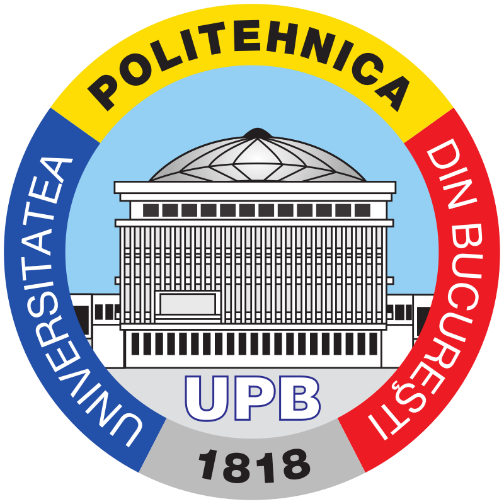
# Algoritmi de optimizare pentru amplasarea

# statilor de incarcare electrica

# in Bucuresti



## Marin Florin-Daniel

### 324 AA, cursul de Tehnici si Optimizare

### Facultatea de Automatica si Calculatoare,

### Universitatea Politehnica din Bucuresti.

#### Prof. curs Ion Necoara

#### Prof. seminar si laborator Daniela Lupu

# Introducere

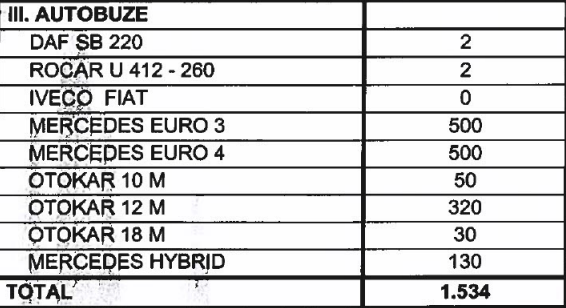
Consumul de carburanti reprezinta o problema reala in cadrul zonelor metropolitane aglomerate. Traficul reprezinta o sursa de poluare, iar state precum China, Japonia, Canada si U.E. au implementat masuri de reducere a traficului, a carburantilor derivati din petrol, astfel imbunatatind calitatea aerului si a vietii in general. Masurile bio sunt considerate cele ce renunta la resursele epuizabile ale solului, in favoarea resurselor neepuizabile. Aceste masuri previn incalzirea globala si mentin un mediu sanatos.

In prezent, U.E. sustine schimbarea sistemului de transport in comun, de la cel bazat pe carburanti (sistemul de autobuze), la cel bazat pe consum electric: tramvaie, autobuze electrice, troleibuze, trenuri. Studii de fezabilitate au aratat ca sistemele de transport in comun, electrice, au un cost mai mic fata de cele pe carburanti, insa diferenta se observa dupa un numar relativ mare de ani de la implementarea sistemului electric, deoarece costurile initiale: sine electrice, respectiv statiile de incarcare, resprectiv cablurile electrice sunt, evident, un cost nou, comparativ cu transportul bazat pe autobuze.

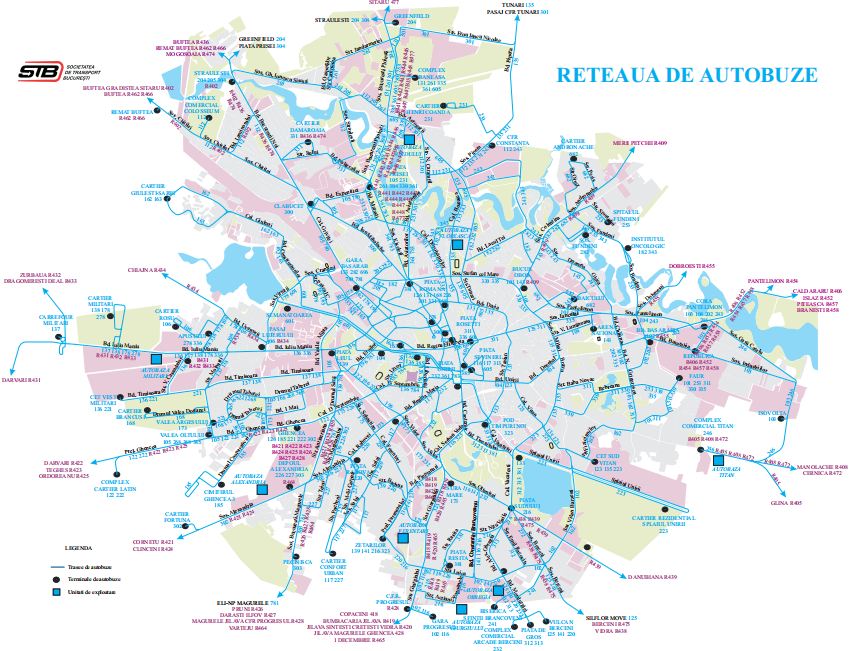
In continuare voi prezenta solutia la problemele legate de poluare in orasul Bucuresti: Innoirea retelei de autobuze clasice, cu autobuze electrice. Din 2021 se incearca achizitia a 100 de autobuze electrice in Bucuresti.

# Date tehnice

STB are in compozitia sa, 1534 de autobuze ce deservesc atat linile urbane, cat si cele regionale.



In zona urbana Bucuresti, STB intretine 103 linii de autobuz. Un traseu are in medie 11 km si sunt, in medie, 10 autobuze pe linie. In total, din cele 1534 de autobuze disponibile, 1200 sunt amplasate pe linile urbane. Scopul acestei lucrari este schimbarea reteaua urbana, deoarece innoirea intregii retele regionale, ar insemna ampasarea statiilor de incarcare electrica la fiecare terminal.



In zona urbana, sunt 78 de terminale (capete de linii) pentru autobuze.

# Statiile de incarcare electrica

Infrastructura amplasarii statilor de incarcare electrica este cel mai important pas in asigurarea unui sistem de transport in comun bazat pe autobuze electrice. In aceasta lucrare am ales implementarea statilor de incarcare electrica DC, care permit sa incarce o baterie a unui autobuz in 15 minute.

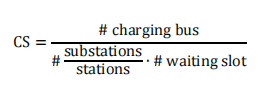
Scopul este de a calcula numarul optim de astfel de statii si de a gasi, pe baza unor algoritmi de optimizare, locatiile de amplasare ale acestora.

Exista 2 optiuni extreme de amplasare: la fiecare statie si respectiv la depozit.



In realitate, acestea se vor amplasa la terminalele linilor, sau, pentru a optimiza costul, intre terminale. Astfel, intre 5 terminale, de ex., se va localiza o statie de incarcare electrica, permitand autobuzelor care au completat traseul, sa circule pana la statia de incarcare si inapoi la terminal, asteptand urmatorul traseu. O astfel de metoda, minimizeaza numarul de statii de incarcare, cu costul maririi timpului dintre curse pentru un autobuz (30 min – 1h).

Numarul de statii de incarcare se poate calcula cu formula urmatoare:



Unde, numaratorul reprezinta numarul maxim de vehicule ce pot fi in acelasi timp la statia de incarcare, iar prima fractie de la numitor reprezinta in medie numarul de substatii de incarcare pentru o statie, iar al doilea factor timpul de asteptare pentru un autobuz, astfel incat sa respecte programul.

Pentru orasul Bucuresti, **CS = 15**.

# Optimizarea amplasarii

Functia obiectiv este:



Si reprezinta totalul **distantelor** de la **fiecare terminal** la **statia de incarcare cea mai apropiata**.

x\_j = coordonata statiei de incarcare pe latitudine

y\_j = coordonata statiei de incarcare pe longitudine

a\_j = coordonata terminalului cea mai apropiata pe latitudine

b\_j = coordonata terminalului cea mai apropiata pe longitudine

N = numarul de statii de incarcare

Minimizarea acestei functii presupune minimizarea distantelor de la terminale la statiile de incarcare cele mai apropiate. Deci, prin minimizare intelegem schimbarea coordonatelor x\_j, y\_j, pentru fiecare terminal, respectiv schimbarea variabilelor x, y.

Constrangerile sunt urmatoarele:

1. Coordonatele statilor de incarcare trebuie sa fie in interiorul zone urbane Bucuresti
2. Impunem o distanta minima intre 2 statii de incarcare
3. Impunem un numar minim de iteratii a algoritmilor

# Datele numerice

Date despre statiile de incarcare:

x = [x1 x2 … xN]

y = [y1 y2 … yN]

N = 14;

Date despre terminale:

a = [a1 a2 .. .am],

b = [b1 b2 … bm].

m = 78;

Constrangeri:

1. N: 44.54 long, S: 44.33 long. V: 25.95 lat, E: 26.22
2. Min\_distance = 0.015 (1.5 km)
3. Nr\_iteratii: 2000

# Algoritmii de optimizare

## Metoda Gradient Descent

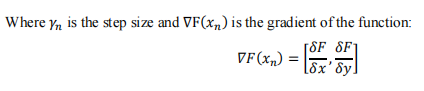
Este o metoda de ordinul 1 iterativa, un algoritm de optimizar inventat de Cauchy in 1847. Conceptul este foarte usor de inteles, insa metoda este robusta si foarte utilizata. Metoda gradient descent poate computa pentru multe iteratii pana ajunge la un minim local.

O metoda mai imbunatatita este metoda Newton, care calculeaza Hessiana functiei obiectiv la fiecare iteratie, deci fiecare iteratie are un cost mai mare, dar numarul de iteratii este mai mic.

La fiecare pas al algoritmului, gradientul functii eroare este calculat, iar parametrul liber al sistemului (in cazul acesta: coordonatele) sunt updatate.

Metoda se bazeaza pe observatia ca daca functia F(x) este definita si diferentiabila intr-o vecinatate a unui punct “a”, atunci F(x) scade mai rapid daca se merge din a spre directia gradientului negativ al lui F in x, 

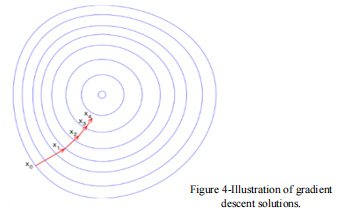




Y\_n reprezinta pasul algoritmului, si poate sa fie constant sau ajustabil. Pasul constant se calculeaza cupa y\_k = 1/L

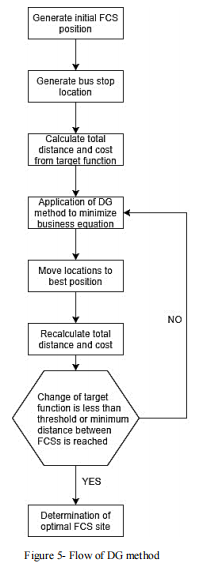
unde L este constanta Lipschitz.

Deci si va converge catre un minimum local:

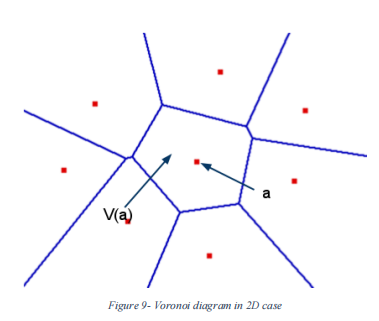


Algoritm:

1. Generez cele N puncte ale statilor de incarcare initiale,
2. Pozitionez in areea urbana punctele m ale terminalelor liniilor de autobuz
3. Calculez distanta totala de la fiecare terminal la statia de incarcare cea mai apropiata
4. Aplic algoritmul descent gradient pentru a minimiz distanta totala, cu rezultatele noi fiind noile coordonate ale statior
5. Mut locatile la cele noi
6. Recalculez distanta totala si ma asigur ca nu sunt incalcate restrictiile. Daca restrictile au fost atinse, rezultatele sunt coordonatele optime, daca nu, ma intorc la punctul d.



## 6.2 Diagrama Voronoi



Folosesc diagrama Voronoi pentru a putea vizualiza statiile de incarcare si terminalele autobuzelor aferente.

# Codul Matlab

1. %%datele numerice
2. A = [44.363445 26.113015;
3. 44.359452 26.139076;
4. 44.360901 26.130089;
5. 44.362403 26.123089;
6. 44.365975 26.090950;
7. 44.369737 26.089105;
8. 44.385636 26.059393;
9. 44.378352 26.046974;
10. 44.386193 26.077606;
11. 44.379248 26.099150;
12. 44.377019 26.120312;
13. 44.337033 26.146689;
14. 44.383988 26.145767;
15. 44.393794 26.121808;
16. 44.401250 26.098168;
17. 44.394529 26.201010;
18. 44.408025 26.120181;
19. 44.416068 26.113688;
20. 44.425778 26.102295;
21. 44.425796 26.103190;
22. 44.416252 26.083089;
23. 44.403581 26.066926;
24. 44.389785 26.005224;
25. 44.403892 26.021521;
26. 44.398178 26.044341;
27. 44.414885 26.046270;
28. 44.404968 25.990272;
29. 44.413109 26.014802;
30. 44.411915 26.017072;
31. 44.415941 26.015597;
32. 44.418508 26.002314;
33. 44.422216 25.982745;
34. 44.430558 26.086232;
35. 44.427744 26.100968;
36. 44.434337 26.076113;
37. 44.434007 26.059071;
38. 44.445496 26.046257;
39. 44.434875 26.034641;
40. 44.437749 26.014307;
41. 44.441405 26.003688;
42. 44.446332 25.982422;
43. 44.435421 25.958062;
44. 44.429092 26.102444;
45. 44.427684 26.104017;
46. 44.432549 26.103962;
47. 44.429560 26.107329;
48. 44.436092 26.106472;
49. 44.436539 26.101757;
50. 44.437668 26.095055;
51. 44.445334 26.097724;
52. 44.451734 26.085474;
53. 44.445707 26.075102;
54. 44.481635 26.147613;
55. 44.485989 26.113368;
56. 44.496830 26.098813;
57. 44.507808 26.086235;
58. 44.514824 26.114980;
59. 44.538284 26.092608;
60. 44.503307 26.024157;
61. 44.491466 26.014095;
62. 44.488078 26.050176;
63. 44.471273 26.051520;
64. 44.471452 26.000543;
65. 44.475566 26.072153;
66. 44.405916 26.148251;
67. 44.406437 26.202304;
68. 44.414977 26.217733;
69. 44.434083 26.185125;
70. 44.439983 26.184771;
71. 44.436227 26.169045;
72. 44.426799 26.178817;
73. 44.470446 26.176282;
74. 44.462871 26.155643;
75. 44.465017 26.148199;
76. 44.466689 26.156246;
77. 44.448406 26.125918;
78. 44.446500 26.143738;
79. 44.439500 26.150737;
80. ];
82. x0 = [26.09,26.165,26.015,26.09,26.165,26.218,25.94,26.015,26.09,26.165,26.218,26.015,26.09,26.1654,26.09];
83. y0 = [44.34,44.34,44.39,44.39,44.39,44.39,44.44,44.44,44.44,44.44,44.44,44.49,44.49,44.49,44.54];
85. load matrix.mat
86. for j = 1: length(A(:,2))
87. for i = 1: length (x0)
88. if matrix(j,i) ~= 0
89. matrix\_2(j,i) = A(j,1);
90. end
91. end
92. end
94. plot(A(:, 2), A(:, 1), 'rx')
95. hold on
96. plot(x0, y0, 'bx')
97. %%algoritm:
99. min\_distance = 1;
100. cont = 0;
101. s = 0;
103. %%calcul constanta lipschitz:
104. for j = 1:length (x0)
105. for i = 1:length (A(:,1))
106. norma = sqrt((x0(j) - matrix(i,j))^2 + (y0(j) - matrix\_2(i,j))^2);
107. s = s + norma \* norma;
108. end
109. end
111. L = s/4;
112. alfa = 1/L;
113. %%
115. while min\_distance > 0.025 && cont < 10000
116. %%Metoda Gradient Descent
117. cont = cont + 1;
119. %derivatele partiale
120. fx = zeros (1 , length (x0));
121. fy = zeros (1 , length (x0));
122. d(cont, length(A(:,1))) = 0;
124. %%
125. vett0 = [x0; y0];
127. %1st cycle pentru fiecare statie de incarcare
128. for j = 1:length (x0)
129. %2nd cycle pentru fiecare terminal
130. for i = 1:length (A(:,1))
131. if matrix (i,j) ~= 0
132. fx(j) = fx(j) + (x0(j) - matrix (i,j))/sqrt((x0(j) - matrix (i,j))^2+(y0(j)-matrix\_2(i,j))^2);
133. fy(j) = fy(j) + (y0(j) - matrix\_2 (i,j))/sqrt((x0(j) - matrix (i,j))^2+(y0(j)-matrix\_2(i,j))^2);
135. d(cont, i) = d(cont, i) + sqrt((x0(j)-matrix(i,j))^2+(y0(j)-matrix\_2(i,j))^2);
136. end
137. end

140. %anti gradientul
141. p=-[fx;fy];
142. for i = 1:length(p)
143. if p(i) == 0
144. p(i) = 0.01;
145. end
146. end
148. %noile coordonate pentru statiile de incarcare
149. vett1 = vett0 + alfa \* p;
151. %ri
152. x0 = vett1 (1,:);
153. y0 = vett1 (2,:);
155. %distanta minima dintre statile de incarcare
156. for k = 1:length(x0)
157. for t = 1:length(x0)
158. if k ~= t
159. distance(k,t) = sqrt((x0(k) - x0(t))^2 + (y0(k) - y0(t))^2);
160. end
161. if k == t
162. distance (k,t) = 5;
163. end
164. end
165. end
166. for k = 1:length(x0)
167. for t = 1:length(x0)
168. if distance(k,t) < min\_distance
169. min\_distance = distance (k,t);
170. end
171. end
172. end
174. end
175. end
176. %%
177. plot (d(:,78))
178. %%
179. plot(A(:, 2), A(:, 1), 'rx');
180. hold on;
181. voronoi(x0, y0, '-g');

# 8. Genetic Algoritm

global R A

nvars = 30;

R = 0.025;

x0 = [26.09,26.165,26.015,26.09,26.165,26.218,25.94,26.015,26.09,26.165,26.218,26.015,26.09,26.1654,26.09];

y0 = [44.34,44.34,44.39,44.39,44.39,44.39,44.44,44.44,44.44,44.44,44.44,44.49,44.49,44.49,44.54];

LB = [25.95 25.95 25.95 25.95 25.95 25.95 25.95 25.95 25.95 25.95 25.95 25.95 25.95 25.95 25.95 44.33 44.33 44.33 44.33 44.33 44.33 44.33 44.33 44.33 44.33 44.33 44.33 44.33 44.33 44.33];

UB = [26.22 26.22 26.22 26.22 26.22 26.22 26.22 26.22 26.22 26.22 26.22 26.22 26.22 26.22 26.22 44.54 44.54 44.54 44.54 44.54 44.54 44.54 44.54 44.54 44.54 44.54 44.54 44.54 44.54 44.54];

objectivefun = @objectiveFun;

Constraintf = @constraintsFun;

%options

options = gaoptimset ('MutationFcn', @mutationadaptfeasible, 'PopulationSize',20,'Generations',1000);

options = gaoptimset (options, 'PlotFcns', {@gaplotbestf}, 'Display', 'iter');

options = gaoptimset (options, 'InitialPopulation', X0);

[X,FVAL,EXITFLAG,OUTPUT,POPULATION,SCORES] = ga(@objectivefun,nvars,[],[],[],[],LB,UB,[],options);

**Cu functia objective:**

function y = objectiveFun (x0)

load matrix.mat

for j = 1: length(A(:,2))

for i = 1: 15

if matrix(j,i) ~= 0

matrix\_2(j,i) = A(j,1);

end

end

end

d = 0;

for j=1:length(A(:,1))

for i = 1:15

pos = zeros(1, 15);

if matrix(j,i) ~= 0

pos(1,i) = 1;

d = d + pos(1) \* sqrt((x0(1) - matrix (j,1))^2+(x0(16)-matrix\_2(j,1))^2) + pos(2) \* sqrt((x0(2) - matrix (j,2))^2+(x0(17)-matrix\_2(j,2))^2)

end

end

end

y = d;

end

# 9. Concluzii

**Cu functia fmicon**

fminunc(@objectiveFun, x0,options)

distanta minima = 0.0202

## Algoritmul gradient descent s-a descurcat mult mai bine, deoarece a gasit un minim de 1.5 km (fig1), fata de algoritmul genetic, care a gasit un minim de aproximativ 10 km (fig2).

## In concluzie, am utilizat primul algoritm impreuna cu diagramele voronoi.

## Rezultatul amplasarii statiilor de incarcare se poate vedea in figura 3.

## In figura 4 sunt aratate punctele initiale de la care s-a pornit (coordonatele statilor de incarcare) cu albastru, impreuna cu coordonatele terminalelor, cu rosu.

Se poate implementa un algoritm mai bun, algoritmul newton, pentru ca setul de date este mic.

# figura 1

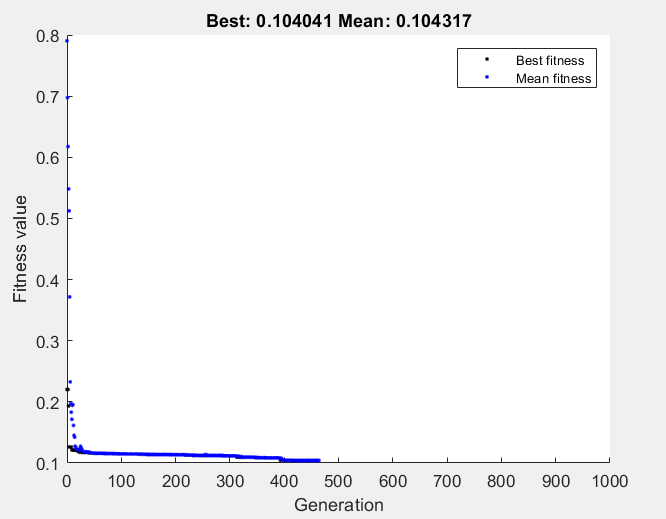
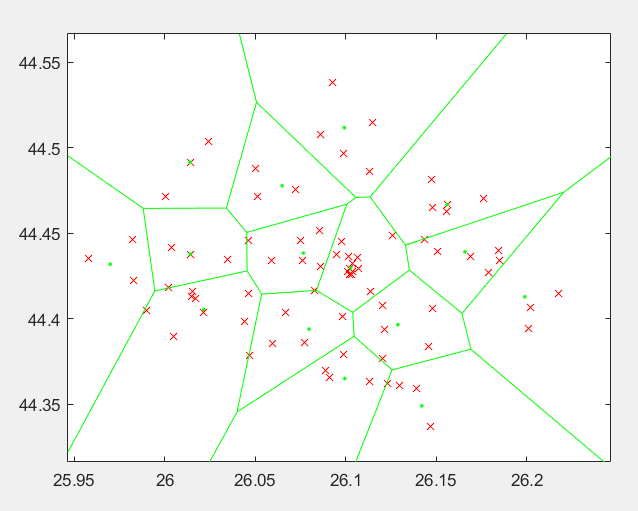
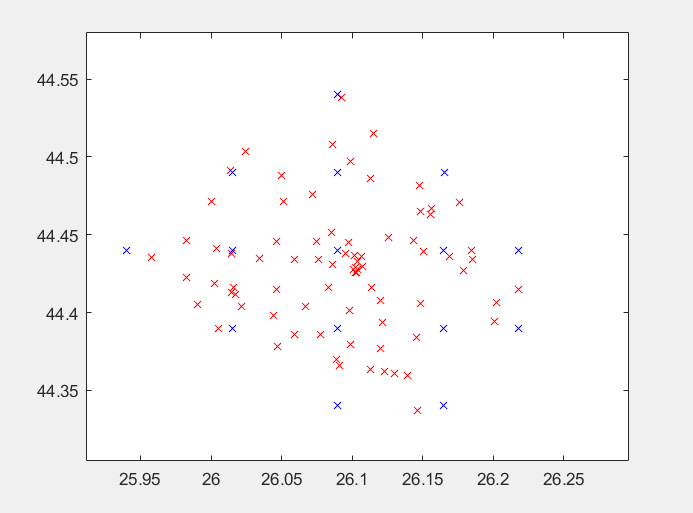
figura 2

figura 3

 figura 4