Multiple-Depot Vehicle Scheduling Problem

Silviu Ilas

June 2021

Contents

1 Introducere

Problema rutarii vehiculelor (VRP) este o generalizare a problemei comisului voiajor (TSP). Diferenta este ca se adaugă un punct (putem să-l numim depou) din care toate rutele trebuie sa porneasca și in care trebuie sa se termine. Știind ca TSP este o problema NP dificilă, putem deduce și ca VRP este cel puțin la fel de grea. Adaugand restricția de timp acestei probleme (VRP), "clienții" pot fi deserviți doar la anumite intervale orare, ajungem la problema programării vehiculelor (VSP). Se poate demonstra că aceasta problema poate fi rezolvata în timp polinomial atunci cand avem un singur depou. Însă, dacă avem 2 sau mai multe depouri, aceasta problema devine una dificila, la care nu se stiu algoritmi care o pot rezolva eficient în timp polinomial.

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

Deoarece problema MDVSP este una NP dificila, stim ca nu avem un algoritm care poate rezolva problema într-un timp rezonabil pentru instanțe mari. De aceea mai mulți algoritmi euristici[4] și metaeuristici [6] au fost propuși pentru a găsi o aproximare a soluții minime. Dezavantajul acestor algoritmi este ca nu avem garanția că soluția dată de ei este cea optimă, dar timpul de execuție este mult mai mic pentru instanțe medii sau mari. Ei,însă, sunt susceptibil in a fi blocati în minime locale, nereusind de multe ori sa ajungă la optimul global. Totuși, de cele mai multe ori se prefera o soluție sub optimala livrată într-un timp rezonabil decât o soluție optimă care poate dura cateva sute de ani sa fie gasita.

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

2 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 $(d_1, d_2, ...d_m)$. Fiecare depou are un număr exact de vehicule asignat. Avem și un număr de sarcini reprezentate de un set de călătorii $(c_1, c_2, ...c_n)$ care trebuie asignate unui vehicul. O pereche ordonată de călătorii (c_i, c_j) este posibila 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 (σ_i), un timp de final (χ) si timpul necesar (θ_i) pentru a parcurge călătoria (c_i, c_j) de la finalul călătoriei c_i. Utilizand aceste notații, călătoria (c_i, c_j) este posibila 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 adaugă și costul așteptării, costul drumului sau alți factori. Pe deasupra avem și costul scoaterii vehiculului din depou și costul întoarcerii lui. Toate aceste lucru formează o matrice de costuri pătratică ,de dimensiune n+m, în care +inf înseamnă un drum imposibil. Din punctul de vedere algorithmic, 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 reprezinta posibilitatea relatiei date de c.

Termeni des folosiți în contextul MDVSP sunt "pull-out trips", reprezentand drumurile de la depou la prima sarcina, "pull-in trips" sunt opusul, ele reprezinta drumul de la ultima sarcina din tur la depou.

O soluție a MDVSP este o atribuire a sarcinilor la vehicule astfel incat sa minimizeze suma totala și să respecte următoarele reguli : fiecare sarcina trebuie îndeplinită o singura data, fiecare vehicul trebuie sa termine la același depou din care a început, numărul de vehicule folosit de fiecare depou poate fi depasit.

3 Alte tehinci folosite

Multi

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()).

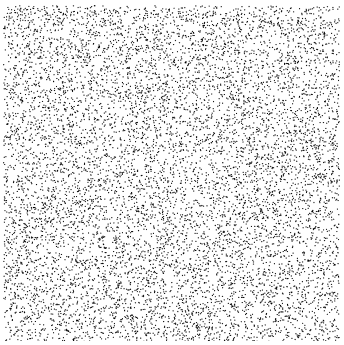

Figure 1: Puncte distribuite aleatoriu

4.1.2 Abordarea realista

In general, pe instantele din lumea reala a acestei probleme, putem observa niste tipare care vizează asezarea geografia 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.

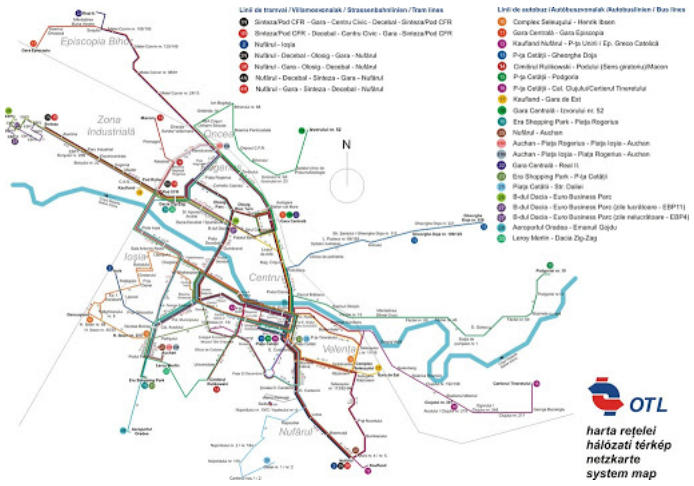

Figure 2: Harta mijloacelor de transport in comun Iași

In poza de mai sus preluata de pe `http://harti.tramclub.org/` este reprezenta harta mijloacelor de transport în comun din Iași, care poate fi considerată o instanța de MDVSP. Chiar și în cazul prezentat mai sus, afirmațiile anterioare sunt valabile.

Distributia Normala

Este cea mai comuna distributie folosita pentre analize statistice. Cea standard are 2 parametri, mediană și deviația standard. Pentru cea normala, 68 % din observatii sunt la +/- o unitate de mediană, 95% sunt la +/- doua unitati, iar 99.7% sunt la trei unități. Cand este pusă pe un graf, informațiile urmează forma unui clopot, cele mai multe informații adunandu-se în jurul unei zone centrale și scazand cu cat se apropie mai mult de margini.

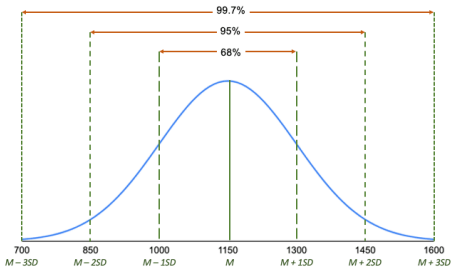

Figure 3: Distributia normala standard

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

comuna, putem sa ne asteptam sa fie o corelație asemănătoare intre clienti si distanța lor fata de centrul. Aceasta afirmație este intarita 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 schimband funcția "random" cu "random-Gauss". Aceasta functie întoarce o valoare în intervalul stabilit urmand distributia normala.

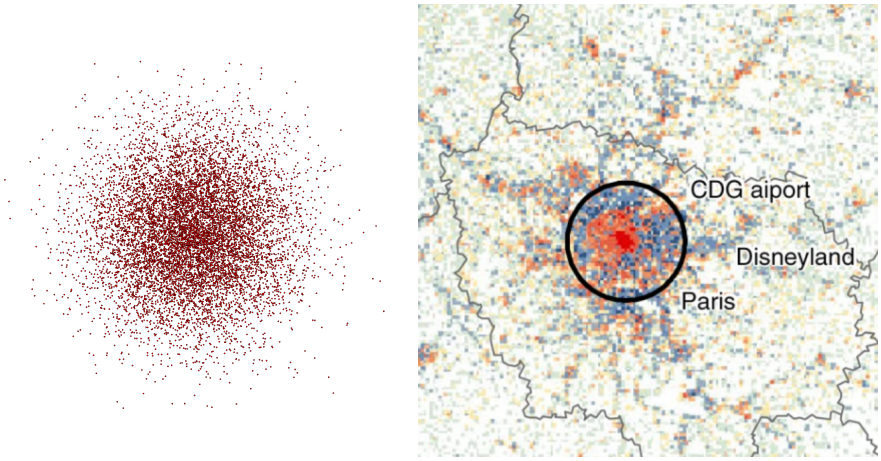

Figure 4: Comparatie între punctele generate de noi și densitatea populatiei parisului [1]

În partea stângă se găsește generarea clienților pe baza regulilor descrise anterior, în partea opusă este densitatea populatiei 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".

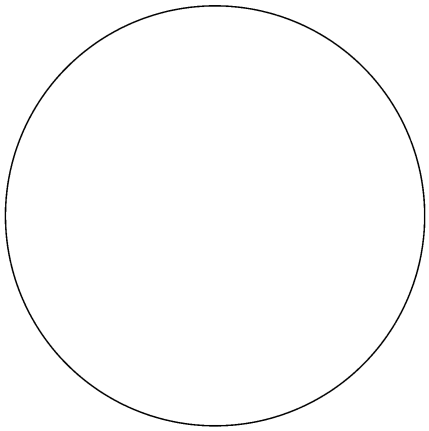

Figure 5: Depourile generate

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 atat de comuna încât nici nu putem identifica nici un nod individual, nodurile colectiv luând aspectul unui cerc perfect.

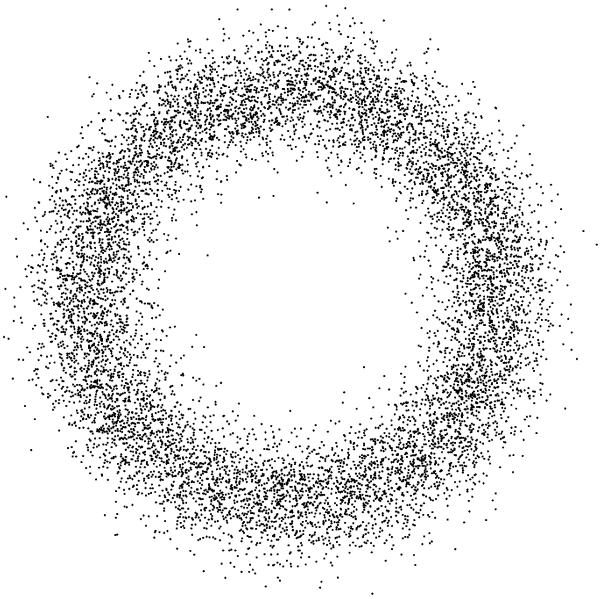

Figure 6: Depourile generate adaugand zgomot

Adaugand acel “zgomot”, care se foloseste de distributia normala, in 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 euclidiana 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 2 . 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 reprezentarii grafice si vor fi detaliate în 4.3.2. În meniu mai sunt disponibile și 2 butoane, cele de "preview" (4.3.2) și "start" (**??**).

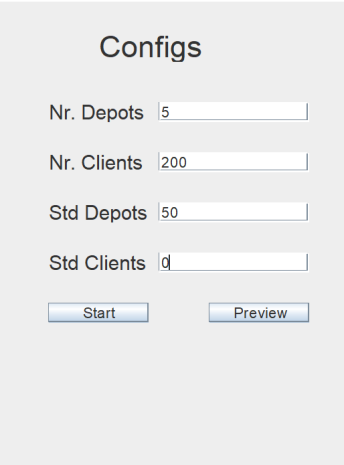

Figure 7: Meniul interfetei

4.3.2 Previzualizarea

Butonul de previzualizare deschide o fereastra în care se pot vedea efectele ultimilor doi parametrii. Parametrul "Std clients" reprezinta deviatia standard a clientilor. Cu cat aceasta este mai mica, cu atat ei probabil vor fi mai apropiati de centru, dând aspectul unui oraș aglomerat. Parametrul "Std depots" reprezinta deviata standard a depourilor. În capitolul 4.1.2 am explicat poziționarea depourilor, astfel parametrul influențează deviația de la marginea cercului a unui depou. Acești parametrii iau o valoare între 0 si 100. Valorile sunt abstractizate de implementarea lor efectiva. Dacă am permite deviația 0, toți clienții s-ar concentra într-un punct iar depourile ar forma un cerc perfect 5. Pentru a evita acest scenariu, 0 reprezinta deviatia minima care respecta trasaturile cautate in generarea interfetei descrie in 4.1.2. Aceeași mentalitate a fost adoptată în stabilirea celeilalte extremitati, 100 reprezinta deviatia maxima care este încă relevantă.

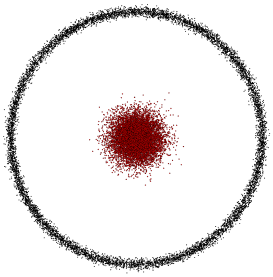

Figure 8: 0-0

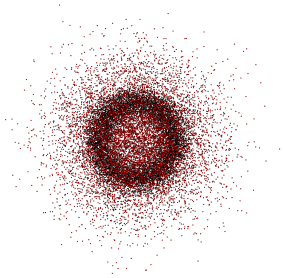

Figure 9: 100-100

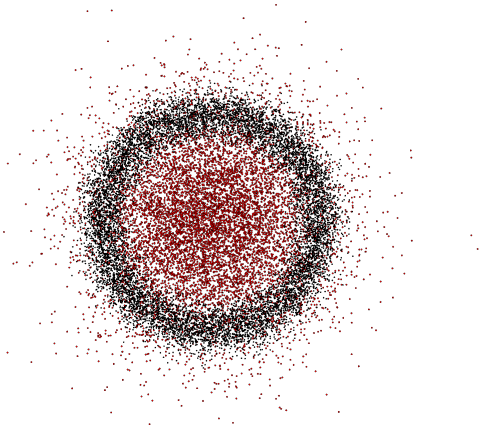

Figure 10: 50-50

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

4.3.3 Start

Butonul de start genereaza o instanta MDVSP si o reprezinta grafic utilizatorului. Aceasta instanța poate fi trimisă unui rezolvator care raspunde cu o soluție. Soluția poate fi reprezentată pe graful rezultat. O muchie intre doua noduri reprezinta un drum de la un client/depot la altul. La început, fiecare depot 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

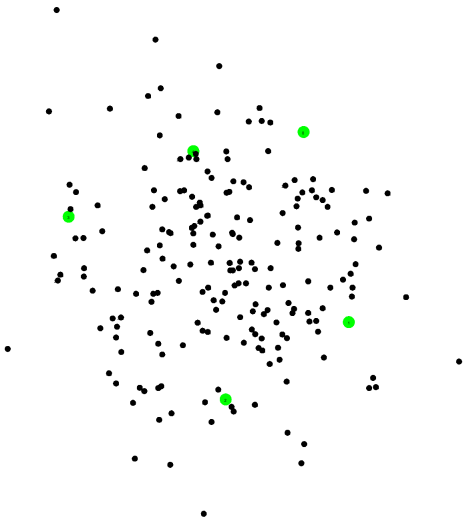

Figure 11: Graful la inceput

Instanța generata de algoritm reprezentată în figura 11 are 5 depouri, 200 de clienți și deviația standard 50. Pe aceasta instanța relativ mica, nu este la fel de evidenta distribuția uniforma folosita ca in 10, dar totuși se pot observa trăsăturile caracteristice : probabilitatea găsirii unui client scade cu cat ne departam de centru și depourile sunt așezate în jurul clienților.

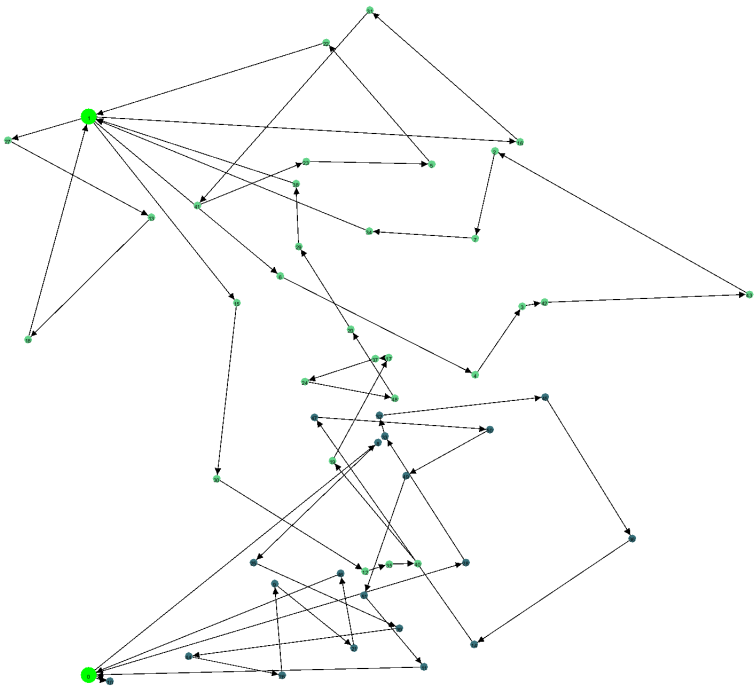

Figure 12: Graful dupa ce primeste solutia

Figura 12, are 2 depouri, 50 de clienți și deviația standard 50. Diferenta principala fata de 11 este ca, pe langa faptul ca are mai puține depouri și clienți (pentru a putea vizualiza nodurile mai bine), reprezentarea grafica a primit o soluție de la un rezolvator. Astfel clienții au fost colorati, luand culoarea asignata la început depourilor și muchiile traseelor au fost trasate.

5 Optimizarea Coloniei de Furnici

5.1 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 înainte și înapoi mai des între cuib și mâncare. Acest caz este exemplificat în figura de mai jos.

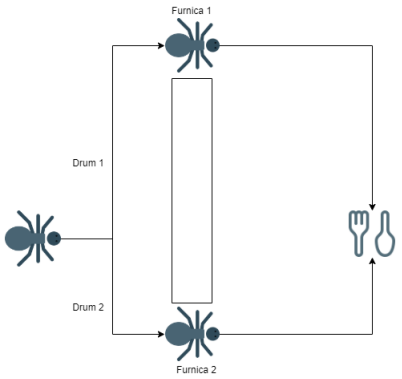

Figure 13: Modelul problemei

Se poate vedea ca furnica 2 va putea face mai multe drumuri de la mancare 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 varianta.

5.2 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 metaheuristică: rețele neuronale, algoritmi genetic, optimizarea coloniei de furnicii etc.

5.3 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 utilizata in astfel de algoritmi este cea a comunicarii pe baza de feromoni. Prin aceasta comunicare se face un fel de inteligenta de grup. Deși fiecare furnica artificială este relativ simpla, interacțiunea dintre ele și mediul lor înconjurător poate da apariția unei "inteligente" colective, necunoscuta de fiecare la nivel individual. Marcarea drumurilor cu feromoni artificiali este un fel de informație numerică distribuită care este modificata 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. Adițional, 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 ca atunci cand o furnica își construiește traseul, va fi mai probabil sa aleagă calea cu mai multi feromoni, face ca statistic drumurile mai bune să fie din ce în ce mai folosite.

6 ACO aplicată pe MDVSP

6.1 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 informația 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) reprezinta un arc iar t este numărul iteratiei. 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 reprezinta clienti, m depourile. Am adaugat un depou suplimentar, numit depou virtual, care se conecteaza la toate celelalte depouri cu arce de cost 0. Cum fiecare depou este conectat la toți clienții, avand acest depou principal ne este garantat ca avem un graf conex. Astfel, orice furnica poate vizita fiecare nod de pe graf într-o singura iterație. [2]

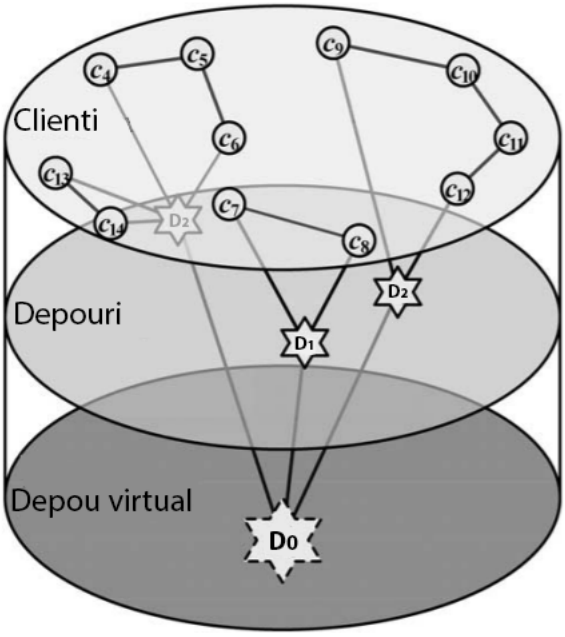

Figure 14: Modelul problemei reprezentat grafic

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 că 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.

6.2 Cod relevant

6.2.1 Sceleton 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 solutie suboptimala, 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 doua, incercand sa ne oprim cand credem ca rezultatele nu se vor îmbunătăți prea curând. Acestul lucru se poate face empiric, uitandu-ne la date putem observa punctul în care este improbabil sa se gaseasca o îmbunătățire, sau se pot face diferite metode de a detecta cand algoritmul stagneaza. O astfel de varianta des intalnita este oprirea după trecerea a X iterații de la cel mai buna soluție.

Funcția "generateAnts()"

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

Funcția "evaulateAnts()"

Construiește soluții apeland funcția run() a fiecărei furnici din iterația curentă, astfel la finalul acestia fiecare furnica va avea drumul construit în acea iterație.

Funcția "updatePheromones()"

Aplica procesul prin care se modifica cantitatea de informație(feromonii) din graf. Aceasta poate fie sa creasca pe o anumită secțiune, dacă a fost o soluție buna, fie sa scadă, datorită evaporării feromonilor. Depozitarea feromonilor crește probabilitatea ca acea componenta sa fie folosită în viitor de alte furnici. Evaporarea feromonilor este o forma de uitare care ajuta la evitarea convergenței premature a algoritmului în o regiune suboptimala, astfel favorizant explorarea.

6.2.2 Scheleton 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.

6.2.3 Restrictiile furnicilor

Functia 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 conditiile. Condiția generala a unei coloni de furnici este că ca nodurile ținta pot fi vizitate o singura data atat timp cat nu a fost specificat altfel. Daca conditia de baza a fost indeplinita se itereaza prin celelalte conditii specifice problemei.

Codul care descrie conditiile aditionale a acestui model pe aceasta problema.

```
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 pull out a depot if you don't have enough vehicles
    if (edgeType == EdgeType.PULL_OUT) {
        .
    }
```

```
    DefaultWeightedEdge lastEdge = this.getLastPickedEdge();
    EdgeType lastPickedEdgeType =          this.getMdvspAntColonyGraph().getEdgeTyp
     // we should not go to a depot that has no more remainingDepots 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, sa forțăm furnicile sa se întoarcă la depoul inițial și să respecte numărul de vehicule dat. Următoarele 3 sunt legate de modul în care am modelat problema. Ele asigura ca traseul drumului este respectat în modul menționat mai sus.

6.2.4 Ordonarea furincilor

În aceasta problema, nu toate solutiile sunt valide, așadar algoritmul propus încearca sa minimizeze 2 funcții:

Prima este numărul de clienți satisfăcuți. Algoritmul presupune că, indiferent de cost, o soluție care satisface mai puțini clienți este inferioara. De obicei, aceasta funcție este minimizata la începutul algoritmului.

Costul a doua soluții este relevant doar atunci cand ambele soluții satisfac toți clienții. Așadar a doua funcție de minimizat, cea a costului, este luată în considerare doar în aceasta situatie.

Pentru a determina calitatea soluțiilor determinate de algoritm, sortam furnicile în modul următor:

```
    ants.sort((ant1, ant2) -> {
        int unsatisfiedNr1 = ant1.getNumberOfNotVisitedVertexes();
        int unsatisfiedNr2 = ant2.getNumberOfNotVisitedVertexes();
        if (unsatisfiedNr1 == 0 && unsatisfiedNr2 > 0)
            return -1;
        else if (unsatisfiedNr1 > 0 && unsatisfiedNr2 == 0)
            return 1;
        else if (unsatisfiedNr1 == 0 && unsatisfiedNr2 == 0)
```

```
            return ant1.getCurrentCost() - ant2.getCurrentCost();
        else
            return unsatisfiedNr1 - unsatisfiedNr2;
    });
```

Făcând asta, respectăm ordinea impusă de cele doua functii de minimizat.

6.3 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.

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

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) si 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. α și β sunt influența relativă a traseelor feromonilor și respectiv vizibilității. Daca α este 0, atunci selecția va fi bazată doar pe informația euristica a algoritmului, acest lucru corespunde cu un algoritm greedy. Pe cand atunci cand β este 0, algoritmul va alege doar pe baza nivelului de feromoni, ducand astfel rapid la stagnare într-un punct în general suboptimal. Tabu este setul de noduri imposibile.

6.4 Regula de actualizare

În funcție de varianta de ACO folosită, regula de actualizare poate fi diferită. Totuși, toate au în comun faptul ca urmăresc drumul al cel puțin unei furnici și cresc cantitatea de feromoni folosită pe fiecare muchie parcursa de ea utilizand urmatoarea formula:

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

unde $0 < \rho \leq 1$ este evaporarea urmelor feromonilor. Se utilizează parametrul ρ 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 in mod standard este definit in 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 funricii 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.

7 Variante de ACO

7.1 Ant System

Este primul algoritm ACO și este în mare masura asemanator cu ce am descris mai sus. A fost dezvoltat in anul 1992 de Dorigo, V. Maniezzo și inițial au fost propuse trei variante diferite numite "ant-density", "ant-quantity" si "ant-cycle". Pe cand "ant-density" si "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.

7.2 Elitist System

Vine ca o primă îmbunătățire a variantei originale si 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 turneu 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.

7.3 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 lacomă poate agrava problema stagnării premature a căutării. Prin urmare, cheia pentru a obține o buna performanta cu algoritmii 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 soluti 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 solutiei 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 limitata, cu consecința existentei 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 componentele 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 in ce mai frecvent pe parcursul algoritmuluii.

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 incetineste pană 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 depinzand direct de informatiile euristice si cantitatea de feromoni. Informațiile euristice sunt, de obicei, dependențe de problema și statice pe tot parcursul algoritmului. Dar, prin limitarea influenței cantității de feromoni,

putem evita cu usurinta acest caz. Pentru a atinge acest obiectiv MMAC impune limite explicite, τ_{min} și τ_{max}, astfel incat cantitatea de feromoni este limitata la $\tau_{min} \leq \tau_{ij}(t) \leq \tau_{max}$. Pentru a impune aceasta condiție, după fiecare iterație putem sa ne asigurăm ca aceste limite au fost respectate. Dacă exista $\tau_{min} > \tau_{ij}(t)$ atunci stabilim $\tau_{ij}(t) = \tau_{min}$, analog pentru τ_{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 principial de determinare a acestor valori. O sa începem cu introducerea noțiunii de convergență pentru MMAS. Spunem că MMAS a convers, dacă pentru fiecare punct de alegere, una dintre muchii are τ_{max} cantitate de feromoni, în timp ce toate celelalte alternative au τ_{min}. Dacă MMAS a convers, soluția construită prin alegerea muchiilor cu cantitate de feromoni maximă va corespunde de obicei cu cea mai bună soluție găsită de algoritm. Conceptul de convergența 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.

7.3.1 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 buna soluție posibilă pentru problema specifica. 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, setam τ_{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 data cand se imbunatateste cea mai buna solutie gasita de la începutul algoritmului schimbăm valoarea τ_{max}, ducând de fapt la o valoare dinamică în schimbare a $\tau_{max}(t)$

7.3.2 Modul de alegere a minimului

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

Pentru a determina valoarea minimului facem următoarele doua presupuneri :

- Prima este ca în jurul soluțiilor bune este o sansa relativ mare de a găsi soluții și mai bune. Validitatea acestui argument depinde de caracteristică 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 diferenta de informații euristice. Acest lucru este posibil dacă influența informațiilor euristice este scăzut și poate fi ajustată modificand parametrul β.

Avand in vedere aceste presupuneri, valori bune pentru τ_{min} sunt găsite coreland convergenta algoritmului cu minimul cantității de feromoni. Astfel, putem actualiza minimul, la fiecare iterație, după următoarea formula.

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

unde $(1-\rho)$ este parametrul folosit pentru evaporarea feromonilor, υ este o valoare de similaritate între soluțiile găsite de furnici și σ este o variabila arbitrară folosită pentru a controla căutarea algoritmului. De menționat ca atunci cand σ este prea mic, τ_{min} poate deveni mai mare decat τ_{max}. În acest caz setam $\tau_{min} = \tau_{max}$ și ar corespunde cu algoritmul luand decizii doar pe baza informațiilor euristice. Dacă alegem $\sigma = 1$ atunci τ_{min} va tinde spre 0, algoritmul transformandu-se in varianta ES. Alegerea valorilor pentru σ este direct corelată explorarea algoritmului MMAS cand a convers. Așadar, σ oferă o buna metoda de a investiga efectele τ_{min} în performanta algoritmului MMAS.

7.3.3 Modul de inițializare a feromonilor

În MMAS initializam cantitatea de feromonii in asa fel incat în prima generatie toți feromoni sa corespundă cu $\tau_{max}(1)$. Acest obiectiv poate fi atins facand cantitatea de feromoni sa fie egala cu un număr arbitrar mare. După prima iterație toate valorile for fi forțate sa ia valori în limitele impuse, astfel ele se vor seta la $\tau_{max}(1)$. Aceasta decizie este luată pentru a mari explorarea algoritmului în primele stadii ale căutării.

8 Parametrii si particularitati

Cu excepția cazului în care se indica explicit astfel, se folosesc următorii parametrii. Folosim $\beta = 2, \alpha = 1, m = 20$, unde m este numărul de furnici, și $\rho = 0.2$. Pentru MMAS, feromoni sunt actualizati doar de cea mai buna furnica de la începutul algoritmului. Limitele feromonilor au fost alese după metoda descrisă mai sus cu $\sigma = 0.5$.

Calcularea valori υ 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 υ este apropiat de 0, atunci cel mai probabil drumurile formate de furnici sunt foarte diferite, iar cand υ este apropiat de 1 putem ști ca populatia stagneaza.

Standard, informația euristica 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 ν. El reprezinta numarul de noduri in care poate ajunge din acel punct. Presupunem ca daca actualizam informația euristica a problemei sa ia în calcul acest număr, rezultatele vor fi îmbunătățite. Așadar, formula de actualizare devine :

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

9 Rezultate

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

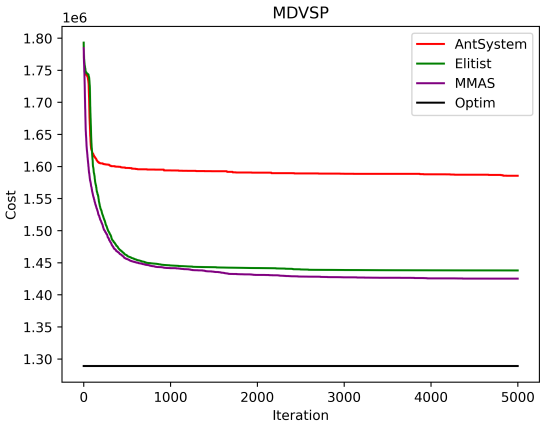

Figure 15:

Se poate observa ca prima varianta descrisă, AntSystem, nu converge în de ajuns de repede pentru a da un răspuns bun într-un timp rezonabil. Varianta elitistă, în care o singura furnica actualizează feromoni, imbunatatesti rezultatele considerabil. Cele mai bune rezultate au fost obtinute cu MMAS. Se poate observa ca particularitatile algoritmului MMAS au reușit sa pastreze explorarea soluțiilor a algoritmului elitist, îmbunătățind însă explorarea algoritmului.

9.1 Îmbunătățirea informației euristice

În secțiunea 8 s-a vorbit despre modificarea informației euristice în problema anterioara. Aceasta modificare a adus îmbunătățiri considerabile algoritmului.

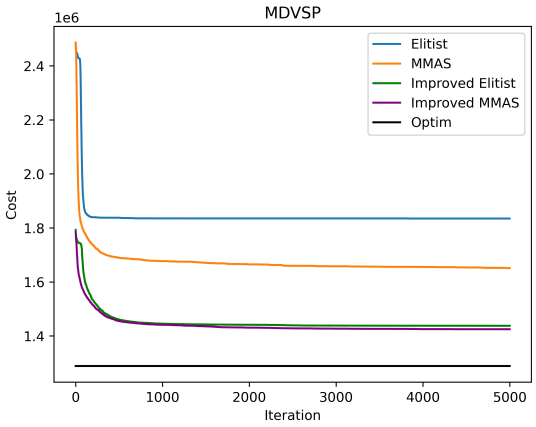

Figure 16:

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

9.2 Influența parametrului σ

În continuare vom vedea influența parametrului σ în calitatea soluției a algoritmului MMAS.

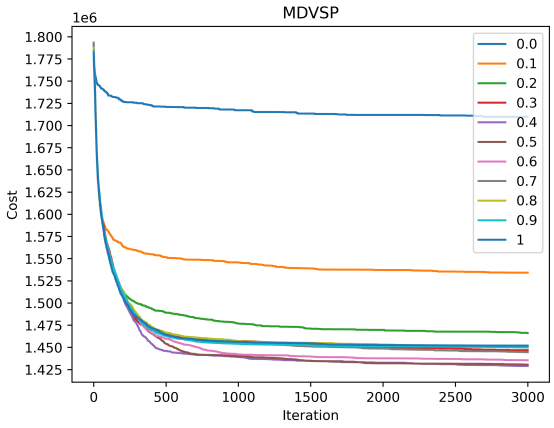

Figure 17: Influenta parametrului σ

Din datele colectate, putem deduce ca parametrul σ influențează atât convergenta algoritmului cat și calitatea soluției găsite. Cazul extrem, în care σ

este 0, arăta rezultatele găsite de algoritm atunci când $\tau_{min} = \tau_{max}$. Făcând asta, algoritmul va lua în considerare doar informația euristica. Următorul caz testat, în care parametrul ia valoarea 0.1, imbunatateste simnificativ cea mai buna solutie. Algoritmul începe sa ia în considerare și feromoni lăsați de furnici. Calitatea și convergenta soluțiilor continua sa creasc̆ă până când valoarea σ este egala cu 0.4. Din acest punct, cu cat am crescut valoarea parametrului cu atat calitatea rezultatelor a început sa se diminueze. Acest lucru ne determina sa alegem = 0.4 ca și parametrul folosit în testele conduse. Mentionam și ca atunci cand valoarea parametrului este egala cu 1, τ_{min} va putea ajunge la 0. Acest lucru sa si intamplat în toate rularile făcute. Dacă toate muchiile în afara de una ajung la τ_{min} în aceasta situatie, algoritmul va stagna fără a avea nici o sansa se îmbunătățire.

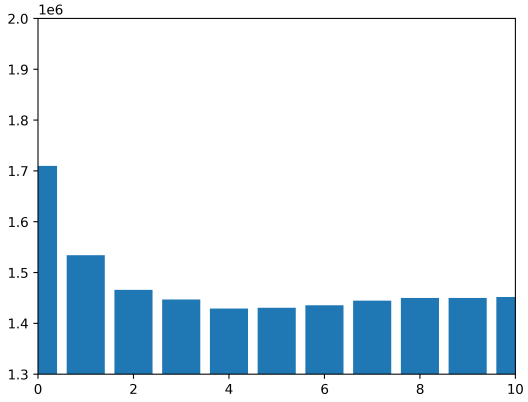

Figure 18:

In graficul 18, axa x reprezinta valoarea σ si pe y regasim costul celei mai bune soluție găsite de algoritm. Cu cat costul este mai mic cu atat soluția este mai buna.

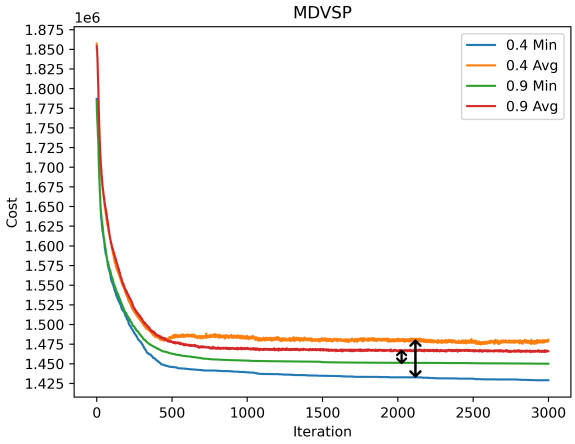

Figure 19:

În cazul ales, σ ia valoarea 0.4 respectiv 0.9. Atunci cand valoarea parametrului este 0.4, chiar dacă răspunsul final este mai bun, din graficul rezultat putem observa ca în medie, furnicile construiesc un drum mai bun atunci cand sunt limitate de celalat parametru. Însă evidențiem faptul ca "distanța" dintre soluția medie și soluția cea mai bună este mai mica în al doilea caz. Acest lucru este marcat pe grafic cu cele doua linii verticale. Atunci cand "distanta" intre minim si medie, în anumite situații, îi permite algoritmului să iasă din minime locale chiar dacă soluțiile sunt în medie de o calitate mai slabă. În situația prezentată, alegand valoarea 0.9, exploatarea algoritmului este prea mare.

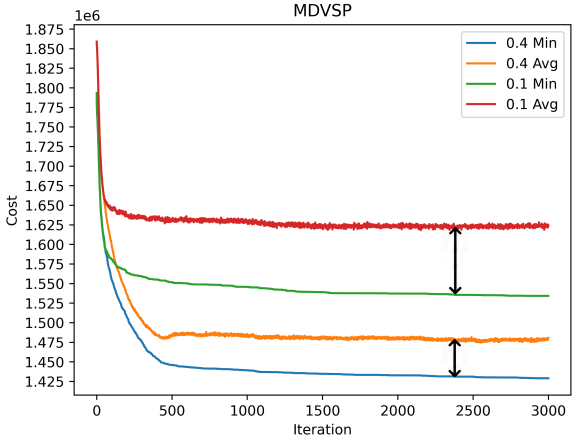

Figure 20:

Aici, exemplificam situația în care, deoarece soluțiile sunt in medie prea diferite una de cealalta, explorarea algoritmului este prea mare și nu imbunatateste în de ajuns de repede soluțiile găsite.

9.3 Influența parametrului ρ

În varianta clasică a algoritmului ACO, o valoare crescută a parametrului ρ determina o rata scăzută de convergere. Totuși, datorită faptului ca MMAS permite doar unei singure furnici să actualizeze feromonii, algoritmul va "uita" mai rapid toate celelalte soluții. Acest lucru reintareste diferenta dintre feromonii de pe cea mai buna soluție aleasă și ceilalți. Astfel, efectul parametrului ρ este inversat. Cand parametrul este prea mare, algoritmul converge prea rapid și ajunge rapid într-un minim local. Cand este prea mic, rezultatul final nu este garantat să fie îmbunătățit și cea mai buna soluție este găsită mai târziu în execuție. Un parametru adecvant permite un compromis intre viteza si calitate.

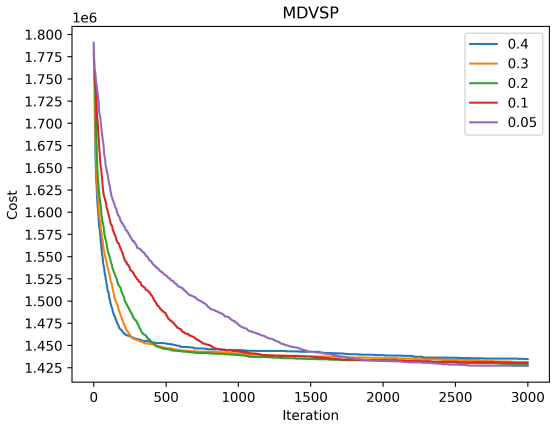

Figure 21:

Din figura 21 reiese faptul ca parametrul ρ influențează foarte mult rata la care algoritmul converge. Cand parametrul este prea mare, algoritmul converge prea rapid și ajunge rapid într-un minim local. Cand este prea mic, rezultatul final nu este garantat să fie îmbunătățit și cea mai buna soluție este găsită mai târziu în execuție. Un parametru adecvant permite un compromis intre viteza si calitate.

9.4 Performanța pe diferite instanțe

În tabela de mai jos, instanțele au fost generate de algoritmul descris in [4]. Putem astfel compara cea mai buna soluție posibilă cu cea găsită. Pentru

instanțe mici, algoritmul da rezultate bune, însă, calitatea rezultatelor începe sa scadă rapid pe instanțe mai mari.

Instanta	Optim	ACO	Eroare(%)
m2n50s0	164707	165214	0.3%
m2n75s0	238402	240671	0.95%
m2n100s0	312547	318999	2%
m3n200s0	622206	660375	6%
m3s500s0	1571950	1679453	6.8%
m4s1000s0	3142127	3411120	8.5%
m4s2000s0	6305846	6860259	8.7%

10 Concluzie

Pentru găsirea algoritmilor care reușesc sa rezolve eficient și rapid aceasta problema, este imperativ sa îi putem testa pe instanțe reprezentative. Aceste instanțe sunt generate de multe ori în mod aleatoriu, pentru a testa algoritmul pe o varietate de scenarii, dar pot include și instanțe reale. Aceasta lucrare descrie o metoda de a genera instanțe automat, dar totuși ancorand problema în realitate, generandu-se după anumite reguli observabile în cele mai multe instanțe reale. Aceasta abordare mixtă, permite reprezentarea grafica a instanței generate, un lucru care poate fi util în înțelegerea și prezentarea problemei. Spre deosebire de alte generatoare de instanțe cum ar fi cele descrise în [4], modalitatea abordată de generare nu oferă și optimul global, utilă în testarea scalabilitatii și preciziei algoritmilor de către dezvoltatori. Un alt dezavantaj ar fi ca numărul scenariilor incluse în o astfel de generare este mai limitat decât în generarea aleatorie. Acestea fiind spuse, generatorul propus vine ca un adaos în testarea algoritmilor, nu o inlocuire.

Pe MDVSP, aceasta lucrare arată faptul ca ACO poate fi utilizat pentru rezolvarea problemei. Algoritmul oferă rezultate bune pentru instanțele mici, dar pentru instanțele mari rezultatele nu au fost competitive cu alți algoritmi euristici sau metaeuristici.

Din testele efectuate, putem deduce ca atat calitatea soluțiilor cat și rata de convergenta este strans legată de valoarea parametrului σ. Mai putem adăuga că acest parametru permite modificarea raportului dintre explorare și exploatare pe parcursul algoritmului. Acest lucru poate fi observat în figurile de la 19 și 20. Astfel alegerea potrivita a σ poate deveni chiar o subproblema importantă a algoritmului.

References

[1] *Uncovering temporal changes in Europe's population density patterns using a data fusion approach.* 2020.

[2] Z-Z Yang1 B Yu1 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. Colorni. *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.