6. domaća zadaća – mravlji algoritmi

Zadatak.

Napisati implementaciju algoritma *Max-Min* Ant System koja rješava problem trgovačkog putnika. Problem trgovačkog putnika je problem pronalaska najkraćeg ciklusa kroz *n* gradova pri čemu trgovački putnik mora posjetiti svih *n* gradova i u svaki smije doći samo jednom. Poznata biblioteka ovakvih problema dostupna je na Internetu pod nazivom TSPLIB:
http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

U okviru ove zadaće razmatramo simetrične TSP-ove koji su dostupni na:
http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/

Kontrole radi, duljine optimalnih ciklusa mogu se pronaći na sljedećoj adresi:
http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html

Algoritam *Max-Min* Ant System (MMAS) obradili smo na predavanju. Opis algoritma Ant System dan je u slideovima koji su dostupni u repozitoriju a razlike između tog algoritma i algoritma *Max-Min* Ant System naveli smo na predavanju. S obzirom da performanse mravljih algoritama padaju kod problema koji imaju vrlo veliko grananje, implementirajte inačicu MMAS algoritma koja koristi liste kandidata. Kod te inačice za svaki se grad pripremi lista kandidata koji će se razmatrati za sljedeći grad. Ovisno o kriteriju po kojem se biraju gradovi koji će biti kandidati, ove se liste mogu graditi dinamički tijekom izvođenja algoritma ili statički prilikom pokretanja programa. Vaš je zadatak napisati potporu za statički izgrađene liste kandidata: po pokretanju programa kao argument naredbenog retka dobit ćete željenu veličinu liste kandidata – označimo je *k*. Za svaki grad *g* od postojećih *n* gradova pronaći ćete upravo *k* njemu najbližih gradova i njih ćete smjestiti u listu kandidata cl_g. Time ćete dobiti *n* lista kandidata, svaka veličine *k*.

Prilikom izgradnje rješenja, svaki mrav pamti do tada posjećene gradove. Kada se nađe u gradu *i*, mrav najprije dohvaća listu kandidata cl_i te razmatra ima li unutra gradova koji još nisu posjećeni. Ako ima, mrav proporcionalnim-slučajnim pravilom kao sljedeći grad bira jedan od tih neposjećenih gradova. Tek ako su svi gradovi iz liste kandidata cl_i već posjećeni, mrav uporabom proporcionalnog-slučajnog pravila bira jedan od svih neposjećenih gradova.

Napišite program `hr.fer.zemris.optjava.dz6.TSPSolver` koji preko komandne linije prima sljedeće argumente:

- *file*: datoteka s opisom TSP problema,
- *k*: veličina liste kandidata,
- *l*: broj mrava u koloniji,
- *maxiter*: maksimalni broj generacija.

Parametre *alfa*, *beta* i *a* u vašoj implementaciji također izdvojite iz samog algoritma ali i postavite u metodi `main`; njih nije potrebno predavati kao argumente naredbenog retka.

Naputci za implementaciju

1. Udaljenost između gradova računajte kao euklidsku: u datoteci s opisom problema nalazit će se, za svaki grad, njegove *x* i *y* koordinate u 2D koordinatnom sustavu – udaljenost računajte na uobičajeni način. Ove udaljenosti možete izračunati unaprijed (po pokretanju programa) i možete ih zapamtiti u nekoj pomoćnoj strukturi. Štoviše, ako tijekom izvođenja programa parametar *alfa* ne mijenjate, kompletnu heurističku informaciju koja se koristi u slučajnom-proporcionalnom pravilu možete unaprijed izračunati (primjerice, umjesto da pamtite matricu udaljenosti $[d_{ij}]$, možete odmah izračunati i pamtiti $[(1/d_{ij})^{beta}]$ čime ćete izbjeći

potrebu za skupim dijeljenjem i još skupljim potenciranjem svaki puta.

2. Nemojte kao početni grad za svakog mrava koristiti isti grad: neka svaki mrav prilikom konstrukcije rješenja slučajno odabere početni grad.

3. Parametar *a* ne mora biti hardkodiran: možete ga dinamički odrediti prilikom pokretanja programa ovisno o veličini problema (tj. broju gradova) i veličini liste kandidata.

4. Razmislite kako ćete odrediti koji je prikladan uvjet za detekciju stagnacije: MMAS očekuje da se prilikom stagnacije napravi reinicijalizacija feromonskih tragova na aktualni τ_{max} čime će ponovno krenuti postupak istraživanja koji bi mogao dovesti do novog boljeg rješenja. Budite svjesni da je to, međutim, dosta skupa operacija: kako algoritam radi s malom stopom isparavanja, nakon reinicijalizacije trebat će dosta vremena da algoritam ponovno počne "proizvoditi" dobra rješenja – pazite da Vam se ne dogodi da reinicijalizaciju radite prečesto pa da algoritam nakon jedne reinicijalizacije prekinete novom prije no što je ovaj uopće uspio dovoljno modulirati feromonske tragove da bi mogao generirati išta dobroga.

5. Razmislite na koji ćete način utvrđivati koji mrav radi ažuriranje feromonskih tragova: najbolji u trenutnoj iteraciji ili globalno najbolji. Sjetite se što smo pričali na predavanju: neka istraživanja pokazuju da je u kasnijim koracima algoritma bolje da ažuriranje češće radi globalno najbolji mrav (onaj *virtualni*) dok u ranijim fazama može biti bolje da to radi najbolji mrav trenutne iteracije.

6. Prilikom izračuna ukupne duljine ciklusa nemojte zaboraviti uračunati i put koji čini povratak iz *n*-tog grada rute u 1. grad rute.

Po završetku rada, algoritam treba na zaslon ispisati najkraću rutu, i to počev s prvim gradom koji je naveden u datoteci (ako gradove indeksiramo, njegov bi indeks bio 1). Kako je ruta ciklus, to uvijek možete napraviti. Također, potrebno je ispisati kolika je duljina tog ciklusa.

Rad programa ispitajte na sljedećim problemima iz TSPLIB-a:

- bays29
- att48
- ch150
- pr2392

Iako nije nužno da Vaš algoritam uspije pronaći optimalan ciklus za problem pr2392 (dapače, za bilo koji veći problem), spomenut ću kao zanimljivost da je taj put moguće pronaći uz vrlo malu veličinu liste kandidata: dovoljno je da lista kandidata sadrži 8 najbližih gradova (što znači da algoritam mora pronaći jedan pravi odabir između poprilici njih $1.55 * 10^{2160}$, što je ipak značajno manje od broja permutacija s 2392 grada).

Napomene:

Rok za predaju Eclipse projekta je sljedeći četvrtak do termina predavanja (14:00).