RZHP, 5. domača naloga

Tadej Ciglarič, Matija Čufar

Opis problema

V plesni šoli se otroci učijo ples v vrsti. Žal se vsi med seboj ne razumejo, nekateri pa si želijo plesati čim bližje, zato jih je potrebno čim bolje rezporediti v vrsto.
Imamo seznam oseb in poleg katerega je napisano kdo si s kom želi plesati. Kvaliteto razporeditve dobimo tako:

- Za vsako osebo, ki stoji med dvema osebama, kjer prva hoče plesati z drugo dobimo eno negativno točko.
- Za vsako osebo, ki stoji med dvema osebama, kjer prva noče plesati z drugo dobimo eno pozitivno točko.
- Če imata obe osebi iste želje, se točke podvojijo, če imata nasprotne, pa se izničijo.

Problem je zelo zahteven, saj za n oseb obstaja $n!$ možnih razporeditev.

Predstavitev problema v kodi in točkovanje rešitev

Problem sva predstavila z matriko sosednosti. Matriko sva naredila zaradi lažjega računanja naredila simetrično.
Primer matrike za prvi set podatkov:

```
##          Patricia Lea Peter Anna Matt John
## Patricia        0  -2    -1    0   -1    0
## Lea            -2   0    -1    0    0    0
## Peter          -1  -1     0    0    0    0
## Anna            0   0     0    0   -1   -1
## Matt           -1   0     0   -1    0    1
## John            0   0     0   -1    1    0
```

Dobra stran matrike sosednosti je tudi to, da je iz nje lahko izračunati kvaliteto razporeditve. Funkcija za točkovanje izgleda takole:

```
  # scoreFun : adjacency matrix, permutation -> score
  scoreFun <- function(A, perm)
  {
    n <- nrow(A)
    A <- A[perm, perm]

    # row indices
    Ar <- matrix(rep(1:n, n), nrow=n)

    # distance between two students is equal to:
    #   col.index - row.index - 1 * -weight
    sum(((t(Ar) - Ar - 1) * -A)[upper.tri(A, diag=F)])
}
```

Metode

Za metodi, ki jih bova uporabila sva si izbrala variable neighbourhood search (VNS) in memetic algorithm (MA). Za VNS sva se odločila zato, ker za naš problem obstaja veliko možnih soseščin. Za MA sva se odločila, ker se nama je algoritem zdel zanimiv in sva ga hotela preizkusiti.
Vse algoritme sva implementirala sama.

Local search in VNS

Ker je local search poseben primer VNS, sva ta dva algoritma razvijala vzporedno.
Preizkusila sva naslednje soseščine:

1. Zamenjaj dva soseda.
2. Razdeli permutacijo na dva dela in jih zamenjaj.
3. Zamenjaj poljubna dva elementa v permutaciji.
4. Izberi del permutacije in v njem obrni vrstni red.

Implementirala sva tudi funkcijo, ki združuje dve soseščini. To stori tako, da vrne permutacije, ki so v drugi soseščini permutacijam prve soseščine.

Prvi dve soseščini sta manjši od drugih dveh, saj pri vhodni permutaciji velikosti n vrneta n sosedov, drugi dve pa n^2. Zaradi pre velikih okolic teh dveh nisva kombinirala z ostalimi. Najboljša okolica je bila tretja, najslabša, druga pa je bila skoraj neuporabna. Algoritme sva testirala tako, da sva jim dala fiksno količino časa jih z naključnimi začetnimi pogoji zaganjala dokler se čas ni iztekel. Najboljši rezultat je dal običajen local search z okolico 3.

Memetic algorithm

Memetski algoritem sva implementirala tako, da sva najprej implementirala preprost genetski algoritem in ga nato razširila z dodatnimi koraki.

Algoritem začne z generiranjem velike količine naključnih osebkov.
V prvem koraku algoritem iz populacije z uporabo enega od naslednjih algoritmov izbere tiste osebke, ki bojo šli naprej:

1. Roulette wheel selection z rangiranimi vrednostmi: Vsak osebek je izbran z verjetnostjo, ki je odvisna od tega katera po vrsti je njegova kvaliteta.
2. Roulette wheel selection s centriranimi vrednostmi: Vsak osebek je izbran z verjetnostjo, ki je odvisna od njegove kvalitete. Zaradi možnih negativnih vrednosti kvalitete, se na začetku od vseh kvalitet odšteje minimalna.
3. Deterministično tournament selection: Naključno izbira pare osebkov iz poola in izbere tistega, ki je boljši.
4. Izbira n najboljši osebkov iz populacije.

Najbolj sta se obnesla 3. in 4., saj sta prva dva generirala preveč naključne populacije.

V drugem koraku se na vseh osebkih v izbrani populaciji izvede izbrani optimizacijski algoritem. Tu sva v glavnem uporabljala local search z omejenim številom korakov. Ker je bil ta korak za mnoge nastavitve daleč najbolj zahteven sva ga paralelizirala z uporabo knjižnice `snowfall`.

V tretjem koraku algoritem naključno izbere pare iz izbrane populacije in jih med seboj združuje z uporabo "Ordered crossover" algoritma. Ta algoritem sva si izbrala zato, ker deluje na genomih pri katerih je vrstni red pomemben in ker ga je lahko implementirati vektorsko.

V četrtem koraku algoritem na novi populaciji naključno izvedaja mutacije. Za mutacijo sva si izbrala zamenjavo dveh naključnih elementov v permutaciji. Mutacija se lahko na istem osebku ponovi večkrat, a verjetnost za to eskponentno pada.

Ti štirje koraki se ponavljajo dokler število iteracij ne doseže izbranega števila. Algoritem med izvajanjem sledi najboljši permutaciji na katero je med izvajanjem naletel. Na koncu na to permutacijo še dodatno optimizira z local searchom in rezultat vrne.

V praksi je algoritem najbolje deloval z majhnim številom iteracij (okrog 10), z vmesno optimizacijo s 100 koraki, izbiro osebkov z metodo "tournaments selectionom", ali pa "izberi n najboljših" in visoko verjetnostjo za mutacije (okrog 0.1).

Rezultati

Pri prvih dveh setih podatkov so vsi algoritmi dobili iste rezultate. Ker so iste rezultate dobili tudi algoritmi, ki so jih uporabljali ostali, sklepava, da gre za optimalno rešitev.

Pri tretjem setu podatkov sva presenetljivo najboljše rezultate dobila s klasičnim local search algoritmom z veliko količino ponovnih zagonov. Razlog za to je verjetno dejstvo, da je ta algoritem najpreprostejši in zato tudi najhitrejši, tako da v istem času opravi znatno večjo količino iskanj.

Podobne rezultate kot local search je dal tudi MA, le da je računanje trajalo dlje in da je bilo potrebno vložiti več dela v pravilne nastavitve parametrov. MA je imel tudi problem, da imajo rezultati, ki jih daje visoko variabilnost, sploh če osebke izbiramo po roulette wheel algoritmu.

Najslabše rezultate je dal algoritem VNS, verjetno zato, ker uporaba različnih okolic ni prinesla tako velike izboljšave, kljub temu pa rezultat algoritma ni bil slab.

Zaključek

V splošnem sva s svojimi rezultati zadovoljna. Če bi nalogo opravljala še enkrat, bi verjetno uporabila podoben pristop. Možna izboljšava bi bila pisanje algoritmov v kakšnem hitrejšem jeziku, saj gre za zelo časovno zahtevne algoritme, je pa res, da bi v temu primeru verjetno za razvoj porabila veliko več časa.