MATEMATIČKI FAKULTET

Univerzitet u Beogradu

MASTER RAD

Raca Todosijević

1018/10

Mentor:

prof. dr Đorđe Dugošija

Beograd, 2011.

Nova heuristika za rešavanje problema trgovačkog putnika bazirana na metodi promena okolina

1.Problem trgovačkog putnika

Problem trgovačkog putnika je jedan od najznačajnijih problema *kombinatorne optimizacije.*

Problem trgovačkog putnika sastoji se u sledećem:
Pretpostavimo da trgovački putnik želi da obiđe izvestan broj gradova.Postavlja se pitanje:
Kojim redosledom trgovački putnik treba da obilazi gradove, tako da svaki grad obiđe jedanput, da se po obilasku svih gradova vrati u početni grad i da pri tome predje najmanje moguće rastojanje?

Neka trgovački putnik treba da obidje *n* gradova.
Kada zeli da napusti početni čvor trgovački put se susreće sa dilemom u koji od (n-1) gradova da se uputi.
U sledećem koraku trgovački putnik treba da se odluči za jedan od preostalih (n-2) gradova, itd.
Ukupan broj različitih tura koje može da generiše trgovački putnik jednak je (n-1)* (n-2)...3*2*1=(n-1)!.U slučaju da ukupna dužina rute ne zavisi od smera kretanja trgovačkog putnika broj različitih tura je (n-1)!/2.

Matematički problem se može formulisati na sledeci način:

Neka je data natrica cena(dužina) C gde je c[i][j] cena putovanja od grada i do grada j. Naći permutaciju (i_1, i_2, i_3, ..., i_n) brojeva od 1 do n tako da je suma c[i_1][i_2] + c[i_2][i_3] + ... + c[i_{n-1}][i_n] minimalna.

Na osnovu strukture matrice C može se izvršiti podela problema na sledeće podprobleme:

1) ako je c[i][j]= c[j][i] tada je matrica C simetrična pa se problem naziva *simetrični TSP(STSP)*,u suprotnom to je *asimetrični TSP*.
2) Ako za svako i,j,k važi c[i][j]+ c[j][k]<= c[i][k] (važi nejednakost trougla) tada se radi o *metrickom TSP*.
3) Ako je c[i][j] euklidsko rastojanje između gradova i i j onda je to *euklidski TSP*.Očigledno euklidski TSP je ujedno i simetrični i metrički TSP.

Primene TSP-a:

Sa problemom trgovačkog putnika susrećemo se pri vršenju distribucije, sakupljanja ili neke druge opsluge na transportnoj mreži.
U različitim problemima saobraćaja i transporta pod trgovačkim putnikom mogu da se podrazumevaju autobusi, kamioni, brodovi, avioni, posade itd.

U čvorovima koje posećuju saobraćajna sredstva mogu se isporučivati ili preuzimati roba ili putnici, ili se, na primer, istovremeno može i isporučivati i preuzimati roba.

Irski matematičar Sir William Rowan Hamilton (1805-1865) i britanski matematičar Thomas Kirkman (1806-1895) su se bavili većim brojem matematičkih problema koji su značajni za problem trgovačkog putnika.
Karl Menger, Hassler Whitney, Merrill Flood, Mahalanobis, Jessen, Gosh i Marks su dali značajne doprinose formulisanju i proučavanju problema trgovačkog putnika u prvoj polovini prošlog veka. Karl Menger je formulisao problem i razmatrao rešavanje problema isputivanjem svih mogućih tura. Whitney je dao ime problemu. Flood se bavila nalazenjem optimalne rute za školski autobus u Nju Džersiju. Mahalanobis, Jessen, Gosh i Marks su razmatrali primenu TSP-a u poljoprivredi.
Dantzig, Fulkerson i Johnson su 1954. godine rešili problem trgovačkog putnika koji je imao 49 gradova korišćenjem metode celobrojnog linearnog ptogramiranja – metode odsecajućih ravni.
Pedeset godina kasnije (2004. godine), D. Applegate, R. Bixby, V. Chvátal, W. Cook i K. Helsgaun su uspeli da reše problem trgovačkog putnika sa 24978 gradova pomoću LKH algoritma[1]. Dokaz optimalnosti nađene ture je izveden pomoću programa „Concorde".

Neke varijante TSP-a:

1) Uopšteni problem trgovačkog putnika (GTSP) je varijanta TSP-a u kome su gradovi podeljeni u klastere, a trgovački putnik treba da poseti najmanje jedan grad iz svakog klastera tako da put koji pređe tom prilikom bude najkraći.
2) Problem trgovačkog putnika sa profitom je varijanta TSP-a gde je svakom čvoru pridružena težina(profit). Trgovački putnik ne mora da obiđe sve gradove, ali mora da optimizuje pređeni put i ostvareni profit.
3) Problem putujućeg kupca(TPP). Kupac ima zadatak da kupi određeni broj artikala putujući od grada do grada. U svakom gradu se prodaje izvestan broj artikala po unapred poznatoj ceni. Cilj putujućeg kupca je da odabere podskup gradova tako da kupi sve artikle, koji su mu neophodni i da pritom pređe put minimalne dužine i potroši minimalan iznos novca.
4) Max-TSP. Za razliku od klasičnog TSP-a gde se traži tura najmanje dužine, u Max-TSP problemu se traži tura maksimalne dužine.
5) Dinamički TSP je varijanta TSP-a u kome čvorovi nisu fiksni,već se kreću unapred zadatom brzinu. Cilj je obići sve čvorove za najkraće vreme.

2.Algoritmi za rešavanje TSP-a

Uvodni pojmovi:

Neka je G = (N, E) neusmeren težinski graf gde je N = {1, 2, ..., n} skup čvorova i E={(i,j)| i ∈ N, j ∈ N} skup ivica. Svakoj ivici (i,j) je pridružena dužina c(i,j)
Put je skup ivica $\{(i_1,i_2), (i_2,i_3), ..., (i_{k-1},i_k)\}$ gde je $i_p \neq i_q$ za svako $p \neq q$.
Ciklus je skup ivica $\{(i_1,i_2), (i_2,i_3), ..., (i_k,i_1)\}$ gde je $i_p \neq i_q$ za svako $p \neq q$.
Tura je ciklus gde je k = n.
Za svaki skup $S \subseteq E$ definišimo njegovu dužinu L(S) sa $L(S) = \sum_{(i,j)\in S} c(i,j)$.
Optimalna tura je tura najkraće dužine.
Graf G je *povezan* ako sadrži put koji povezuje bilo koja dva čvora iz G.
Stablo je povezani graf koji ne sadrži cikluse. *Razapinjajuće stablo* grafa G koji ima n n čvorova je stablo sa n-1 ivicom iz G .*Minimalno razapinjajuće stablo* je razapinjajuće stablo minimalne dužine.
1-stablo grafa G=(N, E) je razapinjajuće stablo nad skupom N\{1} prošireno sa dve ivice iz skupa E koje su susedne sa čvorom 1.
Minimalno 1-stablo je stablo minimalne dužine.

Vremenska složenost algoritma je broj operacija potrebnih za izvršavanje algoritma u f-ji od veličine ulaznih podataka.

NP problem je problem sa karakteristikom polinomijalne proverljivosti rešenja. Takav je problem trgovačkog putnika jer za svaku turu T i dati realan broj c u polinomijalnom vremenu možemo proveriti da li je dužina ture manja od c.
Problem *X* je *NP-težak* problem ako je svaki problem iz klase NP polinomijalno svodljiv na *X*.
Problem *X* je *NP-kompletan* problem ako pripada klasi NP i ako je *X* NP-težak.
Ričard Karp je 1972. pokazao da problem trgovačkog putnika spada u grupu NP kompletnih problema. Za probleme iz te klase ne postoji polinomijalan algoritam koji dovodi do rešenja problema. Ako bi se za neki NP-težak problem pronašao polinomijalni algoritam tada bi svaki NP problem imao polinomijalni algoritam zato što je svaki NP problem polinomijalno svodljiv na bilo koji NP-težak problem.

Algoritmi za rešavanje TSP-a se dele u dve grupe:

1) tačni algoritmi
2) približni algoritmi(heuristike)

Tačni algoritmi

Najpoznatiji algoritam iz ove grupe je algoritam *grananja i ograničavanja*.
Ovaj algoritam se realizuje na sledeći način:
Prvo se odredi minimalno razapinjajuće 1-stablo u grafu (graf se sastoji od n gradova, a matrica dužina ivica je matrica C).Ako je minimalno razapinjajuće 1-stablo tura onda je kraj. U protivnom postoji čvor čiji je stepen veći od dva.Neka je to čvor v, neka je njegov stepen d i $e_1,...,e_d$ ivice incidentne sa v. Sada se problem grana na d potproblema tako što se u i-tom podproblemu dužina ivice e_i postavlja na beskonačno ,a potom traži minimalno razapinjajuće 1-stablo i ponavlja se analogna procedura.Takođe vrši se i ograničavanje tako da se ne razmatraju oni podproblemi čije rešenje neće biti bolje od trenutno najboljeg rešenja. Rešenje podproblema(tura koja se dobija) neće biti bolje od trenutno najboljeg rešenja(trenutno najbolje ture) ako je dužina minimalnog razapinjajuceg 1-stabla veća od trenutno najboljeg rešenja.
Pored ovog algoritma u ovo grupu algoritama spada algoritam grube sile koji ispitaju sve moguće ture i pronalazi najkraću. Složenost ovog algoritma je O(n!). Zatim u toj grupi je i metod odsecajućih ravni.
Vremenska složenost ovih algoritama nije polinomijalna i oni dovode do tačnog rešenja.

Približni algoritmi

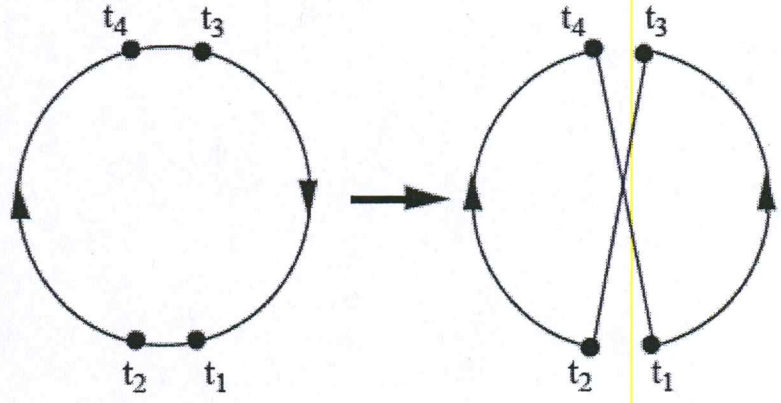
Za razliku od tačnih algoritama ovi algoritmi pronalaze rešenja,ali ne garantuju da će pronaći optimalna rešenja.Ovi algoritmi su jednostavni i imaju relativno malo vreme izvršavanja.Neki od ovih algoritama daju rešenja koja su samo nekoliko procenata iznad optimalnog , u slučajevima kada je optimalno rešenje poznato.

Približni algoritmi se dele u tri klase:

1) "Algoritmi koji konstruišu turu"
2) "Algoritmi koji poboljšavaju turu"
3) "Složeni algoritmi"

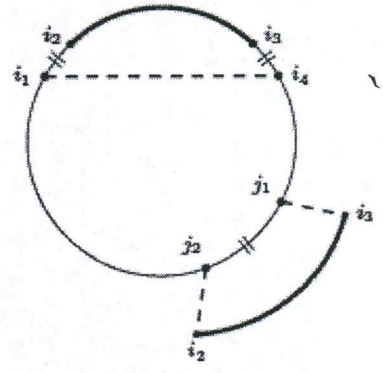
"Algoritmi koji konstruišu turu" spadaju u grupu pohlepnih algoritama. Ovi algoritmi su veoma jednostavni,ali su dobijene ture puno lošije od optimalnih tura.
Najjednostavniji algoritmi ovog tipa je *nearest-neighbor* algoritam.Osnovni koraci ovog algoritma su:
1)Krenuti iz nekog grada
2)Sve dok postoji neposećeni grad iz poslednje posećenog grada posetiti grad koji mu je najbliži
3) Na kraju se vratiti u polazni grad.
" Algoritmi koji poboljšavaju turu" poboljšavaju trenutnu turu primenjujući određene izmene na turi.
Neki od algoritama iz ove klase su:

1) ***2-opt*** algoritam
Ideja ovog algoritma je:zameniti dve grane iz trenutne ture sa drugim dvema granama koje nisu u trenutnoj turi tako da je dobijena tura kraća(ovaj postupak se naziva 2-opt move).Primenjivati ovaj postupak sve dok ima poboljšanja.Kao rezultat rada ovog algoritma dobija se tura koja je 2-opt optimalna,ali ne i globalno optimalna.
Analogno 2-opt algoritmu postoje i k-opt algoritmi(k>=2) koji poboljšavaju trenutnu turu menjajući k grana iz trenutne ture sa drugih k grana koje nisu u trenutnoj turi tako da je dobijena tura kraća.



Slika1:2-opt move

2)***Or-opt*** algoritam radi sledeće:
Najpre se izabere *lanac* gradova ,unapred zadate dužine, iz trenutne ture,a zatim se pokušava da se taj lanac umetne između neka dva grada iz trenutne ture tako da se dobije kraća tura.Pod lancem gradova se podrazumeva zadati broj susednih gradova iz trenutne ture.Primenjivati ovaj postupak dokle god ima poboljšanja.
U zavisnosti od dužine lanca gradova razlikuju se Or-opt algoritmi.Najčešće se koriste Or-opt algoritmi sa lancima dužine 1,2ili3.



Slika2:Or-opt move

3)***2.5-opt*** algoritam je poboljšanje 2-opt algoritma koje se dobija tako sto se dozvoljava da gradovi t1 i t2 u 2-opt move-u mogu biti isti.Odnosno,ovaj algoritam je kombinacija 2-opt algoritma i Or-opt algoritma sa lancem dužine 1.

4)*1-opt* algoritam je pojednostavljeni 2-opt algoritam.On pokušava da popravi trenutnu turu pomoću 2-opt move primenjenog na gradovi t1,t2,t4 i t3 koji se pojavljuju u datom redosledu u trenutnoj turi.

"Složeni algoritmi" kombinuju algoritme koji poboljšavaju turu i algoritme koji konstruišu turu. Ovi algoritmi, u prvom koraku, pronalaze dopustivu turu koristeći algoritam koji konstruiše turu, a zatim poboljšavaju nađenu turu koristeći algoritam koji poboljšava turu. Tako dobijena tura zavisi od izbora početene ture. Jedan od prvih algoritam ove vrste je opisan u [6]. Za nalaženje početne ture korišćen je algoritam koji konstruiše proizvoljnu dopustivu turu ("random tour"), a kao algoritam koji poboljšava turu korišćen je 3-opt algoritam.

3. Lin-Kernighan algoritam

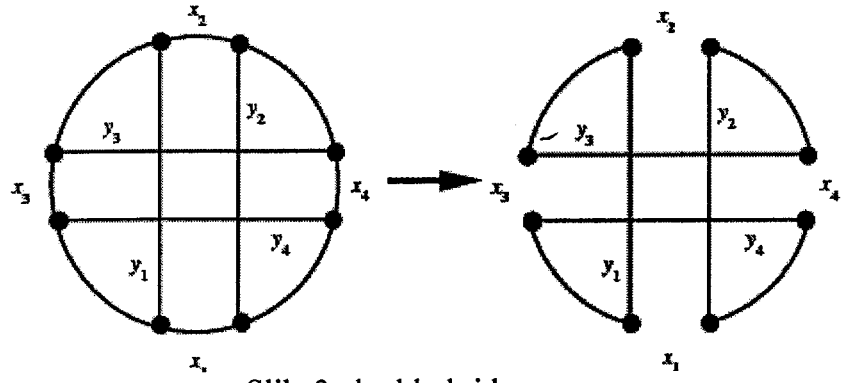
Lin-Kernighan (LK)algoritam je algoritam iz grupe algoritma koji poboljsavaju turu. LK algoritam izvodi takozvane k-opt move-ove nad trenutnom turom. K-opt move menja trenutnu turu tako što k-ivica iz ture zamenjuje sa drugih k ivica,koje nisu u trenutnoj turi, tako da je dobijena tura kraća.

Neka je T trenutna tura.U svakoj iteraciji algoritam pokušava da pronađe dva seta ivica, $X = \{x_1, ..., x_k\}$ i $Y = \{y_1, ..., y_k\}$,tako da ako su ivice iz X izbrisane iz T i zamenjene sa ivicama iz Y,rezultujuća tura je bolja od trenutne.
Skupovi X i Y se konstruišu element po element.U početku X i Y su prazni.U koraku i par ivica, x_i i y_i,dodaje se ,redom,u X i Y.

Da bi se dobilo na efikasnosti algoritma jedino ivice koje potpuno zadovoljavaju sledeće uslove mogu pripadati skupovima X i Y:

1) kriterijum sekvencijalnog menjanja:

x_i i y_i moraju imati zajednički čvor,kao i x_{i+1} i y_i .Na osnovu toga zaključujemo da niz $(x_1, y_1, x_2, y_2, x_3, ..., x_k, y_k)$ gradi lanac susednih ivica.Takva razmena ivica zove se sekvencijalna.
Neophodan uslov da se menjanjem ivica iz X ivicama iz Y dobije ture je da je lanac zatvoren.Međutim to nije i dovoljan uslov.
U opštem slučaju poboljšanje trenutne ture može da se dobije i nesekvencijalnom razmenom ivica.Najjednostavniji nesekvencijalni razmena je iz grupe 4-opt move-ova koja se zove *double-bridge move*.Zato ako se trenutna tura ne može poboljšati sekvencijalnom razmenom onda se pokušava sa double-bridge move-om.



Slika3: double-bridge move

2) kriterijum dopustivosti

On zahteva da se $x_i = (t_{2i-1}, t_{2i})$ (t_{2i-1}, t_{2i} su krajevi ivice xi) bira tako da kad se t_{2i} spoji sa t_1 dobije se tura.Ovaj kriterijum se koristi za $i>2$ i garantuje da je moguće zatvoriti turu.Ovaj kriterijum je uveden da bi se pojednostavilo kodiranje i smanjilo vreme izvršavanja.

U konkretnoj implementaciji korišćeno je da je osnovni move sekvencijalni 5-opt move. Za $i>0$ zahtevano je da se $x_{5i} = (t_{10i-1}, t_{10i})$ bira tako da spajanje t_{10i} sa t_1 prouzrokuje dobijanje ture.Stoga je svaka razmena ivica u algoritmu niz jednog ili više 5-opt move.Međutim,sa konstrukcijom razmene staje se momentalno ako je otkriveno da se zatvaranjem dobija bolja tura od trenutne.

3) kriterijum pozitivnog dobitka

Zahteva da se y_i bira tako da je ukupan dobitak razmene G_i pozitivan.Pretpostavimo da je $g_i = c(x_i) - c(y_i)$,gde je $c(x_i)$ dužina ivice x_i ,dobitak razmene x_i sa y_i .Tada je G_i jednak sumi $g_1 + g_2 + ... + g_i$ .Na prvi pogled deluje da je ovaj kriterijum previše restriktivan.Međutim,to nije tačno zbog sledeće činjenice: ako niz brojeva ima pozitivnu sumu tada postoji permutacija tih brojeva takva da je svaka parcijalna suma pozitivna.Ovaj kriterijum igra glavnu ulogu u efikasnosti algoritma.

4) kriterijum disjunktnosti

Zahteva da su skupovi X i Y disjunktni. Ovaj kriterijum je uveden da bi se pojednostavilo kodiranje i smanjilo vreme izvršavanja.

5) kriterijum za izbor kandidata

Potraga za ivicom $y_i = (t_{2i}, t_{2i+1})$ je ograničena na pet najbližih suseda t_{2i}.

Pseudo kod:

```
int LinKernighan() {
int t1, t2;
int X2, failures;
double G, Gain;
double Cost = 0;
ForAllNodes(t1)
Cost += C(t1,SUC(t1));
do {
failures = 0;
ForallNodes(t1, failures < broja gradova) {
for (X2 = 1; X2 <= 2; X2++) {
t2 = X2 == 1 ? PRED(t1) : SUC(t1);
G = C(t1,t2);
while (t2 = BestMove(t1, t2, &G, &Gain)) {
if (Gain > 0) {
Cost -= Gain;
StoreTour();
Failures = 0;
goto Next_t1;
}
}
Failures++;
RestoreTour();
}
Next_t1: ;
}
} while (Gain > 0);
Nonseq();
}
```

Objašnjenje:
Promenljive t1,t2 predstavljaju dva grada. U promenljivoj Cost čuva se dužiina trenutne ture.Grad koji sledi grad t1 u trenutnoj turi dobija se pozivom funkcije SUC(t1),a grad koji prethodi gradu t1 u trenutnoj turi dobija se pozovom PRED(t1).Pozivom f-je BestMove pokušava se da se popravi trenutna tura primenom sekvencijalnih 5-opt move-ova.Promenljiva gain pamti dobitak ako se izvrši BestMove.Promenljiva failures pamti broj neuspelih pokušaja da se popravi trenutna tura pozivom f-je BestMove.Ako nije moguće popraviti turu sekvencijalnim 5-opt move-ova onda se pokušava popravka ture double-bridge move-om pozivom f-je Nonseq() .

```
Node *BestMove(Node *t1, Node *t2, long *G0, long *Gain) {
    Node *t3, *t4, *t5, *t6, *t7, *t8, *t9, *t10;
    Node *T3, *T4, *T5, *T6, *T7, *T8, *T9, *T10 = 0;
    long G1, G2, G3, G4, G5, G6, G7, G8, BestG8 = LONG_MIN;
    int X4, X6, X8, X10;

    *Gain = 0; Reversed = SUC(t1) != t2;
    ForAllCandidates(t3, t2->CandidateSet) {
        if (t3 == PRED(t2) || t3 == SUC(t2) ||
            (G1 = *G0 - C(t2,t3)) <= 0)
            continue;
        for (X4 = 1; X4 <= 2; X4++) {
            t4 = X4 == 1 ? PRED(t3) : SUC(t3);
            G2 = G1 + C(t3,t4);
            if (X4 == 1 && (*Gain = G2 - C(t4,t1)) > 0) {
                Make2OptMove(t1, t2, t3, t4);
                return t4;
            }
            ForAllCandidates(t5, t4->CandidateSet) {
                if (t5 == PRED(t4) || t5 == SUC(t4) ||
                    (G3 = G2 - C(t4,t5)) <= 0)
                    continue;
                for (X6 = 1; X6 <= 2; X6++) {
                    Determine (T3,T4,T5,T6,T7,T8,T9,T10) =
                              (t3,t4,t5,t6,t7,t8,t9,t10)
                    such that
                       G8 = *G0 - C(t2,T3) + C(T3,T4)
                                - C(T4,T5) + C(T5,T6)
                                - C(T6,T7) + C(T7,T8)
                                - C(T8,T9) + C(T9,T10)
                    is maximum (= BestG8), and (T9,T10) has
                    not previously been included; if during
                    this process a legal move with *Gain > 0
                    is found, then make the move and exit
                    from BestMove immediately;
                }
            }
        }
    }
    *Gain = 0;
    if (T10 == 0) return 0;
    Make5OptMove(t1,t2,T3,T4,T5,T6,T7,T8,T9,T10);
    *G0 = BestG8;
    return T10;
}
```

Objašnjenje:Ovo je pseudo kod f-je BestMove koja se koristi u LinKernighan f-ji kako bi se trenutna tura popravila pomoću sekvencijalnih 5-opt move-ova.

4. VNS

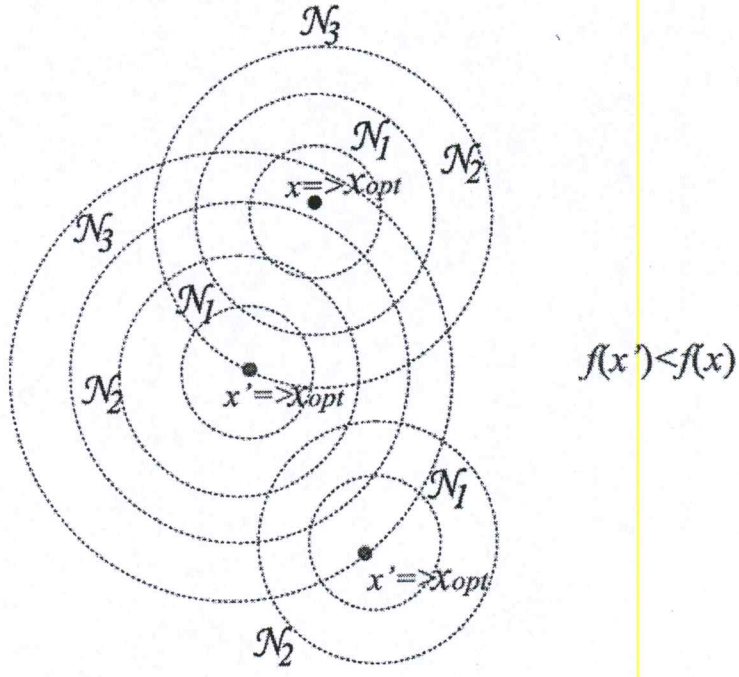
Metodu promenljivih okolina (Variable Neighborhood Search, VNS) predložili su Mladenović i Hansen, a prvi put ideja metode izložena je 1995. godine. Osnovna ideja metode je veoma jednostavna: sistematska promena okolina unutar lokalnog pretraživanja. Dakle, neophodno je uvesti više okolina, bilo da se menja metrika u odnosu na koju se definiše okolina, bilo da se povećava rastojanje u odnosu na istu metriku. Metoda je do sada uspešno primenjena na veliki broj problema kombinatorne optimizacije i veštačke inteligencije. Predloženo je nekoliko modifikacija i poboljšanja, uglavnom namenjenih uspešnom rešavanju problema velikih dimenzija.
VNS metaheuristika zasnovana je na tri jednostavne činjenice :

(1) lokalni minimum u odnosu na jednu okolinu ne mora biti i lokalni minimum u odnosu na neku drugu okolinu

(2) globalni minimum je lokalni minimum o odnosu na sve okoline;

(3) za većinu problema lokalni minimumi u odnosu na razne okoline su međusobno bliski.

Prva činjenica opravdava uvođenje više okolina. Druga činjenica ne garantuje da rešenje koje je lokalni minimum u odnosu na svaku od izabranih okolina predstavlja i globalni minimum, već nešto malo slabije: ako neko rešenje nije lokalni minimum u odnosu na neku okolinu, onda sigurno nije globalni minimum, stoga takođe ima smisla uvođenje više okolina. Kao posledica činjenice 3 (koja je empirijska, a ne teoretska) javlja se potreba detaljnog istraživanja najbliže okoline lokalnog minimuma jer se tu očekuje popravljanje tekućeg najboljeg rešenja.



Slika4: Upotreba više okolina u lokalnom pretraživanju

Definicija. Neka je x iz X proizvoljno dopustivo rešenje i neka je N_k, ($k = 1, \ldots, k_{max}$), konačna kolekcija unapred definisanih okolina. Tada je $N_k(x)$skup rešenja u k-toj okolini od x, tj. skup rešenja koja se u odnosu na usvojenu metriku (broj transformacija) nalaze na rastojanju k od rešenja x.

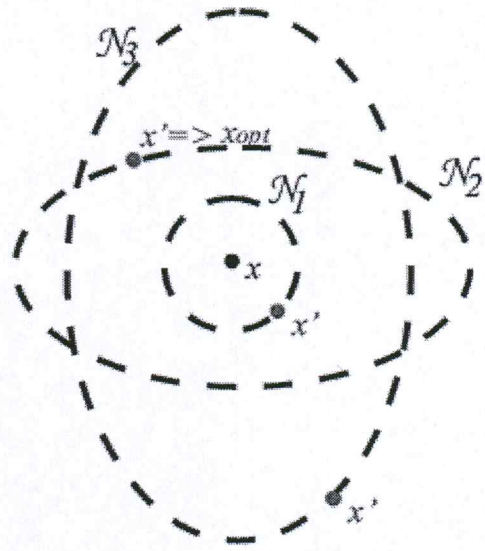
Slika4 ilustruje upotrebu više okolina u rešavanju optimizacionih problema. Mnoge metaheurističke metode bazirane na lokalnom pretraživanju koriste jednu, najviše dve okoline.

Upotreba više okolina postavlja dodatna pitanja na koja treba odgovoriti u okviru konkretne implementacije VNS algoritma :

- izbor okolina koje će se koristiti: zavisi od toga koje se metrike mogu definisani i primeniti u datom slučaju;
- uređenje (redosled) okolina: definisano je usvojenom metrikom i treba da bude takvo da u odnosu na tu metriku rastojanje među rešenjima raste;
- veličine okolina: odnosi se na restrikcije, tj. smanjenja okolina uočavanjem i razmatranjem samo onih suseda koji potencijalno mogu da obezbede poboljšanje;
- strategije pretraživanja: koje mogu biti do prvog poboljšanja (FI) ili naći najbolje u okolini (BI);
- spust i penjanje: odnos između koraka koji obezbeđuju poboljšanja i ostalih (koji omogućavaju izbegavanje zamke lokalnog minimuma);
- smer pretraživanja: određuje redosled kojim će se pretraživati okoline (kod VNS metode on je definisan parametrima k_{min}, k_{max} i k_{step} o kojima će više reči biti kasnije);
- intenzifikacija i diversifikacija pretraživanja: načini na koje će se pretraživanje koncentrisati u okolinama u kojima se očekuje poboljšanje, ili usmeriti pretraga ka novim, neistraženim regionima prostora rešenja čime se povećava šansa nalaženja globalnog minimuma.

Pri konkretnoj implementaciji VNS metode polazi se od tri ranije navedene činjenice ((i), (ii) i (iii)). Ove tri činjenice mogu se iskoristiti na tri različita načina: deterministički, stohastički ili kombinovano. Time se dobijaju tri prve verzije metode promenljivih okolina.

1) **Metoda promenljivog spusta**. Kada se koristi deterministička varijanta, dobija se metoda koju nazivamo metoda promenljivog spusta (Variable Neighborhood Descent, VND). Ona se sastoji u tome da se izabere k_{max} okolina, N_k, $k = 1, 2, \ldots, k_{max}$, odredi početno rešenje x i startuje LS procedura u odnosu na svaku od okolina, a počev od izabranog rešenja x. Ukoliko je $k_{max} = 1$, reč je o običnom lokalnom pretraživanju.
Grafički prikaz upotrebe više okolina u okviru VND metode dat je na slici 5. Grafičkom ilustracijom je istaknuto da okoline ne moraju biti "ugnježdene", tj. da ne moraju biti indukovane iz iste metrike (ili, ukoliko je reč o definiciji okoline u odnosu na neke zadate transformacije rešenja, okoline se mogu definisati primenom različitih transformacija).



Slika5: Upotreba više okolina u metodi promenljivog spusta (VND)

Konkretno, VND znači da se pretražuje okolina N1(x) dok god u njoj postoji mogućnost popravljanja trenutno najboljeg rešenja. Kada se odredi lokalni minimum x′ u odnosu na okolinu N1, ažurira se tekuće najbolje rešenje (postavi $x_{opt} = x'$), startuje se LS procedura u odnosu na okolinu N2(x_{opt}). čim dođe do popravke trenutno najboljeg rešenja xopt, pretraživanje se vraća u okolinu N1 najboljeg rešenja.Ukoliko se xopt ne popravi,procedura lokalnog pretraživanja se usmerava na narednu neispitanu okolinu Nk(x_{opt}). VND procedura se završava ako je nemoguće popraviti trenutno najbolje rešenje xopt ni u jednoj okolini, tj. ako je x_{opt} lokalni minimum u odnosu na sve okoline $N_1, N_2, \ldots, N_{k^{max}}$.

1. *Inicializacija*. Izabrati početno rešenje x, $x_{opt} = x$, $f_{opt} = f(x)$.

2. **Ponavljaj**
(a) k=1
(b) **Ponavljaj**
i. **LS**. Primeniti LS proceduru počev od x u okolini N_k;
neka je x' dobijeni lokalni optimum.
ii. *Provera rešenja*.
ako $f(x') < f(x_{opt})$ onda $x_{opt} = x'$, $f(x_{opt}) = f(x')$, k = 1 inače k = k + 1
dok je $k <= k_{max}$
dok ima poboljšanja

2) **Redukovana metoda promenljivih okolina**. U slučaju stohastičke definicije metode koraci su utoliko jednostavniji što ne postoji proces lokalnog pretraživanja. Dakle metoda se sastoji u sistematskoj promeni okolina i izboru jednog slučajnog rešenja u svakoj od okolina. Koraci odlučivanja bazirani su na tom jednom slučajnom rešenju. Metoda koja se na ovaj način dobija naziva se redukovana metoda promenljivih okolina (Reduced Variable Neighborhood Search, RVNS).
RVNS je izuzetno korisna kod primera velikih dimenzija jer se izbegava složena i dugotrajna LS procedura. Ova metoda veoma liči na Monte-Carlo metodu, mada je donekle sistematičnija. Dok Monte-Carlo bira slučajno rešenje u celom prostoru pretraživanja, RVNS se u svakom koraku ograničava na neku, strogo definisanu okolinu. Ipak, obzirom na stepen slučajnosti, najbolji rezultati se postižu kombinacijom ove metode sa nekom drugom varijantom. Na primer, RVNS se koristi za dobijanje početnog rešenja, a zatim se primenjuje neka varijanta koja sistematično pretražuje okoline tako dobijenog početnog rešenja.
Pseudokod RVNS metode može se zapisati u sledećem obliku:

1. *Inicializacija*. Izabrati početno rešenje x, $x_{opt} = x$, $f_{opt} = f(x)$.

2. **Ponavljaj**
   (a) k=1
   (b) Ponavljaj
      i. *Razmrdavanje.*
      Izabrati slučajno rešenje x' u okolini $N_k(x)$;
      ii. *Provera rešenja.*
      ako $f(x') < f(x_{opt})$ onda $x_{opt} = x'$, $f(x_{opt}) = f(x')$, k = 1 inače k = k + 1
      dok nije $k = k_{max}$
dok nije zadovoljen kriterijum zaustavljanja.

Uobičajeni kriterijum zaustavljanja u ovom slučaju je maksimalni broj iteracija između dva poboljšanja. Naravno, mogu se koristiti i svi ostali navedeni kriterijumi, zavisno od konkretne primene i zahteva koje treba zadovoljiti prilikom rešavanja svakog pojedinačnog optimizacionog zadatka.

3) **Metoda promenljivih okolina**. Kombinacijom prethodna dva principa, tj. sistematskom (determinističkom) promenom okolina, slučajnim izborom početnog rešenja u tekućoj okolini i primenom LS procedure počev od tog, slučajnog rešenja dobija se osnovna metoda promenljivih okolina ((Basic) Variable Neighborhood Search, VNS). Ovo je najrasprostranjenija varijanta metode promenljivih okolina jer obezbeđuje više preduslova za dobijanje kvalitetnijih konačnih rešenja.
Osnovni koraci VNS metode sadržani su u petlji u okviru koje se menja indeks okoline k, određuje slučajno rešenje iz k-okoline, izvršava se procedura lokalnog pretraživanja i proverava kvalitet dobijenog lokalnog minimuma. Ovi koraci se ponavljaju sve dok ne bude zadovoljen neki kriterijum zaustavljanja. Početno rešenje u okolini k generiše se na slučajan način kako bi

se obezbedilo pretraživanje različitih regiona prilikom sledećeg razmatranja okoline k. Okoline se mogu razlikovati po osnovu rastojanja (broja transformacija) ili po osnovu metrike (vrste transformacija). Važno je napomenuti da okoline za razmrdavanje (izbor slučajnog rešenja) i lokalno pretraživanje ne moraju biti istog tipa.
Opisani koraci mogu se ilustrovati pseudokodom na sledeći način:

Inicializacija. Izabrati početno rešenje x iz X;
definisati kriterijum zaustavljanja; STOP = 0.

Ponavljaj
1. k = 1;
2. **Ponavljaj**
   (a) *Razmrdavanje*. Generisati slučajno rešenje x' u k-toj okolini od x, (x' iz $N_k(x)$);
   (b) **LS**. Primeniti neku proceduru lokalnog pretraživanja počev od x';
   označiti sa x'' dobijeni lokalni minimum;
   (c) *Provera rešenja*.
   Ako je lokalni minimum bolji od trenutnog minimuma, preći u to rešenje ($x = x''$);
   nastaviti od novog početnog rešenja u okolini N_1 (k = 1);
   inače preći u sledeću okolinu, tj. k = k + 1.
   (d) *Provera završetka*. Ako je zadovoljen kriterijum zaustavljanja, STOP = 1.
   dok nije k == k_{max} ili STOP == 1;
dok nije STOP == 1;

Kriterijum zaustavljanja može buti maksimalno dozvoljeno CPU vreme,maksimalan broj iteracija ili maksimalan broj iteracija između dve popravke

5.Implementacija i rezultati

Rešavan je euklidski TSP. Implementirani su basic VNS i VND. Kodovi su pisani na jeziku C++. U oba slučaja početno rešenje je slučajna permutacija gradova. Kao struktura podataka korišćena je dvostruko povezana lista. Sva testiranja su obavljena na Intel Pentium IV procesoru sa 1.83 GHz, pod Windows-om.

Cilj nam je bio da testiramo novu metodu dobijenu implementacijom VNS gde je kao lokalna pretraga korišćena **Lin-Kernighan** pretraga opisana u sekciji 3. Ta pretraga je modifikovani LK algoritam koji je predložio Helsgaun u [1]. Modifikacija se ogleda u tome što se kao standardni "move" koristi 5-opt move za razliku od standardnog koji koristi 2-opt move. Eksperimentalnim putem je utvrđeno da ta modifikacija bitno utiče na efikasnost LK algoritma Do sada ni u jednoj implementaciji VNS-a nije korišćena gore opisana **Lin-Kernighan** pretraga.

1) VND

Implementiran je VND sa četiri okoline.Prva okolina predstavlja 1-opt okolinu ture;druga okolina je 2-opt okolina; treća okolina je Or-opt1 okolina i četvrta okolina je Or-opt2 okolina.

Pseudo kod:

```
void vnd()
{  ind=1;
   int k = 1;
   while(k < 5)
    {
     switch(k) {
      case 1:
       oneopt();
       if (nijepoboljšanatura) k++;
       break;
      case 2:
       twoopt();
       if (nijepoboljšanatura) k++;
       else k=1;
       break;
      case 3:
       Oropt1();
       if (nijepoboljšanatura) k  = 1;
       else k++;
       break;
      case 4:
       Oropt2();
       if (nijepoboljšanatura) k  = 1;
       else k++;
       break;
     }
    }
}
```

Primenjene su dve strategije pretrage.Prva strategija je bila do prvog poboljsanja,a druga do najboljeg poboljšanja.
Rezultati koje su dale te dve strategije kada su primenjene na neke test primere iz TSPLIB-a dati su usledećim tabelama. Sve instance su puštane samo jedanput.Tabele sadrže sledeće podatke:
Broj gradova je broj gradova u odgovarajućim test primer.Svaki grad je dat svojim koordinatama(x,y).
Rešenje predstavlja dužinu ture koja je dobijena primenom određene metode na odgovarajući test primer.
Greška predstavlja odstupanje rešenja od najbolje poznate dužine ture za odgovarajući test primer i računa se na sledeći način:
Greška=(rešenje - najbolje poznata dužina)/najbolje poznata dužina.

naziv test primera	broj gradova	rešenje	greška (%)	vreme	najbolje poznata dužina
eil 51	51	430	0.90%	0.00s	426
berlin 52	52	8033	6.51%	0.02s	7542
eil76	76	552	2.6%	0.00s	538
kroC100	100	21527	3.75%	0.02s	20749
rd100	100	8049	1.75%	0.00s	7910
ch150	150	6727	3.05%	0.01s	6528
tsp225	225	4078	4.14%	0.03s	3916
ts225	225	129143	1.97%	0.01s	126643
pcb442	442	53721	5.79%	0.09s	50778
u574	574	38389	4.02%	0.25s	36905
u724	724	43767	4.43%	0.50s	41910
pr1002	1002	275078	6.19%	1.81s	259045

Reziltati dobijeni VND-jem sa strategijom prvo poboljšanje

naziv test primera	broj gradova	rešenje VNS-om	greška (%)	vreme	najbolje poznata dužina
eil 51	51	438	2.82%	0.00s	426
berlin 52	52	8089	7.25%	0.02s	7542
eil76	76	562	4.46%	0.22s	538
kroC100	100	21654	4.36%	0.02s	20749
rd100	100	8323	5.22%	0.02s	7910
ch150	150	6948	6.43%	0.03s	6528
tsp225	225	4159	6.20%	0.06s	3916
ts225	225	126962	0.25%	0.06s	126643
pcb442	442	52907	4.19%	0.55s	50778
u574	574	39233	6.31%	1.22s	36905
u724	724	44833	6.97%	2.62s	41910
pr1002	1002	277427	7.96%	7.97s	259045

Reziltati dobijeni VND-jem sa strategijom najbolje poboljšanje

Prilikom testiranja strategija pretrage u VND-ju ispostavilo se da je strategija prvog poboljšanja daleko brža od strategije najboljeg poboljšanja , a i u većini primera daje bolja rešanja.

2)Basic VNS

U konkretnoj implementaciji VNS-a korišćeno je pet okolina.Okolina $N_k(x)$ ture x predstavlja skup svih tura y takvih da se tura y može dobiti od ture x primenom tačno k 2-opt move-ova.

Npr.neka je tura x predstavljena sa 1-7-3-6-2-10-4-8-5-9-1. Tada tura 1-9-2-10-4-8-5-6-3-7-1 pripada N_1 okolini od x jer se može dobiti od ture x raskidanjem ivica 2-6 i 5-9 u 2-opt move-u.

U razmrdavanju tačka x' iz k-te okoline tačke x se generiše primenom k puta slučajnog 2-opt mova na turu x.Slučajni 2-opt move je 2-opt move u kome se ivice koje se raskidaju biraju slučajno.

Npr.neka je tura x predstavljena sa 1-7-3-6-2-10-4-8-5-9-1 i neka nam je potrebna slučajna tačka x' iz N_2 okoline.Tačku x' generišemo na sledeći način:

Izaberimo slučajno dve ivice koje raskidamo iz ture x.Neke su to ivice 2-6 i 5-9.Tada nakon 2-opt move-a dobijamo turu 1-9-2-10-4-8-5-6-3-7-1.Sada biramo slučajno dve ivice iz novodobijene ture koje ćemo raskinuti u 2-opt move-u.Neka su to ivice 2-10 i 3-7.Tada nakon 2-opt move-a dobijamo turu 1-7-10-4-8-5-6-3-2-9-1 koja predstavlja tačku x' iz N_2 okoline. Analogno se generiše tačka x' iz bilo koje okoline.

Kao kriterijum zaustavljanja korišćeno je da je maksimalno vreme izvršavanje VNS-a 100 sekundi.

U sledećoj tabeli dati su rezultati dobijeni kada je gore opisani VNS primenjen na neke test primere iz TSPlib-a sa sledećim lokalnim pretragama:

1) lokalna pretraga je 2.5-opt pretraga

naziv test primera	broj gradova	rešenje VNS-om	greška (%)	min.vrema rada VNS-a	najbolje poznata dužina
eil 51	51	426	0%	0.84s	426
berlin 52	52	7542	0%	0.03s	7542
eil76	76	538	0%	0.06s	538
kroC100	100	20749	0%	0.03s	20749
rd100	100	7910	0%	0.06s	7910
ch150	150	6528	0 %	0.07s	6528
tsp225	225	3919	0.08%	0.05s	3916
ts225	225	126643	0%	1.87s	126643
pcb442	442	51047	0.53%	1.75s	50778
u574	574	36978	0.20%	27.62s	36905
u724	724	42083	0.41%	10.91s	41910
pr1002	1002	260321	0.49%	94.21s	259045

Minimalno vreme rada VNS-a predstavlja minimalno vreme neophodno da bi se dobilo rešenje koje je isto kao rešenje dobijeno kada je VNS pušten da radi 100 sekundi.

2) lokalna pretraga je Lin-Kernighan pretraga

naziv test primera	broj gradova	rešenje VNS-om	greška (%)	min.vrema rada VNS-a	najbolje poznata dužina
eil 51	51	426	0%	0.02s	426
berlin 52	52	7542	0%	0.11s	7542
eil76	76	538	0%	0.05s	538
kroC100	100	20749	0%	0.11s	20749
rd100	100	7910	0%	0.16s	7910
ch150	150	6528	0 %	44.32s	6528
tsp225	225	3919	0.08%	11.78s	3916
ts225	225	126643	0%	1.18s	126643
pcb442	442	50809	0.06%	36.30s	50778
u574	574	37037	0.36%	26.5s	36905
u724	724	42030	0.29%	87.47s	41910
pr1002	1002	260026	0.38%	67.66s	259045

Na osnovu dobijenih rezultata zaključujemo da nova heuristika pronalazi optimalno rešenje za test primer eil51, berlin 52, eil76, kroC100, rd100, ch 150, ts225. Na ostalim testiranim primerima nova heuristika je pronašla ture koje su veoma bliske optimalnoj turi (odstupanje u najgorem slučaju od optimalne ture je 0.38%).

U poređenju sa rezultatima koje su dale ostale heusristike implementirane u ovom radu nova heuristika je najbolja. Ture dobijene heuristikama baziranim na VND-ju su znatno duže nego ture dobijene novom heuristikom. Takođe, u poređenju sa heuristikom baziranom na VNS-u sa 2.5-opt pretragom nova heuristika je pronašla lošiju turu samo na test primeru u574, ali je na svim ostalim test primerima,na kojim nije pronašla optimalnu turu, pronšla bolju turu nego heuristika sa 2.5-opt pretragom. Na test primeru u574 nova heuristika je pronašla turu za 0.36% lošiju od optimalne, a heuristika sa 2.5-opt pretragom je pronašlu turu za 0.20% lošiju od optimalne. Sa druge strane,na test primeru pcb442 nova heuristika je pronašla turu za 0.06% lošiju od optimalne, a heuristika sa 2.5-opt pretragom je pronašlu turu za čak 0.53% lošiju od optimalne.

U poredjenju sa Helsgaunovom heuristikom datom u [1], nova heuristika ne zaostaje značajno što je od izuzetnog značaja imajući u vidu da je ta heuristika trenutno najbolja. Najveće odstupanje u kvalitetu tura je na test primeru pr1002 na kome Helsgaunova heuristika pronalazi optimalnu turu, a nova heuristika nalazi turu za 0.38% lošiju.

6.Zaključak

U radu je izložena nova heuristika za rešavanje euklidskog TSP-a koja se uz male modifikacije može primeniti i na rešavanje simetričnog,asimetričnog i metričnog TSP-a.

Nova heuristika je pokazala zavidne performanse u pogledu nalaženja optimalne ture u poređenju sa drugim heuristikama koje su ovde izložena kao i u poredjenju sa trenutno najboljom heuristikom, Helsgaunovom heuristikom.

Plan daljeg istraživanja je ispitivanje zavisnosti dužine pronađene ture pomoću nove heuristike od kriterijumu zaustavljanja koji se koristi u VNS, kao i od izbora okolina koje se koriste u VNS-u. Takođe, ideja je da se nova heuristika primeni na rešavanje problema sa veoma velikim brojem gradova.

Literatura:

1)K. Helsgaun,
"An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic",
European Journal of Operational Research, **12**, pp. 106-130 (2000).

2)K. Helsgaun,
"An Effective Implementation of *K*-opt Moves for the Lin-Kernighan TSP Heuristic"
Writings on Computer Science, No. 109, Roskilde University, 2006 Revised November 20, 2007

3)P.Hansen,N.Mladenović,
"Variable neighborhood search:principles and applications",European Journal of Operational Research, **130**, pp. 449-467 (2001).

4)D.Cvetković,M.Čangalović,Đ.Dugošija,V.Kovačević-Vujčić,S.Simić,J.Vuleta,
"Kombinatorna optimizacija:matematička teorija i algoritmi",*DOPIS Beograd(1996)*.

5) "*Traveling salesman problem web*",
http://www.tsp.gatech.edu/

6) S. Lin,
"Computer Solutions of the Traveling Salesman Problem," Bell System Tech.J.44, 2245-2269 (1965)

7)D.Feillet, P. Dejax, M. Gendreau,
"Traveling Salesman Problems with Profits", Transportation Science 39 (2),pp.188–205 (2004)

8)M. Fischetti, J.J. Salazar-Gonzalez,P. Toth,
"A Branch-and-Cut algorithm for the Symmetric Generalized Traveling Salesman Problem", Operations Research 45 (3),p.p 378–394 (1997)

9)J.Riera-Ledesma, J.J. Salazar-Gonzalez,
"A Heuristic approach for the Traveling Purchaser Problem", European Journal of Operational Research 162,p.p 142–152 (2005).

10) M. Hammar , B. J. Nilsson,
"Approximation Results for Kinetic Variants of TSP", Proc. International Colloquium on Automata, Languages, and Programming, p.p. 392-401(1999)

11) G. Dantzig, R. Fulkerson, S. Johnson,
"Solution of a large-scale traveling-salesman problem", Operations Research 2,p.p 393-410 (1954)

12) D. Applegate, R.E. Bixby, V. Chvátal, W.J. Cook,
" Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems", Mathematical Programming 97,p.p. 91-153 (2003)