BABEŞ–BOLYAI UNIVERSITY OF CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

Diploma Thesis

Critical node detection problem in complex networks

Abstract

EZ AZ OLDAL NEM RÉSZE A DOLGOZATNAK!

Ezt az angol kivonatot külön lapra kell nyomtatni és alá kell írni!

A DOLGOZATTAL EGYÜTT KELL BEADNI!

Kötelező befejezés:

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

2020 BÉCZI ELIÉZER

ADVISOR:
ASSIST PROF. DR. GASKÓ NOÉMI

Babeş–Bolyai University of Cluj-Napoca
Faculty of Mathematics and Informatics
Specialization: Computer Science

Diploma Thesis

Critical node detection problem in complex networks

Advisor:
Assist prof. dr. Gaskó Noémi

Student:
Béczi Eliézer

2020

Universitatea Babeş–Bolyai, Cluj-Napoca
Facultatea de Matematică şi Informatică
Specializarea Informatică

Lucrare de licenţă

Titlu lucrare licență

Conducător ştiinţific:
Lector dr. Gaskó Noémi

Absolvent:
Béczi Eliézer

2020

Babeş–Bolyai Tudományegyetem Kolozsvár
Matematika és Informatika Kar
Informatika Szak

Szakdolgozat

Kritikus csomópontok meghatározása komplex hálózatokban

Témavezető:

dr. Gaskó Noémi,
egyetemi adjunktus

Szerző:

Béczi Eliézer

2020

Tartalomjegyzék

1. Bevezető **3**
1.1. Áttekintés . . . 3
1.2. Hozzájárulásaink . . . 3
2. Egycélú CNDP **4**
2.1. Páronkénti konnektivitás . . . 4
2.2. Mohó algoritmus . . . 4
2.2.1. Általánosan . . . 4
2.2.2. Saját mohó algoritmus . . . 5
2.3. Genetikus algoritmus . . . 5
2.3.1. Általánosan . . . 5
2.3.2. Saját genetikus algoritmus . . . 6
3. Kétcélú CNDP **9**

1. fejezet

Bevezető

1.1. Áttekintés

Hálózatok terén nem minden csomópont egyforma fontosságú. A kulcsfontosságú csomópontok keresésével hálózatokban széles körben foglalkoznak, különösképpen olyan csomópontok esetén, melyek a hálózat konnektivitásához köthetők. Ezeket a csomópontokat általában úgy nevezzük, hogy Kritikus Csomópontok.

Kritikus Csomópontok Meghatározásának Problémája (CNDP) egy optimalizációs feladat, amely egy olyan csoport csomópont megkereséséből áll, melyek törlése maximálisan rontja a hálózat konnektivitását bizonyos predefiniált konnektivitási metrikák szerint.

A CNDP számos alkalmazási területtel rendelkezik. Például, közösségi hálók nagy befolyással bíró egyedeinek azonosítása, komputációs biológiában kapcsolatok definiálására jelút vagy fehérje-fehérje kölcsönhatás hálózatokban, smart grid sebezhetőségének vizsgálata, egyének meghatározása védőoltással való ellátásra vagy karanténba való zárásra egy fertőzés terjedésének gátlása érdekében.

A CNDP egy $\mathcal{NP}$-teljes feladat. Adva van egy $G = (V, E)$ gráf, ahol $|V| = n$ a csomópontok száma, és $|E| = m$ pedig az élek száma. A feladat k kritikus csomópont meghatározása, amelyek törlése a bemeneti gráfból minimalizálja a hálózat páronkénti konnektivitását. Az alapján, hogy mit értünk egy hálózat konnektivitása alatt, a CNDP-nak van egycélú illetve többcélú megfogalmazása is.

1.2. Hozzájárulásaink

Ebben a dolgozatban többek között egy bi-objektív megfogalmazásával fogunk foglalkozni a CNDP-nak. Standard evolúciós algoritmusokat fogunk összehasonlítani egymással különböző szintetikus bemenetekre, illetve való világból inspirált bemenetekre, ugyanakkor célunk egy új hibrid algoritmus fejlesztése, melynek eredményei összehasonlíthatók a standard algoritmusok eredményeivel.

Az algoritmusokat Python-ban fogjuk bemutatni, és a NetworkX könyvtárat [Hagberg et al., 2008] fogjuk használni ahhoz, hogy gráfokat tudjunk manipulálni.

Benchmark tesztelés végett egy olyan gráfhalmazt fogunk használni, amelyben 4 alapvető típus jelenik meg, mindegyik a maga jellegzetességeivel.

2. fejezet

Egycélú CNDP

2.1. Páronkénti konnektivitás

Egycélú CNDP esetén a kihívás abban áll, hogy találjunk egy olyan konnektivitási metrikát, amely alkalmazási területtől függően megfelelően leírja egy gráf összefüggőségét. S-el fogjuk jelölni a törlendő csomópontok halmazát, míg az $f(S)$ jóság függvény fogja jellemezni a $G[V \setminus S]$ feszített részgráf összefüggőségét. Ha H-val jelöljük a $G[V \setminus S]$ feszített részgráf összefüggő komponenseinek a halmazát, akkor a jóságfüggvény a következő képlettel írható le:

$$f(S) = \sum_{h \in H} \frac{|h| \cdot (|h| - 1)}{2}, \tag{2.1}$$

amelyet az irodalom [Aringhieri et al., 2016; Ventresca, 2012] úgy tart számon, hogy **páronkénti konnektivitás**. Tehát a feladat a 2.1 függvénynek a minimalizálása:

$$\min_{S \subseteq V} f(S). \tag{2.2}$$

A 2.1 fitness függvény implementációját a 2.1 kódrészlet szemlélteti Python-ban. A továbbiakban tárgyalt 3 algoritmus ezt a fitness függvényt fogja használni.

Listing 2.1. Páronkénti konnektivitás

```
def pairwise_connectivity(G):
    components = networkx.algorithms.components.connected_components(G)
    result = 0

    for component in components:
        n = len(component)
        result += (n * (n - 1)) // 2

    return result
```

2.2. Mohó algoritmus

2.2.1. Általánosan

Egy mohó algoritmus egy egyszerű és intuitív algoritmus, amely gyakran használt optimalizációs feladatok megoldására. Az algoritmus helyi optimumok megvalósításával próbálja megtalálni a globális optimumot.

Habár a mohó algoritmusok jól működnek bizonyos feladatok esetében, mint pl. Dijkstra-algoritmus, amely egy csomópontból kiindulva meghatározza a legrövidebb utakat, vagy Huffman-kódolás, amely adattömörítésre szolgál, de sok esetben nem eredményeznek optimális megoldást. Ez annak köszönhető, hogy míg a mohó algoritmus függhet az előző lépések választásától, addig a jövőben meghozott döntésektől független.

Az algoritmus minden lépésben mohón választ, folyamatosan lebontva a feladatot kisebb feladattá. Más szavakkal, a mohó algoritmus soha nem gondolja újra választásait.

2.2.2. Saját mohó algoritmus

A CNDP esetén a mohó algoritmust a 2.2 kódrészlet szemlélteti.

Listing 2.2. Saját mohó algoritmus

```
def greedy_cnp(G, k):
    S = networkx.algorithms.approximation.min_weighted_vertex_cover(G)

    while len(S) > k:
        B = objective_function.minimize_pairwise_connectivity(G, S)
        i = random.choice(B)
        S.discard(i)

    return S
```

A mohó algoritmus kiindul a gráf csúcslefedéséből. [1] Ez lesz a kezdeti S megoldásunk. A maradék csomópontok $V \setminus S$ a gráf maximális független csúcshalmazát [2] *MIS* alkotják. Mivel majdnem biztos, hogy $|S| > k$, ezért mohón elkezdünk kivenni csomópontokat S-ból, majd ezeket hozzáadni *MIS*-hoz, amíg $|S| > k$. A hozzáadott csomópont az lesz, amelyiket ha visszatesszük az eredeti gráfba, akkor a minimum értéket téríti vissza a páronkénti konnektivitásra a keletkezett gráfban.

Mivel több olyan csomópont lehet, amelyeket ha visszateszünk az eredeti gráfba, akkor ugyanazt a minimális értéket adják vissza a páronkénti konnektivitásra, ezért ezeket eltároljuk a B halmazban, és minden lépésben random módon határozzuk meg, hogy melyik kerüljön vissza a *MIS*-ba.

Ezzel az eljárással garantáljuk, hogy a mohó algoritmusunk különböző megoldásokat fog adni többszöri futtatások esetén.

2.3. Genetikus algoritmus

2.3.1. Általánosan

A genetikus algoritmus a metaheurisztikák osztályába tartozik, és a természetes kiválasztódás inspirálta. Egy globális optimalizáló, amely gyakran használt optimalizációs és keresési problémák esetében, ahol a sok lehetséges megoldás közül a legjobbat kell megkeresni. Azt hogy egy megoldás mennyire jó, a fitness függvény mondja meg.

1. Angolul: vertex cover.
2. Angolul: maximal independent set.

A genetikus algoritmus mindig egy populációnyi megoldással dolgozik. A populációba egyedek tartoznak, amelyek egyenként egy-egy megoldásai a feladatnak. Az algoritmus minden iterációban egy új populációt állít elő az aktuális populációból úgy, hogy a **szelekciós operátor** által kiválasztott legrátermettebb szülőkön alkalmazza a **rekombinációs** és **mutációs operátorokat**.

Ezen algoritmusok alapötlete az, hogy minden újabb generáció az előzőnél valamelyest rátermettebb egyedeket tartalmaz, és így a keresés folyamán egyre jobb megoldások születnek.

2.3.2. Saját genetikus algoritmus

A CNDP esetén a genetikus algoritmust a 2.3 kódrészlet szemlélteti.

Listing 2.3. Saját genetikus algoritmus

```python

def genetic_algorithm(G, k, N=100,
                      pi_min=5, pi_max=50, delta_pi=5, alpha=0.2,
                      tmax=10000):
    def fitness_function(S):
        subgraph = networkx.subgraph_view(G,
                                          filter_node=lambda n: n not in S)
        metric = connectivity_metric.pairwise_connectivity(subgraph)
        commonalities = S.intersection(best_S)

        return metric + gamma * len(commonalities)

    def my_cmp(a, b):
        return fitness_function(a) - fitness_function(b)

    t = 0
    P = []
    pi = pi_min
    my_key = functools.cmp_to_key(my_cmp)

    while len(P) < N:
        P.append(generate_random_solution(G, k))

    best_S = P[0].copy()
    gamma = update(G, best_S, P, alpha)
    best_S_fitness = fitness_function(best_S)

    while t < tmax:
        new_P = new_generation(k, N, P)
        mutation(G, k, N, new_P, pi)

        P.extend(new_P)
        P.sort(key=my_key)
        P = P[:N]

        curr_S = P[0]
        curr_S_fitness = fitness_function(curr_S)

        if curr_S_fitness < best_S_fitness:
            best_S = curr_S.copy()
            best_S_fitness = curr_S_fitness
            pi = pi_min
        else:
            pi = min(pi + delta_pi, pi_max)

        gamma = update(G, best_S, P, alpha)
        t += 1

    return best_S, best_S_fitness
```

Egy Genetikus Algoritmus (GA) standard algoritmikus keretrendszerét használjuk fel. Generálunk egy kezdeti populációt megoldásokkal. Utána keresztezzük őket, hogy új megoldásokat kapjunk, amelyeket pedig mutálunk. Ezután rendezzük a régi és új megoldásokat egy fitness függvény alapján, és létrehozunk egy új populációt eltávolítva a rossz megoldásokat. A folyamatot addig ismételjük, amíg az iterációk száma el nem ér egy felső korlátot. Az algoritmus végén visszatérítjük a legjobb megoldást.

Inicializáció

A kezdeti populáció egyedeit random generáljuk ki. Ez azt jelenti, hogy minden egyed kromoszómája egy k csomópontból álló részhalmaza lesz a bemeneti gráf csomóponthalmazának. Ezt szemlélteti a a 2.4 kódrészlet.

Listing 2.4. Random inicializáció

```
def generate_random_solution(G, k):
    S = list(G)

    while len(S) > k:
        S.pop(random.randrange(len(S)))

    return set(S)
```

Egy új fitness függvényt vezetünk be egy-egy egyed jóságának felmérése végett. Ez abban tér el a 2.1 részben tárgyaltaktól, hogy nem csak a páronkénti konnektivitás mértékét vesszük figyelembe egy egyed esetén, hanem hogy az eddigi talált legjobb megoldástól mennyire tér el. Ezt a fitness függvényt a következő képlettel írjuk le:

$$g(S, S^*) = f(S) + \gamma \cdot |S \cap S^*|. \tag{2.3}$$

A képletben szereplő S^* jelenti az eddig talált legjobb megoldást. A γ egy változó, amely abban segít, hogy fenntartsuk a változatosságot a populáció egyedei között, megbüntetve azokat, amelyek túl közel vannak a legjobbhoz. A γ változót minden iterációban a következő képlettel számoljuk újra:

$$\gamma = \frac{\alpha \cdot f(S)}{\langle |S \cap S^*| \rangle_{S \in P}}, \tag{2.4}$$

ahol a nevező a populáció egyedeinek átlagos hasonlóságát jelenti a legjobb egyedhez.

A 2.4 képlet implementációját láthatjuk a a 2.5 kódrészletben.

Listing 2.5. γ inicializálása

```
def update(G, best_S, P, alpha):
    subgraph = networkx.subgraph_view(G, filter_node=lambda n: n not in
        best_S)
    metric = connectivity_metric.pairwise_connectivity(subgraph)
    avg = 0

    for S in P:
        avg += len(S.intersection(best_S))

    avg /= len(P)
    return float('inf') if avg == 0 else (alpha * metric) / avg
```

Reprodukció

3. fejezet

Kétcélú CNDP

Irodalomjegyzék

Aringhieri, R., Grosso, A., Hosteins, P., és Scatamacchia, R. A general evolutionary framework for different classes of critical node problems. *Engineering Applications of Artificial Intelligence*, 55: 128–145, 2016.

Hagberg, A., Swart, P., és S Chult, D. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

Ventresca, M. Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem. *Computers & Operations Research*, 39(11):2763–2775, 2012.