Multiple-Depot Vehicle Scheduling Problem

Cristian Adam, Alexandru-Denis Chipăruș
Al. I. Cuza University of Iași
November 2021

I. Descrierea problemei

Multiple-Depot Vehicle Scheduling Problem (MDVSP) este o bine cunoscută și importantă problemă de optimizare. Aceasta constă în atribuirea unui set de călătorii unui set de vehicule, care se află în mai multe depouri, pentru a minimiza un anumit cost total.

Problema se poate defini astfel: există un număr m de depouri și un număr n de noduri de tip trip (călătorii), iar pentru fiecare depou, există un număr maxim de vehicule care pot fi stocate, numit capacitate a depoului. Se dorește atribuirea unui vehicul fiecărui nod de tip trip, astfel încât:

1) fiecare trip să fie atribuit unui singur vehicul;
2) fiecare vehicul din soluție, să parcurgă un drum fezabil format din noduri trip (perechile consecutive de noduri să fie fezabile);
3) fiecare vehicul care parcurge un drum, să îl termine în același depou din care a plecat;
4) numărul de vehicule utilizate în fiecare depou să nu depășească capacitatea de stocare a depoului;
5) suma costurilor asociate arcelor utilizate în fiecare drum parcurs de vehiculele alese să fie minim.

Considerând graful problemei $G = (V, A)$, unde V este mulțimea nodurilor, iar A mulțimea arcelor fezabile, și mulțimile W, mulțimea nodurilor de tip depou, și N, mulțimea nodurilor de tip trip, ($W \cup N = V$) atunci MDVSP poate fi formulată ca o problemă de programare cu numere întregi (MIP):

$$MDVSP: \ min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}$$

$$\sum_{i \in V} x_{ij} = r_j, \quad j \in V \tag{1}$$

$$\sum_{j \in V} x_{ij} = r_i, \quad i \in V \tag{2}$$

$$\sum_{(i,j) \in P} x_{ij} \leq |P| - 1, \quad P \in \Pi \tag{3}$$

$$x_{ij} \in \{0, 1\}, \quad i, j \in V, \tag{4}$$

unde $r_i = 1$ pentru $i \in N$ (și 0 în rest), Π reprezintă mulțimea tuturor drumurilor dintre două depouri, iar c_{ij} reprezintă elementele matricii costurilor arcelor fezabile.

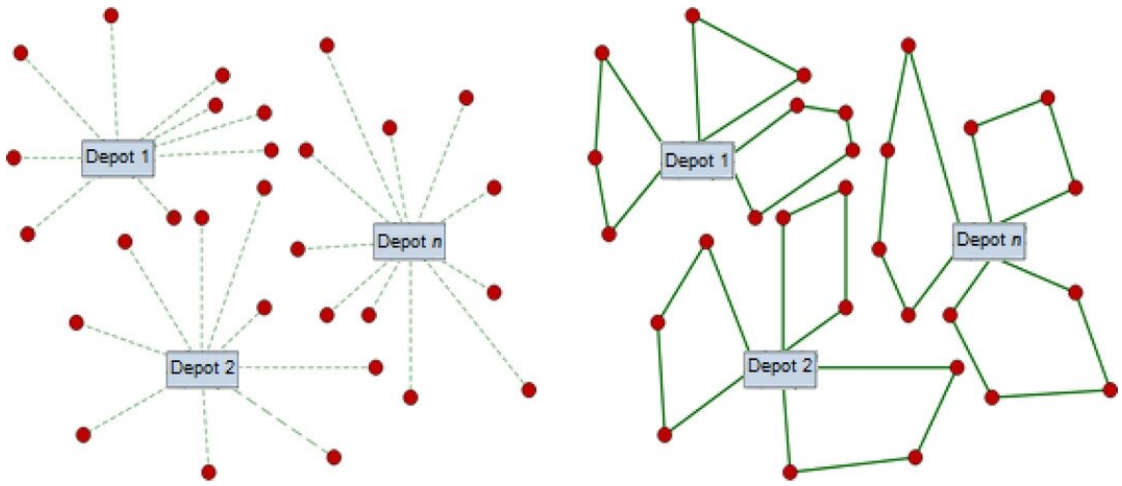

Fig. 1: Un exemplu de instanță a MDVSP și o soluție pentru aceasta. Punctele roșii reprezintă noduri de tip trip.

II. Metode cunoscute din literatură

Când numărul de depouri este mai mare sau egal cu 2, MDVSP devine o problemă NP-hard [3]. Din acest motiv, au fost propuse mai multe abordări pentru această problemă printre care se numără meta-euristici precum Neighborhood Search și Tabu Search [6], Iterated Local Search [5] și abordări de programare liniară pe valori întregi (ILP) fiind cele mai frecvent utilizate metode.

III. Metode utilizate

A. Ant Colony Optimization

Ant Colony Optimization (ACO) este o metaeuristică bazată pe populație care poate fi utilizată pentru a găsi soluții aproximative la probleme dificile de optimizare. Propus inițial de Marco Dorigo în 1992 în teza sa de doctorat, primul algoritm ACO urmărea căutarea unui drum optim într-un graf, fiind bazată pe comportamentul furnicilor care caută cel mai scurt drum între colonia lor și o sursă de hrană. Ideea de optimizare care stă la baza algoritmului este reprezentată de urmele de feromoni, substanță utilizată de furnici pentru a comunica și a se deplasa în mediul lor de viață. Dacă o furnică găsește un drum eficient către o sursă de hrană, atunci acea furnică se va deplasa mai des pe acel drum, lăsând în urma sa o concentrație mai mare de feromoni. Astfel, ea va indica și celorlalte furnici calea către o sursă bună de hrană, în funcție de concentrația de feromoni de pe fiecare drum.

În ACO, un set de agenți software numiți furnici caută soluții bune pentru o anumită problemă de optimizare. Pentru a aplica ACO, problema de optimizare este transformată în problema găsirii drumului de cost minim într-un graf. Furnicile construiesc soluții progresiv, deplasându-se pe grafic. Procesul de construire a soluției este stocastic și este bazat pe un set de parametri asociați cu componentele graficului (fie noduri, fie muchii), numite feromoni, ale căror valori sunt modificate de furnici în timpul rulării algoritmului.

Pentru rezolvarea problemei MDVSP am ales să experimentăm două abordări inspirate din algoritmul Ant Colony System (ACS): ACS aplicat pe o partiție a grafului în funcție de depouri și ACS aplicat pe întregul graf al problemei.

1) ACS aplicat pe o partiție a grafului problemei: Prima abordare din acest proiect implică partiționarea grafului problemei astfel încât fiecărui depou să-i corespundă un subgraf. Această partiționare se realizează astfel:

1) pentru fiecare nod de tip trip se calculează gradul nodului (gradul interior + gradul exterior)
2) se sortează descrescător lista acestor noduri în funcție de grad
3) se repartizează apoi, în ordine, câte un nod trip pentru fiecare depou până toate nodurile trip sunt asociate unui subgraf

Acest mod de a partiționa graful ne asigură un oarecare echilibru între subgrafurile rezultate din punctul de vedere al numărului de noduri și al gradului nodurilor incluse. Ideea partiționării grafului poate fi observată și la rezolvarea problemei MDVSP (Multiple-Depot Vehicle Routing Problem) în [1], unde se repartizează nodurile trip în funcție de cel mai apropiat depou.

Prin partiționarea grafului, MDVSP se reduce la rezolvarea mai multor VSP, câte una pentru fiecare subgraf rezultat. Pentru rezolvarea acestor VSP, utilizăm un algoritm ACS, în cadrul căruia fiecare furnică alege următorul nod folosind formulele următoare:

$$next\_node(i) = \begin{cases} \underset{(ij)\in M_k}{\arg\max} \{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}\} \ , \ dacă \ q \leq q_0 \\ \\ choose\_neighbour(\{p_{ij}^{k}, j \in neighbourhood(i)\}) \ , \ în \ caz \ contrar \end{cases} ,$$

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum\limits_{(im)\in M_k} (\tau_{im}^{\alpha} \cdot \eta_{im}^{\beta})} \ , \ dacă \ (ij) \in M_k \\ \\ 0 \ , \ în \ caz \ contrar \end{cases} ,$$

unde τ_{ij} este cantitatea de feromoni de pe arcul (ij), η_{ij} reprezintă informația euristică (în cazul nostru, egală cu inversul costului arcului (ij)), M_k reprezintă mulțimea nodurilor nevizitate încă de furnica k până la iterația curentă, $choose\_neighbour(p)$ este o funcție care alege vecinii în funcție de probabilitățile descrise de p, q este un număr generat aleatoriu, iar q_0, α și β sunt parametri dați de utilizator.

Matricea feromonilor se actualizează în două moduri: local, la finalul pașilor constructivi ai fiecărei furnici (după construirea soluției furnicii), și global, la finalul fiecărei iterații ale algoritmului (după terminarea pașilor constructivi ai tuturor furnicilor din colonie).

Actualizarea locală a matricii feromonilor se realizează după formula:

$$\tau_{ij} = (1 - \phi) \cdot \tau_{ij} + \phi \cdot \tau_0,$$

unde ϕ este factorul de scădere a feromonilor, iar τ_0 reprezintă valoarea inițială a feromonilor (de la începutul algoritmului).

Actualizarea globală a matricii feromonilor se realizează pe baza formulei:

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

unde ρ reprezintă factorul de evaporare iar $\Delta\tau_{ij}^{best}$ reprezintă cantitatea de feromoni depusă de cea mai bună furnică de la iterația curentă. Această cantitate de feromoni este data de:

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{L_{best}} \text{ , } dacă\ furnica\ trece\ prin\ arcul\ ij \\ \\ 0 \text{ , } în\ caz\ contrar \end{cases},$$

unde L_{best} reprezintă costul drumului construit de cea mai bună furnică.

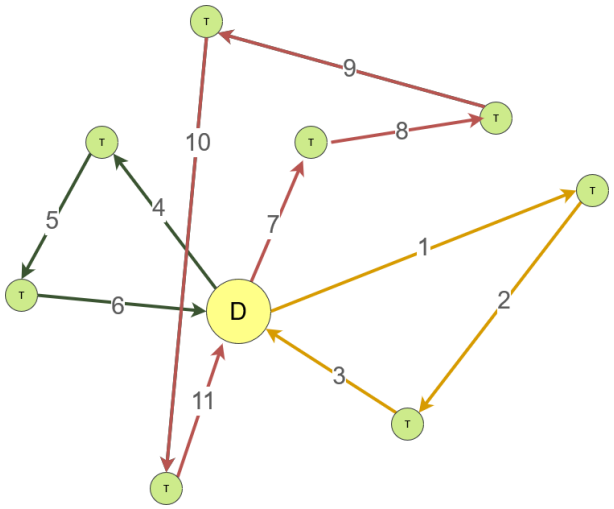

Fig. 2: O posibilă soluție pentru unul dintre subgrafurile problemei.
Arcele sunt numerotate în ordinea în care sunt parcurse de furnică.

Pentru a putea crea mai multe circuite în fiecare subgraf, o furnică are, la fiecare pas constructiv, nodul depou (cel de la care au plecat) în mulțimea vecinilor nevizitați. Astfel, de fiecare dată când furnica se întoarce în depou, se va considera finalul unui circuit care implică utilizarea unui vehicul.

Însă, cum nicio furnică nu este restricționată cu privire la numărul de vizite pe care le poate face în depou, după construirea întregii soluții pentru subgraful curent, numărul de vehicule utilizate poate depăși capacitatea maximă de stocare a depoului. Pentru a evita construirea unor astfel de soluții, de fiecare dată când o furnică construiește o soluție nefezabilă din punctul de vedere al capacității depoului, se reiterează pașii de construcție pentru a obține o altă soluție.

Pe lângă această reiterare, pentru a reduce totuși numărul de mașini utilizate, după obținerea fiecărei soluții (fezabile sau nefezabile) se aplică o euristică de reducere a numărului de vehicule utilizate. Această euristică implică căutarea de noduri fezabil adiacente care au, în soluția obținută, arc direct cu depoul. Dacă un nod care intra în depou este adiacent (conform matricii costurilor din problemă) cu un nod care iese din depou, atunci se elimină din soluție arcele acestora către și dinspre depou și se adaugă arc de la nodul care intra în depou către nodul care ieșea din depou. Așadar, dacă există astfel de perechi de noduri fezabil adiacente, atunci aplicarea euristicii descrise va reduce numărul de vehicule utilizate cu o unitate pentru fiecare pereche de noduri găsită și are potențialul de a repara unele soluții nefezabile obținute la finalul pașilor constructivi ai unei furnici.

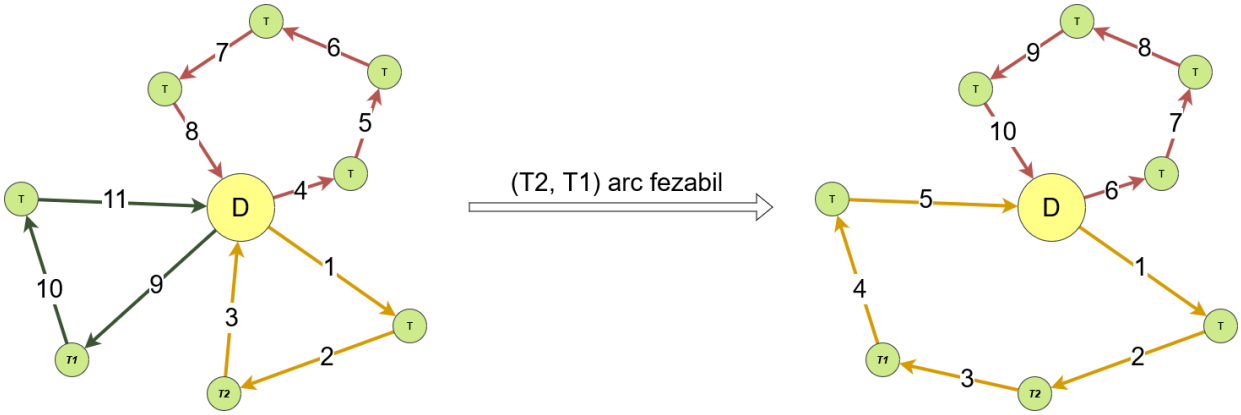

Fig. 3: Euristica de reducere a numărului de vehicule utilizate prin unirea circuitelor în vecinătatea depoului.

Pentru a obține o soluție integrală la problema MDVSP, pentru fiecare subgraf obținut din partiționare, se executa câte o iterație a unei colonii (ACS) iar apoi se asamblează cele mai bune soluții parțiale obținute de fiecare colonie la iterația curentă.

2) ACS aplicat pe întregul graf al problemei: Algoritmul descris anterior, poate fi modificat și aplicat pe întregul graf, fără a se mai partiționa, astfel:

1) Inițial, se alege un depou start, folosind probabilități proporționale cu numărul de vehicule disponibile în fiecare depou;
2) Se construiește un circuit cu ajutorul unei furnici, ignorând la fiecare pas nodurile vecine de tip depou care sunt diferite de depoul inițial;
3) Odată întoarsă în depoul de start, se elimină un vehicul din cele disponibile din acest depou și se generează un număr aleatoriu t_k.
4) Dacă $t_k \leq T$, unde T este un parametru dat de utilizator, atunci se alege un nou depou, în aceeași manieră (deci există o probabilitate să fie ales același depou), în caz contrar, se continuă cu același depou;
5) În cazul în care, la selecția unui nou depou, s-a depășit numărul de vehicule disponibile din fiecare depou, atunci se alege depoul cu numărul cel mai mic de vehicule peste capacitatea sa maxima;
6) Pașii anteriori se iterează, până când se obține o soluție completă;
7) Pe soluția obținută, se aplică euristica de reducere a numărului de vehicule utilizate (descrisă în abordarea anterioară) pe fiecare depou.

Dacă algoritmul descris mai sus nu oferă o soluție fezabilă, atunci acesta este executat din nou, până când soluția obținută este fezabilă.

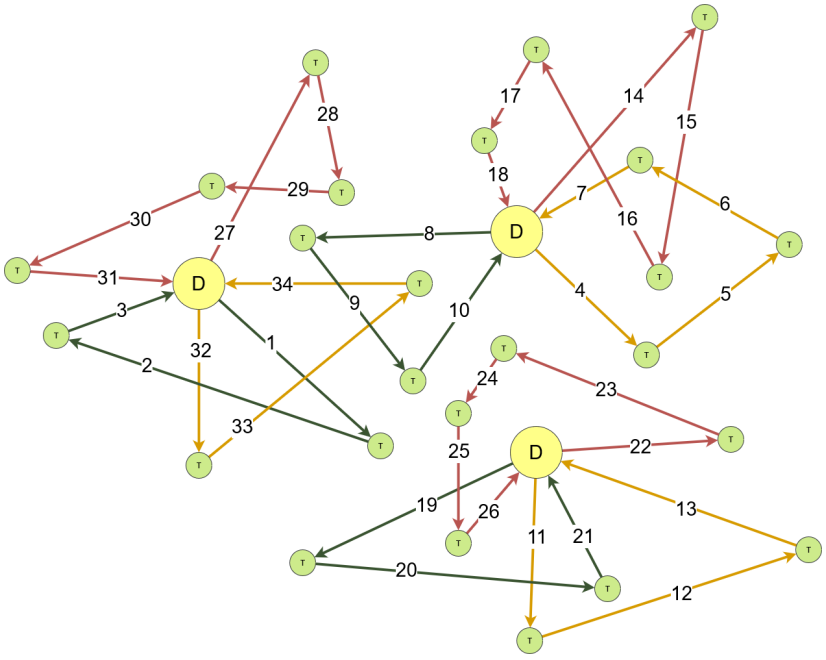

Fig. 4: O posibilă soluție pentru întregul graf al problemei. Arcele sunt numerotate în ordinea în care sunt parcurse de furnică. În acest caz, furnica poate alege să sară de la un depou la altul atunci când încheie un circuit.

3) Euristica de reducere a vehiculelor generalizată: În secțiunile anterioare am descris o euristică de reducere a vehiculelor prin unirea a două circuite la contactul cu depoul. Totuși, două circuite pot fi unite și într-o manieră complexă:

1) Se parcurge fiecare arc din circuit și se caută un arc cu proprietatea: nodul incident interior este adiacent exterior cu primul nod trip din celălalt circuit iar nodul incident exterior este adiacent interior cu ultimul nod trip din celălalt circuit.
2) Dacă se găsește un astfel de arc în circuit, atunci se pot uni cele două circuite, ca în cele două figuri de mai jos.

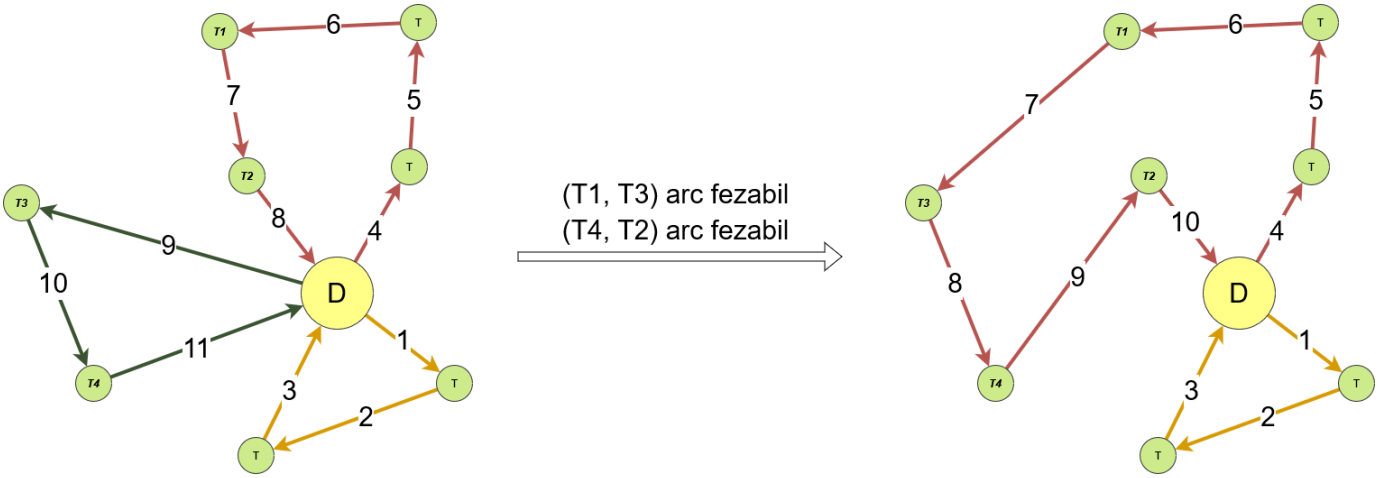

Fig. 5: Euristica generalizată de reducere a numărului de vehicule utilizate prin unirea circuitelor de la același depou.

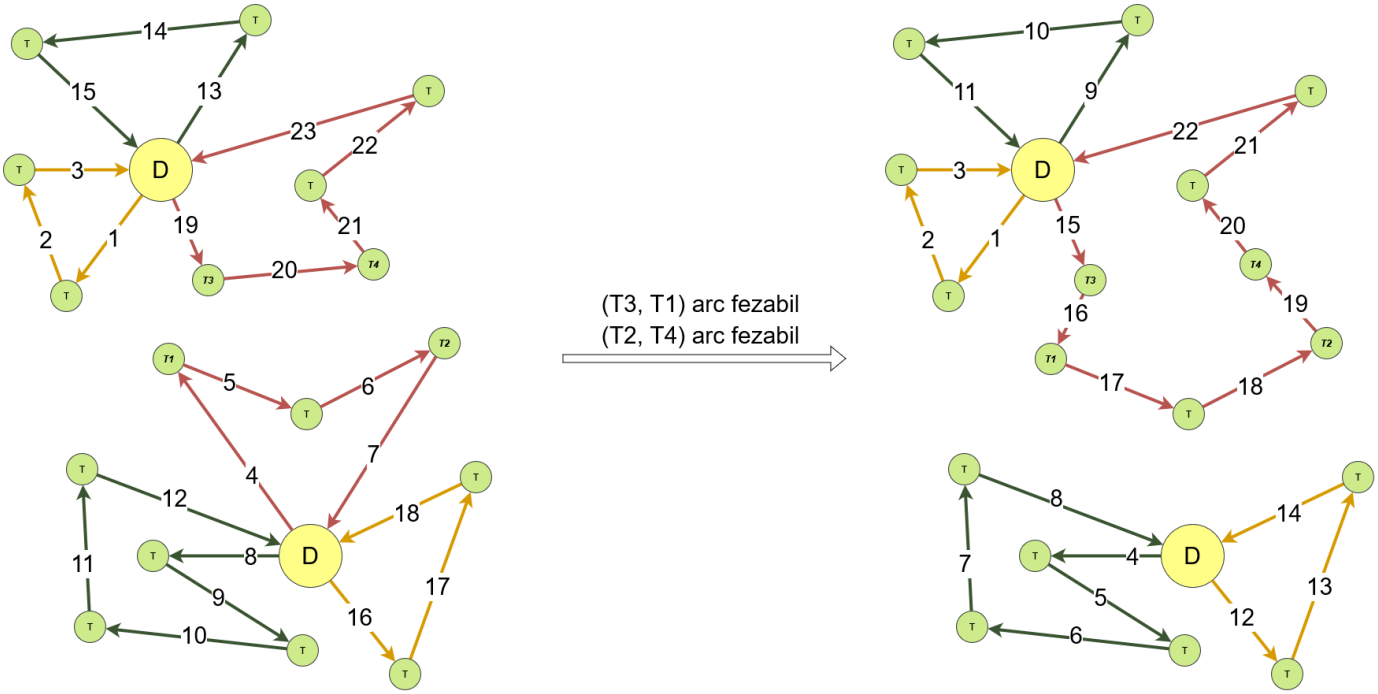

Fig. 6: Euristica generalizată de reducere a numărului de vehicule utilizate prin unirea circuitelor de la depouri diferite.

După cum se poate observa, euristica generalizată poate fi aplicată unind și circuite de la depouri diferite, fără a utiliza un vehicul nou din depoul la care se asignează circuitul rezultat. De asemenea, dacă la pasul 1 al euristicii se iau în considerare și arcele care unesc depoul de primul și ultimul nod trip din circuit, atunci algoritmul ia în calcul și cazurile inițiale de unire a circuitelor lângă depou.

În cazul nostru, euristica generalizată este aplicată iterativ pentru fiecare circuit existent inițial sau obținut după o anumită iterație, până când nu se mai poate uni niciun circuit. Pentru a favoriza unirea circuitelor cu număr mic de noduri, la fiecare iterație a euristicii se sortează lista crescător după numărul de noduri din circuit.

B. Particle Swarm Optimization

PSO a fost propus pentru prima oară de Kennedy și Eberhart [7] în 1995. În procedura PSO, soluțiile sunt modelate ca un roi de particule împrăștiate în spațiul de soluție. Locația fiecărei particule reprezintă o soluție candidată la problemă. La fiecare nouă iterație a metodei, particulele se deplasează cu o anumită viteză într-o nouă regiune a spațiului de soluții pentru a căuta soluții noi și îmbunătățite. În efectuarea mișcării particulelor, trei vectori sunt folosiți ca referință, și anume, vectorul de viteză (velocity) al particulei, cea mai bună locație pe care a reușit particula să o viziteze până acum (adică soluția optimă individuală), și cea mai bună locație vizitată de oricare dintre particulele din roi până în prezent (adică soluția optimă globală). Să presupunem că n particule sunt împrăștiate într-un spațiu de soluții de dimensiune L. Roiul de particule poate fi astfel reprezentat ca $X_i = \{X_1, X_2, ..., X_n\}$ unde locația fiecărei particule este dată de $X_{ij} = \{X_{i1}, X_{i2}, ..., X_{iL}\}$, X_{ij} referindu-se la a j-a componentă de locație a particulei i. În plus, vectorul viteză al fiecărei particule este reprezentat de $V_{ij} = \{V_{i1}, V_{i2}, ..., V_{iL}\}$, soluția locală a particulei este $p_{ij} = \{p_{i1}, p_{i2}, ..., p_{iL}\}$ și soluția globală a roiului este $g = \{g_1, g_2, ..., g_L\}$. Procesul de găsire a soluției în PSO este formulat astfel:

$$v_{ij}^{new} = v_{ij} * w + c_1 * r_1 * (p_{ij} - x_{ij}) + c_2 * r_2 * (g_j - x_{ij})$$
$$x_{ij}^{new} = x_{ij} + v_{ij}^{new}$$

unde, v^{new} se referă la viteza actualizată a particulei și x^{new} indică noua locație a particulei după aplicarea vitezei. În plus, c_1 și c_2 sunt factorii de învățare, iar r_1 și r_2 sunt numere aleatorii cu valori variind de la 0 la 1. Acestea sunt folosite pentru a genera diverse direcții de mișcare ale particulelor astfel încât particulele să poată explora noi regiuni ale spațiului de soluții.

1) Schema de codificare a particulelor:

PSO-ul este aplicat direct pentru a rezolva problema deja descrisă MDVSP. În implementarea metodei, schema de codificare a particulelor propusă este concepută astfel încât să poată rezolva toate cele trei subprobleme a MDVSP-ului în același timp: "Ce depou ar trebui alocat fiecărui client în parte?", "Ce vehicul ar trebui să ajungă la clienții alocați depoului?" și "În ce ordine ar trebui să fie vizitați clienții alocați fiecărui vehicul în parte?". Având la dispoziție D depouri și n clienți la care trebuie să se ajungă, rezultă împărțirea clienților în D grupuri. În plus, dacă presupunem la fiecare depou sunt disponibile m vehicule, clienții deserviți de un anumit depou sunt împărțiți în m grupuri. Cu alte cuvinte, rezolvarea problemei implică împărțirea clienților în grupuri de dimensiune $D * m$. Deci, MDVSP este rezolvată folosind o schemă extinsă de codificare a particulelor care ia în considerare $D * m - 1$ puncte de segmentare în plus față de cele n componente corespunzătoare celor n clienți. Rezultă că fiecare particulă codificată conține $n + D * m - 1$ componente, unde $X = \{x_1, ..., x_n, x_{n+1}, ..., x_{n+D*m-1}\}$

Particulele sunt decodificate folosind o metodă bazată pe cheii aleatorii. În schema de codificare, particula X corespunde unui set de valori cheie, mai exact $KEY = \{c_1, c_2, ..., c_n, S_{11}, ..., S_{1m}, ..., S_{D1}, ..., S_{D(m-1)}\}$, în care S_{ij} semnifică valoarea cheie a segmentării pentru al j-lea vehicul din depoul i. În continuare, X este sortat în ordine crescătoare, iar valorile corespunzătoare cheilor de decodare sunt rearanjate și ele odată cu sortarea. După sortarea lui X, punctele de segmentare definesc nu numai ce vehicul din fiecare depou vizitează ce clienți dar și succesiunea în care acești clienți sunt vizitați. De remarcat faptul că clienții de la sfârșitul secvenței sortate (adică clienții care se află după ultimul punct de segmentare) sunt pur și simplu alocați ultimului vehicul a ultimului depou.

i	1	3	…	*n-1*	*n*	*n+1*	...	*n+(D×m-1)*
Key	C_1	C_2	...	C_{n-1}	C_n	S_{11}	...	$S_{D(m-1)}$
X	X_1	X_2	...	X_{n-1}	X_n	X_{n+1}	...	$X_{n+(D\times m-1)}$

Fig. 7: Codificare

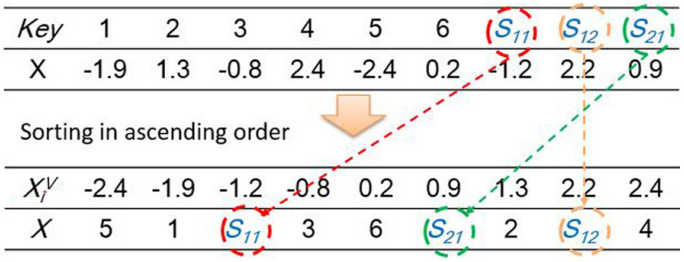

Fig. 8: Sortare

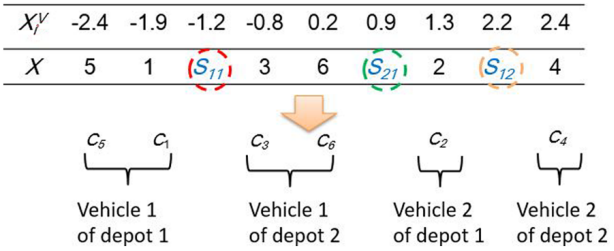

Fig. 9: Decodificare

2) Metodă euristică de generare soluție inițială:

Soluțiile inițiale aleatorii determină ca algoritmul PSO să caute în spațiul soluțiilor într-o manieră destul de imprecisă și drept consecință algoritmul va converge foarte lent. Pe cealaltă parte, soluțiile inițiale bune limitează spațiul soluțiilor și permite ca metoda să conveargă mai rapid spre soluția optimă. Drept urmare, se propune o euristică ca mecanism de obținere a unei anumit număr de soluții inițiale fezabile pentru PSO. Așa cum este scris mai sus, MDVSP presupune rezolvarea a trei sub-probleme, și anume gruparea clienților la depouri, determinarea vehiculelor din fiecare depou care ar trebui să deservească anumiți clienți și stabilirea ordinii în care fiecare vehicul ar trebui să viziteze clienții alocați. Prin urmare, în căutarea soluțiilor inițiale adecvate pentru MDVSP, abordarea euristică începe prin gruparea clienților folosind o metodă similară cu cea folosită la metoda ACS, adică fiecărui client i se atribuie un depou în urma partiționării bazate pe gradul fiecărui nod de tip trip.

Pentru a realiza o grupare inițială mai diversă a clienților, clienții sunt repartizați la anumite depouri folosind fie o metodă greedy, fie la întâmplare. Alegerea uneia dintre cele 2 metode se realizează folosind valoarea probabilității de grupare gp. Mai exact, o valoare aleatorie este generată în intervalul $[0, 1]$, iar metoda greedy este aleasă dacă valoarea este mai mică decât probabilitatea gp, altfel alegându-se metoda aleatoare. Valoarea lui gp este 0,8, adică metoda greedy este aleasă cu o probabilitate de 80%, în timp ce cealaltă metodă este aleasă cu o probabilitate de 20%. După gruparea clienților la depouri, se folosește algoritmul "savings" [8] pentru fiecare depou pentru a găsi o atribuire fezabilă între clienți și vehiculele disponibile depoului rezultând astfel mai multe trasee care respectă constrângerile problemei MDVSP.

Dacă toate soluțiile inițiale sunt generate de metoda euristică propusă, algoritmul va converge rapid, dar există o șansă destul de mare să conveargă la o soluție locală suboptimală. În consecință, parametrul ratei euristice inițiale IHR prezent în ecuația de mai jos este utilizat pentru a determina cât de mult să fie folosită metoda euristică propusă la pasul de inițializare a PSO-ului. Variabila $rand$ din ecuație se referă la o valoare generată aleatoriu în intervalul $[0, 1]$.

$$soluție\ inițială = \begin{cases} generare\ euristică & : dacă\ rand < IHR \\ generare\ aleatoare & : altfel \end{cases}$$

3) Ajustare pondere inerție:

În mod ideal, spațiul soluțiilor la începutul metodei ar trebui să fie cât mai mare cu putință și mai restrâns pe măsură ce regiunea fezabilă a soluțiilor este găsită. Cu alte cuvinte, procesul de căutare ar trebui să evolueze treptat de la un comportament de căutare de tip explorare globală la un comportament de căutare de tip exploatare locală pe măsură ce numărul de iterații crește. Metoda implementată ia în considerare metoda curbei sigmoide prezentată în ecuația de mai jos, în care i este numărul iterației curente și i_{max} este numărul maxim de iterații.

$$w_i = \frac{w_{start} - w_{end}}{1 + e^{-u*(i-n*i_{max})}} + w_{end}$$
$$u = 10^{log(i_{max})-2}$$

unde u este constanta care reglează precizia funcției sigmoid, iar n este constanta care setează partiția aceleiași funcții.

IV. Experimente

A. Descrierea instanțelor

Pentru a testa metodele descrise în secțiunea anterioară, am ales să utilizăm instanțele MDVSP din [13]. Aceste conțin niște grafuri cu un număr de depouri $m \in \{4, 8\}$ și 500 de noduri de tip trip, fiind descrise de câte o matrice a costurilor. Această matrice este, de fapt, o matrice de adiacență, în care primele m noduri sunt depouri, arcele nefezabile sunt marcate cu -1, iar cele fezabile sunt marcate cu un număr pozitiv (care reprezintă costul arcului).

B. Parametrii algoritmilor utilizați

TABLE I: Parametri ACS

număr de iterații	50
număr de furnici	10
τ_0	1.0
α	1.0
β	2.0
ρ	0.1
ϕ	0.1
q_0	0.5
T	0.5

TABLE II: Parametri PSO

număr de iterații	1000
dimensiune roi particule	40
viteză minimă (v_{min})	-6
viteză maximă (v_{max})	6
pondere inerție început (w_{start})	0.6
pondere inerție sfârșit (w_{end})	0.2
primul factor accelerare (c_1)	2.05
al doilea factor accelerare (c_2)	2.05
rată euristică inițială (IHR)	0.3
probabilitate grupare (gp)	0.8
n	0.75

C. Rezultate experimentale

Instanțe	Cea mai bună soluție cunoscută	30 rulări				
		Minim	Maxim	Medie	Mediană	Deviație Standard
m4n500s0.inp	1 289 114	1 882 071	1 929 922	1 907 471,87	1 908 397,0	10 084,15
m4n500s1.inp	1 241 618	1 857 760	1 905 835	1 882 290,9	1 880 777,0	11 720,31
m4n500s2.inp	1 283 811	1 825 920	1 869 951	1 849 791,6	1 849 369,5	10 054,28
m4n500s3.inp	1 258 634	1 844 700	1 882 980	1 862 938,1	1 864 602,5	9 266,48
m4n500s4.inp	1 317 077	1 891 821	1 932 061	1 917 115,43	1.918.072	9 997,04
m8n500s0.inp	1 292 411	1 892 534	1 937 146	1 918 195,6	1 916 585,5	11 893,56
m8n500s1.inp	1 276 919	1 879 640	1 918 470	1 899 107,27	1 898 847,5	10 728,11
m8n500s2.inp	1 304 251	1 886 424	1 925 907	1 911 267,93	1 913 451,5	8 911,84
m8n500s3.inp	1 277 838	1 911 357	1 967 082	1 943 364,53	1 941 777,5	13 352,08
m8n500s4.inp	1 276 010	1 923 538	1 966 176	1 949 359,6	1 952 653,0	11 520,48

TABLE III: Rezultate ACS pe graf partiționat

Instanțe	Cea mai bună soluție cunoscută	30 rulări				
		Minim	Maxim	Medie	Mediană	Deviație Standard
m4n500s0.inp	1 289 114	1 752 124	1 808 325	1 792 976,2	1 795 360,0	12 091,06
m4n500s1.inp	1 241 618	1 746 828	1 793 445	1 772 177,17	1 775 394,0	11 963,11
m4n500s2.inp	1 283 811	1 717 714	1 764 185	1 744 799,5	1 747 290,5	10 973,61
m4n500s3.inp	1 258 634	1 742 220	1 781 741	1 763 072,3	1 762 813,0	9 833,39
m4n500s4.inp	1 317 077	1 782 496	1 841 550	1 818 209,6	1 822 598,5	14 196,33
m8n500s0.inp	1 292 411	1 804 677	1 878 045	1 849 191,53	1 853 831,5	17 102,97
m8n500s1.inp	1 276 919	1 806 521	1 855 264	1 830 007,6	1 828 883,0	12 248,77
m8n500s2.inp	1 304 251	1 785 585	1 837 422	1 814 465,3	1 815 727,0	14 306,07
m8n500s3.inp	1 277 838	1 828 106	1 871 483	1 855 759,23	1 856 913,5	10 716,53
m8n500s4.inp	1 276 010	1 790 573	1 847 647	1 829 893,0	1 829 553,5	12 297,62

TABLE IV: Rezultate ACS pe întregul graf (euristica simplă - ES - de reducere lângă depouri)

Instanțe	Cea mai bună soluție cunoscută	30 rulări				
		Minim	Maxim	Medie	Mediană	Deviație Standard
m4n500s0.inp	1 289 114	1 650 983	1 689 330	1 669 779,47	1 667 607,0	9 957,24
m4n500s1.inp	1 241 618	1 612 548	1 667 700	1 647 177,1	1 649 374,0	12 845,93
m4n500s2.inp	1 283 811	1 597 376	1 638 356	1 622 559,77	1 624 104,5	9 393,92
m4n500s3.inp	1 258 634	1 582 628	1 635 091	1 621 051,27	1 622 678,0	10 541,13
m4n500s4.inp	1 317 077	1 676 982	1 708 634	1 692 345,97	1 691 737,5	8 939,85
m8n500s0.inp	1 292 411	1 612 601	1 643 718	1 629 913,67	1 630 847,5	8 426,52
m8n500s1.inp	1 276 919	1 569 668	1 619 552	1 601 974,17	1 603 688,0	11 553,42
m8n500s2.inp	1 304 251	1 600 332	1 637 124	1 618 260,03	1 619 309,0	9 594,33
m8n500s3.inp	1 277 838	1 632 736	1 665 951	1 648 850,1	1 648 453,0	8 683,61
m8n500s4.inp	1 276 010	1 597 456	1 637 364	1 623 609,67	1 625 706,0	9 774,076

TABLE V: Rezultate ACS pe întregul graf (euristica generalizată - EG - de reducere lângă depouri)

Instanțe	Cea mai bună soluție cunoscută	30 rulări				
		Minim	Maxim	Medie	Mediană	Deviație Standard
m4n500s0.inp	1 289 114	2 097 166	2 147 467	2 123 793,0	2 126 746	20 641,18
m4n500s1.inp	1 241 618	1 998 201	2 091 014	2 031 471,66	2 005 200	42 199,63
m4n500s2.inp	1 283 811	1 922 508	2 015 672	1 976 068,33	1 990 025	39 293,54
m4n500s3.inp	1 258 634	2 032 131	2 056 820	2 048 497,66	2 056 542	11 573,54
m4n500s4.inp	1 317 077	1 982 390	2 017 332	1 998 835,66	1 996 785	14 338,52
m8n500s0.inp	1 292 411	2 056 152	2 077 731	2 067 106,0	2 067 435	8 812,66
m8n500s1.inp	1 276 919	2 063 100	2 088 441	2 075 172,33	2 073 976	10 379,95
m8n500s2.inp	1 304 251	2 044 180	2 083 195	2 062 448,33	2 059 970	16 023,92
m8n500s3.inp	1 277 838	2 112 541	2 135 535	2 121 903,66	2 117 635	9 860,6
m8n500s4.inp	1 276 010	2 064 807	2 094 427	2 084 087,33	2 093 028	13 645,21

TABLE VI: Rezultate PSO

D. Teste statistice

1) Statistici One Way : Testul Kruskal-Wallis:

Valoare statistică = 36.44049
$p_{value} = 0.000000060 < 0.05$

Ipoteza nulă (mediana este egală pentru toate metodele) este respinsă.

Testul Dunn (post-hoc)

	ACS Graf Partiționat	ACS Graf Întreg ES	ACS Graf Întreg EG	PSO
ACS Graf Partiționat	1.0	0.365189	0.000846	0.320263
ACS Graf Întreg ES	0.365189	1.0	0.320263	0.000846
ACS Graf Întreg EG	0.000846	0.320263	1.0	5.741609e-08
PSO	0.320263	0.000846	5.741609e-08	1.0

TABLE VII: Valori p_{value} obținute la testul Dunn

Între metodele ACS Graf Partiționat și ACS Graf Întreg EG există o diferență statistică semnificativă cu un

$$p_{value} = 0.000846 \ .$$

Între metodele ACS Graf Întreg ES și PSO există o diferență statistică semnificativă cu un

$$p_{value} = 0.000846 \ .$$

Între metodele ACS Graf Întreg EG și PSO există o diferență statistică semnificativă cu un

$$p_{value} = 5.741609 \cdot 10^{-8} \ .$$

2) Statistici Two Way : Testul Friedman bazat pe Distribuția Chi Square:

Valoare statistică = 30.00000
$p_{value} = 0.000001380 < 0.05$

Ipoteza nulă (fiecare clasare a metaeuristicilor în cadrul fiecărei instanțe este la fel de probabilă) este respinsă.

Testul Nemenyi Friedman (post-hoc)

	ACS Graf Partiționat	ACS Graf Întreg ES	ACS Graf Întreg EG	PSO
ACS Graf Partiționat	1.0	0.30713	0.00299	0.30713
ACS Graf Întreg ES	0.30713	1.0	0.30713	0.00299
ACS Graf Întreg EG	0.00299	0.30713	1.0	0.00100
PSO	0.30713	0.00299	0.00100	1.0

TABLE VIII: Valori p_{value} obținute la testul Nemenyi Friedman

Între metodele ACS Graf Partiționat și ACS Graf Întreg EG există o diferență statistică semnificativă cu un

$$p_{value} = 0.00299 \ .$$

Între metodele ACS Graf Întreg ES și PSO există o diferență statistică semnificativă cu un

$$p_{value} = 0.00299 \ .$$

Între metodele ACS Graf Întreg EG și PSO există o diferență statistică semnificativă cu un

$$p_{value} = 0.001 \ .$$

Paired Comparison [14] (post-hoc)

	ACS Graf Partiționat	ACS Graf Întreg ES	ACS Graf Întreg EG	PSO
ACS Graf Partiționat	0.0	10.0	20.0	10.0
ACS Graf Întreg ES	10.0	0.0	10.0	20.0
ACS Graf Întreg EG	20.0	10.0	0.0	30.0
PSO	10.0	20.0	30.0	0.0

TABLE IX: Valori diferență obținute la Paired Comparison

Între metodele ACS Graf Partiționat și ACS Graf Întreg ES există o diferență statistică semnificativă cu o $valoare\ diferență = 10$.

Între metodele ACS Graf Partiționat și ACS Graf Întreg EG există o diferență statistică semnificativă cu o $valoare\ diferență = 20$.

Între metodele ACS Graf Partiționat și PSO există o diferență statistică semnificativă cu o $valoare\ diferență = 10$.

Între metodele ACS Graf Întreg ES și ACS Graf Întreg EG există o diferență statistică semnificativă cu o $valoare\ diferență = 10$.

Între metodele ACS Graf Întreg ES și PSO există o diferență statistică semnificativă cu o $valoare\ diferență = 20$.

Între metodele ACS Graf Întreg EG și PSO există o diferență statistică semnificativă cu o $valoare\ diferență = 30$.

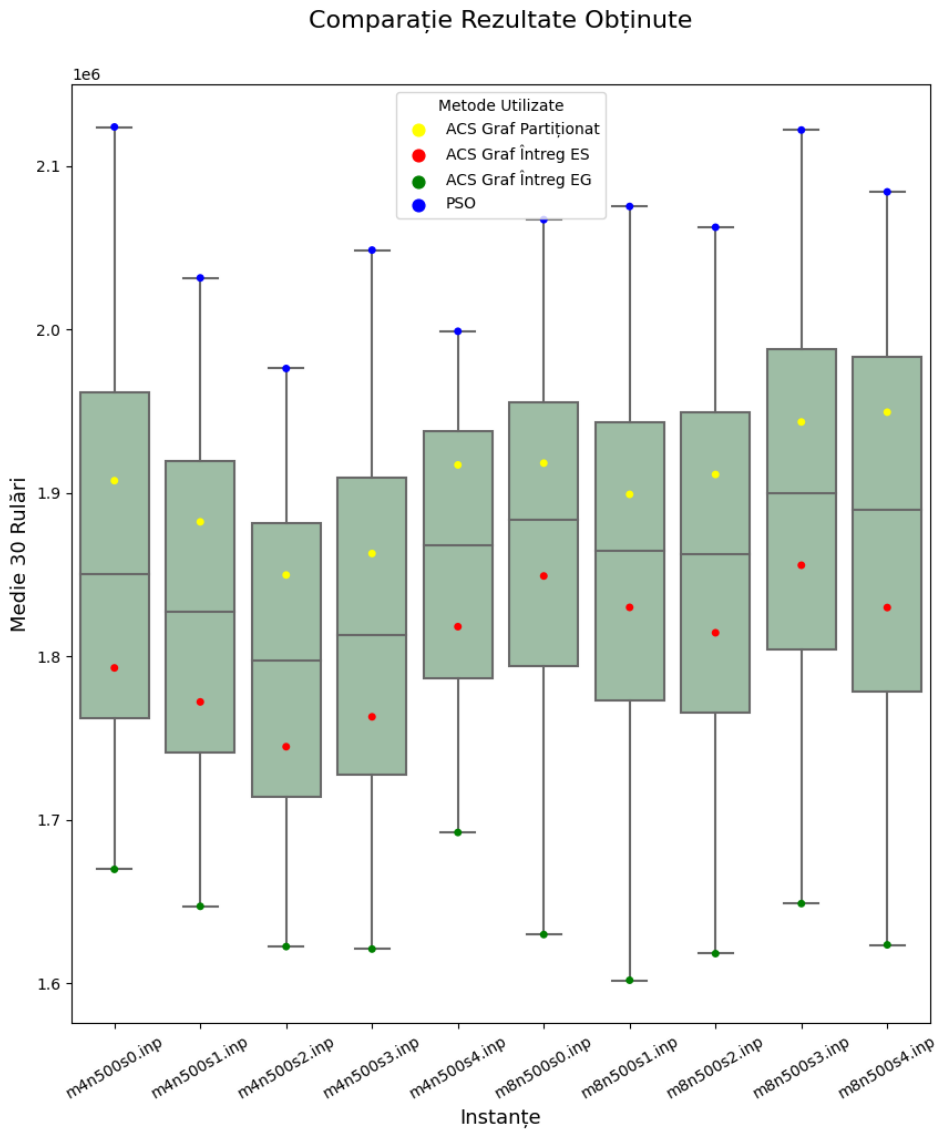

Fig. 10

V. Concluzii

Așadar, după cum s-a putut vedea în rezultatele experimentale, metodele propuse nu reușesc să se apropie foarte mult de cea mai bună soluție cunoscută pentru instanțele utilizate. O posibilă cauză a acestui rezultat o constituie parametrii algoritmilor, care ar putea să nu fie corespunzători pentru MDVSP.

References

[1] Demirel, Tufan & Yılmaz, Şule. (2012). A New Solution Approach To Multi-Depot Vehicle Routing Problem With Ant Colony Optimization. Journal of multiple-valued logic and soft computing. 18. 421-439.

[2] el Bouyahyiouy, Karim & Bellabdaoui, Adil. (2017). An ant colony optimization algorithm for solving the full truckload vehicle routing problem with profit. 142-147. 10.1109/LOGISTIQUA.2017.7962888.

[3] Bertossi, Alan A., Paolo Carraresi, and Giorgio Gallo. "On some matching problems arising in vehicle scheduling models." Networks 17.3 (1987): 271-281.Bertossi, Alan A., Paolo Carraresi, and Giorgio Gallo. "On some matching problems arising in vehicle scheduling models." Networks 17.3 (1987): 271-281.

[4] Milovanovic, Nemanja. (2015). Solving the multiple-depot vehicle scheduling problem. 10.13140/RG.2.1.5057.8804.

[5] Laurent, Benoît, and Jin-Kao Hao. "Iterated local search for the multiple depot vehicle scheduling problem." Computers & Industrial Engineering 57.1 (2009): 277-286.

[6] Pepin, Ann-Sophie, et al. Comparison of heuristic approaches for the multiple depot vehicle scheduling problem. No. EI 2006-34. 2006.

[7] Eberhart, Russell, and James Kennedy. "Particle swarm optimization." Proceedings of the IEEE international conference on neural networks. Vol. 4. 1995.

[8] Lysgaard, Jens. "Clarke Wright's savings algorithm." Department of Management Science and Logistics, The Aarhus School of Business 44 (1997).

[9] Bunte, Stefan & Kliewer, Natalia. (2010). An overview on vehicle scheduling models. Public Transport. 1. 299-317. 10.1007/s12469-010-0018-5.

[10] Fischetti, Matteo & Lodi, Andrea & Toth, Paolo. (1970). A Branch-and-Cut Algorithm for the Multiple Depot Vehicle Scheduling Problem.

[11] www.scholarpedia.org/article/Ant_colony_optimization

[12] https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

[13] https://personal.eur.nl/huisman/instances.htm

[14] https://juangvillegas.files.wordpress.com/2011/08/friedman-test-24062011.pdf