Metaheuristike s jednim rješenjem

Motivacija

- Jednostavna metoda koja može riješiti neki optimizacijski problem
- Što manje parametara
- Iterativno poboljšava rješenja
- Ideja: Iterativna pohlepna metoda

Pohlepni algoritam pretrage

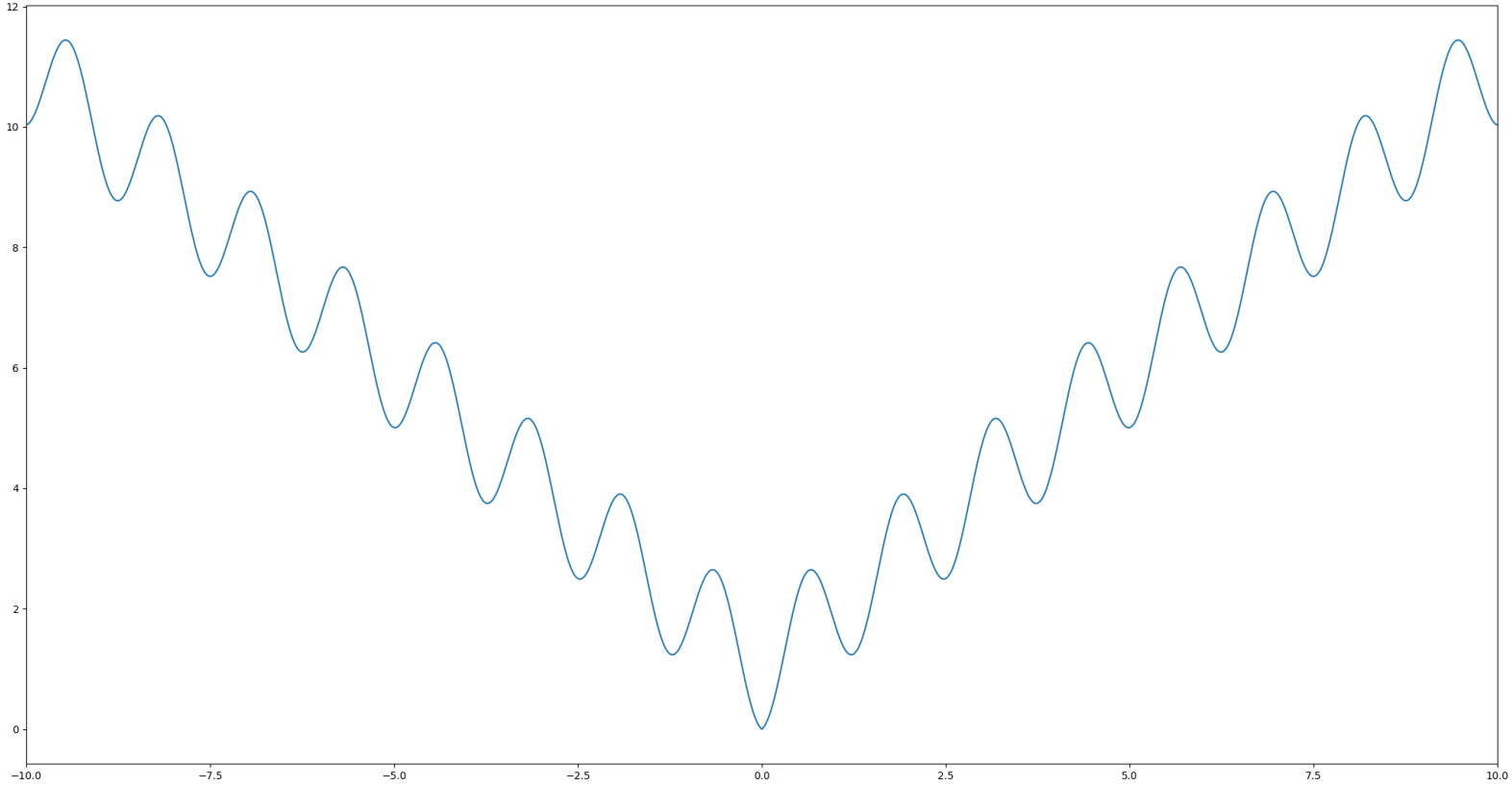
Generiraj početno rješenje *S*

Ponavljaj do kriterija zaustavljanja
    Generiraj susjedno rješenje *S'*
    Ako f(*S'*)>f(*S*)
        S=S'

Vrati *S*

Radni primjer

- Razmatramo minimizaciju funkcije: $f(x) = |x| + 1 - \cos(5x)$ u intervalu $x \in [-10, 10]$
- Minimum je u $x_{min} = 0$ i ima vrijednost $f(x_{min}) = 0$



Funkcija susjedstva

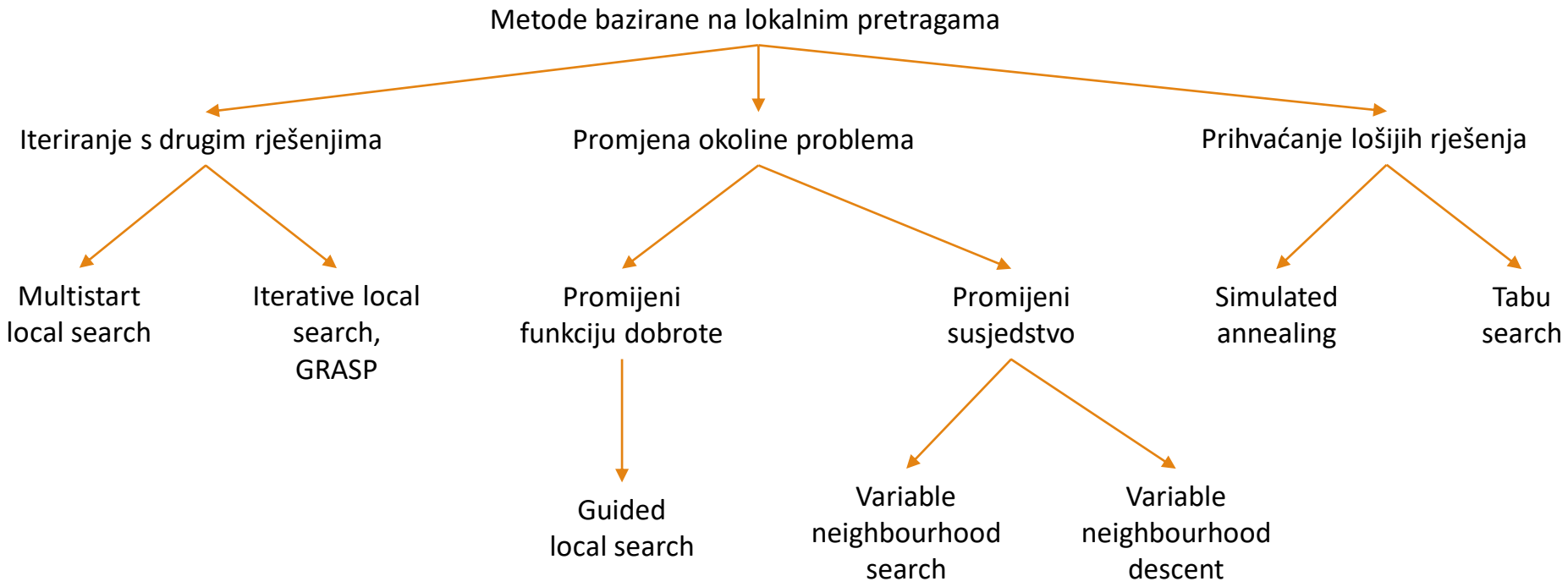
- Susjedno rješenje definiramo kao:
  - $x' = x + unif\_rand(-0.5,0.5)$
- Generiramo samo jedno susjedno rješenje

Pohlepni algoritam pretrage

- Prednosti:
  - Jednostavna implementacija
- Nedostaci:
  - Lako zapne u lokalnom optimumu
  - Nema mehanizam za izbjegavanje lokalnog optimuma!
  - Prihvaćanje isključivo boljih rješenja je loša strategija

- Kako riješiti problem?
  - Koristiti drugačiju funkciju susjedstva
    - Npr. Proširiti interval iz kojeg se generira rješenje
      - Koji je ovdje problem?
  - U algoritam ugraditi mehanizme koji izbjegavaju zapinjanje u lokalnim optimumima

Algoritmi s jednim rješenjem



Algoritmi s jednim rješenjem

- Osnova svih prethodnih algoritama je metoda za generiranja susjedstva
  - Kako generirati susjedna rješenja za neko trenutno rješenje?
  - Obično puno varijanti
  - Jako utječe na ponašanje algoritma -> potrebno dobro odabrati
- Razlike su u strategijama njihove primjene i dodatnim strategijama ugrađenim u algoritme radi poboljšanja uspješnosti
- Također, na različite načine izbjegavaju zapinjanje u lokalnim optimumima

Simulirano kaljenje

Motivacija

- Dozvoliti prihvaćanje lošijih rješenja tijekom pretrage
- Ideja, na početku više dozvoliti prihvaćanje lošijih rješenja kako bi se istražio prostor rješenja (diverzifikacija)
- Pri kraju rjeđe prihvaćati lošija rješenja kako bi se postigla konvergencija
- Kada i pod kojim uvjetima prihvatiti lošija rješenja?

Inspiracija

- Kaljenje metala u metalurgiji:
  - Metal zagrijavamo na veoma visoke temperature
  - Metal se postupno hladi kako bi se dobile jake kristalne strukture
  - Prilikom hlađenja, metal prelazi iz jednog u drugo energetsko stanje:
    - $\Delta \mathrm{E} = \mathrm{E}_2 - \mathrm{E}_1$
    - Ako je $\Delta \mathrm{E} < 0$ prijeđi u to stanje (odnosno uvijek prelazimo u niže energetsko stanje)
    - U suprotnom u više energetsko stanje se prelazi vjerojatnošću

$$\mathrm{P}(\Delta \mathrm{E}) = \mathrm{e}^{(\frac{-\Delta E}{kt})}$$

    - k – Boltzmannova konstanta, t – temperatura
  - Dakle ponekad se tijekom hlađenja dogodi da se prijeđe u stanje više energije kako bi se dalje moglo prijeći u stanje još niže energije

Preslikavanje na optimizaciju

Kaljenje metala	Simulirano kaljenje
Moguća stanja sustava	Prihvatljiva rješenja
Energija sustava	Funkcija kazne
Promjena stanja sustava	Prelazak u susjedno rješenje
Temperatura	Parametar koji simulira temperaturu

Pseudokod

Generiraj početno rješenje S

Postavi početnu temperaturu T

Ponavljaj do kriterija zaustavljanja
    Ponavljaj do stanja ekvilibrija
        Generiraj susjedno rješenje S’
        ΔE=f(S’)-f(S)
        Ako ΔE<0 //Uvijek prihvati bolje rješenje
            S=S’
        Inače
            Prihvati S’ s vjerojatnošću $e^{-\frac{\Delta E}{T}}$
    T=g(T) // Ažuriranje temperature
Vrati najbolje pronađeno rješenje

Odluke pri definiciji SA

- Standardne odluke:
  - Generiranje inicijalnog rješenja
  - Susjedstvo

- Odluke specifične za algoritam
  - Prihvaćanje poteza (susjeda)
  - Raspored hlađenja

Prihvaćanje poteza

- Uvijek prihvaćamo bolje rješenje
- Lošija rješenja prihvaćamo s vjerojatnošću:

$$P = e^{\frac{-\Delta E}{T}}$$

- Rješenje prihvaćamo ako je P>R, pri čemu je R slučajno generirani broj između 0 i 1
- Ako je temperatura *T* veća onda je vjerojatnost prihvaćanja rješenja veća:
  - $T \to \infty, P \to 1$ -> prihvaćamo svako rješenje
  - $T \to 0, P \to 0$ -> ne prihvaćamo lošija rješenja
- Ako je promjena u energiji veća, onda je vjerojatnost prihvaćanja rješenja manja

Plan hlađenja

- Bitan odabir
- Prebrzo hlađenje može dovesti do zapinjanja u lokalnom optimumu
  - Onemogućujemo prihvaćanje lošijih rješenja
- Presporo hlađenje uzrokuje previše nasumičnosti u pretraživanju
  - Može naštetiti konvergenciji
- U literaturi definirano mnogo planova hlađenja, koji smanjuju temperaturu

Linearan plan hlađenja

- Temperatura se u svakom koraku smanjuje za fiksan iznos

$$T = T - \beta$$

- Pri čemu je β zadana konstanta
- Odnosno

$$T_i = T_0 - i \times \beta$$

- gdje i predstavlja trenutnu iteraciju

Geometrijski plan hlađenja

- Temperatura se u svakom koraku smanjuje za fiksan iznos

$$T = \alpha T$$

- pri čemu je $\alpha \in [0,1]$
- Najpopularnija funkcija hlađenja
- Obično se koriste vrijednosti između 0.5 i 0.99

Logaritamski plan hlađenja

- Temperatura se u svakom koraku smanjuje za fiksan iznos

$$T = \frac{T_0}{\log(i)}$$

- Presporo da se primjenjuje u praksi
- Korisno jedino zbog dokaza globalne konvergencije algoritma

Vrlo sporo smanjenje

- Temperatura se u svakom koraku smanjuje za fiksan iznos

$$T = \frac{T_i}{1+\beta T_i}$$

- Ideja je da se hlađenje provodi izuzetno sporo
- No dozvoljava se samo jedna iteracija unutarnje petlje po temperaturi

Ostali planovi

- Nemonotono
  - Svi dosadašnji planovi monotono smanjuju temperaturu
  - Ideja je da se temperatura može ponovo povećati u kasnijim iteracijama
  - Cilj -> povećati diverzifikaciju u kasnijim iteracijama

- Adaptivno
  - Prethodni planovi su statički (plan je definiran a priori)
  - Ideja je da se brzina smanjenja prilagođava tijekom pretrage i prema karakteristikama problema
  - Npr. provođenje velikog broja iteracija na višim/nižim temperaturama

Stanje ekvilibrija

- Koliko se puta pretraga ponavlja za konkretnu vrijednost temperature
  - Unutarnja petlja algoritma
- Ovisi o veličini problema, posebice o veličini susjedstva
- Broj stanja koje će se posjetiti može biti:
  - Statično – broj tranzicija odredi se prije početka pretrage
  - Adaptivno – broj rješenja ovisit će o karakteristikama pretrage

Kriterij zaustavljanja

- Temperatura se je smanjila do određene razine, npr. T=0.01
- Stagnacija -> postignut određeni broj iteracija bez poboljšanja
- Ako u n uzastopnih faza ekvilibrija nije pronađeno novo rješenje

Slične metode

- Threshold acceptance
  - Deterministička varijanta simuliranog kaljenja
  - Prihvaćaju se samo ona rješenja koja su lošija od trenutnog rješenja za određenu razinu
- Record to Record Travel
  - Deterministčan algoritam
  - Rješenje se prihvaća ako je lošije za određenu razinu od najboljeg do sada pronađenog rješenja (record)
  - Devijacija se smanjuje tijekom iteracija

Slične metode

- Algoritam demona
  - Mnogo varijanti baziranih na ovom algoritmu
  - Postoji demon s ograničenom energijom (na početku ima neku inicijalnu vrijednost energije)
  - Kada je $\Delta E<0$ (novo rješenje je bolje) onda sustav gubi energiju i ona se dodaje demonu
  - Kada je $\Delta E>0$ novo rješenje se prihvaća samo ako vrijedi $\Delta E<D$
    - Demon mora imati dovoljnu količinu energije
    - On gubi tu količinu energije nakon što se rješenje prihvati

Pseudokod osnovnog algoritma demona

```
Generiraj početno rješenje S

Postavi početnu vrijednost D

Ponavljaj do kriterija zaustavljanja
        Ponavljaj do stanja ekvilibrija
                Generiraj susjedno rješenje S'
                ΔE=f(S')-f(S)
                Ako ΔE<D
                        S=S'
                        D=D-ΔE
 Vrati najbolje pronađeno rješenje
```

Varijante

- Algoritam ograničenog demona
  - Ako energija demona premaši početnu energiju, postavi ju na nju
- Algoritam kaljenog demona
  - Ako je postignut ekvilibrij, energija demona se umanji nekim planom hlađenja kao u simuliranom kaljenju
- Algoritam nasumičnog ograničenog demona i algoritam nasumičnog kaljenog demona
  - Dodaje se neki Gaussov šum tijekom prihvaćanja rješenja

Slične metode

- Niti jedna od prethodnih varijanti se pretjerano ne koristi
- Najpopularnija je najosnovnija varijanta simuliranog kaljenja

Tabu pretraga

Motivacija

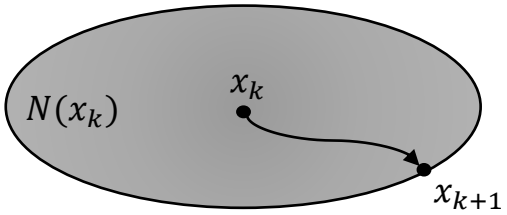
- Razviti algoritam koji dozvoljava prihvaćanje lošijih rješenja radi izbjegavanja zapinjanja u lokalnim optimumima
- Problem: moguće je da neki prostor pretražujemo više puta
- Želimo algoritam forsirati da pretražuje dijelove prostora rješenja koji još nisu istraženi

Izbjegavanje lokalnih optimuma

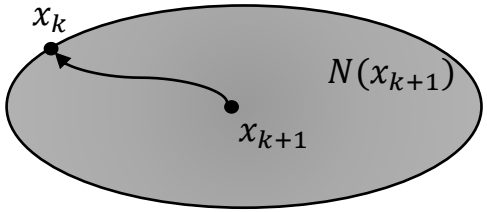
- Varijante pretraživanja:
  - Pretraži cijelo susjedstvo od trenutnog rješenja
    - Prihvati najbolje rješenje u susjedstvu kao trenutno rješenje, bez obzira je li ono bolje od trenutno najboljeg rješenja ili ne
    - Prisiljavamo algoritam da pretražuje neistražena područja
    - Potencijalni problem?
  - Uzmi nasumično jednog susjeda i istražuj dalje od njega
    - Kako znamo da već nismo pretražili rješenja blizu tog susjeda?
    - Opet istražujemo poznato područje i pretraga ne napreduje

Ciklusi

Iteracija k



Iteracija k+1



Tabu lista

- Pratiti rješenja u kojima sam nedavno bio
- Algoritmu ne dozvoliti pomake koji bi nas vratili u neka nedavno posječena rješenja ili njihovu okolicu
- Tabu lista -> lista prethodno posječenih rješenja
- Kada prihvaćamo rješenje:
  - Ako se rješenje ne nalazi u tabu listi
  - Ako se rješenje ne nalazi blizu rješenja koje se već nalazi u tabu listi

Organizacija tabu liste

- Tabu lista može pohranjivati:
  - Posjećena rješenja
    - Pamtimo rješenja koja smo prethodno posjetili
    - Može biti memorijski skupo i skupo za provjeriti nalazi li se rješenje u listi
  - Napravljeni potezi:
    - Pamte se potezi koji su napravljeni
    - Npr. Zamjena dva elementa u rješenju
    - Lakše provjeriti
    - Problem, mogu se zabraniti potezi koji bi u bliskoj budućnosti ponovo bili potrebni

Kriterij aspiracije

- Postoji opasnost da ne prihvatimo neki potez iako nas on vodi k boljem rješenju
  - Zbog tabu liste
- Kriterij aspiracije – dozvoljava da se pod određenim uvjetima neki pomaci koji se nalaze u tabu listi ipak prihvate
- Primjeri:
  - Prihvati pomak ako nas vodi k najboljem do sada postignutom rješenju
  - Prihvati pomak ako nas vodi k najboljem rješenju koje sadrži neko svojstvo
- Na ovaj način osiguravamo da uvijek prihvatimo bolje rješenje ako ga pronađemo

Memorija

- Služi za izbjegavanje ponovnog pretraživanja nekog prostora ili ciklusa u pretrazi
- Ne pomaže nam puno kod diverzifikacije ili intenzifikacije pretrage
- Ideja: uvesti više vrsta memorija s različitim ulogama

Vrste memorija

- Kratkoročna
- Srednjoročna
- Dugoročna

Kratkoročna memorija

- Cilj: izbjegavanje ciklusa
- Mogu se pamtiti:
  - Čitava rješenja (nepraktično)
  - Pomaci napravljeni u operatorima susjedstva
- Pamti se pomak ili inverzni pomak:
  - $s_j = s_i \oplus m$
  - $s_i = (s_i \oplus m) \oplus m^{-1}$

Kratkoročna memorija

- Nije praktično pamtiti sve dosadašnje poteze
- *tabu tenure* – broj iteracija tijekom kojih će se potez nalaziti u tabu listi
  - Prevelika vrijednost – algoritam spor i previše poteza je nedozvoljeno
  - Premalena vrijednost – ciklusi su češći
- Veličina tabu liste:
  - Statična – unaprijed zadana, ovisi o problemu
  - Dinamična – mijenja se ali bez informacija o pretrazi
  - Adaptivna – koristi se informacija o pretrazi za određivanje veličine

Srednjoročna memorija

- Cilj: intenzifikacija
- Iskoristiti informacije o postojećim rješenjima kako bi se pretraga usmjerila u bolje područje
- Primjer:
  - Reinicijalizacija najboljim rješenjem iz srednjoročne memorije
  - Reinicijalizacija prosjekom svih rješenja iz srednjoročne memorije

Srednjoročna memorija

- Šta pamtimo u njoj?
  - Cijela rješenja
  - Karakteristike rješenja
- Kako odrediti dobre karakteristike koje pamtimo?
- Npr. pamtimo koliko su se u prošlosti na nekim mjestima pojavljivali određene karakteristike
- Fiksiramo karakteristike s najvećim brojem uzastopnih pojavljivanja
- Tjeramo algoritam da se fokusira na druge karakteristike

Dugoročna memorija

- Cilj: diverzifikacija
- Usmjeriti algoritam da pretražuje neistražene prostore rješenja
- Slično kao kod srednjoročne memorije pamtimo čitava rješenja ili neke njihove karakteristike
- Najčešće pamtimo pojavljivanja od početka algoritma

Dugoročna memorija

- Kada pretraga zapne, reinicijaliziramo ju s najboljim dosadašnjim rješenjem
- Rješenje se modificira postavljanjem slabije istraženih atributa u rješenju kako bi se pretraga usmjerila u novo područje
- Strategije:
  - Diverzifikacija ponovnim pokretanjem – pokrećemo pretragu s novim rješenjem i njega modificiramo
  - Kontinuirana diverzifikacija – uvodimo kaznu u funkciju cilja koja kažnjava rješenja koja su često posječena

Pregled memorija

Vrsta memorije	Uloga	Reprezentacija
Tabu lista (kratkoročna memorija)	Sprječavanje ciklusa	Posječena rješenja ili potez
Srednjoročna memorija	Intenzifikacija	Praćenje karakteristika nedavnih rješenja
Dugoročna memorija	Diverzifikacija	Praćenje učestalosti rješenja prilikom pretrage

Algoritam tabu pretrage

```
s=s0 // inicijaliziraj početno rješenje
Inicijaliziraj tabu listu, srednjoročnu i dugoročnu memoriju
Ponavljaj
     Pronađi najboljeg susjeda s' //koji nije u tabu listi
     s=s'
     Ažuriraj tabu listu i memorije
     Ako zadovoljen kriterij_intenzifikacije
          Provedi intenzifikaciju rješenja
     Ako zadovoljen kriterij_diverzifikacije
          Provedi diverzifikaciju rješenja
Dok kriterij zaustavljanja nije zadovoljen
Vrati najbolje do sada pronađeno rješenje
```

Primjer

- Kako definirati Tabu listu?
  - Provjera jesmo li u okruženju neke točke gdje smo već prije bili
- Kako definirati srednjoročnu memoriju (intenzifikacija)
  - Ponovno pokretanje algoritma iz najboljeg rješenja u toj listi
  - Izračunaj srednju vrijednost točaka iz liste i pokreni pretragu iz nje
- Kako definirati dugoročnu memoriju (diverzifikacija)?
  - Gledati u kojim intervalima imamo najmanje rješenja u prošlosti i generirati novo trenutno rješenje iz tog intervala
  - Dodati neku kaznu ako se trenutno rješenje kreće u prostoru kojeg smo već dobro istražili

Problemi

- Kako definirati poteze koji će se spremati u tabu listu
  - Za složenije probleme nije trivijalno
- Sporost pri pretraživanju je li neki potez zabranjen
- Kako definirati dugoročnu i srednjoročnu memoriju
- Puno parametara:
  - Veličina tabu liste
  - Kriterij aspiracije
  - *Tabu tenure*
  - Strategije kada koristiti intenzifikaciju ili diverzifikaciju

Zaključno

Zaključak

- Simulirano kaljenje i tabu pretraga su iznimno jednostavne metode za rješavanje optimizacijskih problema
- Relativno uspješno izbjegavaju lokalne minimume
- Koja metoda je bolja?
  - Sjetimo se no-free-lunch teorema
- Korištenje jednog rješenja može dovesti do toga da postupak ipak ne pretraži dobro čitav prostor rješenja
  - Motivacija za korištenje populacijskih algoritama -> metoda koje koriste skup rješenja nad kojim obavljaju različite operacije