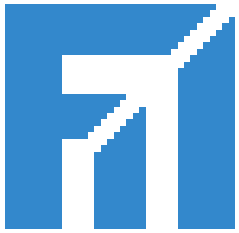
UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ:

Servicii de asistență medicală asistată de drone

propusă de

Ghimp Sergiu

Sesiunea: Iulie, 2020

Coordonator științific

Conf. dr. Raschip Mădălina

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ

Servicii de asistență medicală asistată de drone

Ghimp Sergiu

Sesiunea: Iulie, 2020

Coordonator științific

Conf. dr. Raschip Mădălina

Avizat,

Îndrumător Lucrare de Licență

Conf. dr. Raschip Mădălina _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemntatul(a) …… domiciliul în ……..... născut(ă) la data de ……., identificat prin CNP …….…….……..., absolvent(a) al(a) Universității „Alexandru Ioan Cuza" din Iași, Facultatea de ……. specializarea ……, promoția ……., declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licență cu titlul: _____

_____

_____

_____elaborată sub îndrumarea dl. / d-na _____, pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice in vederea facilitării fasificării de către cumpărător a calității de autor al unei lucrări de licență, de

diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, …… Semnătură student ……

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Servicii de asistență medicală asistată de drone*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași,

Absolvent Sergiu Ghimp

_____

(semnătura în original)

Cuprins

Introducere

Acestă lucrare are ca și scop crearea unei aplicații pentru identifcarea soluțiilor și afișarea rezultatelor problemei serviciilor medicale asistate de drone. Problema ce trebuie rezolvată este problema rutării vehiculelor: identificarea drumurilor minime parcurse de către drone de la centrele de drone la pacienți.

Aplicația permite aplicarea algoritmilor de programare liniară și a algoritmilor euristici (*Hill Climbing* și *Ant Colony Optimization*) pe diferite instanțe. Soluția obținută, costul (instanța drumurilor parcurse de către drone) și timpul de execuție a algoritmului sunt afisate de către aplicație.

Motivație

Calculul traseului minim din punct de vedere al distanței și al timpului mereu s-a identificat în problemele din viața reală a oamenilor.

Am ales acestă temă pentru a arăta că există mai multe metode eficiente de a identifica un număr de trasee ce vor fi parcurse de drone ce au o anumita rază de acțiune. Scopul este de a minimiza timpul și distanța parcursă.

Abordare tehnică

Limbajul de programare folosit este Python. Acesta este ușor, flexibil, puternic și are biblioteci excelente pentru învățarea automată, inteligență artificială, optimizare și modelare statistică. Python este un limbaj de programare ce permite lucrul rapid și poate să integreze sistemele cât mai eficient. În această lucrare am folosit versiunea de Python 3.7.3.

Bibliotecile (și versiunile acestora) ce au fost folosite pentru a crea această aplicație sunt:

1. matplotlib 3.1.0 este o bibliotecă prin care putem genera ploturi, histograme și grafice de tip scatter.
2. docplex 2.14.186 intermediul acestei biblioteci am creat modelele de programare liniară în aplicație.
3. wxPython 4.0.6 acestă bibliotecă este folosită pentru a crea interfața utilizator.
4. pants 0.5.2 utilizată pentru vizitarea unei mulțimi de noduri.

Contribuții

Lucrarea este structurată în trei capitole, în primul capitol sunt descrise informații generale despre problema rezolvată, în capitolul doi sunt oferite detalii despre metodele de implementare și algoritmii utilizați pentru a rezolva această problemă, iar capitolul trei descrie modul de utilizare a aplicației și analiza experimentală a algoritmilor pe instanțe de diferite dimensiuni.

Capitolul 1: Informații despre problema serviciilor de asistența medicală asistată de drone și metodele existente de rezolvare a problemei.

Capitoulul 2: Prezentarea metodelor de rezolvare a problemei conține două subcapitole.

În primul subcapitol este descrisă problema *Set Covering* care are rolul de a elimina centrele de drone redundante și a eficientiza soluția.

Al doilea subcapitol conține cele trei metode de rezolvare a problemei de rutare a vehiculelor:

1. utilizarea programării liniare cu CPLEX
2. algoritmul *Ant Colony Optimization*
3. algoritmul *Hill Climbimg*

La acest capitol am avut cea mai mare contribuție și am reușit să dobândesc cele mai multe cunostințe.

Capitolul 3: Utilizarea aplicației și rezultatele experimentale

Acest capitol are trei subcapitole. În primul subcapitol am prezentat un manual de utilizare a aplicației, modul în care un utilizator poate să aleagă metoda potrivită pentru rezolvarea problemei de rutare și să obțina soluția acesteia. Al doilea subcapitol cuprinde descrierea a diferite tipuri de instanțe generate pentru această problem. În ultimul subcapitol avem rezultele obținute de către algorimii implementați.

1. Introducere

Potrivit Centrelor de Control și Prevenire a Bolilor din Statele Unite ale Americii una dintre cele mai importante preocupări majore ale guvernului și ale pacienților sunt bolile cronice. O componentă superioară a cheltuielilor generale de asistență medicală sunt costurile bolilor cronice. În Statele Unite ale Americii, aproximativ 117 milioane de oameni au avut una sau mai multe boli cronice în anul 2012, ceea ce înseamna că aproximativ jumătate din toți adulții au avut boli cronice care includ boli de inimă, diabet, artrită și obezitate. Pentru controlul de rutină sau cumpărarea medicamentelor pentru tratament, instituțiile medicale îi obligă pe pacienții cu boli cronice să viziteze instituțiile medicale.

Vizitele făcute de pacienți periodic pot suporta cheltuieli majore și costuri medicale care pot fi de câteva ori mai mari în comparație cu pacienții fără boli cronice. Pacienții cu boli cronice cheltuiesc aproape de 2,5 ori mai mult decât cei fără boli cronice datorită vizitelor obișnuite la clinică, a rețetelor, a vizitelor de sănătate la domiciliu și a șederilor în spital.

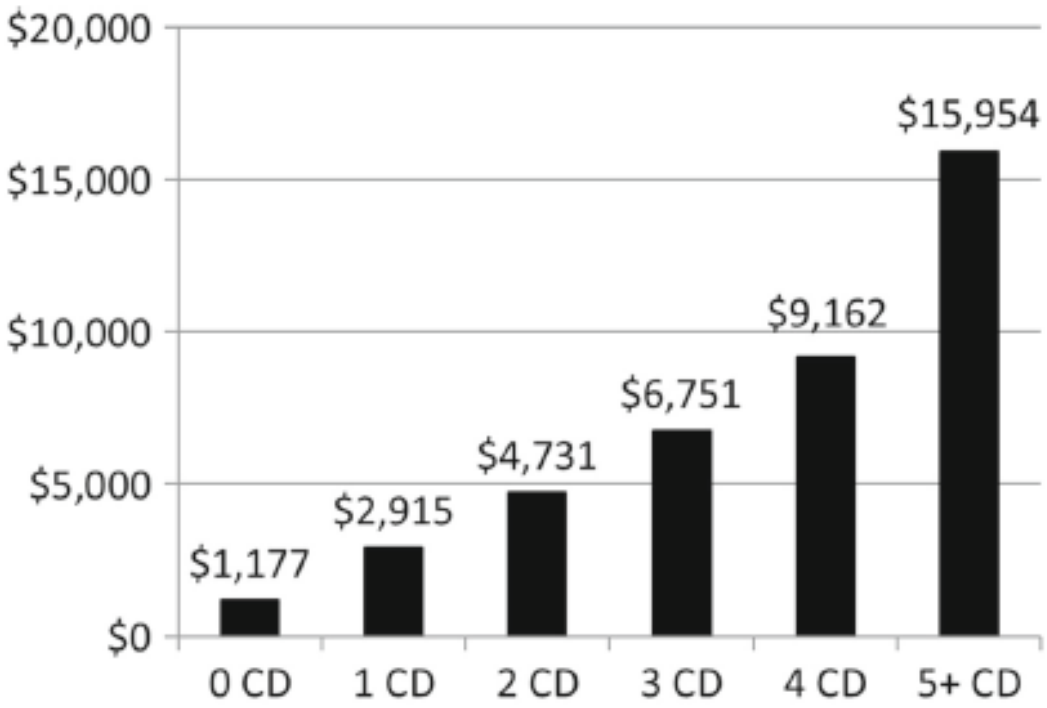


Figura 1: Costul medical al pacienților cronici (în funcție de numărul de condiții cronice); Costul de boli cronice (CD)

Pentru a face teste de rutină sau pentru a solicita servicii medicale pacienții sunt nevoiți să folosească timpul liber sau timpul de la serviciu care reprezintă costuri; la fel, suportă și cheltuieli de transport pentru a participa la programările medicale.

Pacienții din mediu urban sunt avantajați față de cei din mediu rural, care mai au și alte cheltuieli. Locuitorii din mediu urban se întâmpină cu niște bariere pentru a accesa asistența medicală, una dintre acestea este transportul către și de la unitățile medicale.

Locuitorii din zonele rurale au aceleași tipuri de boli cronice ca și cei din zonele urbane. Locuitorii din zonele rurale se confruntă și cu problema limitarilor geografice care presupune și ea niște cheltuieli. Pacienții din zonele rurale sunt dispuși să plătească mai mult pentru a-și trata bolile, dar din cauza lipsei de facilități medicale din apropierea lor, ei nu pot accesa cu ușurință clinicile.

Pentru a atenua problemele de sănătate și pentru a spori accesibilitatea asistenței medicale Comisia Federală de Comunicare din Statele Unite a propus un proiect de dezvoltare a infrastructurii de telemedicină. Datorită acestei infrastructuri, pacienții din zonele rurale nu se vor confrunta cu restricții în ceea ce privește timpul și spațiul și vor putea primi servicii medicale. Serviciul de livrare la domiciliu și programele pilot combinate permit echipelor de medici să monitorizeze și să partajeze informațiile medicale anterioare ale pacienților și rezultatele testelor.

Telemedicina nu poate fi ca și substituent pentru îngrijirea personală a persoanelor sau administrarea de medicamente. Telemedicina poate doar monitoriza și oferi servicii medicale calificate prin fluxuri de voce și video, dar nu poate oferi transport pentru pacienții din mediu rural și nu poate livra medicamente și truse de examinare pacienților.

Un element important în livrarea medicamentelor care care nu poate fi respectat este lipsa transportului.

Locuitorii din mediul rural sunt mai puțini față de cei din mediul urban. Rentabilitatea investițiilor din zonele rurale poate fi mult mai mică în comparație cu zonele urbane, din acest motiv mulți furnizori ezită să investească în astfel de proiecte. În situația în care pentru aceste proiecte de asistență medicală se găsesc furnizori care sunt dispuși să investească, este dificil de găsit, de recrutat și de păstrat personalul medical din cauza condițiilor geografice.

Asociația Americană a Persoanelor în Retragere a stabilit că 89% dintre americani cu vârsta de peste 50 ani doresc să fie asigurați de servicii de asistență în propriile case. Acest lucru este posibil doar prin a angaja mai mulți îngrijitori de către furnizori, pentru a satisface cerințele pacienților.

Aceasta lucrare propune să stabilească modalități de a de atenua problemele de sănătate și depăși limitările metodelor curente de furnizare a serviciilor de asistență medicală în zonele rurale, oferind pacienților din zonele rurale servicii de asistență medicală asistată de drone.

Prin folosirea dronelor, îngrijitorii vor putea livra truse de testare de rutină, aprovizionarea cu medicamente și vor putea ridica truse de examinare ale pacienților, cum ar fi probe de sânge și urină. Dronele vor avea abilitatea de a reduce munca unui îngrijitor și timpul de călătorie.

Dronele pot fi utilizate pentru livrarea și ridicarea articolelor sensibile, indiferent de condițiile de drum de la sol, comparativ cu alte moduri de transport, poșta sau serviciile de curierat. În condiții meteorologice severe dronele nu vor putea fi expediate.

În figura 2 este reprezentat un model de transportare de servicii asistate de drone ce poate fi integrat pentru o localitate rurală.

Figura 2: Conceptul de asistență medicală asistată de drone pentru servicii de livrare și preluare

În timp ce îngrijitorii vor vizita pacienții care au nevoie de îngrijire personală, dronele vor fi utilizate pentru a livra și pentru a ridica truse medicale pentru pacienții din zonele rurale.

Pacienții care au anumite boli cronice nu trebuie să conducă și astfel transportarea truselor medicale, medicamentelor și probelor de analiză cu ajutorul dronelor va contribui la eliminarea cheltuieilor pacienților.

Datorită acestei soluții de livrare a produselor cu ajutorul dronelor îngrijitorii pacienților se vor putea concentra mai mult pe îngrijirea persoanelor; volumul de muncă al îngrijitorilor va fi redus, și calitatea serviciului de îngrijire se va îmbunătăți.

1.1. Servicii de asistență mecicală asistată de drone

O problemă economică pentru guvern și pentru furnizori este creșterea numărului de pacienți cu boli cronice și îmbătrânirea populației. Acesta este unul dintre motivele principale ale creării metodelor de asistență medicală.

Pentru a putea oferi pacienților mai multe tipuri de îngrijiri s-a creat modelul de "Îngrijire la domiciliu". Acest model a apărut acum un secol, când locuința era locul de muncă pentru asistente. De-a lungul timpului modelul s-a dezvoltat și a evoluat datorită tehnologiei și modificărilor cerințelor pacienților. Acest model de "Îngrijire la domiciliu" are mai multe dezavantaje, tratamentele au devenit mai necalificate, a crescut costul de operare și nivelul de servicii a devenit nesatisfăcator.

În zonele rurale "Îngrijirea la domiciliu" s-a confruntat cu multe dificultăți în obținerea de beneficii din cauza mediilor diferite întâlnite față de cele din zonele urbane. Factorii care au contribuit la aceasta sunt: populația scăzuta, lipsa de transport, facilitățile medicale și farmaciile și distanțele lungi între pacienți și unitatea medicală.

Din cauza condițiilor fizice limitate pacienții care necesită îngrijire periodică se confruntă cu anumite bariere. Sunt unele modele care nu pot fi aplicate pentru toți pacienții. Locuitorii din mediul rural nu dispun de transport și îngrijitori capabili să ia medicamente și truse medicale pentru ei.

Utilizarea dronelor este o metodă ce poate fi utilizată pentru a ușura serviciile de asistentă pentru pacienții cu boli cronice care trăiesc în zonele rurale și care întâmpină problema distanțelor lungi până la unitățile medicale, lipsa transportului, lipsa timpului și altor cheltuieli. [1]

1.2. Metode existente

Acest studiu propune două metode de rezolvare: planificarea strategică (SP) și planificarea operațională (OP).

Scopul modelului SP este de a găsi locațiile optime pentru centrele de drone care sa ofere servicii tuturor pacienților dintr-o anumită zonă.

Scopul modelului OP este de a reduce numărul de drone pe centru de drone și identificarea programelor optime de livrare și preluare pentru a satisface cerințele specifice ale pacienților în timpul zborului

Lucrarea prezintă un concept de livrare a serviciilor medicale asistate de drone pentru pacienții cu boli cronice din zonele rurale: dronele pot furniza servicii de livrare și preluare a serviciilor de asistență medicală aeriană pentru a ajuta numărul limitat de îngrijitori și pentru a reduce cheltuielile pentru pacienții din zonele rurale care au nevoie de servicii medicale de rutină.

În cadrul lucrării sunt dezvoltate două metode de planificare pentru livrarea și ridicarea serviciilor medicale asistate de drone: planificarea strategică (SP) pentru a decide în mod optim unde trebuiesc amplasate centrele de drone și câte centre și planificarea operațională (OP) pentru planificarea traseelor dronelor din fiecare centru.

2. Metode implementate

Pe baza datelor despre pacienți, pacienții pot fi împărțiți în două grupuri:

1. cei care are nevoie de îngrijire personală;
2. cei care are nevoie doar de testare simplă sau aprovizionare cu medicamente.

Acest studiu se bazează doar pe cel de-al doilea grup de pacienți, unde dronele pot oferi truse de testare și medicamente. Locațiile și numărul de centre de drone sunt determinate să acopere toți pacienții din acest grup.

Pentru găsirea unei soluții la această problemă, avem nevoie de două modele de rezolvare:

1. Planificare Strategică (SP)
2. Planificare Operațională (OP)

2.1. Planificare Strategica (SP)

Scopul modelului SP este de a găsi locațiile optime pentru centrele de drone care să ofere servicii tuturor pacienților dintr-o anumită zonă.

Aceasta planificare trebuie să asigure ca toate centrele de drone sunt capabile să servească toți pacienții indiferent de nivelul cererii și numărul de boli cronice. Toți pacienții au aceeași prioritate pentru a primi servicii indiferent de circumstanțe.

Pentru a descrie planificarea strategică (SP) avem nevoie de următoarele notații:

1. Mulțimi
   a. I mulțime de pacienți $(i \in I)$;
   b. R_c mulțime de locații de centre de drone $(r \in R_c)$;
   c. R_i mulțime de centre ce acoperă pacientul i, astfel încât $R_i \subseteq R_c$;
2. Parametri
   a. c_r costul inițial pentru a deschide un centru la locația r;
   b. d_{ir} distanța între pacientul i și locația r;
3. Variabile de decizie
   a. x_r 1 dacă locația r este selectată ca și centru, 0 altfel.

Scopul modelului SP este de a determina locațiile de centre de drone cu un cost de investiție minim. Prin urmare, pentru îndeplinirea acestui obiectiv, trebuie satisfăcut modelul:

1. Minimizarea:

$$Z_{SP} = \sum_{r \in R_c} c_r x_r \tag{1}$$

2. Constâgeri:

$$\sum_{r \in R_i} x_r \geq 1, \qquad \forall i \in I$$
$$x_r \in \{0, 1\}, \tag{2}$$

Funcția obiectiv (1) este pentru a minimiza costul inițial al înființării centrelor de drone. Constrângerea (2) specifică faptul că fiecare pacient i din I trebuie să fie acoperit macăr o dată.

2.1.1. Set Covering Problem

Pentru rezolvarea acestei probleme vom folosi o abordare de tip *Set Covering*. Acesta va determina în mod optim unde vor fi amplasate centrele de drone și câte centre de drone să fie active, pentru a acoperi toți pacienții. Soluția problemei *Set Covering* va identifica numărul minim de centre de drone și va specifica fiecărui pacient cărui centru de drone corespunde. Această abordare ajută la eliminarea centrelor de drone care sunt redundante și au o rază de acoperire care se suprapune.

2.1.1.1. Descrierea problemei

Set Covering Problem este o problemă clasică în combinatorică, informatică, cercetarea operațională și teoria complexității. Este una dintre cele 21 probleme *NP – complete* ale lui Karp demonstrate a fi *NP – complete* în 1972.

Scopul problemei este de a selecta numărul minim de submulțimi care conțin toate elementele care sunt incluse în mulțimea inițială. [2]

Input:

1. O mulțime *U* de *n* elemente $U = \{1, 2, ..., n\}$ numită univers
2. O colecție *S* de *m* mulțimi a cărei reuniune este egală cu universul, $S_1 \cup S_2 \cup ... \cup S_m = U$

Set Covering Problem constă în identificarea celui mai mic număr de submulțimi din *S* a căror reuniune să fie mulțimea *U*.

Output:

1. Acoperirea tuturor elementelor din *U*.
2. Eliminarea submulțimilor redundante din colecția de mulțimi *S*.
3. Afișarea celui mai mic număr de submulțimi din *S* care cuprinde toate elementele din *U*.

Exemplu:

Input:

1. Mulțimea univers $U = \{1, 2, 3, 4, 5\}$
2. Colecția de mulțimi $S = \{S_1 = \{1, 2, 3\}, S_2 = \{2, 4\}, S_3 = \{3, 4\}, S_4 = \{4, 5\}\}$
3. $S_1 \cup S_2 \cup S_3 \cup S_4 = U$

Output:

1. Mulțimea $U = \{S_1 = \{1, 2, 3\}, S_4 = \{4, 5\}\} \rightarrow S1, S4$

```
def main():
    universe = set(range(1, 6))
    subsets = [set([1, 2, 3]),
               set([2, 4]),
               set([3, 4]),
               set([4, 5])]
    cover = set_cover(universe, subsets)
    print(cover)

[{1, 2, 3}, {4, 5}]
```

Secțiune de cod 1: *Set Covering Problem* – Output

2.1.1.2. Planificare Strategică – modelare

Input:

1. O mulțime I de i pacienți (coordonatele), $I = \{p_1(x_1, y_1), p_2(x_2, y_2), p_3(x_3, y_3), \ldots, p_n(x_n, y_n)\}$ numita mulțime de pacienți.
2. O colecție R_c de r mulțimi de centre de drone (coordonatele), ce cuprind o anumită rază R, a căror reuniune este egală cu mulțimea de pacienți I:

$S = \{[D_1(xd_1, yd_1) \rightarrow p_1(x_1, y_1), p_2(x_2, y_2)],$

$[D_2(xd_2, yd_2) \rightarrow p_2(x_2, y_2), p_3(x_3, y_3), p_4(x_4, y_4), p_5(x_5, y_5)],$

$[D_3(xd_3, yd_3) \rightarrow p_2(x_2, y_2), p_3(x_3, y_3), p_4(x_4, y_4)],$

$\ldots,$

$[D_n(xd_n, yd_n) \rightarrow p_{n-2}(x_{n-2}, y_{n-2}), p_{n-1}(x_{n-1}, y_{n-1}), p_n(x_n, y_n)]\}$

Set Covering Problem constă în identificarea celui mai mic număr de submulțimi de R_c a căror reuniune să fie mulțimea care acoperă toți pacienții din I.

Output:

1. Acoperirea tuturor pacienților din I;
2. Eliminarea submulțimilor redundante din colecția de mulțimi R_c
3. Afișarea celui mai mic număr de submulțimi din R_c care cuprinde toți pacienții din $I = \{\{p_1(x_1, y_1), p_2(x_2, y_2)\}, \{p_2(x_2, y_2), p_3(x_3, y_3), p_4(x_4, y_4), \ldots p_n(x_n, y_n)\}\}$

2.1.1.3. Planificare Strategică – Caz particular

Input:

1. O mulțime I de i pacienți (coordonatele), $I = \{p_1(1,1), p_2(6,1), p_3(8,3), p_4(6,9), p_5(10,10)\}$ numită mulțime de pacienți.
2. O colecție R_c de r mulțimi de centre de drone a căror reuniune este egală cu mulțimea de pacienți I:

$S = \{[D_1(3, 2) \rightarrow p_1(1, 1), p_2(6, 1)],$

$[D_2(6, 5) \rightarrow p_2(6, 1), p_3(8, 3), p_4(6, 9)],$

$[D_3(9, 8) \rightarrow p_3(8, 3), p_4(6, 9), p_5(10, 10)]\}$

Set Covering Problem constă în identificarea celui mai mic număr de submulțimi de R_c de a cărui reuniune să fie mulțimea care acoperă toți pacienții din I.

Output:

1. Acoperirea tuturor pacienților din I;
2. Eliminarea submulțimilor redundante din colecția de mulțimi R_c;
3. Afișarea celui mai mic număr de submulțimi din R_c care cuprinde toți pacienții din $I = \{p_1(1, 1), p_2(6, 1), p_3(8, 3), p_4(6, 9), p_5(10, 10)\} \rightarrow D_1, D_3$;

```
def SP(centredrona, Ri):
    popular = dict()
    for i in centredrona:
        popular[i] = 0
    for i in Ri:
        for j in i:
            popular[j] += 1
    popular = sorted(popular.items(), key=lambda x: x[1], reverse=True)
    ok = 1
    while ok:
        ok = 0
        for i in range(0, len(popular)):
            centredrona, Ri, t = Scoatere(centredrona, Ri, popular[i][0])
            if t == True:
                ok = 1
                popular.pop(i)
                break
    print(Ri)
    return centredrona, Ri
(Centrele de Drone Initial)R =  [(3, 2), (6, 5), (9, 8)]

(Centrele de Drona dupa Eliminare)R1 = [(3, 2), (9, 8)]
```

Secțiune de cod 2: *Set Covering Problem* – Output: Eliminare Centre de Drone

2.2. Planificare Operațională (OP)

Scopul modelului OP este de a reduce numărul de drone pe centru de drone și identificarea programelor optime de livrare și preluare pentru a satisface cerințele specifice ale pacienților în timpul zborului.

Odată ce locațiile fiecărui centru de drone a fost identificat, numărul de drone per centru de drone poate fi determinat de numărul de pacienți deserviți. Numărul de pacienți asignați fiecărui centru de drone poate fi determinat de rezultatul SP, dar nu asigură că asignările pacienților sunt optime întrucât unii pacienți sunt acoperiți de mai mult de un centru de drone.

Fiecare pacient este deservit doar o singură dată de o dronă și nu există nicio prioritate pentru vizitarea pacienților. Dronele după ce și-au îndeplinit sarcina propusă se întorc înapoi la centru de drone din care au pornit. Pentru a dezvolta drumul unei drone avem nevoie de cantitatea posibilă de livrare a fiecărui pacient spre care va livra.

Pentru a dezvolta modelul de planificare operațională (OP) avem nevoie de următoarele notații:

1. Mulțimi
   a. I mulțime de pacienti $(i, j \in I)$;
   b. R mulțime de locații de centre de drone din modelul SP $(R \subseteq R_c)$;
   c. K mulțime de drone $(k \in K)$;
2. Parametri
   a. A o mulțime de arce, cu $A = \{(i, j) \in R^2 : i \neq j\}$
   b. p_k costul de operare a unei drone k;
   c. P_i cantitatea de preluare a pacientului i;
   d. D_i cantitatea de livrare a pacientului i;
   e. d_{ij} distanța între clienții i si j;
3. Variabile de decizie
   a. x_{ijk} 1, dacă drona k zboară de la pacientul i la pacientul j, 0 altfel.
   b. h_k 1, dacă drona k este folosită pentru a servi, 0 altfel;

Problema constă în:

1. Minimizarea:

$$Z_{OP} = \sum_{k \in K} p_k h_k \tag{3}$$

2. Constângeri:

$$\sum_{i \in I \cup R} \sum_{k \in K} x_{ijk} = 1, \qquad \forall j \in I \tag{4}$$

$$\sum_{i \in I \cup R} \sum_{k \in K} x_{ijk} = 1, \qquad \forall i \in I \tag{5}$$

$$x_{iik} = 0, \qquad \forall i \in I \cup R,\ k \in K \tag{6}$$

3. Calcularea distanței:

$$d_{ij} = \sqrt{(i_1 - i_2)^2 + (j_1 - j_2)^2} \tag{7}$$

Funcția obiectiv (3) constă în minimizarea sumei costului de operare a dronelor, și care ajută implicit în minimizarea numărului de drone.

Constrângerile (4) și (5) asigură că fiecare pacient este servit o singură dată.

Constrângea (6) verifică ca drona să nu viziteze același element de mai multe ori;

(7) Pe baza coordonatelor depozitelor și clienților, algoritmul crează o matrice D care conține distanțele între fiecare cuplu posibil de clienți și distanțele dintre depozit și fiecare client.

Pentru rezolvarea acestei probme vom folosi o modelare similar a problemei de rutare a vehiculelor. Problema de rutare a vehiculelor va determina toate căile posibile parcurse de dronele respective parcurgând drumul minim de la centrul de drone spre pacienți și înapoi. Cu ajutorul problemei de rutare a vehiculelor vom parcurge drumurile în funcție de distanța minimă a drumului parcurs de către drona și în funcție de cantitatea de livrare și preluare. [1]

Exemplul 1:

Pentru parcurgerea drumului vom avea două tipuri de drone, drone de tip 1 (culoare verde) care au o distanța de livrare de 8 km și costă 8$ și drone de tip 2 (culoare galbenă) care au o distanța de livrare de 12 km și costă 12$. Raza pe care o acoperă un centru de drone este de 6 km. Avem un număr de 7 centre de drone prin eliminarea celor redundante obținem 5 centre de drone active și un număr de 16 pacienți.

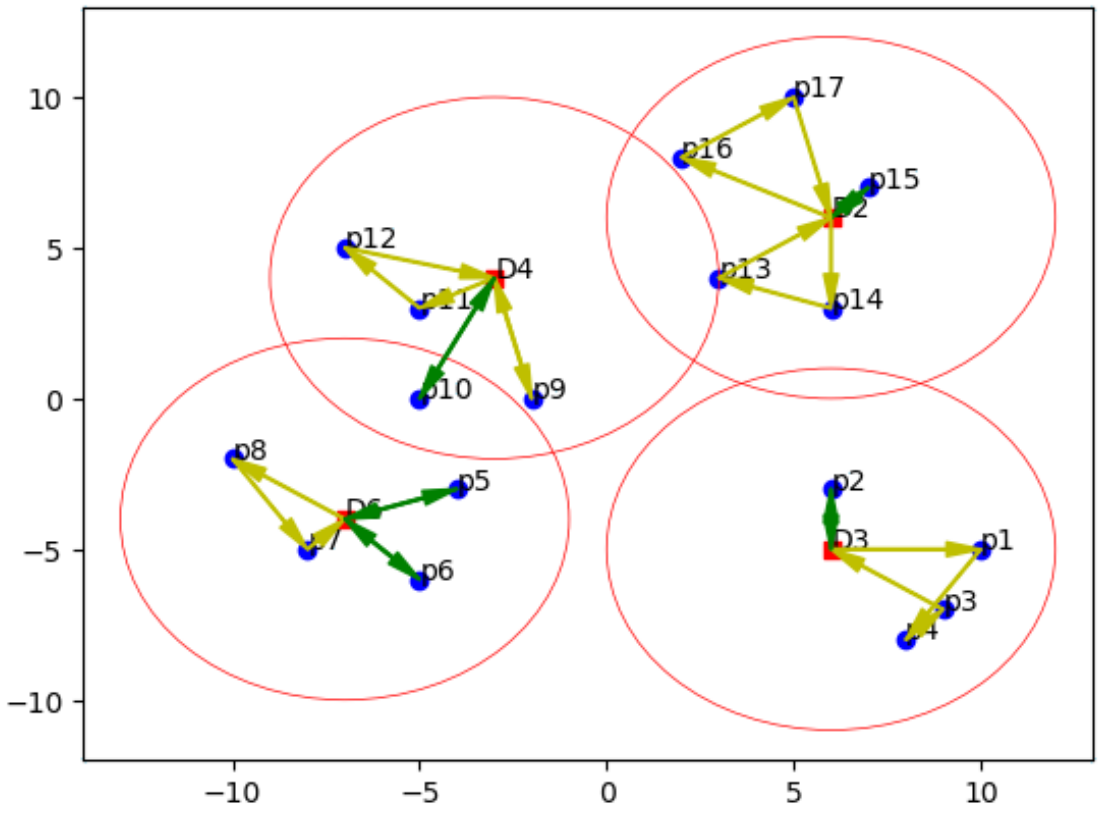


Figura 3: Soluția obținută pentru instanța din exemplul 1

Pacienții care sunt acoperiți de mai multe centre de drone sunt asignați unui singur centru de drone în funcție de drumul minim parcurs de la centrul de drone la pacient și în funcție de cantitatea de livrare și preluare.

	A	B	C	D	E	F	G
1	Centru de Drone	Pacienti	Numar de drone	Cost($)	Traseele si distanta parcursa de ele	Suma Distante	Timpul de exec
2	D2	p14 p16 p17 p13 p15	Tip I:1 Tip II:2	32$	D2 p14 p13 D2 9.77 | D2 p15 D2 2.83 | D2 p16 p17 D2 12.2 |		
3	D3	p1 p3 p2 p4	Tip I:1 Tip II:1	20$	D3 p1 p4 p3 D3 12.63 | D3 p2 D3 4.0 |		
4	D4	p9 p10 p12 p11	Tip I:1 Tip II:2	32$	D4 p9 D4 8.25 | D4 p11 p12 D4 9.19 | D4 p10 D4 8.94 |		
5	D6	p7 p5 p8 p6	Tip I:2 Tip II:1	28$	D6 p6 D6 5.66 | D6 p8 p7 D6 8.63 | D6 p5 D6 6.32 |		
6	Nr Centre: 4	17 Pacienti	11 (Tip I: 5 Tip II: 6)	112$		88.41	2.43 ms

Tabel 1: Valorile costurilor pentru instanța din exemplul 1

2.2.1. Problema de rutare a vehiculelor

Problema de rutare a vehiculelor este o problema de optimizare combinatorie. Scopul acestei probleme este de identifica numărul minim de trasee pe care le poate parcurge un număr de vehicule pentru a livra bunuri la o anumită mulțime de clienți. Determinarea unei soluții optime pentru problema de rutare a vehiculelor este *NP-hard*. Folosind metode de optimizare combinatorie, problema poate fi rezolvata în mod optim. Obiectivul principal al problemei de rutare a vehiculelor este de a reduce costul total al rutei. [3]

Problema de rutare a vehiculelor generalizează problema comis-voiajorului care este la fel o problema *NP-hard* importantă. Scopul problemei comis-voiajorului este de a găsi cea mai scurtă rută posibilă care vizitează fiecare oraș și se întoarce la orașul de origine avem ca input o listă de orașe și distanțele dintre fiecare pereche de orașe.

Input:

1. Un număr *n* de puncte ce reprezintă clienții
2. Un număr *m* de puncte ce reprezintă depozitele
3. Un număr *k* de vehicule care pleacă dintr-un anumit punct, numit depozit, revin la depozit după ce au vizitat clienți.
4. O greutate maximă a sarcinii, numită capacitate, a unui vehicul de transport care se află la un depozit.
5. Fiecare client *n* are o cerere q_i și cererea lor este colectată de un anumit vehicul.
6. Vehiculele de transport au capacitatea de încărcare finite *Q* (limita superioară), iar suma capacităților solicitate de camioane nu trebuie să depășească această valoare.
7. Valoarea maximă *max{qi}* din cantitatea cerută a clientului nu depășește capacitatea *Q* a vehiculului.

Problema de rutare a vehiculelor are următoarele constrângeri:

1. Cantitatea cerută de client a clientului. Suma cererilor clienților nu trebuie să depășească greutatea maximă de transport a camionului (constrângere de capacitate). Dacă cererea clienților depășește valoarea maximă a capacității de încărcare, atunci aceștia sunt împărțiți corespunzător cererii (astfel încât să se afle în limita superioară a capacității de încărcare).
2. Fiecare client trebuie vizitat exact o data parcurgând o distanță minimă de la depozit până revine înapoi la depozit trecând pe la un număr de pacienți.

Output:

1. Găsirea celor mai optime rute ale vehiculelor care satisfac toate cererile clienților (un simplu circuit închis care pleacă de la depozit și se întoarce din nou în depozit și transportă cantitățile cerute de clienți fără să depășească cantitatea maximă a vehiculului). [4]

2.2.1.1. Programare liniară

Optimizarea matematică este o piatră de temelie importantă, care transformă o problemă într-un model matematic și apoi găsește cea mai bună soluție a problemei.

Programarea liniară presupune o formulare matematică a unei probleme de decizie în care funcția obiectiv și constrângerile sunt descrise prin relații liniare și toate variabilele de decizie pot lua valori continue. CPLEX oferă soluții de programare matematică, flexibilă și de înaltă performanță. [5]

În Python există câteva pachete de modelare open-source (CPLEX, Gurobi, PuLP).

Ca și pachet de modelare CPLEX în Python există libraria docplex.

Pentru a implementa modelul de programare liniară cu pachetul de modelare CPLEX în Python este necesar sa adăugăm următoarele linii de cod:

```
import docplex.mp.model as cpx
opt_model = cpx.Model(name="MIP Model")
```

2.2.1.1.1. Modelul de programare liniară

În expedierea unor bunuri din depozitele unor companii prin intermediul vehiculelor, trebuie luată în considerare capacitatea vehiculului, ruta pe care o parcurge și cererea clientului.

Unele dintre cele mai importante obiective ale problemei de rutare a vehiculelor sunt: numărul de locații de depozite, numărul de vehicule necesare și traseul optim parcurs de vehicule.

Problema de rutare a vehiculelor poate fi modelată cu ajutorul grafurilor, unde un nod reprezintă un depozit, iar alte noduri reprezintă locațiile deservite de depozit. Valoarea fiecărui nod este reprezentată de cererea nodului și capacitatea vehiculului.

Distanța și costurile sunt reprezentate cu ajutorul arcurilor. Un arc poate reprezenta distanța dintre un depozit și o locație a clientului sau distanța dintre doi clienți. Pe baza arcelor dintre depozite și clienți, algoritmul crează o matrice care conține distanțele între fiecare cuplu posibil de clienți și distanțele dintre depozit și fiecare client.

Obiectivul cel mai important a problemei de rutare a vehiculelor este de a construi o mulțime de trasee pentru vehicule care sunt capabile sa viziteze toți clienții ce au costuri minime și satisfac constrângerile privind capacitatea vehiculelor.

Input:

1. n un număr de clienți
2. N o mulțime de clienți, cu $N = \{1, 2, \ldots, n\}$
3. V o mulțime de noduri, cu $V = \{0\} \cup N$
4. A o mulțime de arce, cu $A = \{(i, j) \in V^2 : i \neq j\}$
5. c_{ij} costul de călătorie a unui arc $(i, j) \in A$
6. Q capacitatea vehiculului
7. q_i cantitatea de livrare pentru clientul $i \in N$

1. Minimizarea:

$$\sum_{i,j \in A} c_{ij} x_{ij} \tag{1}$$

2. Constângeri:

$$\sum_{j \in V, j \neq i} x_{ij} = 1 \qquad i \in N \tag{2}$$

$$\sum_{i \in V, i \neq j} x_{ij} = 1 \qquad j \in N \tag{3}$$

$$\begin{aligned} &\text{if } x_{ij} = 1 \Rightarrow u_i + q_j = u_j && i, j \in A : j \neq 0, i \neq 0 \\ &q_i \leq u_i \leq Q && i \in N \\ &x_{ij} \in \{0, 1\} && i, j \in A \end{aligned} \tag{4}$$

Funcția obiectiv (1) constă în minimizarea sumei costului de operare, și care ajută implicit în minimizarea traseelor.

Constrângerile (2) și (3) asigură că fiecare client sa fie servit o singură dată.

Constrângea (4) verifică cantitatea vehiculul să nu fie depășită;

Output:

1. Satisfacerea tuturor clienților prin trecerea pe la fiecare client parcurgând o distanță cât mai mică și prin transportarea unei anumite cantități care nu depășește cantitatea maximă a vehiculului.

VehicleRoutingProblemCPLEX.py > ...

```python
import matplotlib.pyplot as plt
import numpy as np
from docplex.mp.model import Model
rnd = np.random
rnd.seed(4)
n = 10  # number of clients
xc = rnd.rand(n + 1) * 20
yc = rnd.rand(n + 1) * 10
N = [i for i in range(1, n + 1)]
print("(set of clients) N = ", N)
V = [0] + N
print("(union of 0 & N) V = ", V)
A = [(i, j) for i in V for j in V if i != j]
print("(set of arcs) A = ", A)
c = {(i, j): np.hypot(xc[i] - xc[j], yc[i] - yc[j]) for i, j in A}
print("(cost of travel) c = ", c)
Q = 20
print("(vehicle capacity) Q = ", Q)
q = {i: rnd.randint(1, 10) for i in N}
print("(the amount that has to be delivered for each custumer) q = ", q)
for i in N:
    plt.annotate('$q_%d=%d' % (i, q[i]), (xc[i], yc[i]))
plt.plot(xc[0], yc[0], c='r', marker='s')
plt.axis('equal')
mdl = Model('MIP Model')
x = mdl.binary_var_dict(A, name='x')
u = mdl.continuous_var_dict(N, ub=Q, name='u')
mdl.minimize(mdl.sum(c[i, j] * x[i, j]for i, j in A))
mdl.add_constraints(mdl.sum(x[i, j] for j in V if j != i) == 1
                                                    for i in N)
mdl.add_constraints(mdl.sum(x[i, j]for i in V if i != j) == 1
                                                    for j in N)
mdl.add_indicator_constraints(mdl.indicator_constraint(
    x[i, j], u[i] + q[j] == u[j]) for i, j in A if i != 0 and j != 0)
mdl.add_constraints(u[i] >= q[i] for i in N)
mdl.parameters.timelimit = 15
solution = mdl.solve(log_output=True)
active_arcs = [a for a in A if x[a].solution_value > 0.9]
for i in N:
    plt.annotate('$q_%d=%d' % (i, q[i]), (xc[i], yc[i]))
for i, j in active_arcs:
    plt.plot([xc[i], xc[j]], [yc[i], yc[j]], c='g', alpha=0.3)
# put on map the Depot coordonates
plt.plot(xc[0], yc[0], c='r', marker='s')
# put on map the Clients coordonates
plt.scatter(xc[1:], yc[1:], c='b')
plt.show()
```

Secțiune de cod 3: Modelare problema de rutare a vehiculelor cu CPLEX

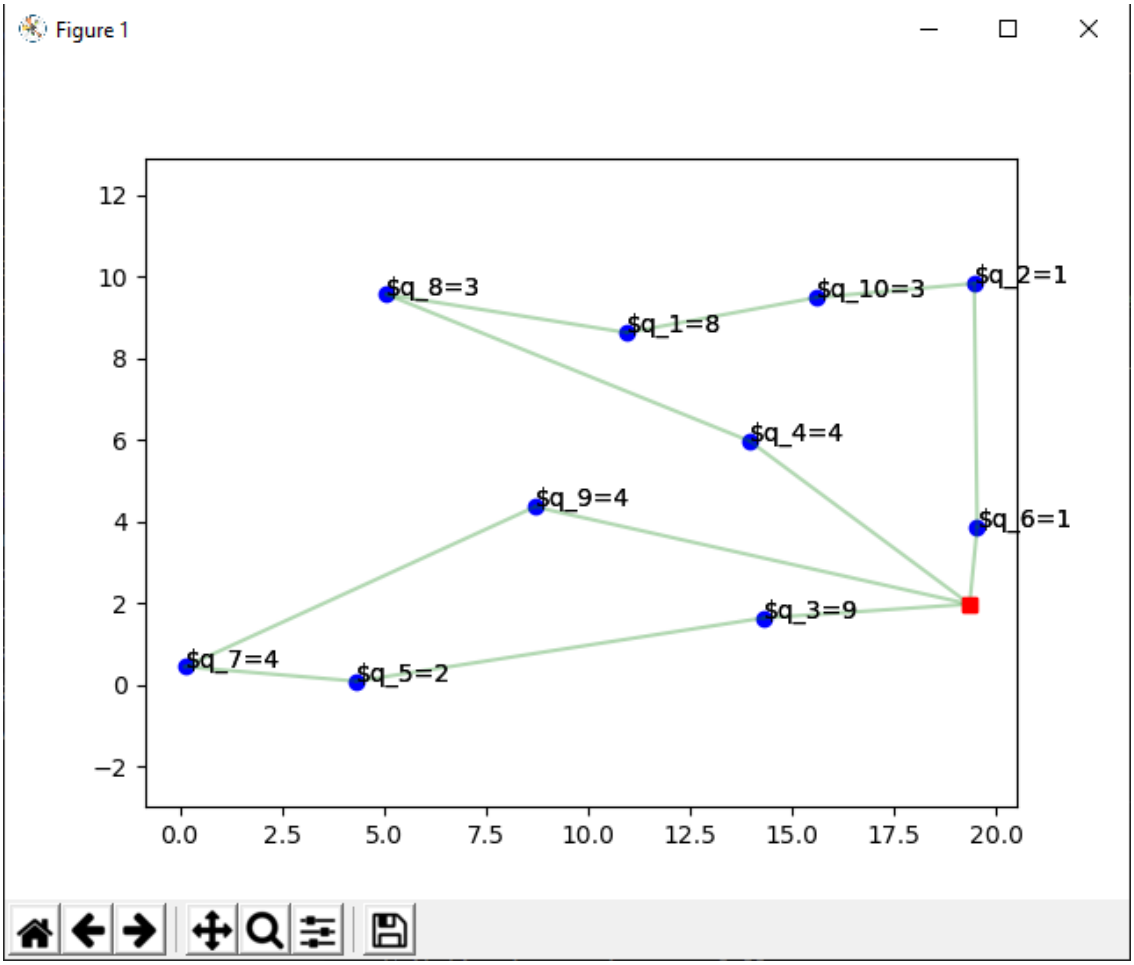


Figura 4: Soluția problemei de rutare a vehiculelor cu CPLEX

2.2.1.2. Metode Euristice

Metodele euristice sunt utilizate în mod obișnuit pentru rezolvare problemelor din viața reală datorită vitezei și abilității lor de a face fața insanțelor de dimensiuni mari.

Prezentăm în continuare algoritmul *Hill Climbing* și algoritmul *Ant Colony Optimization*.

2.2.1.2.1. Hill Climbing

Hill Climbing este o algoritm de căutare euristică utilizat pentru rezolvarea problemelor de optimizare care aparține familiei local search. Este un algoritm iterativ care începe cu o soluție arbitrară și încearcă să găsească o soluție mai bună, printr-o modificare incrementală a soluției. Această modificare se face până se găsește soluția cea mai bună.

Acest algoritm poate fi aplicat pe problema comis-voiajorului, dar și pe problema de rutare a vehiculelor. Algoritmul are rolul de a obține un traseu cât mai scurt parcurgând fiecare client singură dată. [6].

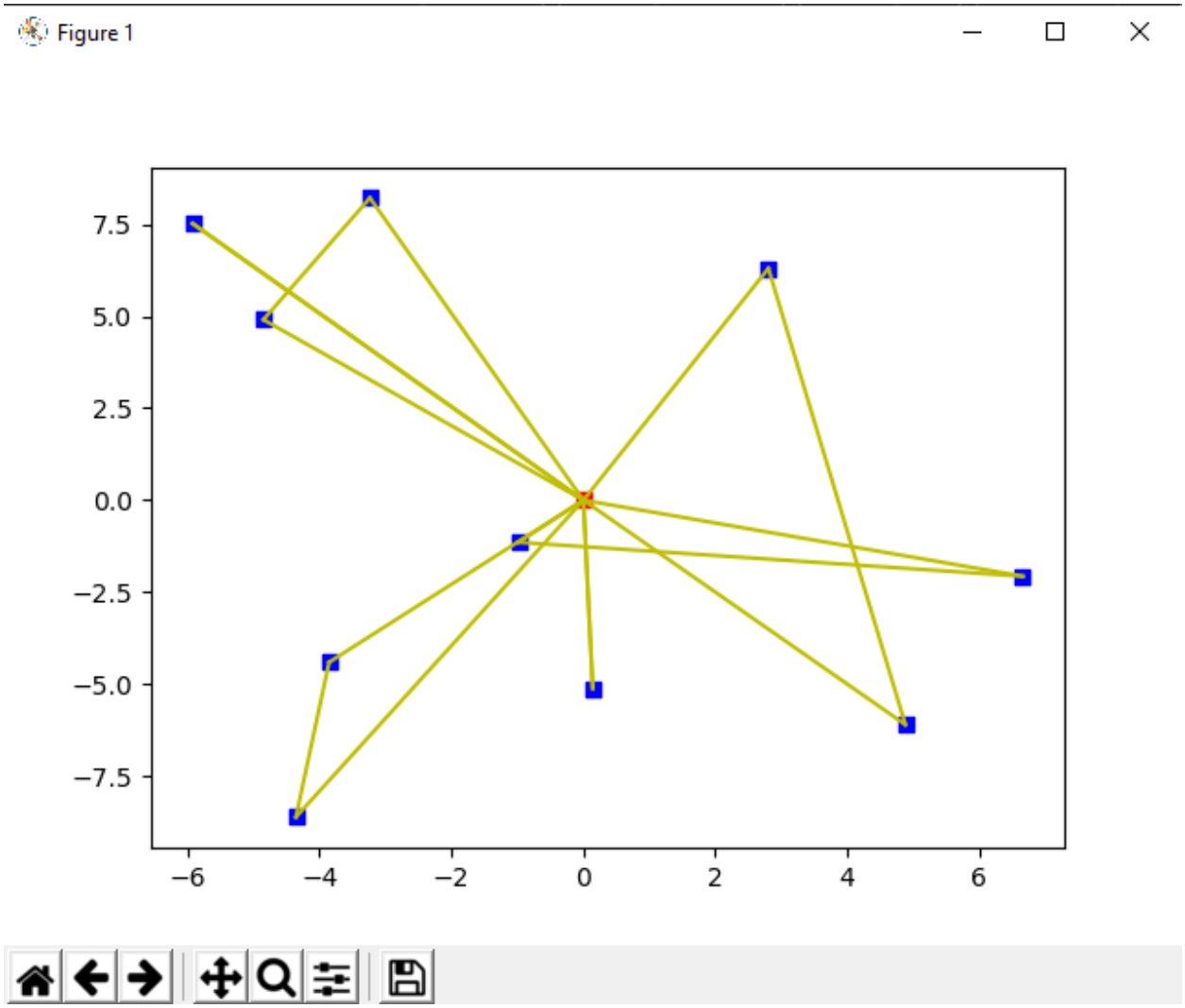


Figura 5: Soluția problemei de rutare a vehiculelor cu algoritmului *Hill Climbing*

În aplicarea acestui algoritm pe problema cu dronele am folosit trei pasi importanți.

Primul pas generează toate permutarile posibile cu centrele de drone existente, fiecare element din permutare reprezintă un centru de drone.

În pasul al doilea luăm fiecare permutare și pentru fiecare element din permutare generăm calea cea mai eficientă fără a depăsi limita distantei și a încărcăturii. Repetăm acest pas pentru pacienții rămași până găsim căi spre toți pacienții din raza centrului de drone.

Pasul al treilea salvăm căile obținute în urma pasului doi și comparăm cu căile obținute din permutarea anterioară dacă există. Dacă căile obținute sunt mai puține decât căile din permutarea anterioară atunci se salvează căile obținute ca fiind cea mai bună soluție. Apoi se apelează pasul doi pentru următoarea permutare. Se repetă pasul trei până parcurgem toate soluțiile permutărilor.

2.2.1.2.2. Ant Colony Optimization

Algoritmul *Ant Colony Optimization* este o tehnică probabilistică pentru rezolvarea problemelor de optimizare. Comportarea insectelor ce trăiesc în colectivități reprezintă o sursă de inspirație pentru dezvoltarea metodei de optimizare. O astfel de abordare se bazează pe comunicare pe cale chimica. Pe drumul parcurs pentru căutarea hranei, furnicile lasă o urmă pentru identificarea ulterioară a traseului care duce la găsirea hranei.

Algoritmii de căutate locală sunt metode des utilizate pentru rezolvarea problemelor de rutare a vehiculelor.

Pentru a implementa algoritmul *Ant Colony Optimization* am folosit libraria "*Pants*" din Python care ne oferă posibilitatea de a determina rapid cum se poate vizita o colecție de noduri interconectate, astfel încât să fie minimizată munca depusă. *Pants* este un intrument de rezolvare pentru problema comis-voiajorului, dar poate fi și un instrument bun și pentru problema de rutare a vehiculelor.

Soluțiile acestei probleme se găsesc printr-un proces iterativ. Pentru fiecare iterație, furnicilor li se permite să găsească o soluție care vizitează fiecare nod. Cantitatea de feromoni de pe fiecare traseu este actualizat în funcție de lungimea soluțiilor în care a fost utilizată. Soluția locală cea mai bună este considerată traseul cu distanța cea mai mică care a fost parcursă de furnică. Dacă soluția locală are o distanța mai scurtă față de orice soluție din iterația anterioară, atunci ea devine cea mai bună soluție. Procesul se repetă până la găsirea celei mai bune soluții globale [7].

Primul pas pentru *Ant Colony Optimization* este alegerea unui nod de pornire aleatoriu. Furnica crează un tur în problema de rutare a vehiculelor, alegând din nodul actual (inclusiv depozit) un alt nod care nu a fost încă vizitat. Pentru fiecare pas, muchia traversată este adăugată la soluție în construcție; când nu mai sunt noduri nevizitate furnica închide turul și se mută de la nodul ei actual la nodul din care a început construcția soluției. Aceasta constucție a soluției implică faptul că furnica are o memorie pentru a stoca nodurile vizitate. Construcția se face cu probabilitați.

După ce furnicile coloniei au finalizat construcția soluției, se realizează evaporarea feromonilor. După cum am specificat și mai sus, acest proces este iterativ și se oprește atunci când este indeplinită condiția de oprire. Presupunem că furnica să fie în nodul v_i, etapa de construcție se face cu probabilitatea: $$\mathbf{p}(e_{i,j}) = \frac{\tau_{i,j}}{\sum_{[k\in[1,\dots,|V|]|v_k\notin T]}\tau_{i,k}}, \quad \forall j \in \{1,\dots,|V|\},\ v_j \notin T.$$

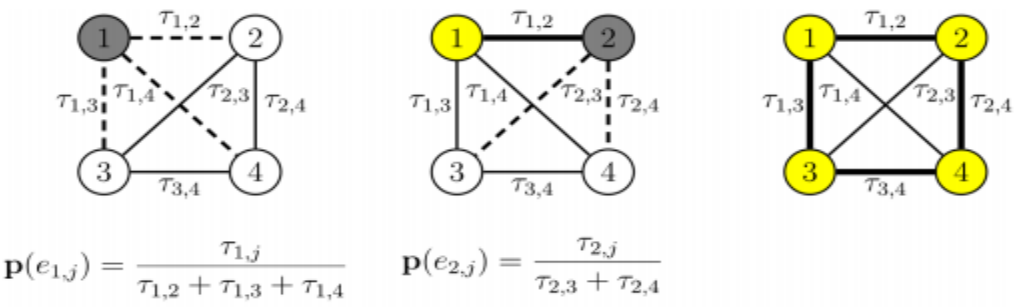


Figura 6: Demonstrație pe etape a algoritmului *Ant Colony Optimization*

(a) Prima etapă a constucției soluției.

(b) A doua etapă a constucției soluției

(c) Soluția completă după ultima etapă a constucției.

După ce furnicile colonieie au finalizat construcția soluției, se realizează evaporarea feromonilor după formula care urmează: $\tau_{i,j} \leftarrow (1-\rho)\cdot\tau_{i,j}, \quad \forall\tau_{i,j}\in\mathcal{T},$

unde T mulțimea tuturor valorilor feromoniei.

Apoi este îndeplinită călătoria de întoarcere. O furnică ce a constuit o soluție, execută pentru fiecare $e_{i,j} \in s$ următorul depozit de feromoni: $\tau_{i,j} \leftarrow \tau_{i,j} + \frac{Q}{f(s)},$

unde Q este o constantă pozitivă și $f(s)$ este valoarea funcțională obiectivă a soluției s. [8].

În figura 7 avem un model de funcționare a algoritmului *Ant Colony Optimization*.

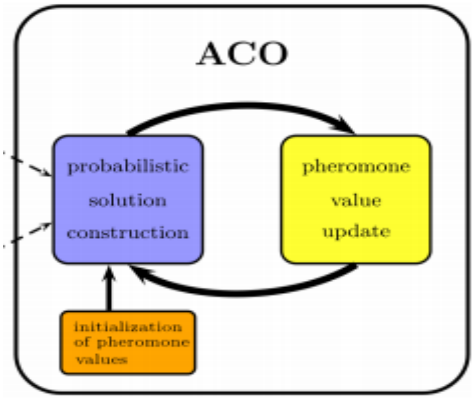


Figura 7: Model de funcționare a algoritmului *Ant Colony Optimization*

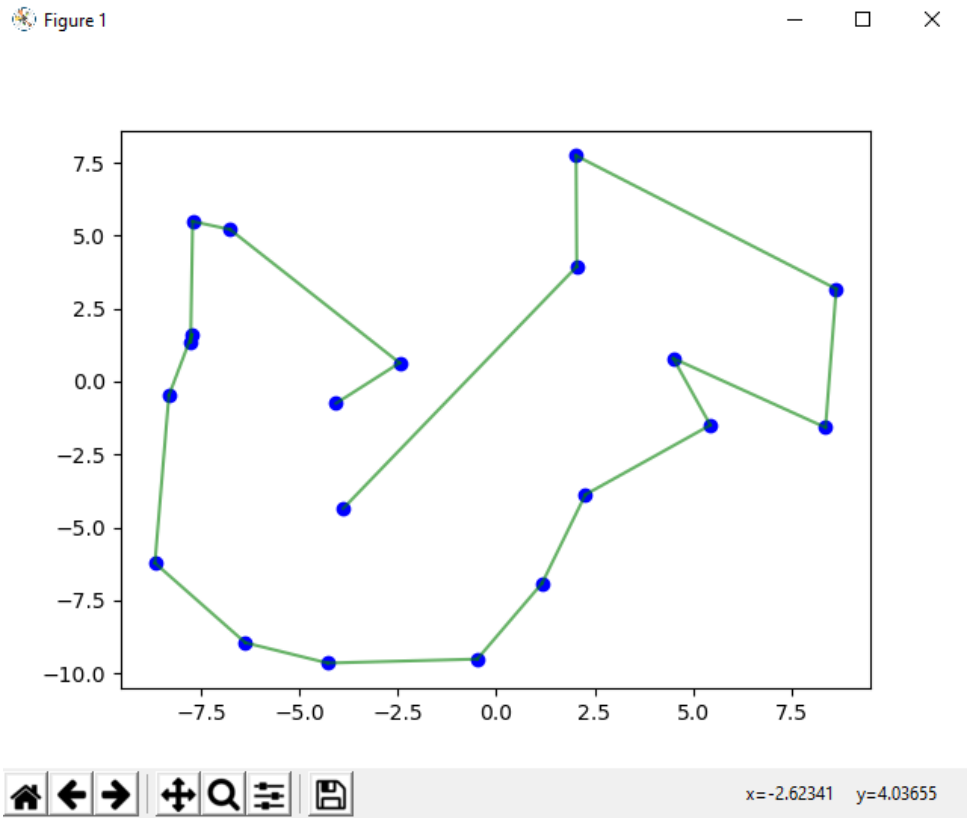


Figura 6: Soluția problemei de rutare a vehiculelor cu algoritmului *Ant Colony Optimization*

Algoritmul *Ant Colony Optimization* reprezintă o optiune bună pentru problema de rutare a vehiculelor, dar depinde de modul în care sunt definiți parametrii săi. Acest algoritm este dependent de un factor probabilistic și din aceasta cauză nu poate garanta existența unei soluții optime.

Algoritmul *Ant Colony Optimization* se bazează pe trei etape importante pentru rezolvarea problemei de rutare a vehiculelor pentru serviciile de asistență medicală asistată de drone.

Prima etapă constă în formarea căilor cele mai eficiente din punct de vedere al distanței și al încărcăturii în jurul fiecărui centru de drone cu pacienții care se află în raza centrului de drone respectiv.

A doua etapă reprezintă verificarea pacienților dacă se afla în mai multe căi și elimină pacienții pe o anumită cale pentru a face costul uniform. În calea în care costul e mai mare eliminăm pacientul care se găseste în ambele căi pentru a reduce din cost.

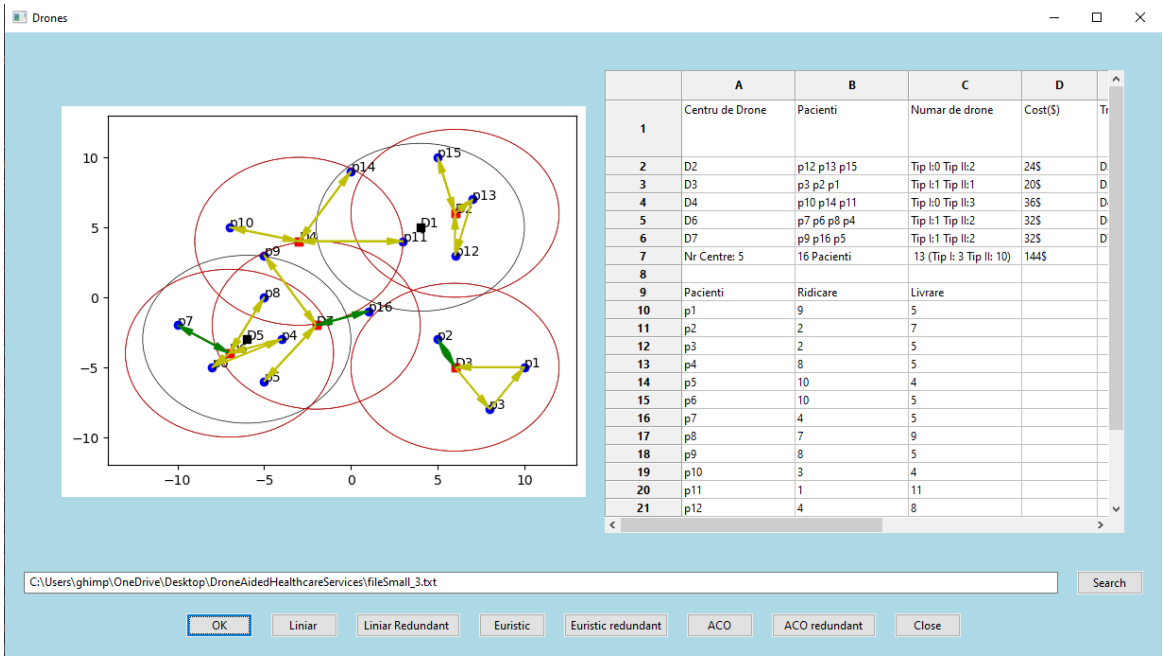
În al treilea pas se va lua calea pentru fiecare centru de drone și se va împărți în mai multe căi care pot fi parcurse de drone. Se ia fiecare pacient din calea creată în ordine și se verifică dacă pacientul adăugat la calea dronei nu depășește condițiile de distanța și cantitate de livrare și preluare. Dacă nu depășește va fi adăugat la calea dronei și se va încerca pentru următorul pacient. Atunci când nu mai poate adăuga pacienți se va salva calea curentă și va repeta același lucru cu pacienții rămasi.

3. Utilizare aplicație & Rezultate experimentale

3.1. Manual de utilizare a aplicației

În acest capitol vom prezenta cum putem folosi aplicația și cum putem selecta diferite variante de algoritmi în rezolvarea problemei. Pentru început este nevoie de un fișier cu date pentru a putea testa algoritmii. În acel fișier avem coordonatele pacienților, coordonatele centrelor de drone, raza de acoperire a centrului de drone, tipul de dronă (tip 1 și tip 2), cantitatea dronei mici și dronei mari.

În urma selectării unei instanțe de test, este afisată avem afișată o imagine în partea dreapta cu pacienții de culoare albastră, centrele de drone active de culoare roșie, centrele de drone redundante de culoare neagră, raza de acoperire cu culoare roșie pentru centrele de drone active și cu culoare neagră pentru centrele de drone redundante, traseele dronelor mari cu culoare galbenă și traseele dronelor mici cu culoare verde. În partea stângă vom avea un tabel care va conține toate informațiile importante despre elementele din imagine din partea stângă: numărul de centre de drone active, pacienții asignați acestor centre de drone, tipurile de drone folosite pentru a parcurge drumul de la fiecare centru de drone la pacienții respectivi și înapoi, costul dronelor din fiecare centru de drone, traseele și distanța parcursă în traseul respectiv, distanța totală parcursă și timpul de execuție pentru fiecare metodă de rezolvare.



	A	B	C	D
1	Centru de Drone	Pacienti	Numar de drone	Cost($)
2	D2	p12 p13 p15	Tip I:0 Tip II:2	24$
3	D3	p3 p2 p1	Tip I:1 Tip II:1	20$
4	D4	p10 p14 p11	Tip I:0 Tip II:3	36$
5	D6	p7 p6 p8 p4	Tip I:1 Tip II:2	32$
6	D7	p9 p16 p5	Tip I:1 Tip II:2	32$
7	Nr Centre: 5	16 Pacienti	13 (Tip I: 3 Tip II: 10)	144$
8				
9	Pacienti	Ridicare	Livrare	
10	p1	9	5	
11	p2	2	7	
12	p3	2	5	
13	p4	8	5	
14	p5	10	4	
15	p6	10	5	
16	p7	4	5	
17	p8	7	9	
18	p9	8	5	
19	p10	3	4	
20	p11	1	11	
21	p12	4	8	

Figura 7: Interfața pentru aplicație

Pentru utilizare avem o listă de butoane în subsolul aplicației prin care putem ușor selecta prin care metoda sa fie rezolvată problema. Fiecare dintre cei trei algoritmi au câte două butoane: primul buton este pentru a afișa toate elementele, iar al doilea buton este pentru a elimina centrele redundante din aplicație. Atunci când avem un exemplu pe o dimensiune mare butoanele asociate metodei de programare liniară sunt inactive.

3.2. Instanțe de testare

Pentru verificarea acestor metode de rezolvare am folosit 10 exemple diferite: 5 instanțe de dimensiuni reduse pentru care putem aplica cele 3 metode de rezolvare și 5 de dimensiuni mai mari pe care am aplicat doar algoritmul *Hill Climbing* și algoritmul *Ant Colony Optimization*. Rezultatele celor trei metode de rezolvare sunt diferite atât din punct de vedere al distanței cât și al timpului de execuție.

Algoritmii euristici au fost executați de 10 ori pentru a identifica media distanței totale parcurse, deviația standard a acesteia și media timpului de execuție și deviația standard a acesteia.

Instanțe de dimensiuni reduse:

Instanță 1:

Are un număr de 5 pacienți, 3 centre de drone inițiale, după eliminare rămân 2 și rază de acoperire a unui centru de drone de 6 km.

Instanța 2:

Are un număr de 17 pacienți, 6 centre de drone inițiale, după eliminare rămân 4 și rază de acoperire a unui centru de drone de 6 km.

Instanța 3:

Are un număr de 16 pacienți, 7 centre de drone inițiale, după eliminare rămân 5 și rază de acoperire a unui centru de drone de 6 km.

Instanța 4:

Are un număr de 18 pacienți, 9 centre de drone inițiale, după eliminare rămân 4 și o rază de acoperire a unui centru de drone de 6 km.

Instanța 5

Are un număr de 18 pacienți, 7 centre de drone inițiale, după eliminare rămân 5 și o rază de acoperire a unui centru de drone de 6 km.

Instanțe de dimensiuni mari:

Instanța 1:

Are un număr de 41 de pacienți, 9 centre de drone inițiale, după eliminare rămân 6 și o rază de acoperire a unui centru de drone de 12 km.

Instanța 2:

Are un număr de 38 de pacienți, 12 centre de drone inițiale, după eliminare rămân 6 și o rază acoperire a unui centru de drone de 16 km.

Instanța 3:

Are un număr de 40 de pacienți, 12 centre de drone inițiale, după eliminare rămân 7 și o rază de acoperire a unui centru de drone de 18 km.

Instanța 4:

Are un număr de 40 de pacienți, 12 centre de drone inițiale, după eliminare rămân 7 și o rază de acoperire a unui centru de drone de 18 km.

Instanța 5:

Are un număr de 40 de pacienți, 10 centre de drone inițiale, după eliminare rămân 6 și o rază de acoperire a unui centru de drone de 18 km.

3.3. Rezultate obținute

Rezultatele aplicării algoritmilor pe cele 10 instanțe se găsesc în Tabelul 2.

	Numar Centre Drone			Distanta Totala Parcursa						Timpul de Executie				
Exemplul	Nr Pacienti	Initial	Dupa Eliminare	Raza	Alg Liniar	Alg Hill Climbing	Deviatia Standard	Alg Ant Colony	Deviatia Standard	Alg Liniar	Alg Hill Climbing	Deviatia Standard	Alg Ant Colony	Deviatia Standard
fileSmall1	5	3	2	6	25.08	25.08	0	29.192	2.305194	1.49	1.234	0.23341	1.38	0.14966
fileSmall2	17	6	4	6	86.1	105.45	0	93.328	6.9669	2.24	1.562	0.0589	2.17	0.1185
fileSmall3	16	7	5	6	88.322	117.57	1.59	94.39	5.0133	2.466	1.566	0.0507	2.33	0.1519
fileSmall4	18	9	4	6	112.258	118.356	2.47	118.2	0.302	1.662	2.55	0.042	2.342	0.1089
fileSmall5	18	7	5	6	119.282	120.75	0	126.226	2.8965	2.282	1.662	0.0506	2.302	0.0694
fileBig1	41	9	6	12	-	599.73	4.4001	472.018	77.7674	-	3.132	0.1103	4.658	1.0149
fileBig2	38	12	7	16	-	841.53	2.8734	807.59	21.5412	-	3.15	0.0827	4.098	0.2926
fileBig3	40	12	7	18	-	940.19	1.27	834.694	29.1338	-	3.29	0.1462	4.69	0.2148
fileBig4	40	12	7	18	-	1034.955	6.6329	896.5695	34.8219	-	4.223	0.5733	6.886	0.3679
fileBig5	40	10	6	18	-	849.766	8.4777	764.806	19.9774	-	3.238	0.3123	4.964	0.3654

Tabel 2: Rezultatele celor trei algoritmi pe cele 10 instanțe

Instanțe de dimensiuni reduse:

Instanța 1:

Rezultatele algoritmului de programare liniară: distanța totală parcursă de 25.08 km, timpul de execuție de 1.49 ms și deviația standard 0.29026.

Rezultatele algoritmul *Hill Climbing*: distanța totală parcursă de 25.08 km cu deviația standard 0 și timpul de execuție 1.234 ms cu deviația standard 0.23341.

Rezultatele algoritmul *Ant Colony Optimization*: distanța totală parcursă de 29.192 km cu deviația standard 2.3051 și timpului de execuție 1.38 ms cu deviația standard 0.14966.

Instanta 2:

Rezultatele algoritmului de programare liniară: distanța totală parcursă 86.1 km și timpul de execuție 2.24 ms cu deviația standard 0.09551.

Rezultatele algoritmul *Hill Climbing*: distanța totală parcursă 105.45 km cu deviația standard 0 și timpului de execuție de 1.562 ms cu deviația standard 0.0589.

Rezultatele algoritmul *Ant Colony Optimization*: distanța totală parcursă 93.328 km cu deviația standard 6.9669 și timpul de execuție 2.17 ms cu deviația standard 0.1185.

Instanța 3:

Rezultatele algoritmului de programare liniară: distanța totală parcursă 88.322 km și timpul de execuție 2.406 ms cu deviația standard 0.1937.

Rezultatele algoritmul *Hill Climbing*: distanța totală parcursă 117.57 km cu deviația standard 1.59 și timpului de execuție de 1.566 ms cu deviația standard 0.0507.

Rezultatele algoritmul *Ant Colony Optimization*: distanța totală parcursă 94.39 km cu deviația standard 5.0133 și timpul de execuție 2.33 ms cu deviația standard 0.1519.

Instanța 4:

Rezultatele algoritmului de programare liniară: distanța totală parcursă 112.258 km și timpul de execuție 1.662 ms cu deviația standard 0.042.

Rezultatele algoritmul *Hill Climbing*: distanța totală parcursă 118.356 km cu deviația standard 2.47 și timpului de execuție de 2.55 ms cu deviația standard 0.042.

Rezultatele algoritmul *Ant Colony Optimization*: distanța totală parcursă 118.2 km cu deviația standard 0.302 și timpul de execuție 2.302 ms cu deviația standard 0.0694.

Instanța 5:

Rezultatele algoritmului de programare liniară: distanța totală parcursă 119.282 km și timpul de execuție 2.282 ms cu deviația standard 0.0486.

Rezultatele algoritmul *Hill Climbing*: distanța totală parcursă 120.75 km cu deviația standard 0 și timpului de execuție de 1.662 ms cu deviația standard 0.0506.

Rezultatele algoritmul *Ant Colony Optimization*: distanța totală parcursă 126.226 km cu deviația standard 2.8965 și timpul de execuție 2.302 ms cu deviația standard 0.0694.

Instanțe de dimensiuni mari:

Instanța 1:

Rezultatele algoritmul *Hill Climbing*: distanța totală parcursă 599.73 km cu deviația standard 4.4001 și timpului de execuție de 3.132 ms cu deviația standard 0.1103.

Rezultatele algoritmul *Ant Colony Optimization*: distanța totală parcursă 472.018 km cu deviația standard 77.7674 și timpul de execuție 4.658 ms cu deviația standard 1.0149.

Instanța 2:

Rezultatele algoritmul *Hill Climbing*: distanța totală parcursă 841.53 km cu deviația standard 2.8734 și timpului de execuție de 3.15 ms cu deviația standard 0.0827.

Rezultatele algoritmul *Ant Colony Optimization*: distanța totală parcursă 807.59 km cu deviația standard 21.5412 și timpul de execuție 4.098 ms cu deviația standard 0.2926.

Instanța 3:

Rezultatele algoritmul *Hill Climbing*: distanța totală parcursă 940.19 km cu deviația standard 1.27 și timpului de execuție de 3.29 ms cu deviația standard 0.1462.

Rezultatele algoritmul *Ant Colony Optimization*: distanța totală parcursă 834.694 km cu deviația standard 29.1338 și timpul de execuție 4.69 ms cu deviația standard de 0.2148.

Instanța 4:

Rezultatele algoritmul *Hill Climbing*: distanța totală parcursă 1034.955 km cu deviația standard 6.6329 și timpului de execuție de 4.223 ms cu deviația standard 0.5733.

Rezultatele algoritmul *Ant Colony Optimization*: distanța totală parcursă 896.5695 km cu deviația standard 34.8219 și timpul de execuție 6.886 ms cu deviația standard de 0.3679.

Instanța 5:

Rezultatele algoritmul *Hill Climbing*: distanța totală parcursă 849.766 km cu deviația standard 8.4777 și timpului de execuție de 3.238 ms cu deviația standard 0.3123.

Rezultatele algoritmul *Ant Colony Optimization*: distanța totală parcursă 764.806 km cu deviația standard 19.9774 și timpul de execuție 4.964 ms cu deviația standard de 0.3654.

În continuare vom detalia rezultatele obținute pentru două instanțe: una de dimensiune reduse și una de dimensiune mare.

Au fost executate 20 de rulări pentru algoritmul de programare liniară și pentru algoritmii euristici. Rezultatele obținute de algoritmul de programare liniară și de algoritmii euristici se găsesc în Tabelul 3.

În urma rulării algoritmilor pe un exemplu de dimensiune redusă cu 18 pacienți, 9 centre de drone inițiale după eliminare rămân doar 4 și raza de acoperire a centrului de drone de 6 km am observat anumite diferențe. Putem observa ușor că distanța parcursă și timpul de execuție pentru fiecare algoritm sunt diferite.

Putem clasifica rezultatele obținute ale algoritmilor în funcție de distanța parcursă sau în funcție de timpul de execuție. Din punct de vedere al distanței parcurse în topul clasificării este algoritmul liniar care în urma mediei făcute pe 20 de rulări avem distanța de 110.0645 km, următorul algoritm eficient din punct de vedere al distanței este algoritmul *Ant Colony Optimization* cu 121.259 km, pe al treilea loc se poziționează algoritmul *Hill Climbing* cu 125.19 km.

Din punct de vedere al timpului de execuție putem observa ca algoritmul *Hill Climbing* este cel mai eficient cu 2.045 ms pe al doilea loc este algoritmul de programare liniara cu 2.357 ms și ultimul este algoritmul *Ant Colony Optimization* cu 2.935 ms.

	Distanta Totala Parcursa			Timpul de Executie		
fileSmall_4	Alg Liniar	Alg Hill Climbing	Alg Ant Colony	Alg Liniar	Alg Hill Climbing	Alg Ant Colony
Rulare 1	110.34	125.19	122.03	2.16	2.13	2.62
Rulare 2	110.34	125.19	112.27	2.39	2.09	2.7
Rulare 3	110.34	125.19	117.96	2.25	2.16	2.16
Rulare 4	110.34	125.19	119.76	2.22	2.13	2.61
Rulare 5	110.34	125.19	112.8	2.23	1.63	2.52
Rulare 6	110.34	125.19	121.19	2.09	1.8	3.66
Rulare 7	110.34	125.19	120.73	2.47	2.17	3.02
Rulare 8	110.34	125.19	122.71	2.27	2.06	2.78
Rulare 9	110.34	125.19	116.84	2.09	1.84	2.51
Rulare 10	110.34	125.19	112.27	2.18	2.1	2.42
Rulare 11	110.34	125.19	131.74	2.06	2.12	2.49
Rulare 12	110.34	125.19	125.83	2.8	2.01	3.22
Rulare 13	110.34	125.19	125.59	2.18	2.07	2.65
Rulare 14	110.34	125.19	117.72	2.15	2.09	2.66
Rulare 15	110.34	125.19	114.69	2.57	1.7	2.95
Rulare 16	110.34	125.19	136.18	2.39	2.33	3.36
Rulare 17	110.34	125.19	118.06	2.58	1.94	3.05
Rulare 18	110.34	125.19	127.59	2.83	2.56	4.51
Rulare 19	110.34	125.19	125.83	2.47	2.12	2.99
Rulare 20	110.34	125.19	123.39	2.76	1.85	3.82
Media	-	125.19	121.259	2.357	2.045	2.935
Deviatia Standart	-	2.91601E-14	6.375806162	0.244327218	0.211821673	0.55879099

Tabel 3: Rezultatele pentru 20 rulări pe o instanță de dimensiune redusă

Aceeași analiză a fost făcută și pentru un exemplu de dimensiune mare cu 40 de pacienți, 12 centre de drone după eliminare rămân doar 7 și pentru o rază de acoperire a centrului de drone de 18 km.

În urma rulărilor algoritmilor pentru un exemplu de dimensiune mare observăm că algoritmul *Ant Colony Optimization* este mai eficient din punct de vedere al distanței parcurse care este 896.5695 km dar nu și din punct de vedere al timpului de execuție cu 6.886 ms.

Pentru algoritmul *Hill Climbing* avem o distanța de 1034.8955 km dar un timp de execuție de 4.223 ms care este mai mic față de timpul algoritmului *Ant Colony Optimization*.

	Distanta Totala Parcursa				Timpul de Executie	
fileBig_4	Alg Liniar	Alg Hill Climbing	Alg Ant Colony	Alg Liniar	Alg Hill Climbing	Alg Ant Colony
Rulare 1	-	1047.05	863.72	-	4.73	6.73
Rulare 2	-	1030.91	911.66	-	5.25	8.19
Rulare 3	-	1040.54	891.27	-	5.85	7.08
Rulare 4	-	1030.91	862.41	-	4.01	6.95
Rulare 5	-	1030.91	891.66	-	3.97	6.89
Rulare 6	-	1040.54	894.7	-	4.08	6.79
Rulare 7	-	1030.91	868.11	-	3.85	6.63
Rulare 8	-	1047.81	865.85	-	4.12	6.67
Rulare 9	-	1030.91	908.05	-	4.06	6.47
Rulare 10	-	1030.91	862.68	-	3.71	6.65
Rulare 11	-	1030.91	913.64	-	4.94	7.02
Rulare 12	-	1030.91	894.79	-	3.65	6.84
Rulare 13	-	1030.91	906.2	-	3.69	6.89
Rulare 14	-	1030.91	902.33	-	3.98	7.01
Rulare 15	-	1039.8	915.77	-	4.16	7
Rulare 16	-	1049.43	838.39	-	4.64	6.41
Rulare 17	-	1030.91	985.83	-	4.08	6.55
Rulare 18	-	1030.91	938.59	-	3.87	7.13
Rulare 19	-	1030.91	948.57	-	3.99	6.97
Rulare 20	-	1030.91	867.17	-	3.83	6.85
Media	-	1034.8955	896.5695	-	4.223	6.886
Deviatia Standart	-	6.632938918	34.82199237	-	0.573329151	0.36795881

Tabel 4: Rezultatele pentru 20 rulări pe o instanță de dimensiune mare

Concluzii

În această lucrare am folosit *o* abordare de tip *Set Covering* pentru a determina unde vor fi amplasate centrele de drone și câte centre de drone să fie active pentru a acoperi cu raza lor toți pacienții. Aceasta mai are și rolul de a identifica numărul minim de centre de drone și de a stabili pentru fiecare pacient cărui centru de drone îi corespunde. Acest prim pas ajută la eliminarea centrelor de drone care sunt redundante și au o rază de acoperire care se suprapune.

Problema de rutare a vehiculelor are ca și scop construirea unei mulțimi de trasee pentru vehiculele care sunt capabile de a vizita clientul cu un cost minim și care satisfac constrângerile privind capacitatea vehiculelor.

Pentru problema de rutare a vehiculelor am folosit programare liniară cu CPLEX care a transformat această problemă într-un model matematic și a găsit cea mai bună soluție. Acest algoritm crează o matrice care conține distanțele între fiecare cuplu posibil de clienți și distanțele dintre depozit și fiecare client. Rezultatul programării liniare este optim din punct de vedere al distanței parcurse, însă poate fi aplicat doar pe dimensiuni reduse ale problemei.

Algoritmul *Hill Climbing* are și el un rol important în rezolvarea problemei de rutare a vehiculelor. Este un algoritm iterativ care începe cu o soluție arbitrară și încearcă să găsească o soluție mai bună, printr-o modificare incrementală a soluției. Algoritmul are rolul de a obține un traseu cât mai scurt parcurgând fiecare client singură dată.

Un alt algoritm pentru rezolvarea problemei de rutare a vehiculelor este *Ant Colony Optimization*. Algoritmul *Ant Colony Optimization* este o tehnică probabilistică pentru rezolvarea problemelor de optimizare. Comportarea insectelor ce trăiesc în colectivități reprezintă o sursă de inspirație pentru dezvoltarea metodei de optimizare. Pe drumul parcurs pentru căutarea hranei, furnicile lasă o urmă pentru identificarea ulterioară a traseului care duce la găsirea hranei. Soluțiile acestei probleme se găsesc printr-un proces iterativ. Pentru fiecare iterație, furnicilor li se permite să găsească o soluție care vizitează fiecare nod.

Bibliografie

1. Seon Jin Kim, Gino J. Lim, Jaeyoung Cho, Murray J. Cot.

Drone-Aided Healthcare Services for Patients with Chronic Diseases in Rural Areas (2017) Tamara Stern. *Set Covering Problem*

https://math.mit.edu/~goemans/18434S06/setcover-tamara.pdf (2006)

2. João Pedro Pedroso and Abdur Rais and Mikio Kubo and Masakazu Muramatsu. *Routing Problems*

https://scipbook.readthedocs.io/en/latest/routing.html?fbclid=IwAR0bvXeqhC_OJoJGH65QmG4AmHkKaQSuUL-Wr9VJa-TyZFjwIpYB53i9aFs (2012)

3. Dominique Bertrand. *IBM Decision Optimization with CPLEX samples*

https://github.com/IBMDecisionOptimization/Decision-Optimization-with-CPLEX-samples (2017)

4. Mohsen Moarefdoost. *Optimization Modeling in Python: PuLP, Gurobi, and CPLEX*

https://medium.com/opex-analytics/optimization-modeling-in-python-pulp-gurobi-and-cplex-83a62129807a (2018)

5. Upasana. *An Introduction to Hill Climbing Algorithm*

https://www.edureka.co/blog/hill-climbing-algorithm-ai/ (2019)

6. Robert Grant. *A Python3 implementation of the Ant Colony Optimization Meta-Heuristic*

https://pypi.org/project/ACO-Pants/ (2014)

7. Christian Blum *Ant colony optimization: Introduction and recent trends*

https://www.ics.uci.edu/~welling/teaching/271fall09/antcolonyopt.pdf (2005)