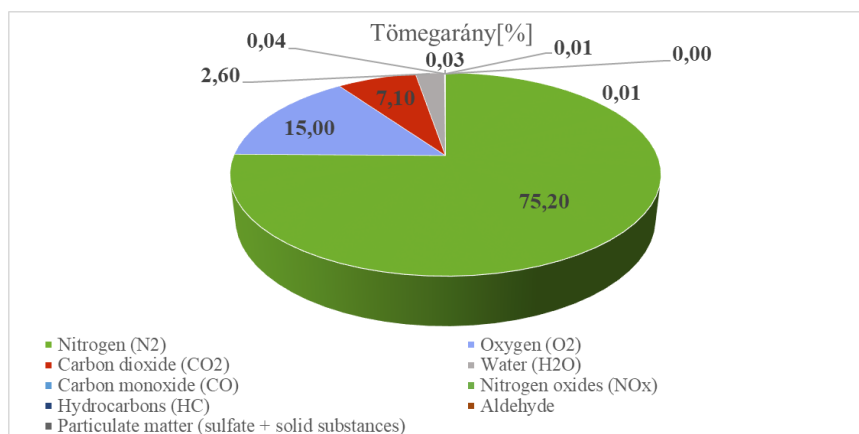
6.1. Kibocsátási értékek

A korábbiakban érintőlegesen foglalkoztunk a belsőégésű motorok, jelen esetben a dízel motorok által az atmoszférába kiengedett szennyező anyagokkal

és a keletkező melléktermékekkel, mint pl. a CO₂ és az NO_x vegyületek. Ezen anyagok lokális és globális problémákat okoznak mind az emberi és állati egészségre nézve, mind pedig a globális felmelegedést tekintve az üvegházhatású tulajdonságuk miatt.

Mivel a budapesti hulladékszállító járműállomány szinte kizárólagosan dízelüzemű járművekből áll, érdemesnek tartottuk kielemezni, hogy ilyen téren milyen kibocsátási értékekre számíthatunk a direkt és az optimalizált útvonalak tekintetében is, illetve fényt derítettünk arra, hogy miért lehet kifizetődő alternatív üzemanyagokra váltani a jövőben ebben a szektorban.

A korábbi számításaink alapján, illetve a dízel üzemanyag égési tulajdonságainak ismeretében ki tudjuk számolni a kibocsátási adatokat. A dízel fajlagos CO₂ kibocsátás értékét $\mu = 2,68 \text{ kg (CO}_2\text{)/l}$ -nek vettük. [13.] Az alábbi képen ábrázoltuk a kipufogógázok összetételét a jelen esetben:



26. ábra: A dízel üzemanyag kipufogógázának összetétele [13.]

Látható a diagramon, hogy a gázok nagy részét az elemi nitrogén teszi ki, számunkra azonban a piros színnel jelölt CO₂ és a szabad szemmel nehezen látható NO_x lesz fontos.

A számítási eredményeink az alábbi táblázatban láthatóak:

Optimalizálás hatása a kibocsátásokra		
	Direkt útvonalak	Optimalizált útvonalak
Távolság [km]	23 477,11	22 268,31
Fajlagos CO2 kibocsátás [kg CO2/l]	2,68	
Üzemanyag fogyasztás [l]	17 138,29	16 255,87
CO2 kibocsátás [kg]	45 930,62	43 565,72
Nox kibocsátás [kg]	219,95	208,62
Havi CO2 megtakarítás [kg]	9 459,59	
Havi Nox megtakarítás [kg]	45,30	

5. tábla: Az optimalizálás hatása a kibocsátási értékekre

Látható, hogy a szén-dioxid kibocsátás megtakarítás rendszerszinten havi **9 459,59 kg**, a nitrogén-oxidok kibocsátás-csökkenése pedig **45,3 kg** volt. A második érték elhanyagolhatónak tűnik a szén-dioxidhoz képest, ám ezen anyagok erős karcinogén hatása miatt kis mennyiségben is felelősek lehetnek daganatos elváltozások kialakulásáért, így a lokálisan okozott kár jelentősebb és kézzelfoghatóbb, mint a CO₂ esetén, ami legfőképpen a globális trendekben játszik szerepet.

A kibocsátási értékek bemutatásával megfontolandó, hogy a zsúfolt, sűrűn lakott városi területek esetén – mint Budapest – a hagyományosan alkalmazott dízelhajtású járművek helyett alternatív üzeműeket alkalmazzanak. Jelenleg két lehetséges opció található a piacon alternatív rendszerekből, amelyek lokális kibocsátása nulla: megfelelő előkészületekkel, illetve az infrastruktúra kiépítésével teljes mértékben karbon-semlegesek lehetnek az elektromos- és a hidrogén hajtású járművek.

Az elektromos járművek az elmúlt években ugrásszerű népszerűség növekedésnek örvendenek, amely jelentős mértékben lecsökkentette mind a gyártási költségeiket, mind a kereskedelmi áraikat. Ennek ellenére egy

elektromos rendszer – a popularitása ellenére – drágább lenne egy hidrogénüzemű, hidrogéncellával hajtott járműhöz képest, mivel ez utóbbinál nem szükséges a hajtásláncot működtető akkumulátort alkalmazni, amely az elektromos autók gyártási költségének nagy részét teszi ki.

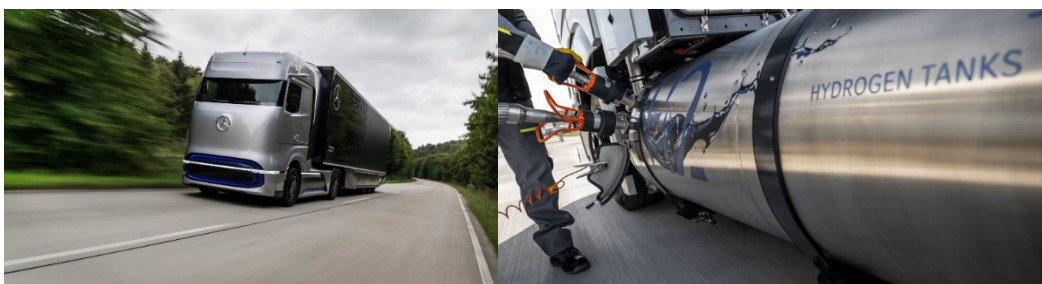
Az akkumulátor gyártás a mai napig a fejlődés ellenére nagyon környezetkárosító folyamat mind az alapanyagok kibányászása szempontjából a forrásországokban (pl.: kobalt Afrikában), mind pedig az összeszereléskor. Ezen felül az akkumulátorok újrahasznosítása a mai napig nem megoldott probléma, amely újabb – talán a gyártási folyamatnál is nagyobb - környezetszennyezési potenciált hordoz magában.

Tovább rontja az elektromos alternatíva lehetőségét, hogy az akkumulátorok korlátozott számú feltöltési ciklussal rendelkeznek, amely határérték megközelítése esetén jelentős teljesítmény és kapacitáscsökkenést tapasztalható.

Ezzel szemben a hidrogén üzemű járművek hidrogén cellája nem rendelkezik ilyen korlátozott élettartammal, a gyártási folyamat kisebb környezeti termeléssel jár, valamint kevésbé energiaintenzív folyamat, az életciklusa végén történő újrahasznosítás nagyságrendekkel kisebb szennyezést jelent a környezetre.

Hidrogénüzemű hulladékszállítás megvalósult már Magyarországon is: Pécsen. A hulladékszállításba bevonásra került 2 db hidrogénnel hajtott szállítójármű, amely hidrogén üzemanyagot a hálózatról, illetve napelemekből szerzett villamos energiával állítanak elő.

A pécsi alkalmazáson túl ígéretes lehetőség, hogy például a Mercedes cég bejelentett egy hidrogén cellával hajtott teherautó-koncepciót. A bejelentés azt jelzi, hogy a hidrogénhajtású technológia területén a közeljövőben további fejlesztéseket várhatunk az általános teherszállító járművek ágazatában is.



27. ábra: A Mercedes GenH2 hidrogén hajtású teherautó [14.]

A teherautó (27. ábra) folyékony halmazállapotban tartja a hidrogént $-253\text{ }^{\circ}\text{C}$ -os hőmérsékleten, jelentősen megnövelve az eltárolható hidrogénmennyiséget és így a hatótávját.

A fenti pontok figyelembevételével a jövőre nézve megfontolandó opciónak tartjuk a H_2 hajtású járművek alkalmazását a hulladékszállításhoz Budapesten.

6.2. A kutatómunka folytatása

A bevezetésben említettük, hogy ezen dolgozat csupán része egy nagyobb – a HUHA 2 hulladékhasznosító erőmű telepítésének teljeskörű elemzésével foglalkozó – tanulmánynak, amelyben az erőmű gazdasági elemzésével foglalkoztunk az instabil nemzetközi gázpiac ár ingadozásainak függvényében. Jelen kutatómunkánk eredményei segítséget jelentenek az erőmű megvalósulásához a TSZH beszállítási költségének csökkentésével, ami befolyásolja az erőmű koncepcionális gazdasági megtérülését valamint az üzemelési költségeit, tekintettel arra, hogy a TSZH egyetlen költsége az erőmű szemszögéből a beszállítási költség. Ennek alapján ez a költség szembeállítható a mindenkori gázárral és rávilágít arra, hogy a helyben termelődő TSZH mennyivel kedvezőbb árú tüzelőanyag, mint a földgáz.

A jelen dolgozatban részletezett eredményeink felhasználásával új szempontok szerint lehet értékelni a tanulmányban már korábban elvégzett

gazdasági számításainkat, még pontosabb képet kapva a megtakarítási lehetőségekről.

A tanulmány folytatásaként terveink között szerepel a következő témák alapos megvizsgálása:

- Az optimalizált TSZH beszállítási útvonalak alapján a TSZH – mint tüzelőanyag – árának meghatározása.
- A rendszer teljes CO₂ kibocsátásának komplex vizsgálata, amely tartalmazza a – jelen dolgozatban kiszámolt - szállítás során kiengedett szennyezőanyagok mértékét is.
- A távhőrendszerre való csatlakozás és a téli-nyári üzemállapotok részletesebb elemzése.
- A HUHA 2 belépésével leterhelt kispesti, kelenföldi és csepeli erőmű hatásfokváltozásának vizsgálata a HUHA 2 kiadott teljesítményének függvényében.
- A hulladéklerakás során felszabaduló depóniagáz okozta CO₂ egyenértékű metán (GWP érték) felszabadulásának megvizsgálása és belekalkulálása az erőmű kibocsátási értékeibe.
- A TSZH és az RDF/SRF tüzelés okozta különbségek gazdasági- és CO₂ kibocsátási különbségei, továbbá az előválogatás során leválasztott, eladható nyersanyagok értékesítésével szerzett többletbevétel vizsgálata.

Ezen tanulmánysorozat elvégzésével teljes képet fogunk kapni a HUHA 2 telepítésének körülményeiről, szükségességéről és hatásairól a budapesti energiaellátásban, amely reményeink szerint ösztönözheti a HUHA 2 megvalósulása felé tett lépéseket.

7. FELHASZNÁLT FORRÁSOK

1. HUHA II. Új Iszapégető és Hulladék-hasznosító Erőmű Műszaki Döntés Előkészítő Tanulmány Rev.1. (S-6 Mérnök Kft, koordinátor: Orbán Tibor, 2017. augusztus)
2. Régi üzemanyag árak jegyzéke: https://net.jogtar.hu/getpdf?docid=000G0000.APE&targetdate=&printTitle=Adatt%C3%A1r&referer=http%3A//net.jogtar.hu/jr/gen/hjegy_doc.cgi%3Fdocid%3D000000003.TXT
3. Kukásautók működése: <https://www.fkf.hu/kukasautokrol>
4. NOx hatásai az emberi szervezetre: <http://www.legszenyvezes.hu/nitrogen-oxidok/>
5. Egy főre jutó éves hulladéktermelés Magyarországon: <https://www.europarl.europa.eu/news/hu/headlines/society/20180328STO00751/hulladekkezeles-az-eu-ban-trendek-es-statisztikak-infografika>
6. Dízelmotor kibocsátási értékei: <https://adoc.pub/szakdolgozat-kipufogogazok-karos-emissziojanak-cskkentese.html>
7. Magyarország Nemzeti Atlasza: https://emna.hu/map/Bp_nepsur_2011/@47.4663625,19.1274506,12.00z
8. Mercedes teherautók fogyasztása: http://www.linardos.gr/brochures/brochures.specific/brochures.chassis.cab.trucks/MERCEDES-BENZ/MERCEDES-BENZ_ECONIC.pdf
9. Parciális gráf: http://www.jgypk.hu/mentorhalo/tananyag/Matematika_mdszertan/138_grfok.html
10. Teljes Gráf: https://hu.wikipedia.org/wiki/Teljes_gr%C3%A1f
11. Google Distance Matrix output: <https://www.geeksforgeeks.org/python-calculate-distance-duration-two-places-using-google-distance-matrix-api/>
12. Google Maps : <https://www.google.com/maps>
13. Dízel üzemanyag kibocsátása: https://en.wikipedia.org/wiki/Diesel_exhaust
14. Mercedes GenH2 teherautó: https://www.google.com/search?q=mercedes+hydrogen+truck&sxsrf=ALiCzsbU8C_8sSCv5ZH_kEoK_06YvJh0pw:1667768107875&source=lnms&tbm=isch&sa=X&ved=2ahUKEwic0lituJr7AhUDjosKHVdeDbQQ_AUoAXoECAIQAw&biw=1862&bih=947&dpr=1.38#imgsrc=Sgdn_s-sVpy1yM