Weak mixed k-metric dimension

Uvod

Najina naloga je bila, da napiševa CLP program za izračun šibke mešane k-metrične dimenzije $wmdim_k(G)$ in določiti $\kappa''(G)$ ter $wmdim_k(G)$ za cikle, polne grafe, dvodelne polne grafe, hiperkocke in kartezične produkte ciklov. S pomočjo tega, sva poskusila uganiti bolj splošne formule za $\kappa''(G)$ in $wmdim_k(G)$.

Nato sva s pomočjo sistematičnega in stohastičnega iskanja (hill-climbing in simulated annealing) iskala grafe za katere je $wmdim_k(G)$ velik oz. majhen.

CLP program

Kot že rečeno sva najprej morala napisati CLP program, ki bo vračal $wmdim_k(G)$. Ker mora najin program izračunati tudi razdalje med vozlišči in povezavami sva zato naprej definirala funkcijo razdalja, ki naredi prav to. Funkcija `razdalja` sprejme tri argumente: `moznost`, `vozlisce` in `G`, kjer je moznost podana kot par `(c,a)`, kjer `c` pove ali imamo povezavo ali vozlišče, `a` pa je potem ta povezava oz. vozlišče, do katerega raučnamo razdaljo. Razdalja od vozlišča do povezave `UV` je definirana kot najkrajša razdalja od vozlišča `vozlisce` do `U` oz. `V`.

```
def razdalja(moznost, vozlisce, G):
    c, a = moznost
    if c == 'e':
        U, V = a
        return min(G.distance(U, vozlisce), G.distance(V, vozlisce))
    else:
        return G.distance(a, vozlisce)
```

CLP program za šibko mešano k-metrično dimenzijo je zelo podobne CLP programu za šibko k-metrično dimenzijo s to izjemo, da moramo pri mešani upoštevati tudi vse povezave. To sva naredila tako, da sva definirala `moznosti` kot seznam povezav in vozlišč in vsako izmed njih poimenovala - da bo funkcija razdalja vedela, ali gre za vozlišče ali povezavo (problem bi drugače nastal pri kartezičnih produktih ciklov). Pri samem programu pa sva si pomagala z vgrajeno funkcijo `MixedIntegerLinearProgram` s katero sva potem dobila `wmdim_k` od grafa `G` za nek določen `k`.

```
def CLP_weak_mixed_k_dim(G, k):
    p = MixedIntegerLinearProgram(maximization=False)
    x = p.new_variable(binary=True)

    V = G.vertices()
    E = G.edges(labels=False)

    moznosti = [('v', v) for v in V] + [('e', e) for e in E]

    p.set_objective(sum(x[v] for v in V))

    for a, b in Combinations(moznosti, 2):
        p.add_constraint(
            sum(abs(razdalja(a, v, G) - razdalja(b, v, G)) * x[v] for v in V) >= k
            )

    wmdim_k = p.solve()
    mnozica_S = [v for v in V if p.get_values(x[v]) > 0.5]

    return (wmdim_k, mnozica_S)
```

Sedaj nam za prvi del preostane samo še izračun $\kappa''(G)$. Tega dobimo tako, da pogledamo do katerega `k` funkcija `CLP_weak_mixed_k_dim` ne dobi napake. Največji tak `k` je ravno $\kappa''(G)$.

```python
def kappa_2_crti(G):
    k = 1
    while True:
        try:
            CLP_weak_mixed_k_dim(G, k)
            k += 1
        except:
            return k - 1
```

Ugotovitve

Po poganjanju kode (glej datoteko **prva_naloga.ipynb**) sva prišla do naslednjih ugotovitev.

Cikli

$$\kappa''(G) = \left\lfloor \frac{n}{2} \right\rfloor$$

$$wmdim_1(G) = 3$$

$$wmdim_k(G) = \begin{cases} 2k, & \text{če } n \text{ sod} \\ 2k+1, & \text{če } n \text{ lih} \end{cases}$$

Polni grafi

$$\kappa''(G) = 1$$

$$wmdim_1(G) = n$$

Dvodelni polni grafi

- Za dvodelne polne grafe za katere je bil m ali n enak 1, je $\kappa''(G) = 1$, $wmdim_k(G)$ je tedaj enak $m \cdot n$. Izjema je dvodelni polni graf, kjer sta m in n enaka 1, tedaj je $\kappa''(G) = 1$, $wmdim_k(G) = 2$, torej $m + n$.
- Za dvodelne polne grafe, za katere m in n nista enaka 1, je $\kappa''(G) = 2$. $wmdim_k(G)$ je za $k = 1$ enak $m + n - 1$, če je m ali n enak 2, v nasprotnem primeru pa je enak $m + n - 2$. Za $k = 2$ je $wmdim_k(G)$ enak $m + n$.

Hiperkocke

$$\kappa''(G) = 2^{n-1}$$

Za $wmdim_k(G)$ žal nisva našla vzorca.

```
- velikost: 3 : kappa = 4
  k = 0, wmdim = 0.0
  k = 1, wmdim = 3.0
  k = 2, wmdim = 4.0
  k = 3, wmdim = 7.0
  k = 4, wmdim = 8.0
- velikost: 4 : kappa = 8
  k = 0, wmdim = 0.0
  k = 1, wmdim = 4.0
  k = 2, wmdim = 5.0
  k = 3, wmdim = 7.0
  k = 4, wmdim = 8.0
  k = 5, wmdim = 11.0
```

```
  k = 6, wmdim = 14.0
  k = 7, wmdim = 15.0
  k = 8, wmdim = 16.0
```

Kartezični produkti ciklov

Pri kartezičnih produktih ciklov je za grafe s k-jem enakim $\kappa''(G)$

$$wmdim_k(G) = m \cdot n.$$

Za ostale k nisva znala določiti splošne formule za $wmdim_k(G)$.

```
Processing Cartesian Product of Cycles C_4 x C_4
  Max k (kappa'') = 8
    k=1: wmdim_k(C_4 x C_4) = 4.0
    k=2: wmdim_k(C_4 x C_4) = 5.0
    k=3: wmdim_k(C_4 x C_4) = 7.0
    k=4: wmdim_k(C_4 x C_4) = 8.0
    k=5: wmdim_k(C_4 x C_4) = 11.0
    k=6: wmdim_k(C_4 x C_4) = 14.0
    k=7: wmdim_k(C_4 x C_4) = 15.0
    k=8: wmdim_k(C_4 x C_4) = 16.0

Processing Cartesian Product of Cycles C_5 x C_6
  Max k (kappa'') = 12
    k=1: wmdim_k(C_5 x C_6) = 4.0
    k=2: wmdim_k(C_5 x C_6) = 6.0
    k=3: wmdim_k(C_5 x C_6) = 8.0
    k=4: wmdim_k(C_5 x C_6) = 10.0
    k=5: wmdim_k(C_5 x C_6) = 13.0
    k=6: wmdim_k(C_5 x C_6) = 15.0
    k=7: wmdim_k(C_5 x C_6) = 18.0
    k=8: wmdim_k(C_5 x C_6) = 20.0
    k=9: wmdim_k(C_5 x C_6) = 23.0
    k=10: wmdim_k(C_5 x C_6) = 25.0
    k=11: wmdim_k(C_5 x C_6) = 28.0
    k=12: wmdim_k(C_5 x C_6) = 30.0

Processing Cartesian Product of Cycles C_6 x C_6
  Max k (kappa'') = 18
    k=1: wmdim_k(C_6 x C_6) = 4.0
    k=2: wmdim_k(C_6 x C_6) = 5.0
    k=3: wmdim_k(C_6 x C_6) = 7.0
    k=4: wmdim_k(C_6 x C_6) = 8.0
    k=5: wmdim_k(C_6 x C_6) = 11.0
    k=6: wmdim_k(C_6 x C_6) = 12.0
    k=7: wmdim_k(C_6 x C_6) = 15.0
    k=8: wmdim_k(C_6 x C_6) = 16.0
    k=9: wmdim_k(C_6 x C_6) = 19.0
    k=10: wmdim_k(C_6 x C_6) = 20.0
    k=11: wmdim_k(C_6 x C_6) = 23.0
    k=12: wmdim_k(C_6 x C_6) = 24.0
    k=13: wmdim_k(C_6 x C_6) = 27.0
    k=14: wmdim_k(C_6 x C_6) = 28.0
```

```
k=15: wmdim_k(C_6 x C_6) = 31.0
k=16: wmdim_k(C_6 x C_6) = 32.0
k=17: wmdim_k(C_6 x C_6) = 35.0
k=18: wmdim_k(C_6 x C_6) = 36.0
```

Če gledamo kartezični produkt ciklov C_m in C_n, velikosti m in n pa je $\kappa''(G)$ definiran s pomočjo:

$$a = \max\{m, n\} \quad \text{in} \quad b = \min\{m, n\}$$

- če je a sod:

$$\kappa''(G) = \begin{cases} \left(\left\lfloor \frac{b}{a} \right\rfloor + 1\right) \cdot a, & b \text{ lih} \\ \left\lfloor \frac{b}{2} \right\rfloor \cdot a, & b \text{ sod} \end{cases}$$

- če je a lih:

$$\kappa''(G) = \begin{cases} a \cdot \left\lfloor \frac{b}{2} \right\rfloor - \left\lfloor \frac{b}{2} \right\rfloor, & b \text{ sod} \\ \left(\left\lfloor \frac{b}{a} \right\rfloor + 1\right) \cdot a, & b \text{ lih} \end{cases}$$

- če sta m in n enaka:

$$\kappa''(G) = \left\lfloor \frac{m}{2} \right\rfloor \cdot m$$

Grafi z malim $wmdim_k(G)$

Za iskanje grafov z malim, torej 1, 2 ali 3, $wmdim_k(G)$ sva napisala kodo, ki jih bo generirala s pomočjo funkcije `nauty_geng`.

```
def poisci_grafe_z_wmdim_k_n(od, do, n):
    for i in range(od, do + 1):
        print(f'Povezani grafi na {i} vozliscih z wmdim_k(G) = {n}:')
        for G in graphs.nauty_geng(f'{i} -c'):
            kappa_2crti = kappa_2_crti(G)
            for k in range(1, kappa_2crti + 1):
                wmdim_k, _ = CLP_weak_k_dim(G, k)
                if wmdim_k == n:
                    G.show()
                    print(f'k = {k}')
```

Funkcija `poisci_grafe_z_wmdim_k_n(od, do, n)` vrača grafe (slike) od velikosti `od` do vključno velikosti `do`. Za vsakega od njih pri vsakem `k` preveri, ali je slučajno `wmdim_k == n` in če se to zgodi, ga vrne.

$wmdim_k(G) = 1$

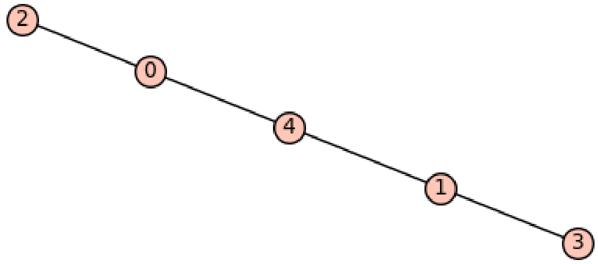
Takih grafov, glede na najine rezultate ni. Sklepava, da pride do tega, ker je do nekega vozlišča in tiste povezave enaka razdalja. Zaradi tega bo

$$\Delta_S(a, b) = \sum_{s \in S} |d(s, a) - d(s, b)| = 0,$$

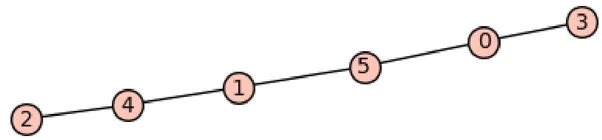
kjer je $S \subseteq V(G),\ a, b \in V(G) \cup E(G)$.

$wmdim_k(G) = 2$

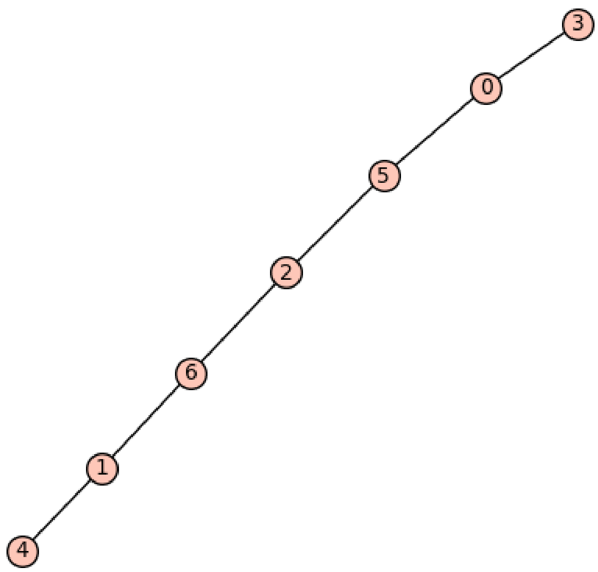
Rezultat tega so poti. Spodaj so prikazani nekateri rezultati.



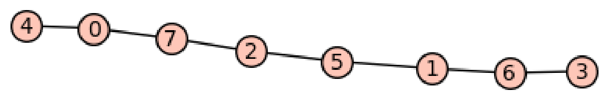
Slika 1: *Graf na petih vozliščih* (k = 1)



Slika 2: *Graf na šestih vozliščih* (k = 1)



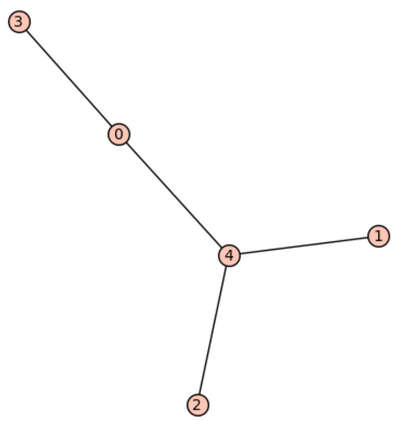
Slika 3: *Graf na sedmih vozliščih* (k = 1)



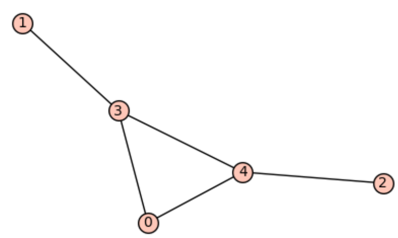
Slika 4: *Graf na osmih vozliščih* (k = 1)

$wmdim_k(G) = 3$

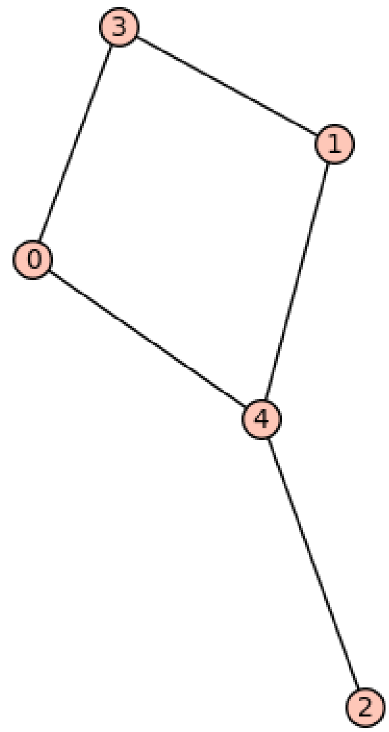
Grafov, ki imajo $wmdim_k(G) = 3$ je zelo veliko. Tukaj bi podala slike za grafe na petih vozliščih. Grafi na več kot petih vozliščih se nahajajo v datoteki "**druga_naloga.ipynb**". Vidimo lahko, da se med rezultati pojavijo tako grafi z vsaj enim ciklom, kot tudi drevesa.



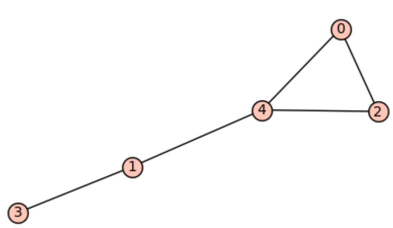
Slika 5 (k = 1)



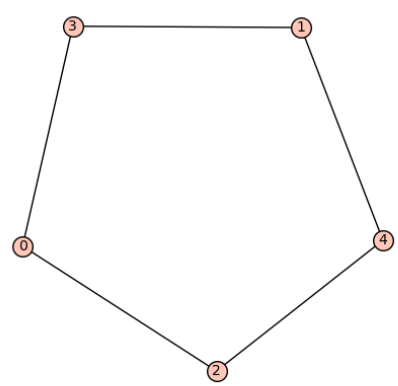
Slika 6 (k = 1)



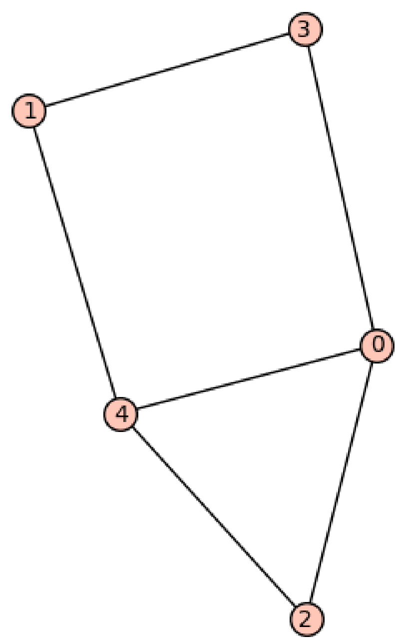
Slika 7 (k = 1)



Slika 8 (k = 1)



Slika 9 (k = 1)



Slika 10 (k = 1)

Grafi z velikim $wmdim_k(G)$

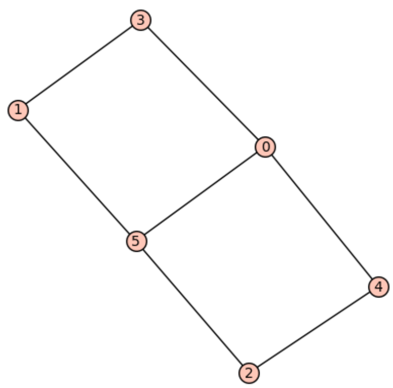
Podobno kot za manjše $wmdim_k(G)$ sva napisala tudi funkcijo za večje, torej

```
def poisci_grafe_z_wmdim_k_n_minus_j(od, do, j):
    for i in range(od, do + 1):
```

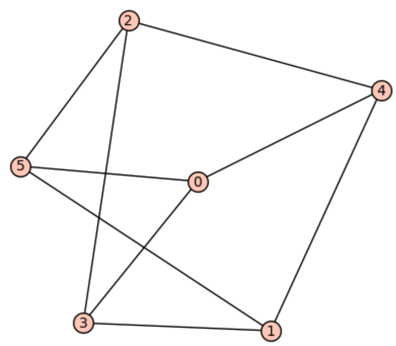
```
        print(f'Povezani grafi na {i} vozliscih z wmdim_k(G) = {i-j}:')
        for G in graphs.nauty_geng(f'{i} -c'):
            kappa_2crti = kappa_2_crti(G)
            for k in range(1, kappa_2crti + 1):
                wmdim_k, _ = CLP_weak_mixed_k_dim(G, k)
                if wmdim_k == i - j:
                    G.show()
                    print(G.adjacency_matrix())
                    print(f'k = {k}')
```

$wmdim_k(G) = n - 2$

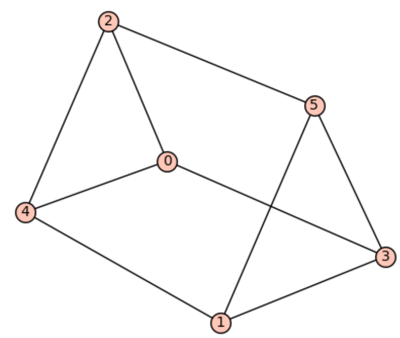
Taki grafi imajo po veliki večini vsaj en cikel ali pa so drevesa z $n-1$ listi.



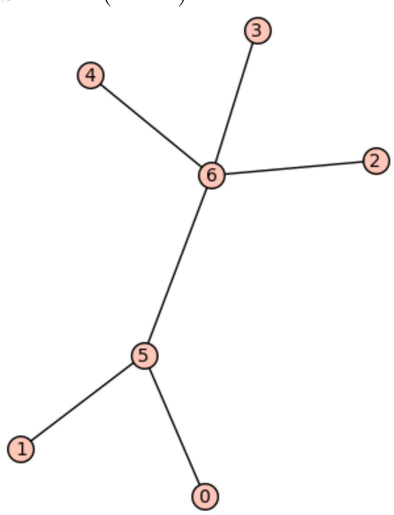
Slika 11 (k = 2)



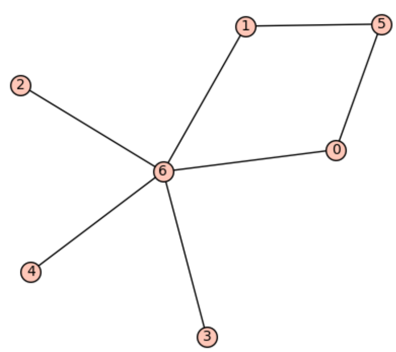
Slika 12 (k = 1)



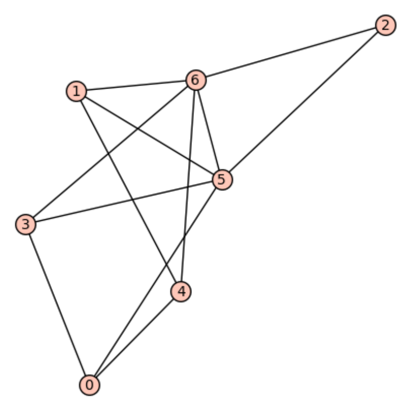
Slika 13 (k = 1)



Slika 14 (k = 1)



Slika 15 (k = 1)

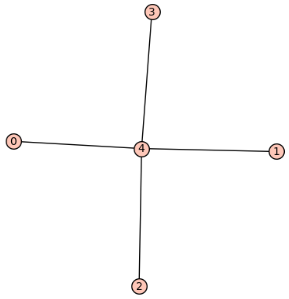


Slika 16 (k = 1)

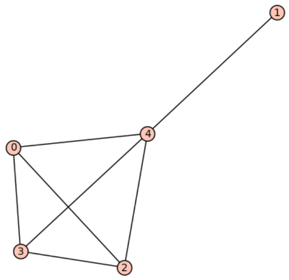
Več rezultatov se nahaja v datoteki **druga__naloga2.ipynb**.

$wmdim_k(G) = n - 1$

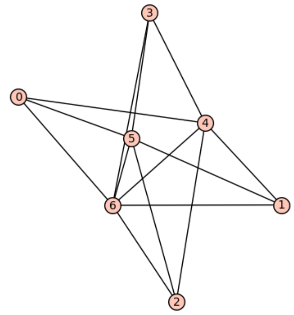
Podobno kot gor tudi tukaj dobimo grafe, ki imajo vsaj en cikel ali pa drevo z $n-1$ listi.



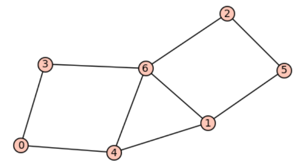
Slika 17 (k = 1)



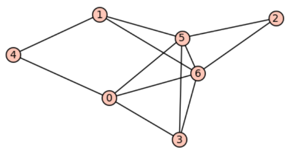
Slika 18 (k = 1)



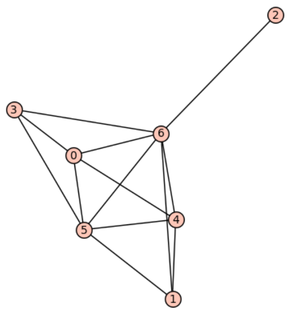
Slika 19 (k = 1)



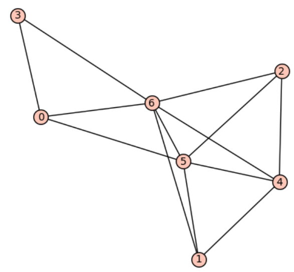
Slika 20 (k = 2)



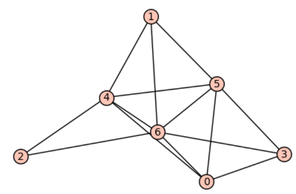
Slika 21 (k = 1)



Slika 22 (k = 1)



Slika 23 (k = 1)

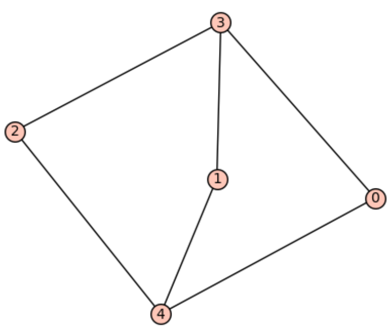


Slika 24 (k = 1)

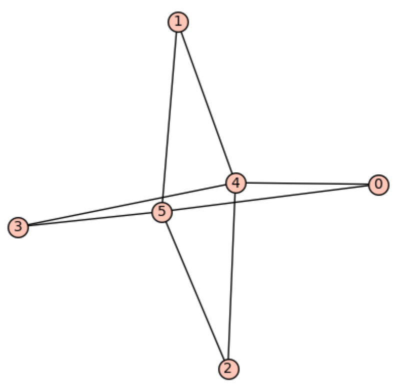
Več rezultatov se nahaja v datoteki **druga__naloga3.ipynb**.

$wmdim_k(G) = n$

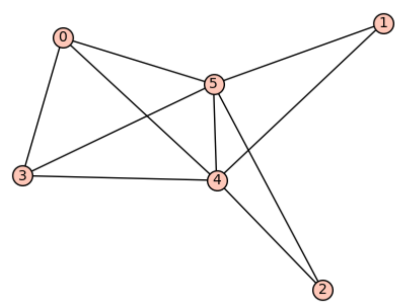
Tudi tu dobimo grafe z vsaj enim ciklom ali pa drevesa z $n-1$ listi.



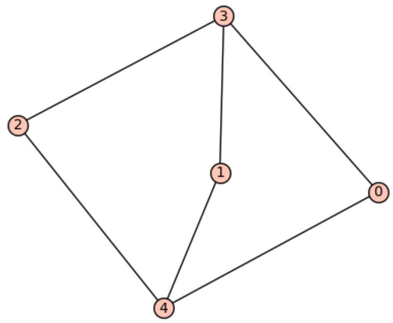
Slika 25 (k = 2)

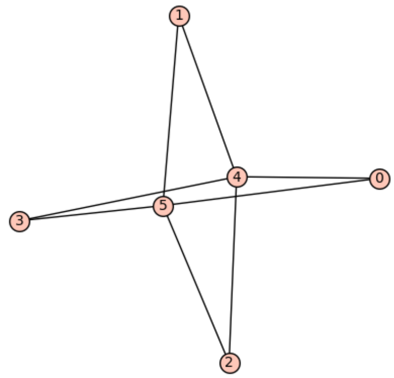


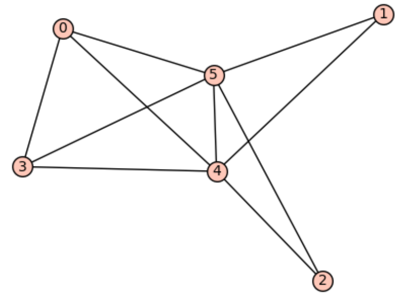
Slika 26 (k = 2)

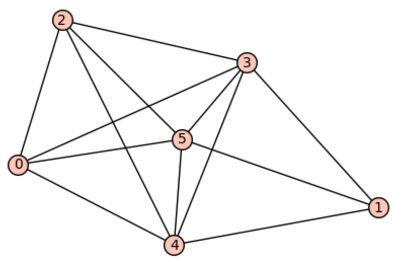


Slika 27 (k = 1)

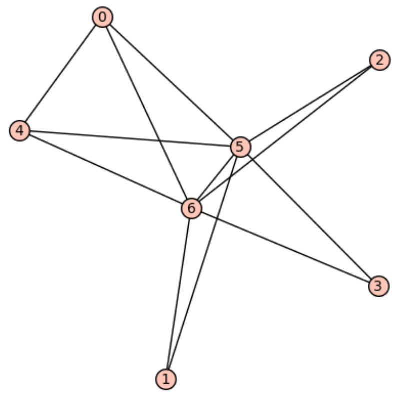




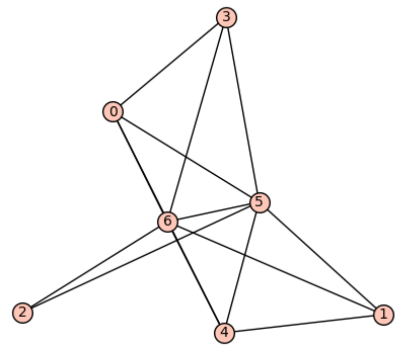




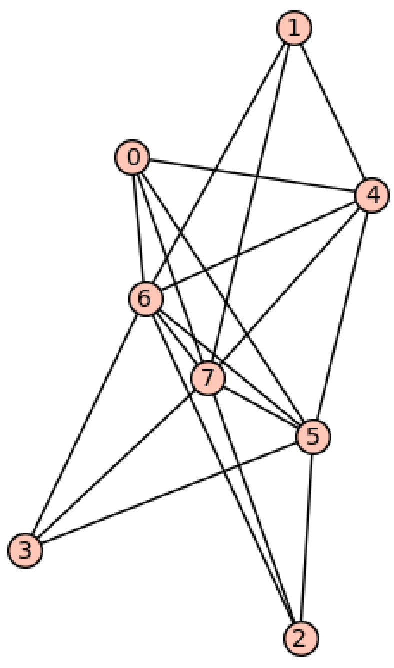
Slika 28 (k = 1)



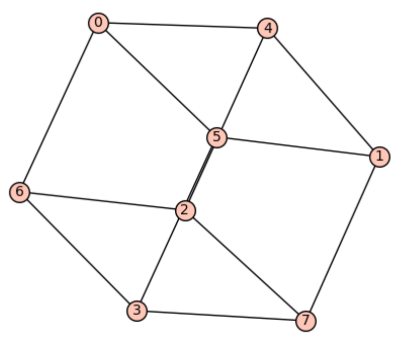
Slika 29 (k = 1)



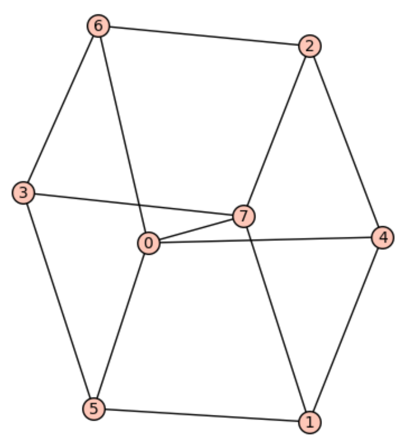
Slika 30 (k = 1)



Slika 31 (k = 1)



Slika 32 (k = 4)



Slika 33 (k = 3)

Več rezultatov se nahaja v datoteki **druga_naloga4.ipynb**.

Opazimo lahko, da se nekateri grafi (npr. C_6) pojavijo tako pri $n-1$, kot tudi pri $n-2$ in n, kar pomeni, da pri različnih k doseže $wmdim_k(G)$ vse tri vrednosti.

Stohastično iskanje $wmdim_k(G)$ za velike grafe

```
def generate_random_connected_graph(n, p):

    while True:
        G = graphs.RandomGNP(n, p)
        if G.is_connected():
            return G

def simulated_annealing(n, k, target_wmdim, max_iterations=1000, initial_temp=10, cooling_rate=0.95):

    current_graph = generate_random_connected_graph(n, random.uniform(0.2, 0.8))
    current_temp = initial_temp

    def objective(graph):
        try:
            wmdim_k, _ = CLP_weak_mixed_k_dim(graph, k)
            return abs(wmdim_k - target_wmdim), wmdim_k
        except (ValueError, MIPSolverException):
            return float('inf'), None

    current_cost, current_wmdim = objective(current_graph)
    best_graph = current_graph
    best_cost = current_cost
    best_wmdim = current_wmdim

    none_count = 0

    for iteration in range(max_iterations):
        if none_count >= 20:
          print("Regenerating graph due to 20 consecutive None values for current_wmdim.")
          current_graph = generate_new_graph()
          current_cost, current_wmdim = objective(current_graph)
          none_count = 0

        new_graph = current_graph.copy()
        if random.random() < 0.5:
            u, v = random.sample(range(n), 2)
            if not new_graph.has_edge(u, v):
                new_graph.add_edge(u, v)
        else:
            if new_graph.size() > n - 1:
                u, v = random.choice(new_graph.edges(labels=False))
                new_graph.delete_edge(u, v)

        if not new_graph.is_connected():
            continue

        new_cost, new_wmdim = objective(new_graph)

        if new_wmdim is None:
          none_count += 1
        else:
          none_count = 0
```

```
        if new_cost < current_cost or random.random() < math.exp((current_cost - new_cost) / current_te

            current_graph = new_graph
            current_cost = new_cost
            current_wmdim = new_wmdim
            if current_cost < best_cost:
                best_graph = current_graph
                best_cost = current_cost
                best_wmdim = current_wmdim

        current_temp *= cooling_rate
        print(f"Iteration {iteration + 1}: Current wmdim_k = {current_wmdim}, Best wmdim_k = {best_wmdi

        if best_cost == 0:
            break

    return best_graph, best_wmdim
```

Za iskanje velikih grafov z majhnim `wmdim_k` (3), oziroma velikim `wmdim_k` (n - 2, n - 1, n) sva uporabila stohastično iskanje, in sicer sva implimentirala funkcijo `simulated_annealing`. Simulirano ohlajanje je optimizacijska metoda, ki se uporablja za iskanje globalnega optimuma funkcije v velikem prostoru rešitev. Najina koda implementira algoritem simuliranega ohlajanja za iskanje grafa, katerega šibko mešana k-dimenzija (`wmdim_k`), se čim bolj približa ciljni vrednosti (`target_wmdim`). Algoritem postopoma prilagaja strukturo grafa, da bi minimiziral razliko med trenutno in ciljno vrednostjo. Obenem pa dopušča tudi "slabše" rešitve, kar preprečuje, da bi algoritem obtičal v lokalnih minimumih.

Parametri funkcije `simulated_annealing`: - **n**: Število vozlišč grafa. - **k**: Dimenzija `k`, za katero se računa `wmdim_k`. - **target_wmdim**: Ciljna vrednost `wmdim_k`, ki jo želimo doseči. - **max_iterations**: Največje število iteracij algoritma. - **initial_temp**: Začetna temperatura. - **cooling_rate**: Faktor, s katerim se na vsakem koraku zmanjša temperatura.

Na začetku koda generira naključni povezani graf s pomočjo funkcije `generate_random_connected_graph`, ki sprejme parametre **n**, torej število vozlišč grafa in **p**, ki predstavlja verjetnost povezave med vozlišči. S tem se zagotovi, da je začetni graf povezan, saj nepovezani grafi niso primerni za izračun `wmdim_k`.

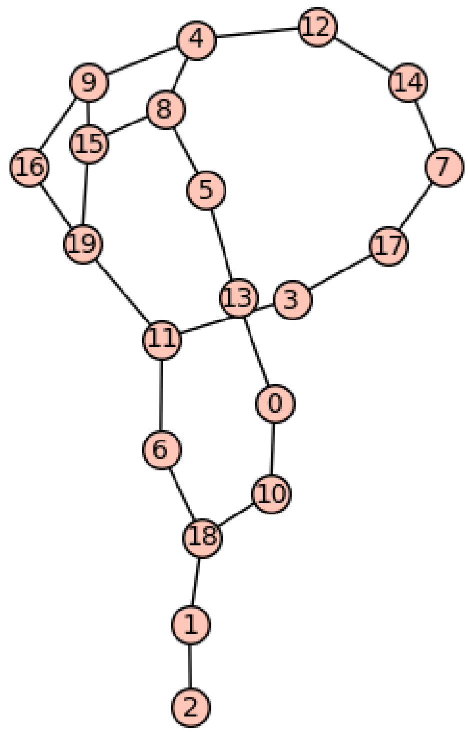
Ciljna funkcija `objective` izračuna razdaljo med trenutno vrednostjo `wmdim_k` grafa in ciljno vrednostjo `target_wmdim_k`. Če izračun `wmdim_k` ni mogoč (generiran graf je "preredek" za smiselno izračunavanje `wmdim_k` oziroma je parameter **k** prevelik glede na strukturo grafa), funkcija vrne vrednost `None`.

V glavnem delu koda `simulated_annealing` na vsakem koraku ustvari novo različico trenutnega grafa, na način, da z verjetnostjo 0.5 doda novo povezavo med naključnima vozliščema, v primeru da ta ne obstaja. Sicer pa odstrani naključno obstoječo povezavo, če graf pri tem procesu ostane povezan. Nato koda izračuna vrednost `wmdim_k` za posodobljeni graf z uporabo ciljne funkcije. Če ta vrne vrednost `None`, se vrednost `none_count` poveča za ena. Če pride do dvajset zaporednih ponovitev vrednosti `None`, se trenutni graf zamenja z novim naključno generiranim povezanim grafom. V primeru, da ciljna funkcija vrne vrednost, ki ni enaka `None` algoritem sprejme spremembo grafa, če se razlika med trenutno in ciljno vrednostjo zmanjša. Z določenim naključjem pa sprejme tudi "poslabšanje" grafa, s čimer se izognemo, da bi obtičali v lokalnih minimumih. Na vsakem koraku prav tako pride do zmanjšanja temperature s faktorjem `cooling_rate`. Nižja temperatura zmanjša verjetnost sprejetja "poslabšanja", kar vodi v stabilizacijo rešitve.

Algoritem se zaključi, ko doseže maksimalno število iteracij ali pa najde graf, katerega `wmdim_k` ustreza ciljni vrednosti.

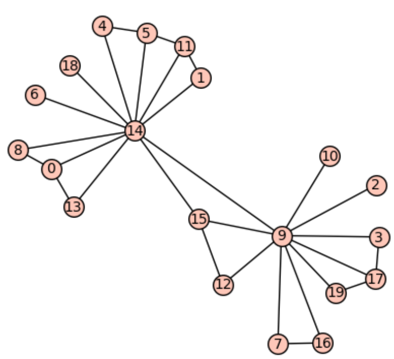
Izvedla sva `simulated_annealing` na grafu z dvajsetimi vozlišči. Za ostale informacije pa si lahko ogledate datoteko **hill_climbing.ipynb**.

$wmdim_k(G) = 3$

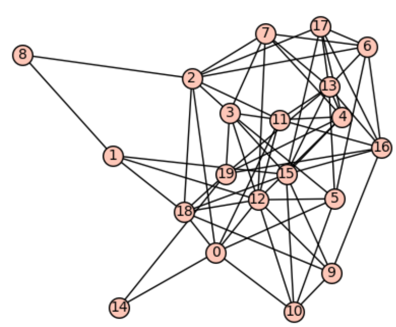


Slika 34: k = 1

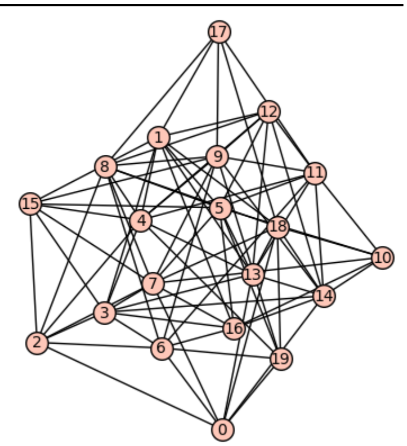
$wmdim_k(G) = n - 2$



Slika 35: k = 1

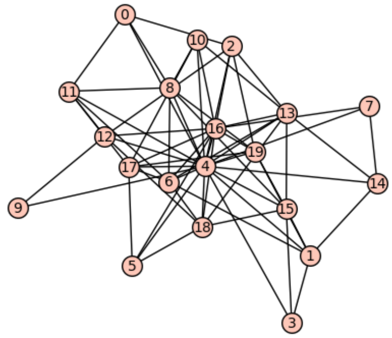


Slika 36: k = 2

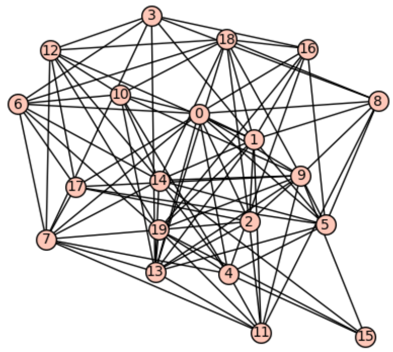


Slika 37: k = 3

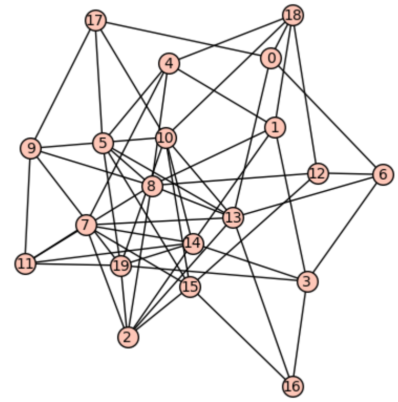
$wmdim_k(G) = n - 1$



Slika 38: k = 1

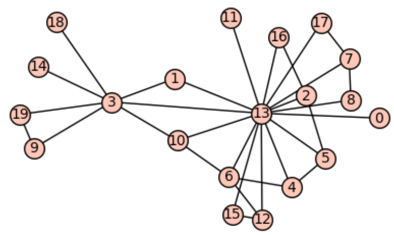


Slika 39: k = 2

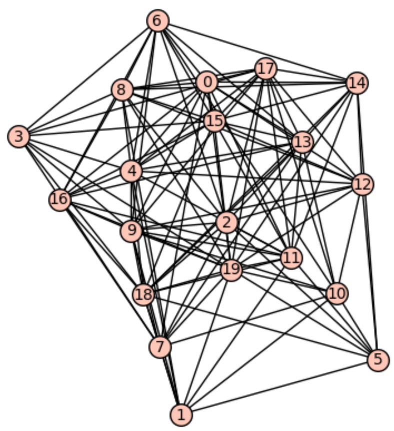


Slika 40: k = 3

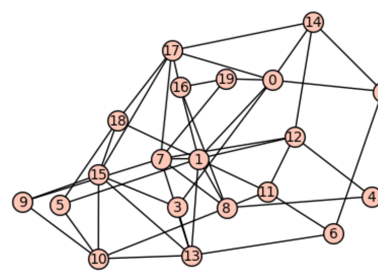
$wmdim_k(G) = n$



Slika 41: k = 1



Slika 42: k = 2



Slika 43: k = 3

Zaključek

V najini projektni nalogi sva uganila formule za $wmdim_k(G)$ in $\kappa''(G)$ za nekatere točno določene grafe. Pogledala sva si tudi kdaj je $wmdim_k(G)$ majhen oz. velik. S pomočjo stohastičnega iskanja pa sva si pogledala še kakšne grafe bi dobila, če bi imela več vozlišč (20). Opazila sva, da imajo taki grafi za velik $wmdim_k(G)$ veliko povezav.