Kodutöö 3, Lokaalne otsing

Urmas Pitsi, 12.okt.2019

Käesolevas töös lahendasin N-Queens probleemi. Vastavalt malereeglitele tuleb paigutada n arv lippe n*n ruuduga malelauale, nii et ükski lipp ei tulista teist. Peamine lahenduskäik seisneb konfliktide minimeerimises igas järgnevas positsioonis. Lõpplahendus on leitud, kui konfliktide arv on null. Et otsinguruumi oluliselt vähendada kodeerisin võimalikud olekud ühte n-pikkusega vektorisse. Vektori iga element näitab, millises veerus paikneb lipp, kui asume antud vektori indeksile vastaval real. Uute käikude tekitamiseks vahetan kaks vektori elementi omavahel, mis omakorda vähendab oluliselt otsinguruumi. Lühidalt on otsingualgoritm järgnev:

1. Genereerime võimalikud käigud.
2. Valime neist parima: väikseima konfliktide arvuga lõppseis.
3. Jätkame kuni leiame lahenduse.

Selline suhteliselt lihtne skeem hakkas kohe tööle ja leidsin mõõduka otsinguajaga lahendused kui n < 50. Järgmise sammuna otsisin võimalusi, kuidas piirata otsinguruumi võimalike käikude genereerimisel. Katsetasin “greedy stochastic gradient descent” (greedy-SGD) meetodit. See seisnes järgnevas:

1. Piirame võimalike järgnevate käikude arvu: “batch_size”.
2. Tagastame jooksva parima käigu kohe, kui on toimunud etteantud arv tulemuste parandusi: “min_num_improvements”.
3. Võimalikud käigud genereerime juhuslikult: valime juhusliku paari, mille veerukoordinaadid me vahetame ära.
4. Juhtudel, kui varajane tagastus ei käivitunud, valime parimatest tulemustest juhusliku. See aitab väljuda tsüklitest ja lokaalsetest miinimumidest. Loomulikult ei pruugi see alati õnnestuda.

Nimetatud “greedy-SGD” meetod aitas mul leida mõistliku ajaga lahendusi kuni n = 2000+. Koondasin tulemused Tabelisse 1, leheküljel 2 on mõned näidislahendused.
Kood on implementeeritud Python 3.7, kasutasin ainult standard teeki, st ei ole väliseid, ega spetsiifilisi teeke, näiteks pole kasutatud numpy, scipy jms teeke. Proovisin lisada numba dekoraatoreid, et kiiremaks saada, aga ei õnnestunud: numba ei kompileeri Set tüüpi, List tüübil on vaja teha täpsemat definitsiooni. Ei hakanud sellele rohkem aega kulutama.

Tabel 1. Greedy Stochastic Gradient Descent otsing.

n	10	50	100	500	1000	2000
aeg sekundites (iteratsioonide arv)	0 s (29)	0.3 s (50)	1 s (50)	60 s (190)	248 s (373)	1353 s (806)

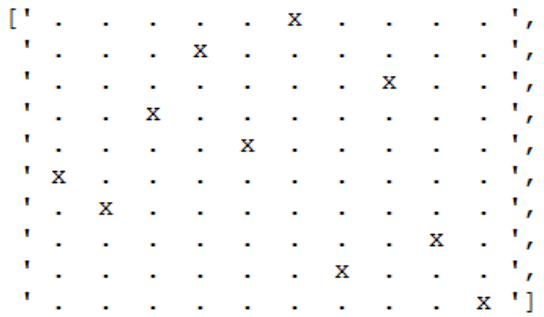
```
batch_size = min(100, 0.5 * n)
min_num_improvements = 3
```

Katsetused näitasid, et erinevad hüperparameetrid võivad anda ka oluliselt paremaid tulemusi. N väärtuse suurenemisel peab “natuke” suurendama otsingu põhjalikkust, et leiaks lahenduse.

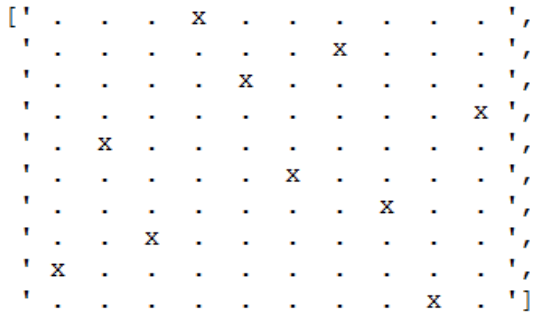
Näidislahendused:

Vasakul juhuslikult genereeritud algkonfiguratsioon ja paremal lahendus.

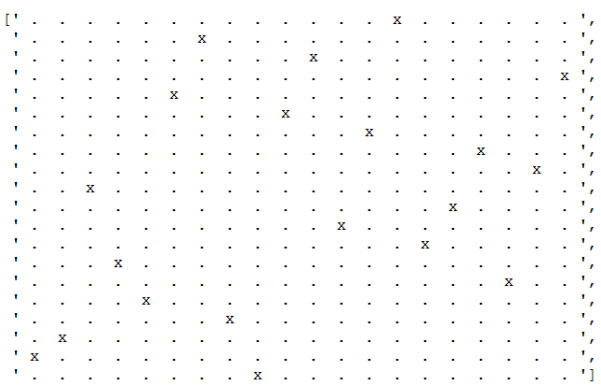
N=10: algkonfiguratsioon, konfliktide arv = 5



lahendus



N=20: algkonfiguratsioon, konfliktide arv = 7



lahendus