

UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 1. stopnja

Klemen Hovnik
Matija Gubanec Hančič
Jan Rudof

**Predpisana drevesa z najmanjšim/največjim Wienerjevim
indeksom**

Projekt v povezavi z OR

Ljubljana, 2018

KAZALO

1. Navodilo	3
2. Uvod	3
3. Opis dela	3
3.1. Enostaven algoritem	3
3.2. Genetski algoritem	5

1. NAVODILO

We want to analyze the structure of trees on a fixed number of vertices n and fixed maximum degree Δ that have Wiener index (i.e. total distance) as small as possible. Similarly, we want to find the structure of trees on a fixed number of vertices n with fixed diameter d that have Wiener index (i.e. total distance) as large as possible. In order to get the answer for very small values of n first, apply an exhaustive search, and next, for larger n , apply a genetic algorithm or any other metaheuristic. Verify for how large n your exhaustive search and your genetic algorithm implementations are efficient.

2. UVOD

Naj bo $G = (V(G), E(G))$ enostaven povezan neusmerjen graf. *Wienerjev indeks* (oziroma *Wienerjevo število* $W(G)$) je definiran kot

$$(1) \quad W(G) = \frac{1}{2} \sum_{u \in V(G)} \sum_{v \in V(G)} d_G(u, v).$$

Tukaj označimo z $d_G(u, v)$ razdaljo med vozliščem u in v v grafu G .

Naša naloga je, da analiziramo lastnosti dreves z določenim številom vozlišč in fiksno maksimalno stopnjo vozlišč, ki imajo najmanjši Wienerjev indeks. Podobno nas zanimajo tudi lastnosti dreves na določenem številu vozlišč s fiksnim premerom, ki imajo največji možni Wienerjev indeks.

Za izvedbo projekta smo si izbrali programski jezik *Sage*, saj ta že vsebuje orodja za delo z grafi, prav tako pa ima tudi generator dreves in že vgrajeno funkcijo za izračun Wienerjevega indeksa.

3. OPIS DELA

Najprej smo se lotili izračuna Wienerjevih indeksov na preprostih grafih z malo vozlišči, da vidimo, kako naj bi ta struktura grafov z minimalnimi oziroma maksimalnimi indeksi izgledala v splošnem.

3.1. Enostaven algoritem.

3.1.1. *Maksimalen Wiener index na drevesih s fiksnim polmerom.*

Definirali funkcijo $drevesa(n)$, ki nam izpiše seznam vseh dreves s številom vozlišč n . Potem smo to funkcijo uporabili v funkciji $drevesa_premer(n)$, ki nam iz prejšnjih dreves generira slovar, kjer so ključi možni premeri naših dreves, vrednosti ključev pa so pripadajoča drevesa. Tako smo si pripravili podlago za enostaven algoritem iskanja maksimalnega Wiener indeksa za drevesa z določenim premerom. Sestavili smo funkcijo $max_Weinerindex(n, N)$, kjer je N fiksni premer. Ta funkcija nam je za vsa drevesa s številom vozlišč n in premerom N izpisala maksimalni Wiener index in nam to drevo tudi izrisala.

```
def drevesa(n):
    t = graphs.trees(n)
    T= next(t)
    k= []
    k.append(T.edges())
    for T in t:
        m = T.edges()
        k.append(m)
    return k
```

```
def drevesa_premier(n):
    L = []
    k = drevesa(n)
    for i in range(len(k)):
        premier1 = Graph(k[i]).diameter()
        L.append([premier1, k[i]])
    from collections import defaultdict
    d1 = defaultdict(list)
    for l, v in L:
        d1[l].append(v)
    d = dict((l, tuple(v)) for l, v in d1.iteritems())
    return d
```

```
def max_Wienerindex(n,N):
    d = drevesa_premier(n)
    x = []
    for i in range(len(d[N])):
        x.append(Graph(d[N][i]).wiener_index())
    print 'drevesa, na ', n, 'vozliščih, s premerom', N, 'imajo maksimalni Wiener index:', max(x)
    for i in range(len(x)):
        if x[i] == max(x):
            Graph(d[N][i]).show()
```

3.1.2. Minimalen Wiener index na drevesih s fiksno stopnjo.

Za iskanje najmanjših Wienerjevih indeksov pri določenem številu vozlišč n in pri fiksni maksimalni stopnji m smo definirali funkcijo *fiksna_stopnja*, ki nam je iz seznama, ki ga vrne funkcija *drevesa(n)* izpisala drevesa z maksimalno stopnjo m . Izmed teh optimalnih dreves pa smo potem s funkcijo *.wiener_index()* izračunali najmanjši Wiener index, ter drevo s tem indeksom tudi izpisali.

```
def fiksna_stopnja(n,m):
    k = drevesa(n)
    zaporedje = []
    optimalna_drevesa = []
    index = []
    for j in range(len(k)):
        zaporedje.append(Graph(k[j]).degree_sequence())

    for l in range(len(zaporedje)):
        T = Graph(k[l])
        if max(zaporedje[l]) == m:
            optimalna_drevesa.append(k[l])
            index.append(T.wiener_index())
    u = min(index)
    pozicija = [i for i, j in enumerate(index) if j == u]
    print 'Minimalni Wiener index s fiksno stopnjo', m, 'je', u
    return Graph(optimalna_drevesa[pozicija[0]]).plot()
```

Hitro smo prišli do ugotovitve, da je naš algoritem za izračun indeksov časovno prepotraten. Zato smo se problema iskanja Wienerjevih indeksov lotili na drugačen način. In sicer z *genetskim algoritmom* katerega ideja je, da izmed dreves na manjših vozliščih gradimo večja drevesa

3.2. Genetski algoritem.

Genetski algoritem je metahevrstika, navdihnjena s strani procesov naravne selekcije in spada v razred *razvojnih algoritmov*. Uporablja se za generiranje kvalitetnih rešitev v optimizaciji, ki temeljijo na operatorjih kot so mutacija, križanje in selekcija.

V genetskem algoritmu se uporabi množica kandidatov za rešitev, ki jih nato razvijamo do optimalne rešitve. Vsak kandidat ima določene lastnosti, katere lahko spremenimo oziroma lahko mutirajo. Evolucija rešitev se ponavadi začne na naključni izbiri kandidatov, katere potem s pomočjo iteracije razvijamo. Na vsakem iterativnem koraku se potem oceni primernost novih kandidatov za optimizacijski problem. Najboljše kandidate potem uporabimo za naslednji korak iteracije in tako dalje. Na koncu se algoritem zaključi, ko doseže maksimalno število iteracijskih korakov oziroma, ko dobi najboljši približek optimalni rešitvi.

Naša začetna množica kandidatov bodo optimalna drevesa, ki smo jih dobili z našim prvim enostavnim algoritmom. Nato bomo ustvarili genetski algoritem, ki bo iz teh začetnih podatkov generiral nova optimalna drevesa. Ta postopek bomo nadaljevali (in pri tem selekcionirali iz novo nastalih dreves le najboljše za naslednje korake), dokler ne bomo prišli do optimalnih dreves za določeno število vozlišč. Pri tem bomo morali paziti, da se bo ohranjala maksimalna stopnja vozlišč, oziroma v drugem primeru, premer.

Na koncu bomo primerjali algoritma in pogledali, kdaj se ustavi naš enostavni algoritem za iskanje grafov z max/min Wienerjevim indeksom oziroma za katero število vozlišč je genetski algoritem še učinkovit.