Ispit iz algoritamskih heuristika april 2024

Izvestaj uz zadatak 1 – SA za resavanje Sudoku puzle

Profesor : Stanisa Dautovic

Student : Petar Stamekovic E1-11/2023

1. Sadrzaj rada

Ova dokumentacija prati rad zadatka broj 1 sa ispita iz predmeta Algoritamske Heuristike aprila 2024. Zadatak 1 je bilo resavanje Sudoku puzzle metaheuristikom simuliranog kaljenja. Ideja je bila da se koristi kostur heuristike preuzete sa turskog sajta *yarpiz[3]* I da se odredjenim modifikacijama kod prilagodi resavanju pomenutog problema. Za ideju resavanja je koriscen dokument "*Metaheuristics can Solve Sudoku Puzzles*"[1] a upravo te ideje cu navesti u ovoj dokumentaciji (taj rad ce biti prilozen uz ovu dokumentaciju). Takodje, prilozen je veci broj instanci problema 3x3 sudoku puzle klasifikovani po tezini (L – lako , S – srednje , T – tesko). Prenosim napomenu profesora da neke teske instance nisu valide, pa u zamenu za njih stoje instance iz foldera *sudoku_instance_2.* Pomenuti kostur se nalazi u folderu *src* I ogradjujem se da ovo nije moj kostur vec sa pomenutog sajta.

2. Uopsteno o Sudoku puzli

Sudoku predstavlja jednu od najpopularnijih i najizazovnijih puzli ikada smisljenih. Potice od stare igre *Latin squares*, a originalno nastaje u Japanu gde doslovno znaci "*solitary numbers*". Puzla je formata $n^2 x n^2$ I podeljena je na kvadrate formata *n x n.* Puzla je data delimicno popunjena, a dopunjava se logickim razmisljanjem gde se moraju zadovoljiti sledeca 3 kriterijuma :

1. Unutar jednog kvadrata sme biti samo po jedna kopija brojeva od 1 do 9.
2. Unutar jedne vrste cele puzle moze biti samo po jedna kopija brojeva od 1 do 9.
3. Unutar jedne kolone cele puzle moze biti samo po jedna kopija brojeva od 1 do 9.

Sustiniski, *n* moze biti proizvoljan broj medjutim, najcesce se koristi red puzle 3 ili eventualno 4. U ovom radu I zadatku se koristi red 3, dakle puzla je formata 9x9 a kvadrati su formata 3x3.

3. Algoritam simuliranog kaljenja (*Simulated annealing)*

Simulurano kaljenje predstavlja jednu od najstariji I najpoznatijih metaheuristika I prakicno se moze reci da spada u *hall of fame* istih. Prati proces obrnut od topljenja metala I koristi **temperaturu** kao kontrolni parametar. Oformljava se kristalna resetka (fazni prelaz) kako se temperatura spusta (materijal se hladi). Postoje **inicijalna** I **finalna** temperatura kao I parametar *alfa* koji diktira brzinu opadanja temperature. Ovo se obuhvata takozvani *cooling schedule* koji je monotono opadajuca funkcija (moze biti linearna, eksponencijalna…) I moze drasticno da utice na rezultate. Od ovoga u opstem algoritmu zavise svojstva novonastalih legura! Sta se zapravo ovde desava? Kako se temperatura smanjuje elektroni imaju sve manje kineticke energije I "*brzo*" se uspostavlja kristalna resetka, gde elektroni zauzimaju neko ravnotezno stanje i stabilizuju se. Polako se pronalazi optimalno resenje uz eventualne pretrage lokalnog minumuma. Obratiti paznju da se veliki skokovi I promene desavaju upravo pri visokim temperaturama. Dakle, optimalno je poceti od relativno visoke temperature I u dovoljnom intervalima vremena sa nekim faktorom smanjivati istu (preporucuje se da *alfa* bude izmedju 0.8 I 0.95). Bitni pojmovi za ovaj rad su sledeci :

- X ce biti prostor potencijalnih resenja
- X' ce biti naredno resenje koje se dobija primenom *neighbourhood* operatora. Ono moze biti **bolje** (sto je dobro) I **losije** (sto nekad mozemo prihvatiti a desava se sa odredjenom verovatnocom $e^{-delta/t}$
- *N* je neighborhood operator I pomocu njega se na neki nacin(odredjeni) dolazi do narednog resenja (susednog). Videcemo da je u ovom radu to zamena dva *non-fixed* polja.
- *C* je *cost* funkcija koju je potrebno minimizovati I pri njenom minimumu dobijamo "*optimalno"* resenje.
- *T* je temperatura I kao sto smo rekli predstavlja kontrolni parametar

Osnovna ideja je sledeca : Generisi pseudoslucajno resenje I pomeri se u naredno (bolje ili nekad cak I losije kako bi pobegao iz eventualnog lokalnog minimuma). Glavni problem je odarbir adekvatnog *cooling schedula* I ovo se uglavnom resava eksperimentalnim putem.

4. Nacin resavanja Sudoku puzle

Pomenuti rad "*Metaheuristics can Solve Sudoku Puzzles*"[1] na pocetku daje ideju o kreiranju instanci problema, ali obzirom da su nama one date, necu zalaziti u detalje oko toga. Predstavljena je ideja o tzv. *Root solution* – inicijalnom resenju koje nastaje od prazne matrice I popunjavanjem iste po nekom algoritmu. Ideja za puzlu je da bude logicki resiva bez potrebe za nagadjem resenja (ne zelimo *backtracking* jer on dize vreme proracuna) I da resenje bude jedinstveno. U tekstu je dat rad koji daje ideju o resavanju Sudoku puzle preko CNF I formula zadovoljivosti, ali kao sto rekoh necu zalaziti u ovo.

U radu su date sledece ideje za realizaciju algoritma koji resava samu Sudoku puzlu:

- *Reprezentacija I neighbourhood operator*
- *Cost funkcija*
- *Algoritam simuliranog kaljenja*

U nastavku cu malo detaljnije reci nesto o svakoj od ovih tacki I kako se ona realizuje u mom kodu.

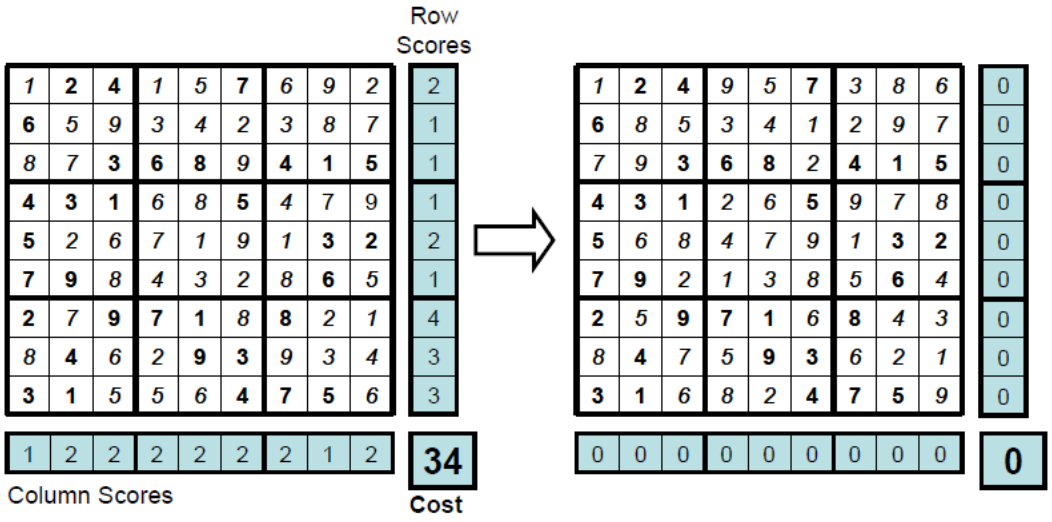
4.1. Reprezentacija I Neighbourhood operator

U prilozenim instancama problema sa vrednostima 0 su oznacena prazna polja ili *non-fixed* kako cu ih ja zvati u ovom radu. Ta polja su inicijalno prazna I upravo ona ce biti ta koja ce se *swap*-ovati preko *neighbourhood* operatora. Pocetno resenje se dobija popunjavanjem *non-fixed* polja nasumicno ali da se postuje kriterijum koji zahteva jednstvenost broja u jednom kvadratu (kriterijum broj 1 od gore). Ovime obezbedjujemo da tokom celog rada algoritma taj kriterum bude ispunjen, jer se taj *swap* od strane *neighbourhood* operatora desava iskljucivo u okviru jednog kvadrata!

4.2. Cost funkcija

Ova funkcija je zaduzena da ispita I kontrolise validnost preostala dva kriterijuma, jedinstvenost brojeva u svim vrstama I kolonama Sudoku puzle. Funkcija ide redom po svakoj vrsti I racuna koliko brojeva iz opsega 1-9 fali u istoj. Ovo cemo nazvati *privremena cena* ili *cena jedna vrste*. Na primer sledeca vrsta 1 5 4 3 2 2 7 8 9 ce imati cenu 1 jer broj 6 fali a vrsta 1 5 5 4 7 8 9 8 1 ce imati cenu 3 jer fale brojevi 2 3 6.

Funkcija prolazi kroz sve vrste I sve kolone I sumira sve *privremene cene* kako bi se dobila *finalna cena* koju je potrebno minimizovati. Resenje sa cenom 0 je jedino validno jer to znaci da nijedan broj nigde ne fali I da nema duplikata po kolonama I vrstama. Slika za ovaj koncept je data u pomenutom radu a ja cu je prikazati na slici 1.



Slika 1 : *Cene kolona/vrsta I finalna cena resenja [1]*

4.3. *SA algoritam*

Pod pretpostavkom da imamo prethodne 3 stvari spremne, integirsanje sa SA algoritmom je trivijalno. Potrbno je promeniti model, cost funkciju I neighbourhood operator I eksperimentisati sa vrednostima I temperaturom. Od kontrolnog parametara *t* zavisi uspesnost SA algoritma. Prvo je bolje dati sto vecu temperaturu kako bi se uglavnom svako resenje prihvatilo (autor preporucuje podesavanje da se oko 80% resenja inicijalno prihvata) a onda postepeno smanjivati (*cooling schedule)* I time ciniti algoritam halavijim (*greedy)*.

5. Strukura mog koda I odradjene modifikacije

U ovom odeljku cu u kratkim crtama objasnti sve fajlove unutar moje varijante matlab koda, sta se kojim postize kao I funkcije u svakom.

1. ***Main.m*** - Ovaj fajl samo pokrece rad *SA* algoritma.

2. ***CreateModel.m*** - U ovom fajlu se pomocu funkcije *readmatrix* cita matrica iz tekstualnog fajla u pomocnu promenljivu I vraca istu kao povratnu vrednost funkcije.

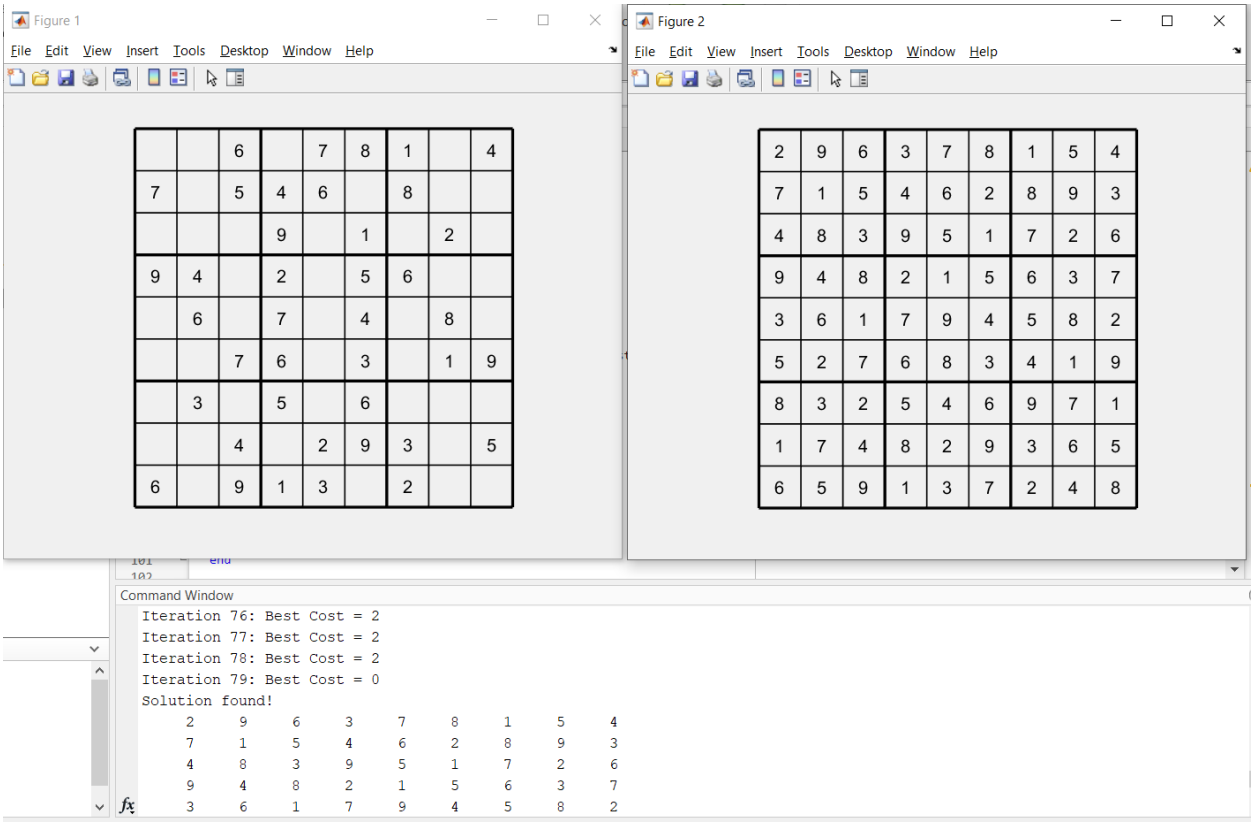
3. ***CreateRandomSolution.m*** – Poziva se funckija koja popunjava matricu po kriterijumu 1 (pomeutno inicijalno resnje) I u konzoli ispisuje medjuresenje.

4. ***CreateNeighbor.m*** – Relizacija *neighbourood* operatora. Ovde se ponovo ucitava inicijalna matrica kako bi ovde mogli da ponadjemo lokacije *non-fixed* celija. Radi se randomizacija kvadrata unutar kojeg se desava *swap*. Nakon toga se pronalaze *non-fixed* lokacije I randomizacijom se biraju dva takva polja. Pomocu standardne *swap with temp* sekvence se menjaju vrenosti celija. Promenjen kvadrat se utiskuje u originalnu matricu koja se vraca kao povratna vrednost.

5. ***CalcDiff.m*** - Cost funkcija koju je potrebno minimizovati. Prvo se ide po svakoj vrsti, uzima se svaka vrsta posebno kao niz, I pomocu funkcije *setdiff* koja vraca niz brojeva kojih nema u nizu 1:9 povecavamo cenu. Ovo se radi I za sve kolone cime se dobija originalna cena koja se vraca kao povratna vrednost funckije.

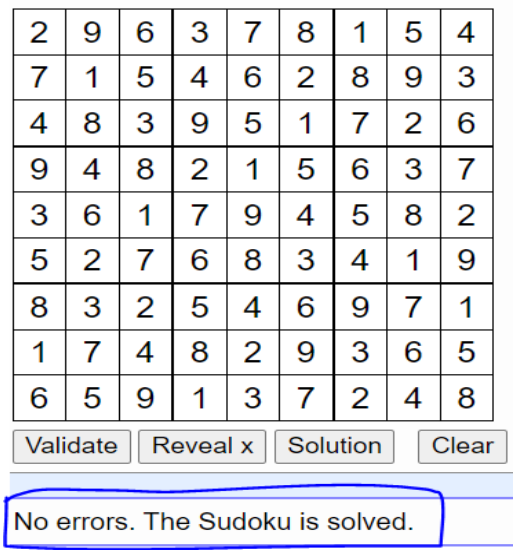
6. ***sa.m*** – Sustinski glavni fajl koji predstavlja I sam algoritam. Sastoji se iz vise blokova koje cu ovde malo detaljnije opisati. Kod pocinje sa ciscecem svih figura I komandog prozora.
    1) *Problem definition* – Sekcija u kojoj se kreira (u nasem slucaju loaduje) instanca problema, tj. prazna matrica koja se ispisuje i definise *cost* funkcija.
    2) *SA parameters* – Sekcija u kojoj se definisu potrebni parametri za rad algoritma, u mom slucaju broj iteracija I sub-iteracija, inicijalna temperatura I parametar *alfa* za smanjivanje temperature.
    3) *Initialization* – Kreiranje inicijalnog resenja pomocu randomizacija I kalkulacija inicijalne cene. Inicijalizacija temperature.
    4) *SA Main loop* – Glavna petlja algoritma u kojoj se prolazi kroz iteracije. Prati gore pomenutu ideju I nacin rada algoritma u vezi prihvatanja boljih/losijih resenja. Smanjivanje temperature I uslov terminacije kada se pronadje konacno resenje.
    5) *Results* – Sekcija u kojoj se samo ispisuje konacno resenje pozivom funkcije za ispis.
    6) *Additional functions* – Definisana funkcija za lepsi ispis resenja. Ispis se takodje radi u standardnoj matricoj formi u konzoli, medjutim za ljudsko oko je lepsi preko figure.

Rad algorima ce biti demonstriran na odbrani a slika koja potvrduje valdinost algoritma I njegovog rada je data na slici 2.



Slika 2 : Rezultat rada algoritma

Vidimo da je algoritam zavrsio svoj rad I pronasao resnje u 79 iteracija u ovom slucaju (lak slucaj) medjutim ovaj broj ce naravno varirati kako od parametara tako I od situacije I same randomizacije. Napomena : Za pokretanje algoritma je potrebno promeniti *filepath* I u *CreateModel* I u *CreateNrighbour* fajlu. Ovo resenje je validirano I na online Sudoku checkeru I to je prikazano na slici 3.



Slika 3 : Potvrda validnosti resenja[2]

6. Zakljucak

Ovime bih zakljucio ovu dokumentaciju I jos jednom se osvrnuo na istu. U ovom radu je receno nesto malo o heuristici simuliranog kaljenja, pracen je clanak "*Metaheuristics can solve sudoku*" I koriscena je ideja iz tog artikla kako bi se realizovao ovaj projekat. Uradjene su modifikacije I algoritam uspesno resave date instance u odredjenom broju iteracija. Prosao sam kroz osnovne pojmove ovog algoritma kao sto su *neighbourhood* operator, inicijalno resenje, pojam bolje I losijeg resenja I slicno. Otvoren sam za sve kritike, komentare I evenutalne dodatke za izradu ovog rada.

7. Literatura I reference

1. R. Lewis – "*Metaheuristics can Solve Sudoku Puzzles*"
2. https://onlinetoolz.net/sudoku
3. https://yarpiz.com/223/ypea105-simulated-annealing
4. Beleske sa predavanja kod profesora S. Dautovic