

# Multiple-Depot Vehicle Scheduling Problem

Silviu Ilas

June 2021

## Contents

<b>1</b>	<b>Introducere</b>	<b>3</b>
<b>2</b>	<b>Optimizarea Coloniei de Furnici</b>	<b>4</b>
2.1	Furnicile din natura . . . . .	4
2.2	Metaeuristica . . . . .	4
2.3	Prezentarea Algoritmului . . . . .	5
<b>3</b>	<b>Problema Programării Vehiculelor cu mai multe Depouri (MD-VSP)</b>	<b>6</b>
<b>4</b>	<b>Generator de instanțe</b>	<b>7</b>
4.1	Generarea locațiilor . . . . .	7
4.1.1	Aleatorie . . . . .	7
4.1.2	Abordarea realista . . . . .	7
4.2	Asignarea programului . . . . .	11
4.2.1	Alegerea orei de start . . . . .	11
4.2.2	Alegerea orei finale . . . . .	11
4.2.3	Alegerea timpului necesar . . . . .	11
4.3	Interfața grafica . . . . .	11
4.3.1	Meniu . . . . .	11
4.3.2	Previzualizarea . . . . .	12
4.3.3	Start . . . . .	13
<b>5</b>	<b>ACO aplicată pe MDVSP</b>	<b>15</b>
5.1	Model . . . . .	15
5.2	Cod relevant . . . . .	16
5.2.1	Skeleton ACO . . . . .	16
5.2.2	Skeleton Furnica . . . . .	18
5.2.3	Restricțiile furnicilor . . . . .	18
5.3	Modul de decizie . . . . .	20
5.4	Regula de actualizare . . . . .	20

<b>6</b>	<b>Variante de ACO</b>	<b>21</b>
6.1	Ant System . . . . .	21
6.2	Elitist System . . . . .	21
6.3	Min Max Ant System . . . . .	21
6.3.1	Modul de alegere a maximului . . . . .	23
6.3.2	Modul de alegere a minimului . . . . .	23
6.3.3	Modul de inițializare a feromonilor . . . . .	24
<b>7</b>	<b>Parametrii si particularitati</b>	<b>24</b>
<b>8</b>	<b>Rezultate</b>	<b>25</b>
<b>9</b>	<b>Concluzie</b>	<b>26</b>

# 1 Introducere

Problema rutarii vehiculelor (VRP) este o generalizare a problemei comisului voiajor (TSP). Diferenta este ca se adauga un punct (putem sa-l numim depou) din care toate rutele trebuie sa porneasca si in care trebuie sa se termine. Stiind ca TSP este o problema NP dificila, putem deduce si ca VRP este cel putin la fel de grea. Adaugand restrictia de timp acestei probleme (VRP), "clientii" pot fi deserviti doar la anumite intervale orare, ajungem la problema programarii vehiculelor (VSP). Se poate demonstra ca aceasta problema poate fi rezolvata in timp polinomial atunci cand avem un singur depou. Insa, daca avem 2 sau mai multe depouri, aceasta problema devine una dificila, la care nu se stiu algoritmi care o pot rezolva eficient in timp polinomial.

Problema programarii vehiculelor cu mai multe depouri (MDVSP) este una des intalnita in viata de zi cu zi a multor companii, cum a companiilor de mijloc de transport in comun. Considerand ca o planificare eficienta a vehiculelor poate reduce considerabil costul intregii operatiuni, este evident ca exista o motivatie pentru a descoperi algoritmi inteligenti care rezolva aceasta problema.

Deoarece problema MDVSP este una NP dificila, stim ca nu avem un algoritm care poate rezolva problema intr-un timp rezonabil pentru instante mari. De aceea mai multi algoritmi euristici[4] si metaeuristici [6] au fost propusi pentru a gasi o aproximare a solutiei minime. Dezavantajul acestor algoritmi este ca nu avem garantia ca solutia data de ei este cea optima, dar timpul de executie este mult mai mic pentru instante medii sau mari. Ei, insa, sunt susceptibil in a fi blocati in minime locale, nereusind de multe ori sa ajunga la optimul global. Totusi, de cele mai multe ori se prefera o solutie sub optimala livrata intr-un timp rezonabil decat o solutie optima care poate dura cateva sute de ani sa fie gasita.

Algoritmul propus in aceasta lucrare pentru rezolvarea problemei descrise mai sus este cel al optimizarii coloniei de furnici (ACO). Este un algoritm metaeuristic, bine cunoscut si aplicat cu succes pentru probleme precum VRP si TSP. Aceasta lucrare va descrie cateva variatii a algoritmului si le va compara pe aceasta problema.

## 2 Optimizarea Coloniei de Furnici

### 2.1 Furnicile din natura

Calitatea principală a coloniilor de insecte, furnici sau albine constă în faptul că fac parte dintr-un grup auto-organizat în care cuvântul cheie este simplitatea. În fiecare zi, furnicile rezolvă probleme complexe datorită unei sume de interacțiuni simple, care sunt efectuate de indivizi.

Când o colonie de furnici se confruntă cu alegerea de a-și atinge hrană prin două căi diferite, dintre care una este mult mai scurtă decât cealaltă, la început alegerea lor este în întregime aleatorie. Cu toate acestea, furnicile care folosesc traseul mai scurt ajung la mâncare mai repede și, prin urmare, merg înapoi și înapoi mai des între cuib și mâncare. Acest caz este exemplificat în figura de mai jos.

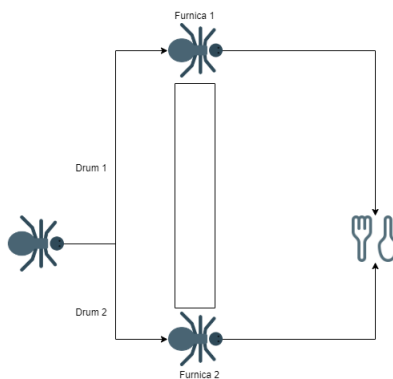


Figure 1: Modelul problemei

Se poate vedea că furnica 2 va putea face mai multe drumuri de la mâncare la cuib, astfel în timp, cantitatea de feromoni de pe acel drum va crește și ambele furnici vor alege din ce în ce mai des acea variantă.

### 2.2 Metaeuristica

O metaeuristică este un set de concepte algoritmice care pot fi utilizate pentru a defini metode euristice aplicabile unei game largi de probleme diferite. Cu alte cuvinte, o metaeuristică poate fi văzută ca o euristică generală construită pentru a ghida o euristică care stă la baza unei anumite probleme. Exemple de metaeuristică: rețele neuronale, algoritmi genetici, optimizarea coloniei de furnici etc.

## 2.3 Prezentarea Algoritmului

ACO este un algoritm probabilistic, metaeuristic, care poate fi folosit pentru rezolvarea problemelor computaționale care pot fi reduse la găsirea unui drum “bun” pe un graf. El se folosește de furnici artificiale care au fost modelate după comportamentul furnicilor din natura. Paradigma des utilizată în astfel de algoritmi este cea a comunicării pe baza de feromoni. Prin această comunicare se face un fel de inteligență de grup. Deși fiecare furnică artificială este relativ simplă, interacțiunea dintre ele și mediul lor înconjurător poate da apariția unei “inteligente” colective, necunoscută de fiecare la nivel individual. Marcarea drumurilor cu feromoni artificiali este un fel de informație numerică distribuită care este modificată de furnici pentru a reflecta experiența lor în rezolvarea problemei.

Primul algoritm, propus în 1992, ACO a fost numit “Ant System” (AS) și a fost aplicat problemei Comisului Voiajor. Pornind de la AS mai multe îmbunătățiri au fost aduse algoritmului propus. În general aceste îmbunătățiri aveau în comun faptul o exploatare mai ridicată a soluției găsite în procesul de căutare al furnicilor. Adicional, cei mai performanți algoritmi ACO folosesc în general și algoritmi de căutare locală.

Principala caracteristică a algoritmul ACO este furnicile care au avut un drum “bun” vor depune o cantitate mai mare de feromoni pe traseul obținut. Acest lucru combinat cu faptul că atunci când o furnică își construiește traseul, va fi mai probabil să aleagă calea cu mai mulți feromoni, face ca statistic drumurile mai bune să fie din ce în ce mai folosite.

### 3 Problema Programării Vehiculelor cu mai multe Depouri (MDVSP)

Problema este definită în felul următor. Avem un set de vehicule de transport, impartie între mai multe depouri ( $d_1, d_2, \dots, d_m$ ). Fiecare depou are un număr exact de vehicule asignat. Avem și un număr de sarcini reprezentate de un set de călătorii ( $c_1, c_2, \dots, c_n$ ) care trebuie asignate unui vehicul. O pereche ordonată de călătorii ( $c_i, c_j$ ) este posibilă doar dacă vehiculul care îndeplinește călătoria  $c_i$  poate fi folosit după pentru a îndeplini călătoria  $c_j$ . Acest lucru este de obicei descris asociind fiecărei călătorii  $c_i$  un timp de start ( $\sigma_i$ ), un timp de final ( $\chi$ ) și timpul necesar ( $\theta_i$ ) pentru a parcurge călătoria ( $c_i, c_j$ ) de la finalul călătoriei  $c_i$ . Utilizând aceste notații, călătoria ( $c_i, c_j$ ) este posibilă doar dacă  $\chi + \theta_i \leq \sigma_j$ .

În general, costul călătoriei ( $c_i, c_j$ ) poate lua în considerare mai mulți factori. Pentru a face călătoria putem adăuga și costul așteptării, costul drumului sau alți factori. Pe deasupra avem și costul scoaterii vehiculului din depou și costul întoarcerii lui. Toate aceste lucruri formează o matrice de costuri pătratică, de dimensiune  $n+m$ , în care  $+\infty$  înseamnă un drum imposibil. Din punctul de vedere algoritmic, nu suntem interesați de timpii menționați, locația sarcinilor sau factorii luați în considerare de cost. Modelul instanței MDVSP este văzut ca o tupla  $(G, m, n, r, c)$ , unde  $G$  este reprezentat de digraful care reprezintă posibilitatea relației date de  $c$ .

Termeni des folosiți în contextul MDVSP sunt “pull-out trips”, reprezentând drumurile de la depou la prima sarcină, “pull-in trips” sunt opusul, ele reprezintă drumul de la ultima sarcină din tur la depou.

O soluție a MDVSP este o atribuire a sarcinilor la vehicule astfel încât să minimizeze suma totală și să respecte următoarele reguli : fiecare sarcină trebuie îndeplinită o singură dată, fiecare vehicul trebuie să termine la același depou din care a început, numărul de vehicule folosit de fiecare depou poate fi depășit.

## 4 Generator de instanțe

În continuare, vom descrie o modalitate de a genera instanțe MDVSP care pot fi reprezentate pe plan 2D (reprezentat de un grila carteziana). Se creeaza mai intai nodurile(clienții și depourile), după trebuie sa le stabilim coordonatele și programul lor.

### 4.1 Generarea locațiilor

#### 4.1.1 Aleatorie

Generarea aleatorie a locațiilor este relativ banală. După ce s-au creat nodurile trebuie sa le asignam o valoare aleatoare din intervalul nostru prestabilit. De exemplu, daca grila noastra carteziana ia valori în intervalul 0-1000 și presupunem ca avem o funcție “random” care întoarce numele aleatorii de pe acest interval, atunci coordonatele unui nod  $x$  ar putea fi (random(), random()).

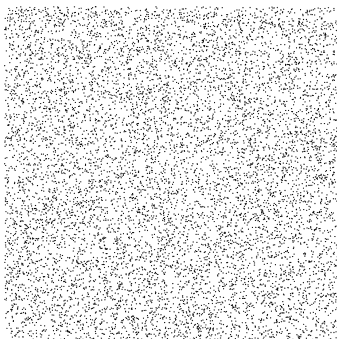


Figure 2: Puncte distribuite aleatoriu

#### 4.1.2 Abordarea realista

În general, pe instanțele din lumea reala a acestei probleme, putem observa niste tipare care vizează asezarea geografica a clienților și depourilor. Putem considera reprezentarea grafică harta unui oraș. În general, este mai probabil sa avem mai mulți clienți în mijlocul orașului, iar probabilitatea lor scade cu cat ne departam mai mult inspre periferie. O alta observatie ar fi că este mai probabil sa găsim depouri la marginea orașului. Putem sa ne folosim de aceste observații pentru a creea o reprezentare mai sugestiva si mai ancorata in realitate.

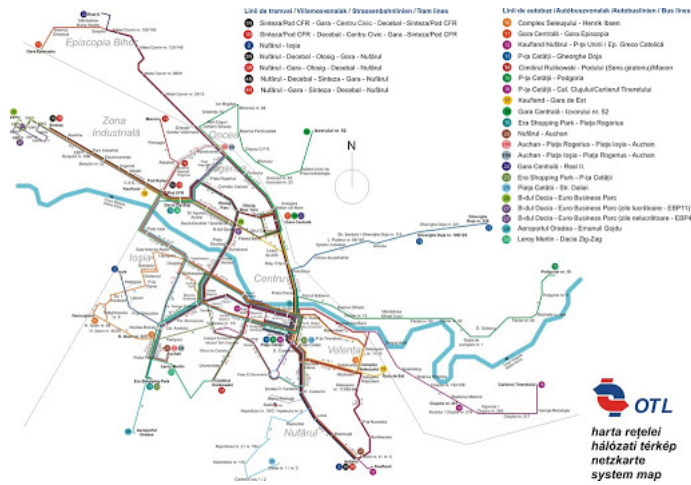


Figure 3: Harta mijloacelor de transport in comun Iași

În poza de mai sus preluată de pe <http://harti.tramclub.org/> este reprezentată harta mijloacelor de transport în comun din Iași, care poate fi considerată o instanță de MDVSP. Chiar și în cazul prezentat mai sus, afirmațiile anterioare sunt valabile.

### Distributia Normala

Este cea mai comună distribuție folosită pentru analize statistice. Cea standard are 2 parametri, mediană și deviația standard. Pentru cea normală, 68 % din observații sunt la  $\pm$  o unitate de mediană, 95% sunt la  $\pm$  două unități, iar 99.7% sunt la trei unități. Când este pusă pe un graf, informațiile urmează forma unui clopot, cele mai multe informații adunându-se în jurul unei zone centrale și scăzând cu cât se apropie mai mult de margini.

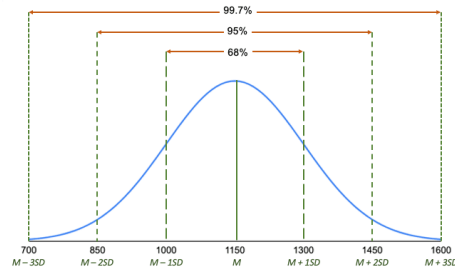


Figure 4: Distribuția normală standard

Această distribuție a fost observată în diferite științe sociale și naturale. Câteva exemple ar fi : înălțimea, abilitatea de a citi, IQ-ul, satisfacția locului de muncă și multe altele. Deoarece acest tip de distribuție de variabile este foarte



comuna, putem sa ne asteptam sa fie o corelație asemănătoare între clienți și distanța lor fata de centrul. Aceasta afirmație este întarita de graficile studiului [7] în care se poate vedea jumatea simetrică a unei distribuții normale.

### Locațiile clienților

Utilizand distributia normala, putem amplasa clientii într-un mod mai apropiat de realitate și putem controla chiar și densitatea lor. Astfel, modificăm regula anterioara de plasare a nodurilor schimbând funcția “random” cu “random-Gauss”. Aceasta functie întoarce o valoare în intervalul stabilit urmand distributia normala.

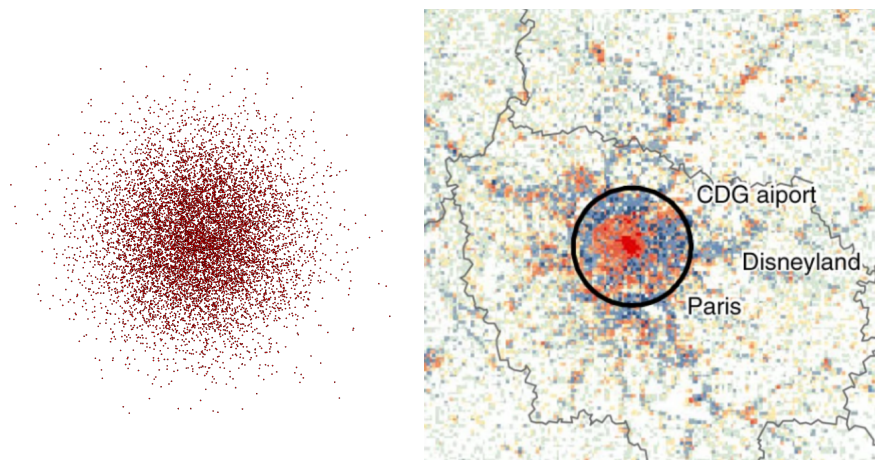


Figure 5: Comparatie între punctele generate de noi și densitatea populației parisului [1]

În partea stângă se găsește generarea clienților pe baza regulilor descrise anterior, în partea opusă este densitatea populației în orașul paris conform studiului [1].

### Locațiile depourilor

Facem presupunerea că, în realitate, depourile o sa fie în apropierea marginii orașului. Avand aceasta presupunere putem crea un cerc în jurul clienților, depourilor vor fi în apropierea marginii cercului. Putem folosi și aici distributia gaussiana pentru a adauga “zgomot”.

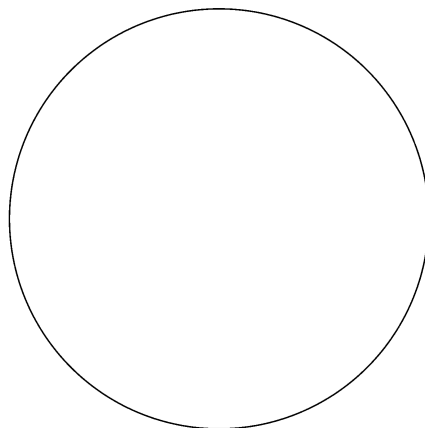


Figure 6: Depourile generate

În poza de mai sus sunt reprezentate 10000 de noduri. Coordonatele lor au fost alese aleatoriu uniform din coordonatele posibile ale marginii unui cerc. Suprapunerea lor este atât de comună încât nici nu putem identifica nici un nod individual, nodurile colectiv luând aspectul unui cerc perfect.

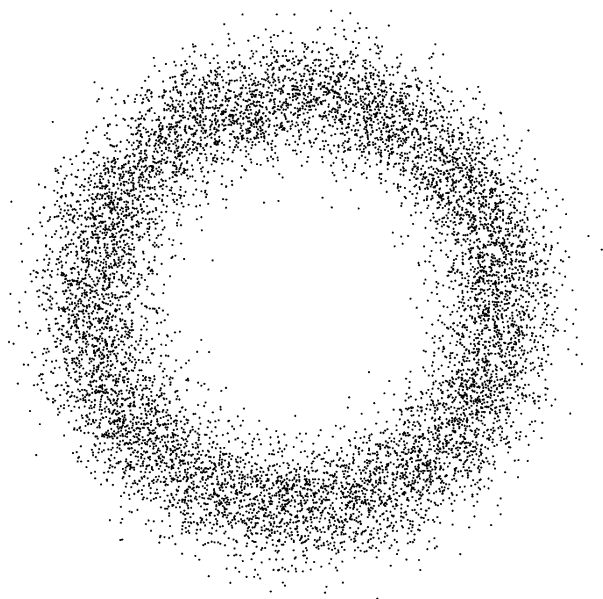


Figure 7: Depourile generate adaugand zgomot

Adaugand acel “zgomot”, care se folosește de distribuția normală, în mod

evident s-a imbunatatit aranjarea nodurilor, luand o forma mai naturala si mai apropiata de lumea reala, in care probabilitatea suprapunerii scade considerabil.

## 4.2 Asignarea programului

### 4.2.1 Alegerea orei de start

Ora de start reprezintă ora la care trebuie servit clientul. În generator a fost aleasă ca un număr aleatoriu, între  $min_s$  și  $max_s$ . Noi am ales  $min^s = 0$  și  $max^s = 1440$  (numărul de minute din zi). În cod există un vector cu n elemente (numărul de clienți) populat cu aceste valori aleatorii. Asadar  $\forall i, \sigma_i = rand(min^s, max^s)$

### 4.2.2 Alegerea orei finale

Ora finală este strans legată de ora de start. Trebuie sa ne asigurăm ca este îndeplinită condiția  $\leq$ . Astfel ea a fost aleasă după următoarea formula :  $\forall i, \chi_i = \sigma_i + t$  unde t este un număr predefinit.

### 4.2.3 Alegerea timpului necesar

La pasul anterior am setat niște coordonate carteziene fiecărui client. Putem sa ne folosim de ele pentru a calcula timpul necesar. Avand coordonatele, putem calcula distanța euclidiană dintre oricare doi clienți pe care o putem asocia timpului necesar.

$$\forall i, j, \theta_{ij} = dist(c_i, c_j)$$

## 4.3 Interfața grafica

După pașii descriși mai sus, avem informațiile necesare pentru a construi o instanță MDVSP și a o reprezenta grafic. Putem encoda o instanță MDVSP într-o matrice de adiacență în care avem -1 pe nodurile inaccesibile și costul drumului pe celelalte noduri. Nodurile inaccesibile sunt cele care nu respectă formula  $\chi + \theta_i \leq \sigma_j$ , descrisă în detaliu în 3 . Reprezentarea grafica a instanței e banală după ce stabilim coordonatele. În continuare vom alege coordonatele după metodele propuse la 4.1.2.

### 4.3.1 Meniu

Meniul generatorului nostru de instanțe permite stabilirea a patru parametrii : numărul de clienți, numărul de depouri, deviația clienților și deviația depourilor. Ultimele 2 sunt specifice reprezentării grafice și vor fi detaliate în 4.3.2. În meniu mai sunt disponibile și 2 butoane, cele de "preview" (4.3.2) și "start" (??).

The image shows a configuration window titled "Configs". It contains four input fields with labels and values: "Nr. Depots" with value "5", "Nr. Clients" with value "200", "Std Depots" with value "50", and "Std Clients" with value "0". Below these fields are two buttons: "Start" and "Preview".

Figure 8: Meniul interfetei

#### 4.3.2 Previzualizarea

Butonul de previzualizare deschide o fereastră în care se pot vedea efectele ultimilor doi parametrii. Parametrul “Std clients” reprezintă deviația standard a clienților. Cu cât aceasta este mai mică, cu atât ei probabil vor fi mai apropiați de centru, dând aspectul unui oraș aglomerat. Parametrul “Std depots” reprezintă deviația standard a depourilor. În capitolul 4.1.2 am explicat poziționarea depourilor, astfel parametrul influențează deviația de la marginea cercului a unui depou. Acești parametri iau o valoare între 0 și 100. Valorile sunt abstractizate de implementarea lor efectivă. Dacă am permite deviația 0, toți clienții s-ar concentra într-un punct iar depourile ar forma un cerc perfect. Pentru a evita acest scenariu, 0 reprezintă deviația minimă care respectă trasaturile cautate în generarea interfetei descrise în 4.1.2. Aceeași mentalitate a fost adoptată în stabilirea celeilalte extremități, 100 reprezintă deviația maximă care este încă relevantă.

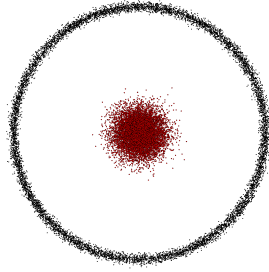


Figure 9: 0-0

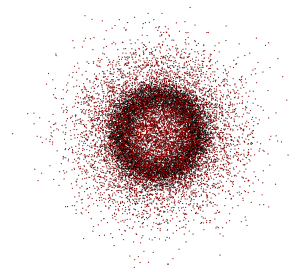


Figure 10: 100-100

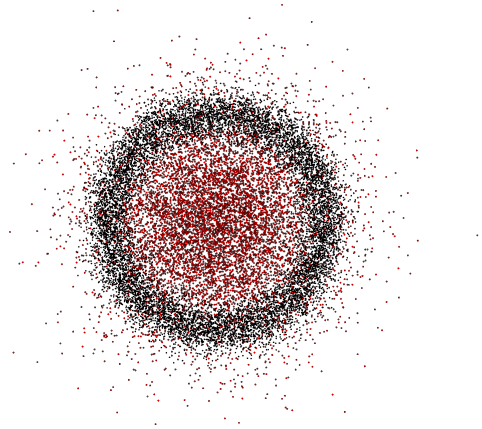


Figure 11: 50-50

În imaginile de mai sus, clienții sunt reprezentați de punctele roșii iar depourile de punctele negre. Imaginile au fost generate alegând 10000 de puncte, atât pentru clienți cât și pentru depouri, conform regulilor descrise. Aceasta previzualizare nu reprezintă graful final, scopul ei este de a determina influența ultimilor doi parametri a interfeței grafice în construcția instanței MDVSP.

#### 4.3.3 Start

Butonul de start generează o instanță MDVSP și o reprezintă grafic utilizatorului. Aceasta instanță poate fi trimisă unui rezolvator care răspunde cu o soluție. Soluția poate fi reprezentată pe graful rezultat. O muchie între două noduri reprezintă un drum de la un client/depot la altul. La început, fiecare depou are culoarea verde, iar clienții au culoarea neagră. Additional, fiecărui depou  $i$  se asociază o culoare aleatorie. Cu ajutorul soluției interfața va trasa muchiile și va colora clienții

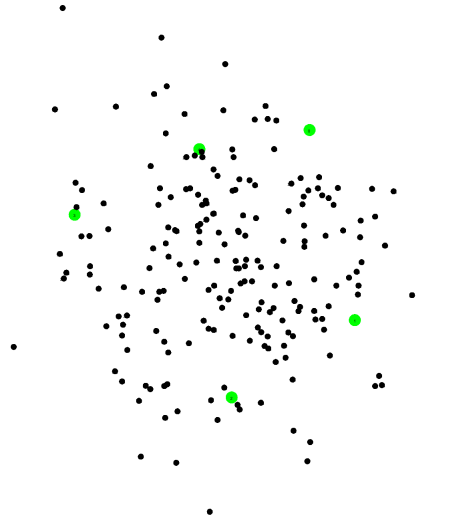


Figure 12: Graful la inceput

Instanța generată de algoritmul reprezentată în figura 12 are 5 depouri, 200 de clienți și deviația standard 50. Pe această instanță relativ mică, nu este la fel de evidentă distribuția uniformă folosită ca în 11, dar totuși se pot observa trăsăturile caracteristice : probabilitatea găsirii unui client scade cu cât ne departăm de centru și depourile sunt așezate în jurul clienților.

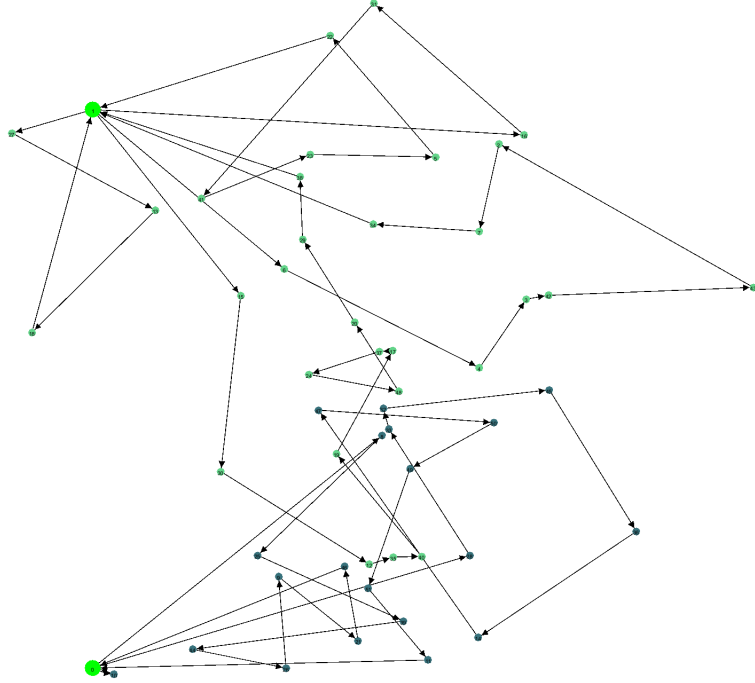


Figure 13: Graful dupa ce primește soluția

Figura 13, are 2 depouri, 50 de clienți și deviația standard 50. Diferența principală față de 12 este că, pe lângă faptul că are mai puține depouri și clienți (pentru a putea vizualiza nodurile mai bine), reprezentarea grafică a primit o soluție de la un rezolvator. Astfel clienții au fost colorați, luând culoarea asignată la început depourilor și muchiile traseelor au fost trasate.

## 5 ACO aplicată pe MDVSP

### 5.1 Model

În ACO, furnicile sunt agenți simpli care construiesc soluții mișcându-se dintr-un punct în altul pe graf. Soluția construită de furnici este ghidată de feromonul artificial și o informație euristica (cum ar fi costul). Putem asocia feromonul artificial cu cantitatea de informație cunoscută. Astfel  $\tau_{i,j}(t)$  este o informație numerică pe care o modificăm în timpul algoritmului unde  $(i,j)$  reprezintă un arc iar  $t$  este numărul iterației. În problema MDVSP este reprezentată de un digraf, deci putem avea  $\tau_{i,j}(t) = \tau_{j,i}(t)$ .

Pentru a modela MDVSP, am folosit un digraf de dimensiune  $n+m+1$ , unde  $n$  reprezintă clienți,  $m$  depourile. Am adăugat un depou suplimentar, numit depou virtual, care se conectează la toate celelalte depouri cu arce de cost 0.

Cum fiecare depou este conectat la toți clienții, având acest depou principal ne este garantat ca avem un graf conex. Astfel, orice furnica poate vizita fiecare nod de pe graf într-o singură iterație. [2]

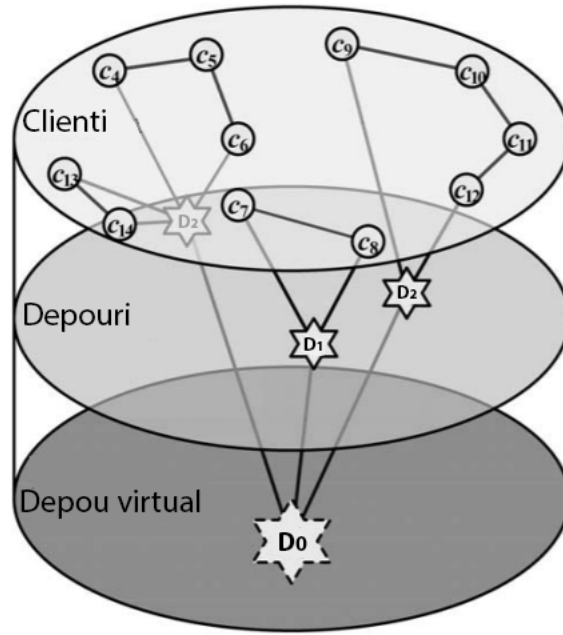


Figure 14: Modelul problemei reprezentat grafic

Modeland astfel problema, traseul unei furnici arată în modul următor: începe în depoul virtual, trece printr-un depozit apoi către clienți și face călătoria de întoarcere la același depozit efectiv fără a depăși constrângerile impuse de problema.

O soluție MDVSP găsește un set de rute de costuri minime pentru a facilita livrarea de la depozitul central virtual prin depozitele efective la mai multe locații ale clienților. Acest lucru este foarte asemănător cu comportamentele de căutare a hranei din coloniile de furnici din natură. Dacă luăm depozitul central virtual ca și cuib, luăm depozitele reale ca intrări ale cuibului și luăm clienții drept hrană, rezolvarea MDVSP poate fi descrisă ca procesul de căutare a alimentelor începând de la cuib printr-o intrare.

## 5.2 Cod relevant

### 5.2.1 Skeleton ACO

```
public Deque<Tour> run() {
```



```

    Ant bestAntSoFar = null;
    while ( condition () ) {
        generateAnts ();
        evaluateAnts ();
        updatePheromones ();
        Ant bestAntThisIteration = getBestAntThisIteration ();
        if ( isFirstAntBetter ( bestAntThisIteration , bestAntSoFar ) ) {
            bestAntSoFar = bestAntThisIteration ;
        }
    }
    return bestAntSoFar.getDequeTour ();
}

```

#### **Funcția “condition()”**

Condiția de oprire a algoritmului este foarte importantă deoarece influențează foarte mult timpul de execuție sau calitatea soluției. Dacă ne oprim prea devreme, ne mărim șansa de a ne opri la soluție suboptimală, în schimb dacă ne oprim prea târziu timpul de rulare al algoritmului crește fără a îmbunătăți rezultatele. De aceea se face un echilibru între cele două, încercând să ne oprim când credem că rezultatele nu se vor îmbunătăți prea curând. Acest lucru se poate face empiric, uitându-ne la date putem observa punctul în care este improbabil să se găsească o îmbunătățire, sau se pot face diferite metode de a detecta când algoritmul stagnează. O astfel de variantă des întâlnită este oprirea după trecerea a  $X$  iterații de la cel mai bună soluție.

#### **Funcția “generateAnts()”**

Generează furnicile care vor fi folosite în această iterație a algoritmului. Sunt două moduri principale de a genera furnicile, aleatoriu sau elitist. Alegerea aleatorie distruge toate furnicile vechi și generează altele noi, iar cea elitistă păstrează un număr de furnici. Din observațiile noastre alegerea elitistă are constant rezultate mai bune pe această problemă.

#### **Funcția “evaluateAnts()”**

Construiește soluții apelând funcția `run()` a fiecărei furnici din iterația curentă, astfel la finalul acesteia fiecare furnică va avea drumul construit în acea iterație.

#### **Funcția “updatePheromones()”**

Aplica procesul prin care se modifică cantitatea de informație (feromonii) din graf. Aceasta poate fi să crească pe o anumită secțiune, dacă a fost o soluție bună, fie să scadă, datorită evaporării feromonilor. Depozitarea feromonilor crește probabilitatea ca acea componentă să fie folosită în viitor de alte furnici. Evaporarea feromonilor este o formă de uitare care ajută la evitarea convergenței premature a algoritmului în o regiune suboptimală, astfel favorizant explorarea.

### 5.2.2 Skeleton Furnica

```
public void run() {  
while (!isFinished()) {  
    List<DefaultWeightedEdge> availableEdges =  
        getAvailableEdges(currentLocation);  
    DefaultWeightedEdge pickedEdge = pickAnEdge(availableEdges);  
    goToNextPosition(pickedEdge);  
}  
}
```

#### Funcția “isFinished”

Verifica daca furnica curentă a terminat. Acest lucru se poate intampla atunci când toți clienți au fost serviți sau mai sunt clienți de servit dar nu mai sunt vehicule disponibile.

#### Funcția “getAvailableEdges”

Returnează o lista care contine toate muchiile care încep în parametrul dat și se termina într-un nod accesibil care respecta toate restricțiile adiționale ale problemei.

#### Funcția “pickAnEdge”

Alege o muchie din cele date ca parametru. Aici se folosește regula de selecție a algoritmului în care trebuie favorizate muchiile cu o cantitate de feromoni ridicată.

#### Funcția “goToNextPosition”

Muta furnica în următoarea poziție și actualizează drumul curent cu muchia obtinuta.

### 5.2.3 Restricțiile furnicilor

Funcția getAvailableEdges a fost implementata in modul urmator

```
List<DefaultWeightedEdge> availableEdges = new ArrayList<>(edges);  
for (DefaultWeightedEdge edge :  
    edges) {  
    Integer target = antColonyGraph.getEdgeTarget(edge);  
    Integer timesVertexWasVisited = timesNodesWhereVisited.get(target);  
    Boolean isVertexRepeatable =  
        antColonyGraph.getIsVertexRepeatable().get(target);  
    if (timesVertexWasVisited != 0 && !isVertexRepeatable) {  
        availableEdges.remove(edge);  
    }  
}
```

```

    } else {
        for (AvailableEdgeRestriction availableEdgeRestriction :
            availableEdgeRestrictions) {
            if (availableEdgeRestriction.shouldRemoveEdge(edge)) {
                availableEdges.remove(edge);
            }
        }
    }
}

```

Itereaza prin toate muchiile nodului dat ca parametru și verifica dacă acea muchie respecta toate condițiile. Condiția generală a unei coloni de furnici este că ca nodurile țintă pot fi vizitate o singură dată atât timp cât nu a fost specificat altfel. Dacă condiția de bază a fost îndeplinită se iterează prin celelalte condiții specifice problemei.

Codul care descrie condițiile adiționale a acestui model pe această problemă.

```

this.addAvailableEdgesRestriction((edge) -> {
    EdgeType edgeType = mdvspAntColonyGraph.getEdgeType(edge);
    Integer target = mdvspAntColonyGraph.getEdgeTarget(edge);
    Integer source = mdvspAntColonyGraph.getEdgeSource(edge);
    // can't pull in a different depot then the one you started at
    if (edgeType == EdgeType.PULL_IN) {
        if (currentDepot != target) {
            return true;
        }
    }
    // don't pull out a depot if you don't have enough vehicles
    if (edgeType == EdgeType.PULL_OUT) {
        .
    }
    DefaultWeightedEdge lastEdge = this.getLastPickedEdge();
    EdgeType lastPickedEdgeType = this.getMdvspAntColonyGraph().getEdgeType(lastEdge);
    // we should not go to a depot that has no more remaining depots left
    if (edgeType == EdgeType.MASTER_PULL_OUT) {
        int remainingTrips = remainingDepotsNr[target];
        if (remainingTrips <= 0)
            return true;
    }
    // we can't pull in the master right after we pulled out
    if (edgeType == EdgeType.MASTER_PULL_IN) {
        if (lastPickedEdgeType == EdgeType.MASTER_PULL_OUT)
            return true;
    }
    // we just pulled in we can't go back until we go to the master depot
    if (lastPickedEdgeType == EdgeType.PULL_IN) {

```

```

        if (edgeType != EdgeType.MASTER_PULL_IN)
            return true;
    }
    return false;
}

```

Primele 2 condiții sunt specifice MDVSP, să forțăm furnicile să se întoarcă la depoul inițial și să respecte numărul de vehicule dat. Următoarele 3 sunt legate de modul în care am modelat problema. Ele asigură ca traseul drumului este respectat în modul menționat mai sus.

### 5.3 Modul de decizie

Luarea deciziilor cu privire la combinarea clienților se bazează pe o regulă probabilistică, luând în considerare atât vizibilitatea, cât și cantitatea de informație de pe o muchie. Astfel, pentru a selecta următorul client  $j$ , furnica folosește următoarea formulă probabilistică.

$$P(i, j) = \begin{cases} \frac{(\tau(i, j))^\alpha \cdot (\eta(i, j))^\beta}{\sum_{l \notin \text{tabu}} (\tau(i, l))^\alpha \cdot (\eta(i, l))^\beta}, & j \notin \text{tabu}. \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

unde,  $P(i, j)$  este probabilitatea de a alege să combine clienții  $i$  și  $j$  pe traseu;  $\tau(i, j)$  este concentrația de feromoni de pe muchia  $(i, j)$  și ne poate spune cât de bună a fost combinația acestor doi clienți  $i$  și  $j$  în iterațiile anterioare;  $\eta(i, j)$  = vizibilitatea pe margine  $(i, j)$ , de obicei asociată unui parametru precum inversul distanței.  $\alpha$  și  $\beta$  sunt influența relativă a traseelor feromonilor și respectiv vizibilității. Dacă  $\alpha$  este 0, atunci selecția va fi bazată doar pe informația euristica a algoritmului, acest lucru corespunde cu un algoritm greedy. Pe când atunci când  $\beta$  este 0, algoritmul va alege doar pe baza nivelului de feromoni, ducând astfel rapid la stagnare într-un punct în general suboptimal. Tabu este setul de noduri imposibile.

### 5.4 Regula de actualizare

În funcție de varianta de ACO folosită, regula de actualizare poate fi diferită. Totuși, toate au în comun faptul că urmăresc drumul al cel puțin unei furnici și cresc cantitatea de feromoni folosită pe fiecare muchie parcursă de ea utilizând următoarea formulă:

$$\tau_{ij}(t+1) = (1 - \rho) \cdot \tau_{ij}(t) + \sum_{k=1}^m \Delta\tau_{ij}^k(t)$$

unde  $0 < \rho \leq 1$  este evaporarea urmelor feromonilor. Se utilizează parametrul  $\rho$  pentru a evita acumularea nelimitată a feromonilor pe un anumit drum și prin aceasta algoritmului să uite deciziile proaste luate anterior. Dacă un arc nu este ales de furnici, puterea sa de feromoni asociată scade exponențial.  $\Delta\tau_{ij}^k(t)$

este cantitatea de feromonii furnica  $k$  pune pe arcurile pe care le-a vizitat; este în mod standard este definit în modul urmator:

$$\Delta\tau_{ij}^k(t) = \begin{cases} 1/L^k(t) & \text{daca muchia } (i, j) \text{ este folosita de furnica } k \\ 0 & \text{altfel} \end{cases}$$

unde  $L^k(t)$  este lungimea turului a furnicii  $k$ . Prin ecuația de mai sus, cu cât suma costurilor drumurilor este mai mica, cu atât se va depozita o cantitate mai mare de feromoni pe acea muchie.. În general, arce care sunt folosite de multe furnici și care sunt conținute în tururile mai scurte vor primi mai mult feromoni și, prin urmare, sunt, de asemenea, mai probabil pentru a fi alese în iterațiile viitoare ale algoritmului.

## 6 Variante de ACO

### 6.1 Ant System

Este primul algoritm ACO și este în mare masura asemanator cu ce am descris mai sus. A fost dezvoltat în anul 1992 de Dorigo, V. Maniezzo și inițial au fost propuse trei variante diferite numite “ant-density”, “ant-quantity” și “ant-cycle”. Pe cand “ant-density” și “ant-cycle” actualizează feromoni direct după ce o furnica face o mișcare pe graf, feromonii algoritmului “ant-cycle” erau actualizat la final după ce toate furnicile și-au construit turul. Deoarece “ant-cycle” a avut rezultate mult mai bune decat celelalte doua variante, o să prezentăm varianta respectiva pe aceasta problema.

### 6.2 Elitist System

Vine ca o primă îmbunătățire a variantei originale și a fost introdusă în [5] [3]. Ideea principala este de a da un adaos considerabil de feromoni pe muchiile aparținând celui mai bun tur găsit de la începutul algoritmului; acest tur este denumit  $T_{gb}$  în cele ce urmează. Acest lucru se realizează prin adăugarea la arcurile turului  $T_{gb}$  cantitatea  $1/L_{gb}$ . Au fost prezentate câteva rezultate limitate în [5] [3] sugerează că utilizarea strategiei elitiste pe problema TSP cu un adecvat numărul de furnici elitiste permite AS: să găsească tururi mai bune și să le găsească vii mai devreme în execuție. Cu toate acestea, atunci cand numărul de furnici elitiste este prea mare, căutarea se concentrează devreme în jurul soluțiilor suboptimale care duc la o stagnare timpurie a căutării.

### 6.3 Min Max Ant System

Cercetările privind ACO au arătat că performanța îmbunătățită poate fi obținută printr-o exploatare mai puternică a celei mai bune soluții găsite în timpul căutării. Cu toate acestea, folosind o căutare mai lăcomă poate agrava problema stagnării premature a căutării. Prin urmare, cheia pentru a obține o buna performanta cu algoritmi ACO este de a combina o exploatare îmbunătățită a

celor mai bune soluții găsite în timpul căutării cu un mecanism eficient pentru evitarea stagnării timpurii a căutării. MAX –MIN Ant System, care a fost dezvoltat special pentru a satisface aceste cerințe, diferă în trei aspecte cheie de AS.

- Pentru a exploata cele mai bune soluții găsite în timpul unei iterații sau în timpul rulării algoritmului, după fiecare iterație doar o singură furnică poate adauga feromoni. Această furnică poate fi cea care a găsit cea mai bună soluție în iterația curentă sau cea care a găsit cea mai bună soluție de la începutul procesului.
- Pentru a evita stagnarea căutării intervalul de posibile trasee de feromoni pe fiecare componentă a soluției sunt limitate la un interval  $[\tau_{min}, \tau_{max}]$ , adică  $\forall \tau_{ij}, \tau_{min} \leq \tau_{ij} \leq \tau_{max}$ .
- În plus, feromonul de pe muchii este inițializat la limita maximă a drumului, acest lucru determina o explorare mai mare la începutul algoritmului.

În MMAS, o singura furnica poate actualiza feromonii după fiecare iterație. Prin urmare, regula modificată a actualizării traseului feromonic este dată de:

$$\tau_{ij}(t+1) = (1 - \rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}^{optim}(t)$$

unde  $\Delta\tau_{ij}^{optim}(t) = 1/f(s^{best})$  și  $f(s^{best})$  denotă costul celei mai bune soluții din aceasta iterație ( $s^{io}$ ) sau costul celei mai bune soluție globale ( $s^{go}$ ). Utilizarea unei singure soluții, fie  $s^{io}$ , fie  $s^{go}$ , pentru utilizarea feromonilor este cel mai important mijloc de exploatarea căutării în MMAS. Prin această alegere, elementele soluției care apar frecvent în cele mai bune găsite soluțiile obțin o întărire mare. Totuși, alegerea dintre cea mai bună iterație și cea mai bună furnică globală pentru actualizarea traseelor feromonilor controlează modul în care se face exploatarea. Când se utilizează numai  $s^{go}$ , căutarea se poate concentra prea repede în jurul acestui aspect, deci explorarea celor mai bune soluții poate fi limitată, cu consecința existenței pericolului de a rămâne prins în o soluție de calitate slabă. Acest pericol este redus atunci când  $s^{io}$  este ales pentru actualizarea traseului feromonilor, dar cele mai bune soluții de iterație pot diferi considerabil de iterație la iterație și un număr mai mare de soluții componente pot primi armături ocazionale. Desigur, se pot folosi și strategii mixte precum alegerea  $s^{io}$  ca implicit pentru utilizarea feromonilor și utilizarea  $s^{go}$  din ce în ce mai frecvent pe parcursul algoritmului.

Indiferent dacă alegem cea mai buna soluție din iterația curentă sau cea globală pot apărea stagnări în căutare. Acest lucru se poate intampla daca, la fiecare punct de alegere, sunt mai semnificativ mai mulți feromoni pe un traseu de cat pe toate celelalte. În aceasta situatie, datorită probabilitati de alegere guvernată de ecuația descrisă mai sus, o furnica va prefera acel traseu peste toate alternativele, accentuand problema la fiecare iterație. Într-o astfel de situația în care furnicile construiesc același traseu în repetate randuri, căutarea spațiului incetinește până poate ajunge chiar sa se și oprească. Evident aceasta stagnare ar trebui evitată.

O modalitate de a realiza acest lucru este de a influența alegerea drumurilor, acestea depinzând direct de informațiile euristice și cantitatea de feromoni. Informațiile euristice sunt, de obicei, dependente de problema și statice pe tot parcursul algoritmului. Dar, prin limitarea influenței cantității de feromoni, putem evita cu ușurință acest caz. Pentru a atinge acest obiectiv MMAC impune limite explicite,  $\tau_{min}$  și  $\tau_{max}$ , astfel încât cantitatea de feromoni este limitată la  $\tau_{min} \leq \tau_{ij}(t) \leq \tau_{max}$ . Pentru a impune această condiție, după fiecare iterație putem să ne asigurăm că aceste limite au fost respectate. Dacă există  $\tau_{min} > \tau_{ij}(t)$  atunci stabilim  $\tau_{ij}(t) = \tau_{min}$ , analog pentru  $\tau_{max}$ . De asemenea, dacă  $\tau_{ij}(t) < \inf$  și  $\tau_{min} > 0$  atunci probabilitatea de a alege o anumită componentă a soluției nu este niciodată 0.

Totuși, trebuie alese valori adecvate pentru limitele cantității de feromoni. În cele ce urmează vom propune un mod principal de determinare a acestor valori. O să începem cu introducerea noțiunii de convergență pentru MMAS. Spunem că MMAS a converș, dacă pentru fiecare punct de alegere, una dintre muchii are  $\tau_{max}$  cantitate de feromoni, în timp ce toate celelalte alternative au  $\tau_{min}$ . Dacă MMAS a converș, soluția construită prin alegerea muchiilor cu cantitate de feromoni maximă va corespunde de obicei cu cea mai bună soluție găsită de algoritm. Conceptul de convergență MMAS diferă într-un aspect ușor, dar important de conceptul de stagnare. În timp ce stagnarea descrie situația în care toate furnicile urmează aceeași cale, în situații de convergență a MMAS nu face acest lucru datorită limitelor cantității de feromoni.

### 6.3.1 Modul de alegere a maximului

Putem rafina conceptul de convergență arătând că traseul maxim de feromoni este asimptotic delimitat.

La fiecare iterație, putem adăuga maxim  $1/f(s^{perf})$ , unde  $f(s^{perf})$  este cea mai bună soluție posibilă pentru problema specifică. Așadar, cantitatea de informație poate fi cel mult

$$\tau_{ij}^{max}(t) = \sum_{i=1}^t (1 - \rho)^{t-i} \frac{1}{f(s^{perf})} + (1 - \rho)^t \tau_{ij}(0)$$

Deoarece  $p < 1$ , suma converge la

$$\frac{1}{p} \frac{1}{f(s^{perf})}$$

În MMAS, setăm  $\tau_{max}$  la o estimare a maximului. Acest obiectiv este atins dacă folosim în ecuația de mai sus  $f(s^{go})$  în loc de  $f(s^{perf})$ . Astfel de fiecare dată când se îmbunătățește cea mai bună soluție găsită de la începutul algoritmului schimbăm valoarea  $\tau_{max}$ , ducând de fapt la o valoare dinamică în schimbare a  $\tau_{max}(t)$

### 6.3.2 Modul de alegere a minimului

În secțiunea de mai jos, propunem o metoda pentru stabilirea minimului.

Pentru a determina valoarea minimului facem următoarele două presupuneri :

- Prima este ca în jurul soluțiilor bune este o șansa relativ mare de a găsi soluții și mai bune. Validitatea acestui argument depinde de caracteristica spațiului de căutare a problemei.
- A doua este ca influența construirii soluției este determinată de diferența dintre partea superioară și cea inferioară a limitelor căilor, în loc de diferența de informații euristice. Acest lucru este posibil dacă influența informațiilor euristice este scăzut și poate fi ajustată modificând parametrul  $\beta$ .

Având în vedere aceste presupuneri, valori bune pentru  $\tau_{min}$  sunt găsite coreland convergența algoritmului cu minimul cantității de feromoni. Astfel, putem actualiza minimul, la fiecare iterație, după următoarea formula.

$$\tau_{min} = \begin{cases} \tau_{min} \cdot (1 - \rho) & \text{daca } v < \sigma \\ \tau_{min} \div (1 - \rho) & \text{altfel} \end{cases}$$

unde  $(1 - \rho)$  este parametrul folosit pentru evaporarea feromonilor,  $v$  este o valoare de similaritate între soluțiile găsite de furnici și  $\sigma$  este o variabilă arbitrară folosită pentru a controla căutarea algoritmului. De menționat că atunci când  $\sigma$  este prea mic,  $\tau_{min}$  poate deveni mai mare decât  $\tau_{max}$ . În acest caz setăm  $\tau_{min} = \tau_{max}$  și ar corespunde cu algoritmul luând decizii doar pe baza informațiilor euristice. Dacă alegem  $\sigma = 1$  atunci  $\tau_{min}$  va tinde spre 0, algoritmul transformându-se în varianta ES. Alegerea valorilor pentru  $\sigma$  este direct corelată cu explorarea algoritmului MMAS când a convers. Așadar,  $\sigma$  oferă o bună metodă de a investiga efectele  $\tau_{min}$  în performanța algoritmului MMAS.

### 6.3.3 Modul de inițializare a feromonilor

În MMAS inițializăm cantitatea de feromonii în așa fel încât în prima generație toți feromonii să corespundă cu  $\tau_{max}(1)$ . Acest obiectiv poate fi atins făcând cantitatea de feromoni să fie egală cu un număr arbitrar mare. După prima iterație toate valorile vor fi forțate să ia valori în limitele impuse, astfel ele se vor seta la  $\tau_{max}(1)$ . Această decizie este luată pentru a mari explorarea algoritmului în primele stadii ale căutării.

## 7 Parametrii și particularități

Cu excepția cazului în care se indică explicit astfel, se folosesc următorii parametri. Folosim  $\beta = 2, \alpha = 1, m = 20$ , unde  $m$  este numărul de furnici, și  $\rho = 0.2$ . Pentru MMAS, feromonii sunt actualizați doar de cea mai bună furnică de la începutul algoritmului. Limitele feromonilor au fost alese după metoda descrisă mai sus cu  $\sigma = 0.6$ .



Calcularea valori  $v$  a fost facuta în modul următor: La finalul fiecărei iterații, fiecărui drum format de către furnici i s-a alocat un număr hash. Cu acest numar, am putut identifica procentul drumurilor similare, formate de furnici. Astfel avem un număr între 0 și 1 care ne poate indica gradul de similaritate a furnicilor. Dacă  $v$  este apropiat de 0, atunci cel mai probabil drumurile formate de furnici sunt foarte diferite, iar cand  $v$  este apropiat de 1 putem ști ca populatia stagneaza.

Standard, informația euristică folosită în ACO este inversul distanței dintre puncte. Însă pe aceasta problema, putem lua în considerare și ora la care se face călătoria. Intuitiv, o sarcina care începe devreme, va fi conectata în matrice cu mai multe alte sarcini. Calculam astfel gradul de “conectivitate” a fiecărui nod și îl notăm cu  $\nu$ . El reprezinta numarul de noduri in care poate ajunge din acel punct. Presupunem ca daca actualizam informația euristică a problemei sa ia în calcul acest număr, rezultatele vor fi îmbunătățite. Așadar, formula de actualizare devine :

$$\tau_{ij}(t+1) = (1 - \rho) \cdot \tau_{ij}(t) + \sum_{k=1}^m \Delta\tau_{ij}^k(t) \cdot \nu$$

## 8 Rezultate

Graficele sunt o medie, din 30 de rulari, a valorilor la fiecare iteratie.

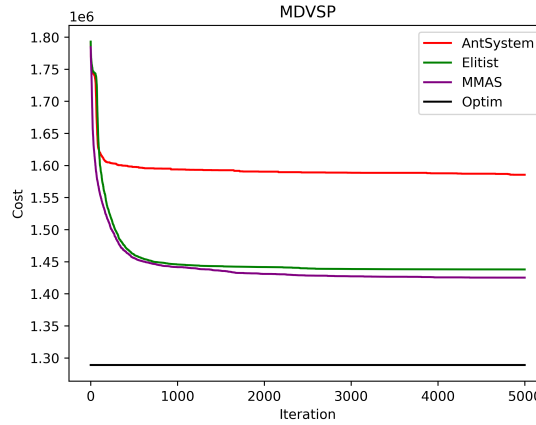


Figure 15:

Se poate observa ca prima varianta descrisa, AntSystem, nu converge in de ajuns de repede pentru a da un raspuns bun intr-un timp rezonabil. Varianta elitista, in care o singura furnica actualizeaza feromoni, are rezultate mai bune. Cele mai bune rezultate au fost obtinute cu MMAS. Se poate observa ca particularitatile algoritmului MMAS au reusit sa pastreze exploatarea solutiilor a

algoritmului elitist, imbuntatind in sa explorarea algoritmului.

În secțiunea 7 s-a vorbit despre modificarea informației euristice în problema anterioara. Aceasta modificare a adus îmbunătățiri considerabile algoritmului.

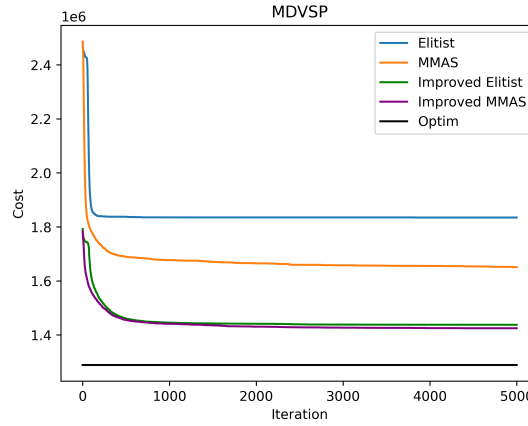


Figure 16:

Aceasta modificare a îmbunătățit atat solutia initiala cat si cea finala a ambilor algoritmi. Diferenta dintre MMAS și Elitist este mai pronunțată în varianta fără îmbunătățirea precizata.

## 9 Concluzie

## References

- [1] *Uncovering temporal changes in Europe's population density patterns using a data fusion approach.* 2020.
- [2] Z-Z Yang<sup>1</sup> B Yu<sup>1</sup> and J-X Xie. *A parallel improved ant colony optimization for multi-depot vehicle routing problem.* 2010.
- [3] M. Dorigo. *Optimization, Learning, and Natural Algorithms (in Italian).* PhD thesis, Dip. Elettronica, Politecnico di Milano. 1992.
- [4] Cristian FRASINARU Emanuel Florentin OLARIU. *Multiple-Depot Vehicle Scheduling Problem Heuristics.* 2020.
- [5] V. Maniezzo M. Dorigo and A. Colomi. *The Ant System: Optimization by a Colony of Cooperating Agents.* IEEE Transactions on Systems, Man, and Cybernetics. 1996.

- [6] Dr.S.Sumathi Surekha P. *Solution To Multi-Depot Vehicle Routing Problem Using Genetic Algorithms*. 1999.
- [7] Iwan Rudiarto Wiwandari Handayani. *Dynamics of Urban Growth in Semarang Metropolitan – Central Java: An Examination Based on Built-Up Area and Population Change*. 2014.