Multiple-Depot Vehicle Scheduling Problem

Silviu Ilas

June 2021

Contents

1 Introducere **2**

2 Optimizarea Coloniei de Furnici **3**
2.1 Furnicile din natura . . . 3
2.2 Metaeuristica . . . 3
2.3 Prezentarea Algoritmului . . . 4

3 Problema Programării Vehiculelor cu mai multe Depouri (MDVSP) **5**

4 ACO aplicată pe MDVSP **6**
4.1 Model . . . 6
4.2 Cod relevant . . . 7
4.2.1 Sceleton ACO . . . 7
4.2.2 Scheleton Furnica . . . 8
4.2.3 Restrictiile furnicilor . . . 9
4.3 Modul de decizie . . . 10
4.4 Regula de actualizare . . . 11

5 Variante de ACO **11**
5.1 Ant System . . . 11
5.2 Elitist System . . . 11
5.3 Min Max Ant System . . . 12
5.3.1 Modul de alegere a maximului . . . 13
5.3.2 Modul de alegere a minimului . . . 14
5.3.3 Modul de inițializare a feromonilor . . . 14

6 Parametrii si particularitati **15**

7 Rezultate **15**

8 Concluzie **17**

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[3] și metaeuristici [5] 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 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 înainte și înapoi mai des între cuib și mâncare. Acest caz este exemplificat în figura de mai jos.

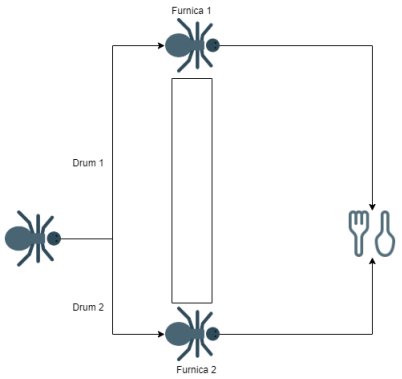

Figure 1: Modelul problemei

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.

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

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 $(d_1, d_2, ...d_m)$. 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 $\chi + \theta_i <= \sigma_j$.

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.

4 ACO aplicată pe MDVSP

4.1 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. [1]

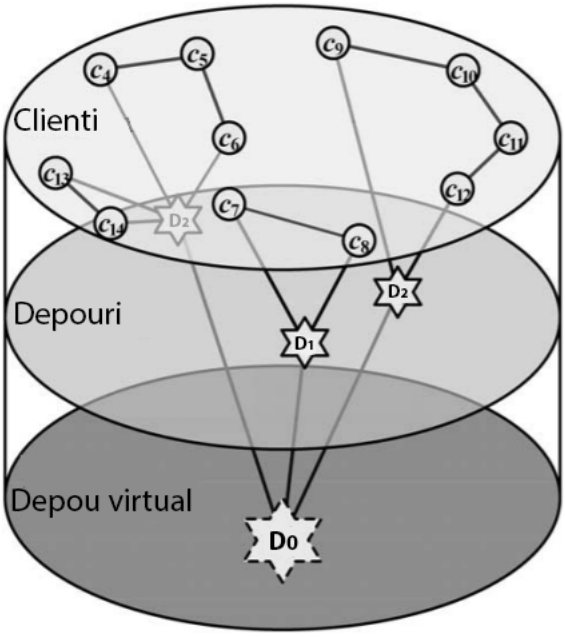

Figure 2: Modelul problemei reprezentat grafic

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.

4.2 Cod relevant

4.2.1 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 aceștia 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.

4.2.2 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.

4.2.3 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.

4.3 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.

4.4 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.

5 Variante de ACO

5.1 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.

5.2 Elitist System

Vine ca o primă îmbunătățire a variantei originale si a fost introdusă în [4] [2]. 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 [4] [2] 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.

5.3 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 arămături 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 opreasc̆ă. 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.

5.3.1 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 $\tau_{max}(t)$

5.3.2 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.

5.3.3 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.

6 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 updatati doar de cea mai buna furnica de la începutul algoritmului. Limitele feromonilor au fost alese după metoda descrisă mai sus cu $\sigma = 0.6$.

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 :

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

7 Rezultate

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

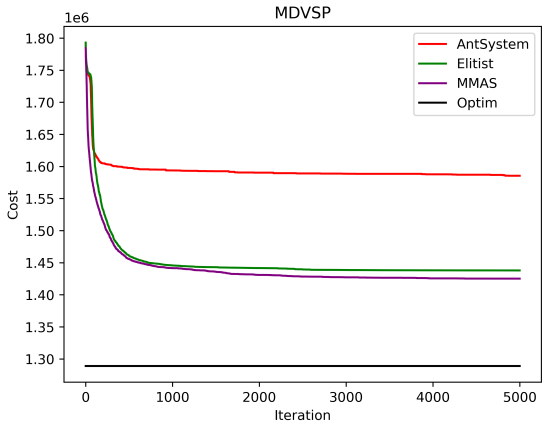

Figure 3:

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 actualizaeaza 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 insa explorarea algoritmului.

În secțiunea section 6 s-a vorbit despre modificarea informației euristice în problema anterioara. Aceasta modificare a adus îmbunătățiri considerabile algoritmului.

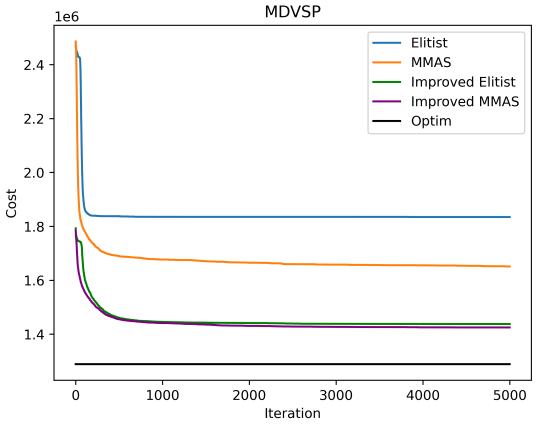

Figure 4:

Aceasta modificare a îmbunătățit atat solutia initialacat si cea finala a am-

bilor algoritmi. Diferenta dintre MMAS și Elitist este mai pronunțată in varianta fără îmbunătățirea precizata.

8 Concluzie

References

[1] Z-Z Yang1 B Yu1 and J-X Xie. *A parallel improved ant colony optimization for multi-depot vehicle routing problem.* 2010.

[2] M. Dorigo. *Optimization, Learning, and Natural Algorithms (in Italian). PhD thesis, Dip. Elettronica, Politecnico di Milano.* 1992.

[3] Cristian FRASINARU Emanuel Florentin OLARIU. *Multiple-Depot Vehicle Scheduling Problem Heuristics.* 2020.

[4] 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.

[5] Dr.S.Sumathi Surekha P. *Solution To Multi-Depot Vehicle Routing Problem Using Genetic Algorithms.* 1999.