# Slovenská technická univerzita v Bratislave

## FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Ilkovičova 3, 812 19 Bratislava


---

# Evolučné Algoritmy
# Case study
# Metóda zakázaného hľadania aplikovaná na TSP

---


**Tomáš Matúšek**


2005/2006

# 1 PREHĽAD DOKUMENTU

Dokument predstavuje case study z predmetu Evolučné algoritmy, konkrétne aplikáciu zakázaného hľadania na problém obchodného cestujúceho.

Na začiatku je stručne vysvetlený problém obchodného cestujúceho a základný princíp metódy zakázaného hľadania, ďalej nasleduje konkrétna aplikácia metódy na daný problém a nakoniec uvádzam dosiahnuté výsledky spolu so zhodnotením.

# 2 ÚVOD DO PROBLEMATIKY

## 2.1 TSP (PROBLÉM OBCHODNÉHO CESTUJÚCEHO)

TSP patrí medzi kombinatorické optimalizačné problémy. Dá sa formulovať nasledovne:

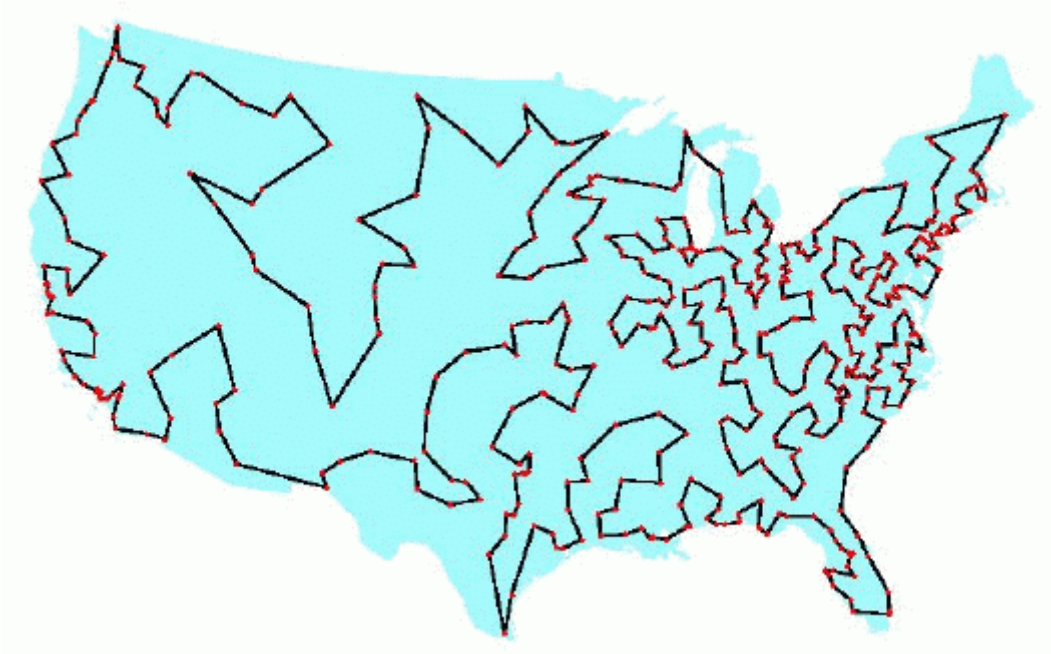
Máme n miest, pričom je definovaná cena cesty medzi každým párom miest osobitne. Úlohou je nájsť takú najkratšiu cestu v grafe miest, aby sme navštívili každé mesto práve raz, okrem mesta ,z ktorého cestu začíname.

Zdanlivo triviálna formulácia problému je zavádzajúca. V skutočnosti sa jedná o problém so zložitosťou n!, ktorí patrí do triedy tzv. NP úplných problémov, ku ktorým sa zvyčajne pristupuje tromi rôznymi spôsobmi:

1) Vytvorenie algoritmu na presné nájdenie riešenia (v rozumnom časovom intervale je možné riešiť len problémy malej veľkosti)

2) Vytvorenie suboptimálnych alebo heuristických algoritmov, tj. algoritmov, ktoré dokážu nájsť pravdepodobne dobré riešenie, ktorého optimálnosť však nie je možné dokázať.

3) Nájdenie špeciálnych prípadov problému, pre ktoré je možné nájsť presné riešenia alebo aspoň lepšie heuristiky



**obr. 1: Ukážka najkratšej cesty 532 miest v USA**

Medzi existujúce spôsoby riešenia TSP patria iteratívne prístupy ako Kernighan - Lin alebo k- opt heuristika, alebo stochastické prístupy kam patria genetické algoritmy, simulované žíhanie, neurónové siete, zakázané hľadanie.

Mojou úlohou bolo aplikovať posledne menovaný prístup, tj. zakázané hľadanie.

## 2.2 TABU SEARCH (ZAKÁZANÉ HĽADANIE)

Základný koncept TS je podľa jedného z jeho autorov (Glover) metaheuristika aplikovaná na nejakú inú heuristiku. TS skúma priestor riešení posúvaním sa od aktuálneho riešenia j najlepšiemu riešeniu z jeho okolia.

Na rozdiel od klasických zostupových metód, TS akceptuje aj horšie riešenie. Aby sa zamedzilo cykleniu, transformácie, ktoré vedú k nedávno navštíveným riešeniam sú označené ako tabu, čiže zakázané. Doba, počas ktorej je transformácia v tabu stave môže byť pevná, alebo sa môže v čase meniť. Tabu stav môže byť ignorovaný v prípade, že sú splnené nejaké špeciálne podmienky. Tieto sa nazývajú ašpiračné kritériá a predstavujú zväčša prípad, keď daná transformácia vedie ku dosiaľ najlepšiemu riešeniu.

Táto tendencia odchyľovať sa od predpokladaného smeru môže znamenať stratu, ale aj zisk. Nové riešenia nie sú generované náhodne, ako pri iných stochastických algoritmoch, ale deterministicky, keďže TS prijme horšie riešenie len v prípade, že sa chce vyhnúť už navštívenej ceste. Toto zaručuje, že budú preskúmané širšie oblasti priestoru riešení a nemalo by dochádzať k zaseknutiu sa v lokálnom minime.

Pseudokód:

- Krok 1. Vyber počiatočné riešenie i z S. Nastav i* = i a k = 0
- Krok 2. Nastav k = k +1 a generuj podmnožinu V z i pomocou transformácií t z T tak, že platí: t sa nenachádza v tabu zozname alebo sú splnené ašpiračné kritéria
- Krok 3. Vyber najlepšie j z množiny V a nastav i = j
- Krok 4. Ak f(i) < f(i*), i* = i
- Krok 5. Uprav tabu zoznam a ašpiračné kritériá
- Ak je splnená ukončovacia podmienka, skonči. Inak choď na krok 2.

Ďalšími dôležitými vlastnosťami TS sú mechanizmy diverzifikácie a intenzifikácie. Sú založené na prístupe ľudskej mysle. Keď človek prehľadáva nejakú časť priestoru riešení daného problému a zdá sa mu, že dlho nedospel k zlepšeniu, alebo sa pohybuje v cykloch, prestane sa sústrediť na danú časť a svoju pozornosť presunie inde, čo môžeme označiť ako diverzifikáciu. Naopak, ak v danej časti priestoru nachádza stále nové lepšie riešenia, bude sa snažiť ju preskúmať čo najpodrobnejšie. Tento jav nazývame intenzifikácia.

TS sa v súčasnosti úspešne aplikuje na rôzne oblasti riešenia najmä náročných kombinatorických problémov. Treba zdôrazniť, že sa jedná len o metaheuristiku, teda je potrebné ho špeciálne prispôsobiť danému problému, čo nie je v žiadnom prípade jednoduchý proces.

# 3 OPIS RIEŠENIA

Mojou úlohou bolo aplikovať Tabu search na problém obchodného cestujúceho, pričom sa vychádza z existujúcej postupnosti (permutácie miest na ortogonálnej mriežke) a operácia mutácie predstavuje výmenu dvoch miest v rámci permutácie. Jedná sa teda o metódu 2-opt, čo v praxi znamená, že každou mutáciou zaniknú dve hrany a vzniknú dve nové.

Ďalej som mal experimentovať s veľkosťou tabu zoznamu a maximálnym počtom iterácií tak, aby výsledkom bolo čo najoptimálnejšie riešenie.

## 3.1 ZÁKLADNÉ ÚLOHY

Medzi základné otázky, na ktoré je potrebné odpovedať pred implementáciou riešenia patrí vytvorenie počiatočného stavu, spôsob, akým bude realizovaný tabu zoznam a čo sa v ňom bude uchovávať, určenie okolia, ktoré sa bude prehľadávať a v neposlednej rade určenie mechanizmu diverzifikácie.

Zo zadania úlohy vyplýva, že nemá zmysel príliš špekulovať nad stratégiou tvorby iniciálneho riešenia, nakoľko mestá sa nachádzajú na ortogonálnej mriežke a napríklad metóda najbližšieho suseda prakticky vytvorí optimálne riešenia. Samozrejme, niečo také funguje len pre špeciálny prípad ortogonálnej mriežky, je dokázané, že existujú prípady, kedy spomínaná metóda nájde práve najhoršie riešenie. Preto som sa rozhodol na vytvorenie iniciálneho riešenia použiť obyčajný náhodný mechanizmus.

Pri realizácii tabu zoznamu som sa najskôr pohrával s myšlienkou použitia nejakej zoznamovej štruktúry, ale nevýhody takéhoto prístupu, ktoré spočívajú najmä v komplikovanej a častej úprave zoznamu ma odradili. Rozhodol som sa pre prístup s využitím dvojrozmerného poľa, kde rozmer predstavuje počet miest na mriežke. Na začiatku je zoznam vynulovaný, po pridaní prvku do zoznamu sa nastaví na dĺžku zoznamu + počet doteraz vykonaných iterácií. Pri zisťovaní, či sa daný prvok v zozname nachádza sa porovná hodnota na danom mieste zoznamu a aktuálny počet vykonaných iterácií. Takýto prístup je pomerne nenáročný na výpočet aj realizáciu.

Existuje viacero informácií, ktoré je možné v tabu zozname uchovávať. Ja som sa zaoberal dvomi možnosťami:

1) Pozície miest v permutácii, ktoré sme v aktuálnom kroku vymenili. Jedná sa o jednoduchý spôsob, ale nie príliš reprezentatívny, keďže po niekoľkých iteráciách môže výmena oboch miest predstavovať už úplne inú situáciu.

2) Hrany, ktoré v danej mutácii zmenili svoj stav, tj. vznikli alebo boli zrušené. Jedná sa o tzv. atribúty danej mutácie, ktoré vo všeobecnosti predstavujú väčšiu výpovednú hodnotu o danej situácii. Potom je potrebné určiť, či sa za tabu pokladá zmena aspoň jedného atribútu alebo len zmena všetkých atribútov. Rozhodol som sa pre druhý spôsob.

Pre porovnanie som sa rozhodol implementovať obidva spôsoby.

Výber okolia, z ktorého sa vyberá najlepšie riešenie sa dá tiež realizovať rôzne. Za okolie môžeme považovať len určitú časť alebo celý priestor riešení. Ak zoberieme celý priestor riešení máme zaručené vyššiu dôkladnosť hľadania, za cenu vyššej časovej náročnosti.

Naopak, pri čiastočnom priestore bude trvať prehľadávanie kratšie, ale dosiahnuté riešenia majú nižšiu pravdepodobnosť totožnosti s globálnym optimom. Po niekoľkých pokusoch som druhú možnosť zavrhol, pretože výsledky, ktoré mi poskytovala boli niekoľkonásobne horšie, bez tendencie zlepšovania sa.

Posledným dôležitým aspektom je realizácia diverzifikácie. Zvolil som tri rôzne spôsoby, ktoré je možné kombinovať.

1) Permutácia s doteraz najlepším riešením. Po určitom počte iterácií sa vrátime k permutácii s doteraz najlepším riešením ,s cieľom lepšie preskúmať priestor okolo nej (je to v podstate zároveň aj intenzifikácia)
2) Otáčanie reťazca určitej dĺžky. Po stanovenom počte iterácií náhodne vyberieme reťazec, ktorý zrkadlovo otočíme. Vhodné na únik z ostrého lokálneho minima.
3) Preusporiadanie reťazca. Reťazec obmedzenej dĺžky preusporiadame metódou najbližšieho suseda. Môže viesť k lepším výsledkom.

## 3.2 REAKTÍVNY TABU ZOZNAM A PREDCHÁDZANIE STAGNÁCIÁM

Veľkým problémom TS je zvoliť správnu veľkosť tabu zoznamu. Pri malých veľkostiach je dôsledkom častejšie cyklenie, pri príliš veľkých naopak vynechanie sľubných lokálnych miním. Často sa stáva, že v rôznej iterácii hľadania je potrebná rôzna dĺžka tabu zoznamu. Preto som sa okrem pevnej dĺžky rozhodol implementovať aj reaktívnu veľkosť zoznamu. Zoznam reaguje opakujúce sa riešenia narastaním a na odlišné riešenia skracovaním. Problémom je správne zhodnotiť, o aký jav sa práve jedná.

Druhým problémom je počet iterácií medzi dvomi procesmi diversifikácie. Pokiaľ proces hľadania stagnuje, čiže dlho nevie nájsť lepšie riešenie, je zrejme potrebná diverzifikácia a nemá zmysel pokračovať. Aj v tomto prípade je potrebné správne vyhodnotiť situáciu.

## 3.3 ALGORITMUS

C++ metóda jednej iterácie:

```
void Tab::iterate()
{
     int index1 = -1, index2 = -1, min = 1000000, temp,
temp2, temp3[2];
     bool cycle = true;
     int sup1, sup2;
```

```
     for(int i = lower; i < upper; i++){
          for(int j = i + 1; j < upper; j++){
               if(i != j){
                    if(i < j){
                         sup1 = i;
                         sup2 = j;
                    }
                    else{
                         sup1 = j;
                         sup2 = i;
                    }
                    temp = altDist(sup1, sup2);
                    if !isTabu(i, j)  ||  temp < bestD){

                         if(temp < min){
                              min = temp;
                              index1 = sup1;
                              index2 = sup2;
                         }
                    }
               }
          }
     }

     if(index1 != -1 && index2 != - 1){
          cestaD = min;
          temp3[0] = mesta[index1][0];
          temp3[1] = mesta[index1][1];
          mesta[index1][0] = mesta[index2][0];
          mesta[index1][1] = mesta[index2][1];
          mesta[index2][0] = temp3[0];
          mesta[index2][1] = temp3[1];
          if(cestaD <= bestD){
               bestD = cestaD;
               for(i = 0; i < this->n * this->n; i++){
                    best[i] = computeIndex(mesta[i][0],
mesta[i][1]);
               }
          }

          addTabu(index1, index2);
     }
     this->iteration++;
}
```

C++ metóda TS aj s diversifikáciou:

```
While(this->iteration < 60000 && this->bestD > (this-
>n*this->n) + 1){
     while(this->iteration < (c+1)*500 && this-
>stagnation <= this->n*this->n/3  && this->bestD > (this-
>n*this->n) + 1){
```

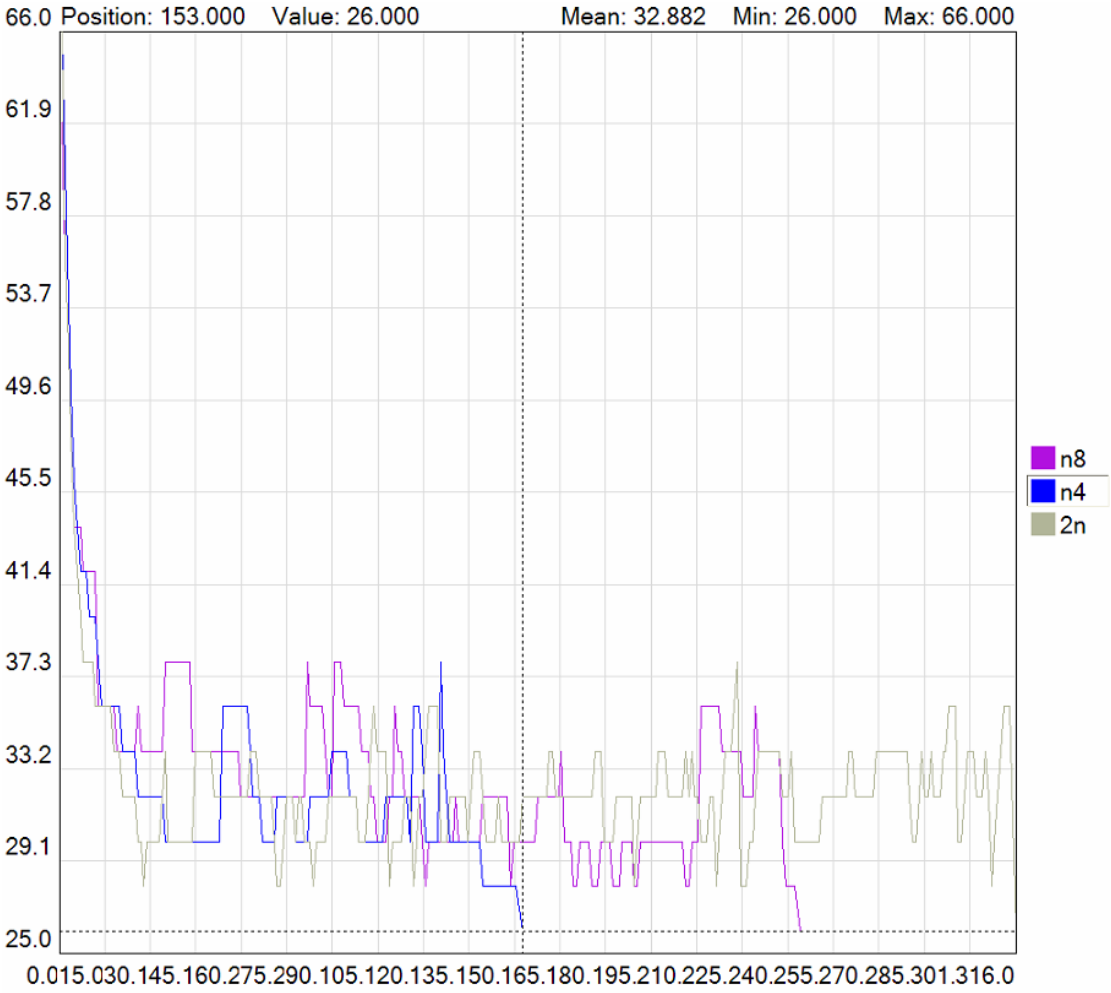
```
                help = bestD;
                this->iterate();
                if(repeatC < 8 && repeat == cestaD &&
!fix){

                     repC++;
                }
                else if(repeatC ==  12 && !fix){

                     if(repC >= 4){
                          if(this->factor < n*n)
                               this->factor++;
                          else
                               this->factor = 0;
                          //this->stagnation++;
                     }
                     else{
                          this->factor--;
                          //this->stagnation--;
                          repeat = cestaD;
                     }
                     repeatC = 0;
                }

                if(this->bestD > (this->n*this->n) + 1)
                     fprintf(f1, "%d     %d\n", this-
>iteration, cestaD);

                if(help == bestD){
                     this->stagnation++;
                }
                else {
                     this->stagnation--;
                }


                repeatC++;

           }

           this->stagnation = 0;
           this->inverseCompute();
           this->turnString();
      //   this->greedyReconnect();
           c++;
      }
```

# 4 EXPERIMENTY

Táto časť predstavuje overenie implementácie na niekoľkých experimentoch.

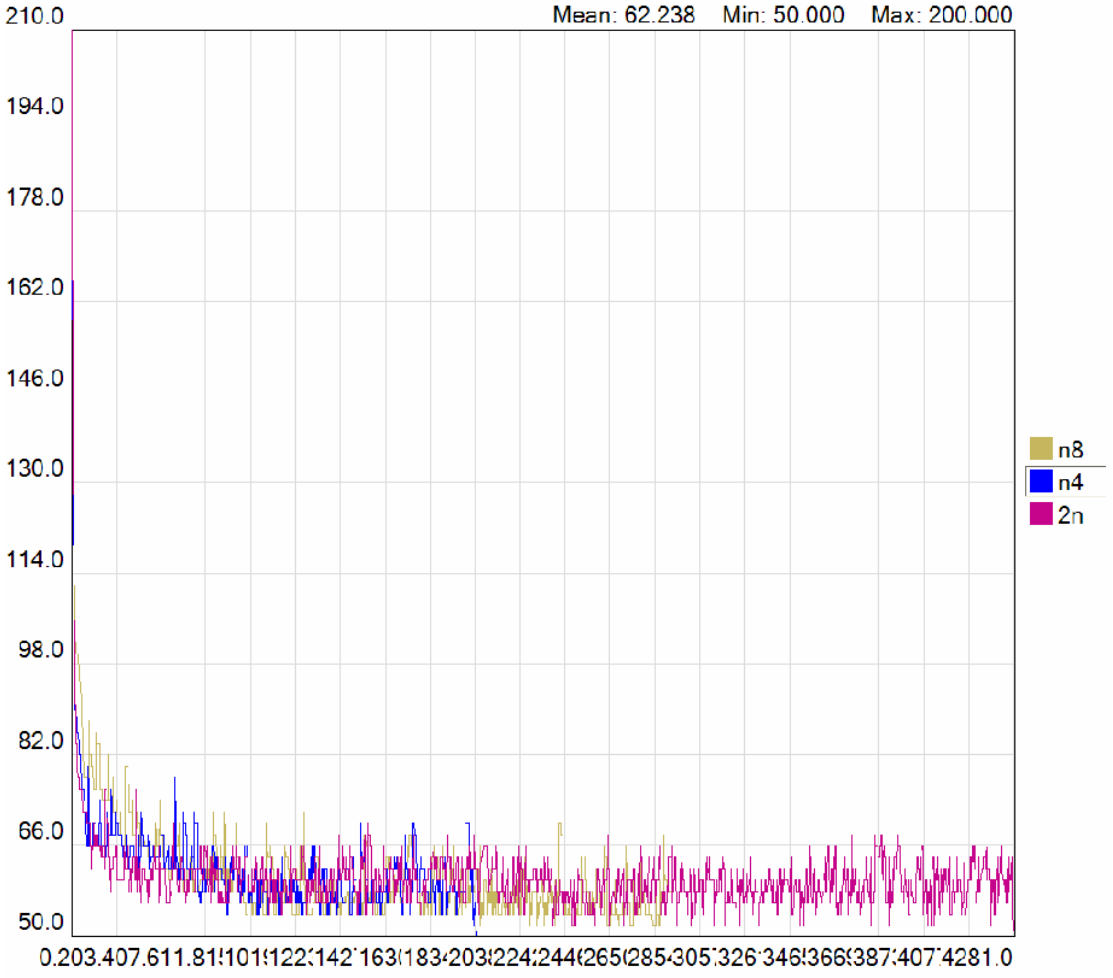
Nasleduje porovnanie priebehu hľadania pre tabu zoznam rôznej fixnej dĺžky pre 25 - 49 miest.

Graf pre n = 25, tabu =n/4, n/8 a 2*n




**obr. 2: porovnanie priebehu hľadania pre rôzne veľkosti tabu zoznamu**

Graf pre n = 49, , tabu =n/4, n/8 a 2*n

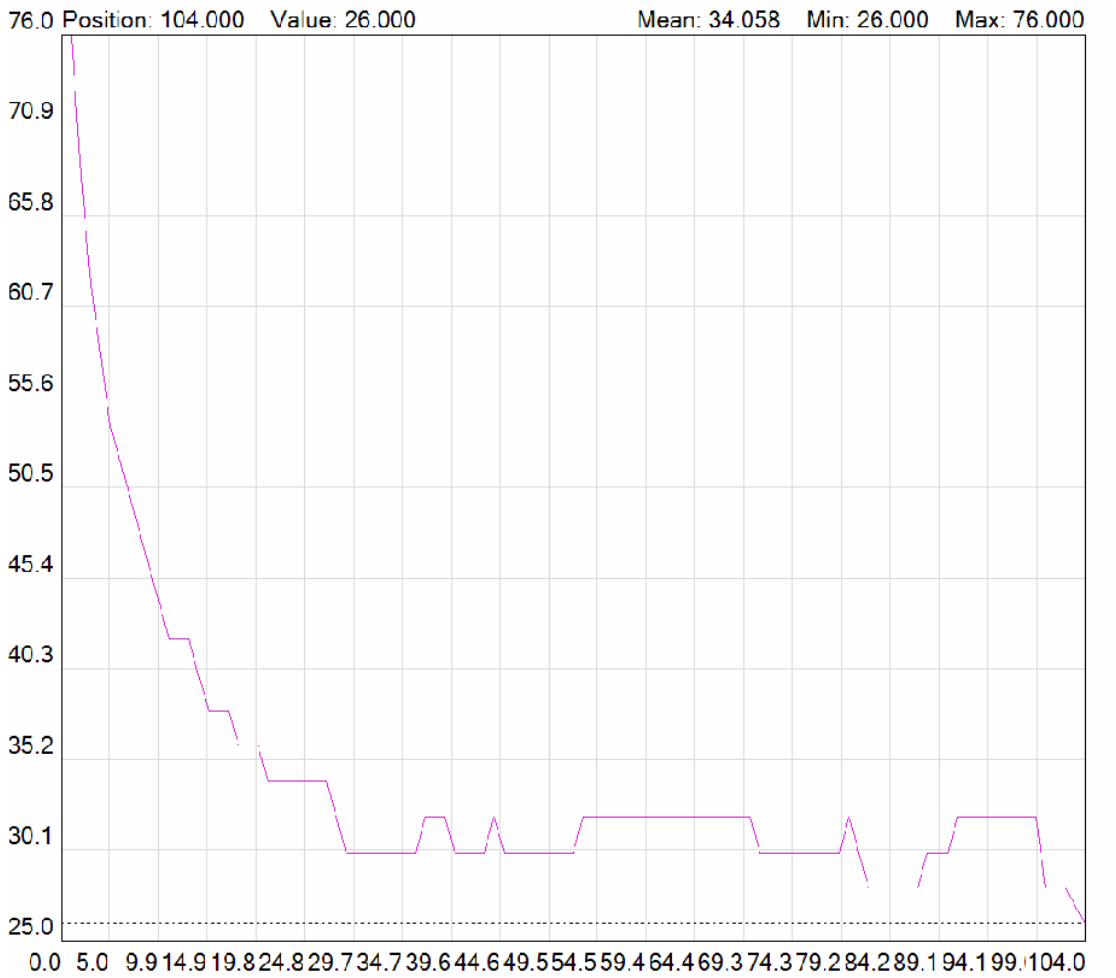



**obr. 3: porovnanie priebehu hľadania pre rôzne veľkosti tabu zoznamu**

Z experimentov vyplýva, že najvhodnejšia veľkosť pevného tabu zoznamu je približne n/4, kde n je počet miest.

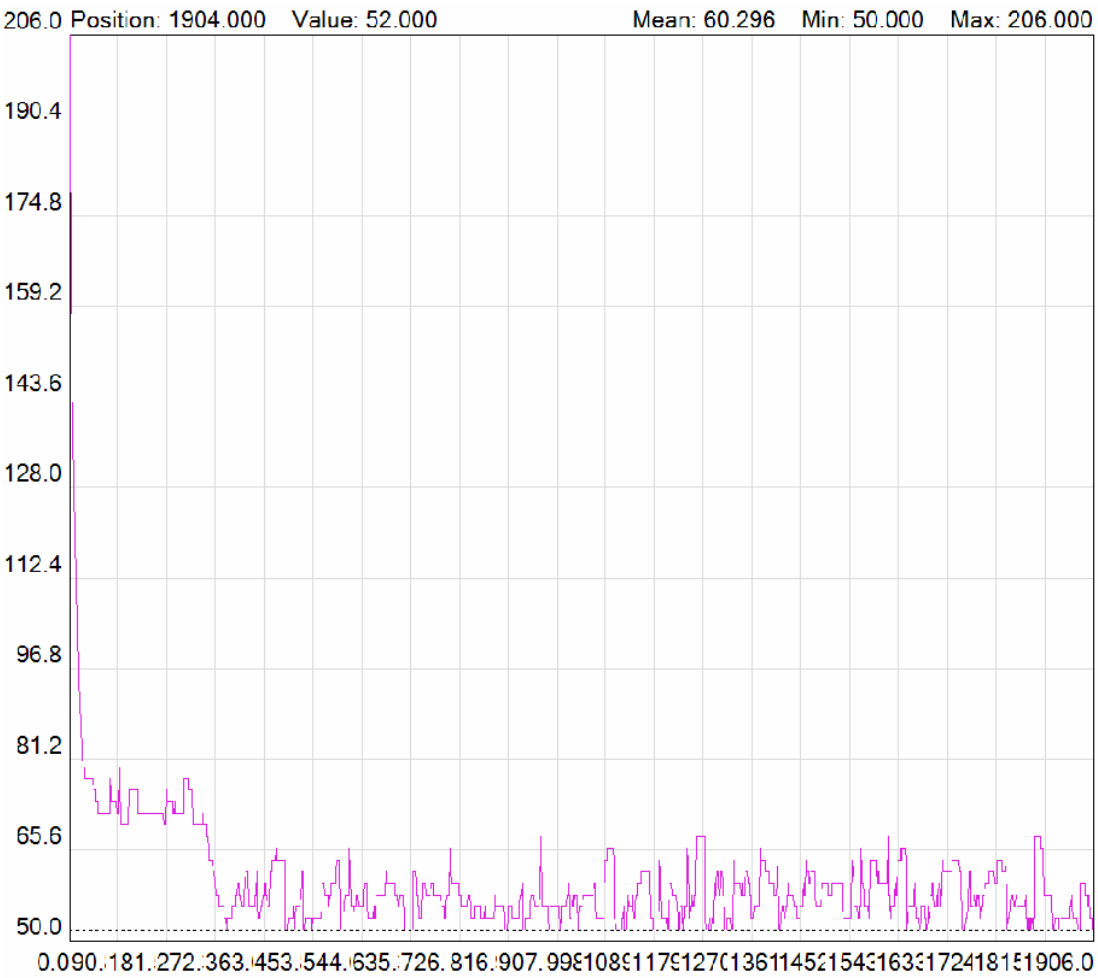
Nasledujúce grafy reprezentujú rovnaký problém pre premenlivú dĺžku tabu zoznamu.

n = 25:




**obr. 4 porovnanie priebehu hľadania pre premenlivú veľkosť tabu zoznamu**

n = 49:

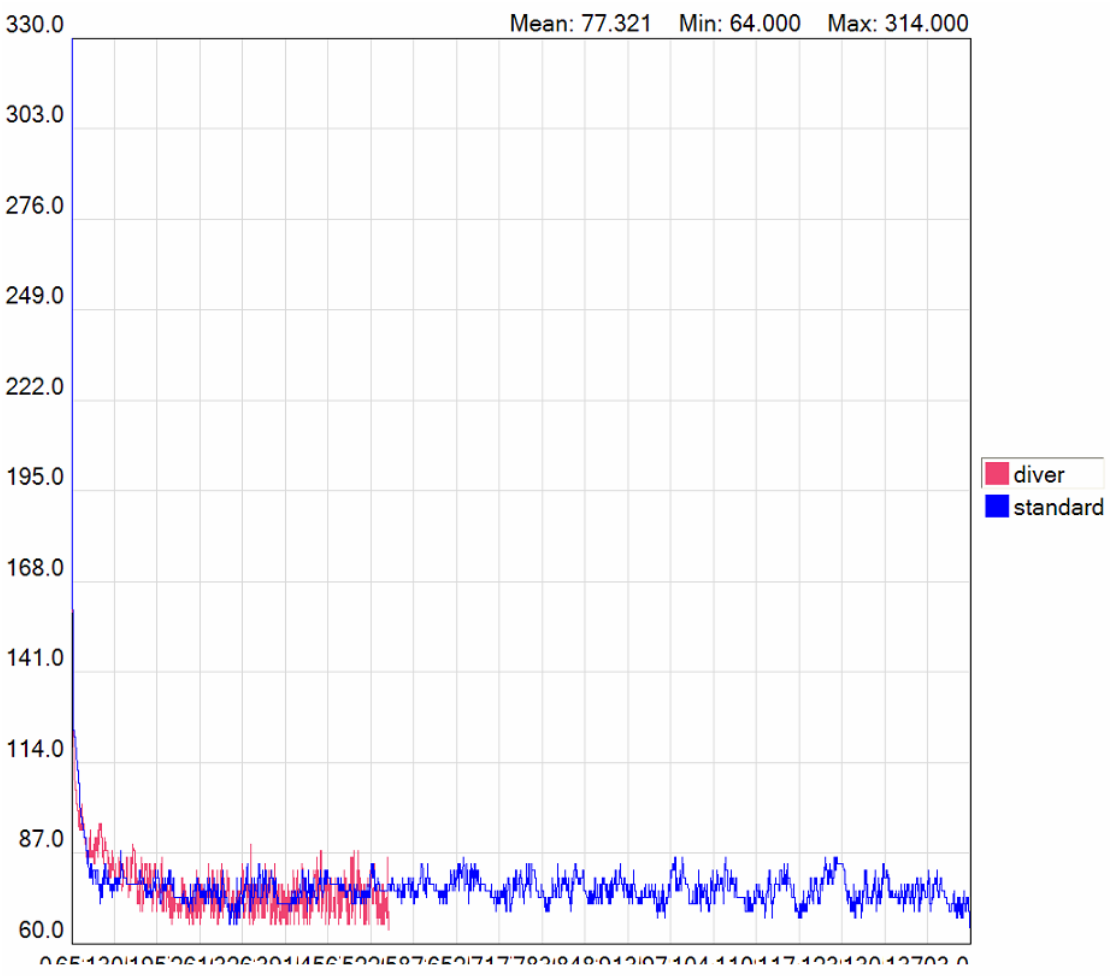



**obr. 5: porovnanie priebehu hľadania pre premenlivú veľkosť tabu zoznamu**

Z experimentov vyplýva, že dynamická veľkosť tabu zoznamu dosahuje podobné výsledky ako fixný tabu zoznam s veľkosťou n/4. Vzhľadom na implementovaný mechanizmus riadenia zmeny veľkosti tabu zoznamu sa dá predpokladať, že pri dokonalejšej implementácii by boli dosiahnuté výsledky pozitívnejšie.

Posledný experiment sa týka porovnania použitia diverzifikačných mechanizmov uvedených v opise riešenia a základného algoritmu (bez divezifikácie).

n = 64




**obr. 6: porovnanie priebehu hľadania pre algoritmus s a bez diverzifikácie**

Na grafe je znázornené priemerný priebeh hľadania s diverzifikáciou (asi 8000 iterácií), pre algoritmus bez diverzifikácie je to minimálny priebeh, nakoľko väčšinou sa jej nepodarí nájsť správne riešenie ani za 60 000 iterácií.

Z výsledkov možno konštatovať ,že diverzifikácia je integrálnou súčasťou TS.

# 5 ZHODNOTENIE

Podarilo sa mi ukázať, že pri vhodnom spôsobe realizácie sa dá veľkosť tabu zoznamu určovať reaktívne, čiže s ohľadom na aktuálne potreby hľadania. Rovnako doba strávená v určitej oblasti sa vďaka diverzifikácii môže redukovať na nevyhnutný počet cyklov.

Musím však konštatovať, že celkové výsledky ma sklamali, nakoľko metóda nedokázala nájsť optimum už pri 100 mestách (najlepší dosiahnutý výsledok bola cesta dĺžky 102 počas 60 000 iterácií), nehovoriac o tom, že proces hľadania trvá v porovnaní s inými metódami, ktoré sa využívajú na daný problém (najmä simulované žíhanie) neporovnateľne dlho. Na základe informácií, ktoré sami podarilo získať prostredníctvom rôznych zdrojov môžem konštatovať, že úspešné fungovanie metódy si vyžaduje jej dôkladné prispôsobenie problému a sofistikovaný mechanizmus intenzifikácie, diverzifikácie a mutácie

# 6 POUŽITÁ LITERATÚRA


[1] GLOVER, F.,LAGUNA, M.: Tabu search
http://www.dei.unipd.it/~fisch/ricop/tabu_search_glover_laguna.pdf

[2] HERTZ, A, TAILLARD, E., WERRA, D.: A tutorial on tabu search
http://www.cs.colostate.edu/~whitley/CS640/hertz92tutorial.pdf

[3] MISEVICIUS, A. 2004. Using iterated tabu search on travelling salesman problem
http://itc.ktu.lt/itc32/Misev32.pdf

[4] COOK, W.: Travelling Salesman Problem
http://www.tsp.gatech.edu/