

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

BABEȘ–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 BABEȘ–BOLYAI, CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licență

Identificarea nodurilor critice în rețele complexe



CONDUCĂTOR ȘTIINȚIFIC:
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
2.4. Hibrid Genetikus Algoritmus	9
3. Kétcélú CNDP	10

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 gráfalmazt 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 függetlenül 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:

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

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

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

A 2.1 fitness függvény implementációját a 2.1. kódrészlet szemlélteti Python-ban.

Listing 2.1. Páronkénti konnektivitás

```
1 def pairwise_connectivity(G):  
    components = networkx.algorithms.components.connected_components(G)  
    result = 0  
  
    for component in components:  
6         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.

2. FEJEZET: EGYCÉLÚ CNDP

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 az 1. kódrészlet szemlélteti.

Algorithm 1 Greedy CNP

```
1: function GREEDY( $G, k$ )
2:    $S \leftarrow \text{VERTEX COVER}(G)$ 
3:   while  $|S| > k$  do
4:      $B \leftarrow \arg \min_{i \in S} f(S \setminus \{i\})$ 
5:      $S \leftarrow S \setminus \{\text{SELECT}(B)\}$ 
6:   end while
7:   return  $S$ 
8: end function
```

A mohó algoritmus kiindul a gráf csúcislefedéséből.¹ Ez lesz a kezdeti S megoldásunk. A maradék csomópontok $V \setminus S$ a gráf maximális független csúcshalmazát² MIS alkotják. Mivel majdnem biztos, hogy $|S| > k$, ezért mohón elkezdünk kivenni csomópontokat S -ből, majd ezeket hozzáadni MIS -hez, amíg $|S| > k$. A hozzáadott csomópont az lesz, amelyiket ha visszateszünk 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 MIS -be.

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.

2. FEJEZET: EGYCÉLÚ CNDP

A genetikus algoritmus mindig egy populációnyi megoldással dolgozik. A populációba egyedek tartoznak, amelyek egyenként egyed esetén 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. kódrészlet szemlélteti.

Algorithm 2 Genetic Algorithm

```
1: function GA( $G, k, N, \pi_{\min}, \pi_{\max}, \Delta\pi, \alpha, t_{\max}$ )
2:    $t \leftarrow 0$ 
3:   INITIALIZE( $N, P, S^*, \gamma, \pi$ )
4:   while  $t < t_{\max}$  do
5:      $P' \leftarrow$  CROSSOVER( $k, N, P$ )
6:     MUTATION( $k, N, P', \pi$ )
7:      $P \leftarrow$  SELECTION( $P, P'$ )
8:      $\gamma, \pi =$  UPDATE( $P, S^*, \pi, \pi_{\min}, \pi_{\max}, \Delta\pi, \alpha$ )
9:   end while
10:  return  $P$ 
11: end function
```

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 2.2 kódrészlet.

Listing 2.2. Random inicializáció

```
1 def generate_random_solution(G, k):
2     S = list(G)
3     while len(S) > k:
4         S.pop(random.randrange(len(S)))
5
6     return set(S)
```

Egy új fitness függvényt vezetünk be egyed esetén egyed jószágának felmérése végett. Ez abban tér el a 2.1 részben tárgyaltaktól, hogy nem csak a páronkénti konnektivitás mértékét vesszük figyelembe egy

2. FEJEZET: EGYCÉLÚ CNDP

egyed esetén, hanem hogy az eddigi talált legjobb megoldástól mennyire tér el. Ezt a fitness függvényt a következő képlettel írjuk le:

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

A képletben szereplő S^* jelenti az eddigi 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:

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

ahol a nevező a populáció egyedeinek és a legjobb egyed közötti átlagos hasonlóságot fejezi ki. Az α pedig a képletben található változók egymás feletti fontosságát befolyásolja. A 2.4 képlet implementációját a 2.3 kódrészlet mutatja be.

Listing 2.3. γ inicializálása

```
def update(G, best_S, P, alpha):
    subgraph = networkx.subgraph_view(G, filter_node=lambda n: n not in
    best_S)
3    metric = connectivity_metric.pairwise_connectivity(subgraph)
    avg = 0
    for S in P:
        avg += len(S.intersection(best_S))
8    avg /= len(P)
    return float('inf') if avg == 0 else (alpha * metric) / avg
```

A π paraméter a mutáció valószínűségét fejezi ki egy egyed esetén. Ezt kezdetben π_{\min} -re állítjuk, de minden iterációban frissítjük aszerint, hogy találtunk-e az új generációban egy olyan megoldást, amely jobb, mint a globális legjobb. Ha találtunk az eddigieknél jobb megoldást, akkor a π értékét π_{\min} -re állítjuk, különben a $\pi = \min(\pi + \Delta\pi, \pi_{\max})$ képlet szerint növeljük. Ez arra jó, hogy fenntartsuk a populáció sokféleségét abban az esetben, amikor nem tudunk javítani az eddig talált legjobb megoldáson, mindezt úgy, hogy megnöveljük a mutációk kialakulásának a valószínűségét.

Reprodukción

A genetikus algoritmus egy kulcsfontosságú fázisa a reprodukció. Itt döntjük el, hogy a meglévő populációból miként jöjjön létre az új generáció. Ez azt jelenti, hogy meghatározzuk, hogy az S_1 és S_2 szülők kromoszómáit hogyan olvasztjuk egybe annak érdekében, hogy egy új S' egyed szülessen. Ezt a folyamatot szemlélteti a 2.4 kódrészlet.

Listing 2.4. Reprodukció

```
def new_generation(k, N, P):
    new_P = []
    for _ in range(N):
5        r1 = random.randrange(N)
        r2 = random.randrange(N)
        while r1 == r2:
```

2. FEJEZET: EGYCÉLÚ CNDP

```
10         r2 = random.randrange(N)
        new_S = P[r1].union(P[r2])
        if len(new_S) == k:
            new_P.append(new_S)
15     else:
        tmp = list(new_S)
        random.shuffle(tmp)
        tmp = tmp[:k]
20     new_P.append(set(tmp))
    return new_P
```

Esetünkben úgy történik egy új egyed létrehozása, hogy random módon kiválasztunk 2 különböző szülőt, és ezek kromoszómáit egybevonjuk: $S' = S_1 \cup S_2$. Mivel majdnem biztos, hogy az így kapott egyed kromoszómája több, mint k csomópontot tartalmaz, ezért szükséges törölnünk belőle nódusokat, amíg $|S'| > k$. Az hogy melyik nódus kerül törlésre az új egyed kromoszómájából, random módon történik.

Fontos megemlítenünk, hogy mivel a szülőket random módon választjuk ki egyed esetén egyed létrehozásához, ezért a populáció egyedei között nem teszünk különbséget. Vagyis keresztezéskor nem nézzük, hogy csak a legrátermettebb szülőket válasszuk, hanem egyenlő eséllyel választunk kevésbé jó fitness értékkel rendelkező egyedet is szülőnek. Ez lelassítja a populáció uniformizálódásának folyamatát, de segíti a megoldástér bejárását. Ez azért jó, mert nem tudjuk előre, hogy a csomópontok mely kombinációja fogja eredményezni a bemeneti gráf maximális szétesését, ha ezeket együtt töröljük a gráf-ból. Ezért a kevésbé jó fitness értékkel rendelkező egyedeket sem kell figyelmen kívül hagyni, mert kombinálva őket jó megoldásokhoz juthatunk.

Mutáció

A következő nagy jelentőséggel bíró fázisa a genetikus algoritmusnak a mutáció. Mutáció alatt azt értjük, hogy vesszük az újonnan létrejött populációt, és a populációban található egyedek génjeit perturbáljuk valamilyen csekély valószínűséggel. A mutáció azért tartozik a nagy döntések halmazába, mert a mutáció révén fenntartjuk a populáció sokféleségét, és elkerüljük a korai konvergenciát.³ A 3. kódrészlet a mutáció műveletét hívatott bemutatni.

A populáció minden egyes új egyede esetén, a mutáció valószínűségét a π paraméter befolyásolja. Generálunk egy egyenletes eloszlású véletlen számot 1 és 100 között, és ha ez kisebb, mint π , akkor módosítjuk a megoldást. A módosítás úgy történik, hogy leszögezzük, hogy a megoldás hány génjét szeretnénk változtatni. Ezt a számot tükrözi az n_g változó, amely értékét a $[0, k]$ intervallumból veszi, és random generáljuk. A következő lépés, hogy kitörölünk n_g csomópontot a megoldásból, de mivel majdnem biztos, hogy a megoldásunk így nem-optimális, mert $|S| < k$, ezért szükséges visszaadogatnunk csomópontokat S -be. Ennek érdekében véletlenszerűen kiválasztunk egy csomópontot a $V \setminus S$ halmazból, és a kiválasztott csomópontot visszatesszük a megoldásba.

3. Angolul: premature convergence.

Algorithm 3 Mutation Operator

```

1: procedure MUTATION( $k, N, P', \pi$ )
2:   for  $i \leftarrow 1, N$  do
3:      $r \leftarrow \text{RAND INT}(1, 100)$ 
4:     if  $r \leq \pi$  then
5:        $S' \leftarrow P'[i]$ 
6:        $n_g \leftarrow \text{RAND INT}(0, k)$  ▷ Number of genes to mutate
7:       for  $j \leftarrow 1, n_g$  do
8:          $idx \leftarrow \text{RAND INT}(1, |S'|)$ 
9:          $S' \leftarrow S' \setminus \{P[idx]\}$ 
10:      end for
11:       $MIS \leftarrow V \setminus S'$ 
12:      while  $|S'| < k$  do
13:         $elem \leftarrow \text{SELECT}(MIS)$ 
14:         $S' \leftarrow S' \cup \{elem\}$ 
15:      end while
16:    end if
17:  end for
18: end procedure

```

Szelekció

Az utolsó fázisa a genetikus algoritmusunknak a szelekció. Itt döntjük el, hogy mely egyedek fogják alkotni a következő nemzedéket. Jelen esetben ez úgy megy végbe, hogy összefésüljük a régi P és az újonnan létrejött P' populációkat, és rendezzük az egyedeket a 2.3 fitness függvény alapján. Növekvő sorrendbe rendezzük őket, mivel nem szabad elfelejtenünk, hogy célunk végső soron a páronkénti konnektivitás minimalizálása. Ezután kiválasztjuk az első \mathcal{N} egyedet, és ezeket visszük tovább a következő iterációba.

2.4. Hibrid Genetikus Algoritmus

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.