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:
ASSOC. 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:
ASSOC. PROF. DR. GASKÓ NOÉMI

STUDENT:
BÉCZI ELIÉZER

2020

UNIVERSITATEA BABEŞ–BOLYAI, CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

Identificarea nodurilor critice în rețele complexe

CONDUCĂTOR ŞTIINŢIFIC:
CONF. 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 DOCENS

SZERZŐ:

BÉCZI ELIÉZER

2020

Tartalomjegyzék

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.

A Kritikus Csomópontok Meghatározásának Problémája (Critical Node Detection Problem - **CNDP**) egy optimalizációs feladat, amely egy olyan csoport csomópont megkereséséből áll, melyek törlése maximálisan rontja a hálózat konnektivitását bizonyos előre meghatározott konnektivitási metrikák szerint.

A CNDP számos alkalmazási területtel rendelkezik. Például, közösségi hálók nagy befolyással bíró egyedeinek azonosítása [Kempe et al., 2005], komputációs biológiában kapcsolatok definiálására fehérje-fehérje kölcsönhatás hálózatokban [Boginski és Commander, 2009; Tomaino et al., 2012], smart grid sebezhetőségének vizsgálata [Nguyen et al., 2013], egyének meghatározása védőoltással való ellátásra vagy karanténba való zárásra egy fertőzés terjedésének gátlása érdekében [Aspnes et al., 2006; Ventresca és Aleman, 2013, 2014].

A CNDP egy $\mathcal{NP}$-teljes feladat [Arulselvan et al., 2009]. 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 különféle gráfműveletek elvégzésére.

1.3. Bemeneti teszt gráfok

Benchmark tesztelés végett a Ventresca [2012] által javasolt gráfhalmazt fogjuk használni, amelyben négy alapvető típus jelenik meg, mindegyik a maga jellegzetességeivel. Az 1.1. táblázat bemutatja az illető gráfhalmazt, feltüntetve mindegyik bemenet esetén a következő tulajdonságokat: a példány nevét, a csomópontok $|V|$ és az élek $|E|$ számát, a törlendő (kritikus) csomópontok számát (k). A következőkben ezeket a modelleket szeretnénk röviden ismertetni.

1.3.1. Barabási–Albert modell

A Barabási–Albert modell olyan komplex hálózatok struktúráját írja le, amelyek skálafüggetlenek, vagyis a fokszámeloszlás nem változik az idő múlásával. Két alapgondolatot foglal magában: *folyamatos növekedés* és *preferenciális kapcsolódás*[1]. Mindkét fogalom széles körben ismert a valós hálózatok terén. A folyamatos növekedés alatt azt értjük, hogy a hálózat csúcsainak a száma egyre növekszik az idő múlásával, mivel újabb és újabb csúcsokat adunk hozzá a gráfhoz. A preferenciális kapcsolódás pedig azt jelenti, hogy minél nagyobb a fokszáma egy csomópontnak, annál nagyobb valószínűséggel fog kapcsolódni egy új csomóponthoz. Az 1.1. ábra (a) alábrája bemutat egy 1000 csomópontból álló Barabási–Albert típusú gráfot.

1.3.2. Erdős–Rényi modell

Az Erdős–Rényi modell véletlen gráfok előállítására szolgáló modell. Két különböző konstrukciót is jelöl: a $G(n, M)$ modellben n csúcsú és M élű gráfok közül választunk azonos valószínűséggel, míg a $G(n, p)$ modellben az n csúcsú gráf minden élét, egymástól függetlenül, p valószínűséggel húzzuk be. Az 1.1. ábra (b) alábrája bemutat egy 466 csomópontból és 700 élből álló Erdős–Rényi típusú gráfot.

1.3.3. Forest-fire modell

Mint a Barabási–Albert modell, a Forest-fire is a preferenciális kapcsolódás megközelítést használja, viszont a csomópontok fokszáma egy *lassan lecsengő eloszlást*[2] mutat, a hálózat átmérője csökkenvén az idő múlásával. Ennek eredményeképpen a modell egy *sűrűsödő hatványtörvényt*[3] követ, ami azt jelenti, hogy a hálózat egy hatványtörvénynek megfelelően sűrűsödik. A modell onnan kapta az elnevezését, hogy növekedésének a mintája egy erdőtűz terjedéséhez hasonlít. Az 1.1. ábra (c) alábrája bemutat egy 500 csomópontból álló Forest-fire típusú gráfot.

1.3.4. Watts–Strogatz modell

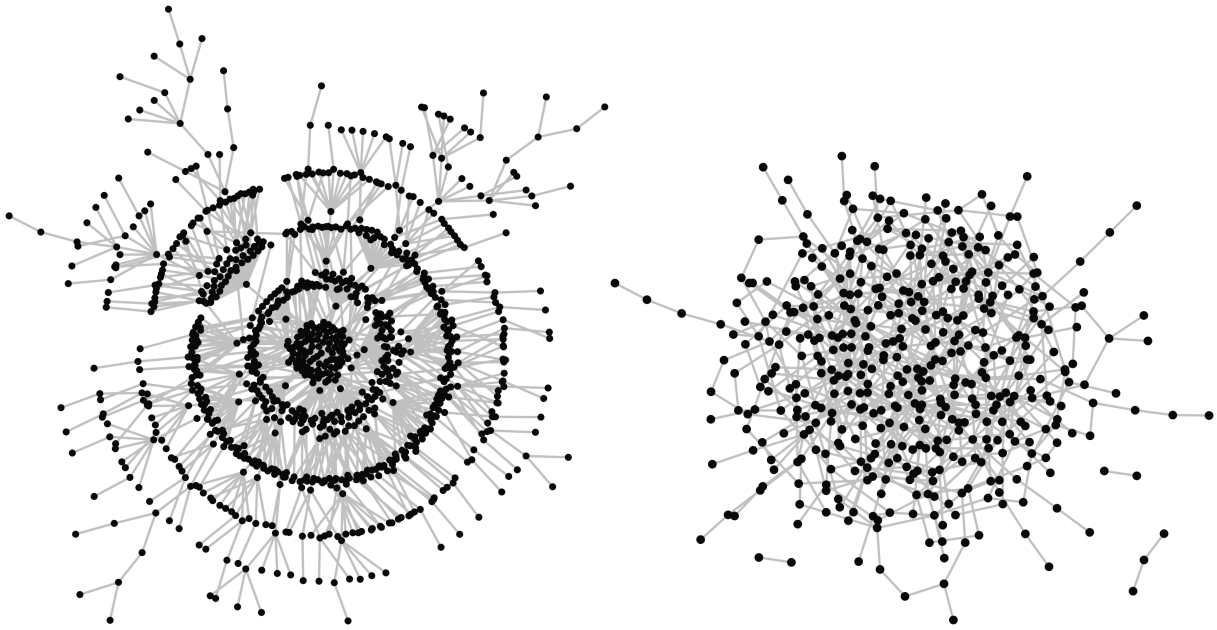
A Watts–Strogatz modell olyan véletlen gráfok előállítására szolgáló modell, amelyek *kis-világ* tulajdonságúak. Ezen hálózatok átmérője kicsi, és a legtöbb csomópont elérhető minden más csomópontból

1. Angolul: preferential attachment.
2. Angolul: heavy-tailed distribution.
3. Angolul: densification power-law.

relatív kevés ugráson vagy lépésen belül. Így a csúcsok közötti átlagos távolság rövid. Továbbá, ezen hálózatok magas klaszterezettségi együtthatóval rendelkeznek, vagyis a csomópontok hajlamosak csoportokba tömörülni. Az 1.1. ábra (d) alábrája bemutat egy 250 csomópontból álló Watts–Strogatz típusú gráfot.

1.1. táblázat. Benchmark tesztelésre használt bemeneti példányok

Gráf	$\|V\|$	$\|E\|$	k
BA500	500	499	50
BA1000	1000	999	75
BA2500	2500	2499	100
BA5000	5000	4999	150
ER250	235	350	50
ER500	466	700	80
ER1000	941	1400	140
ER2500	2344	3500	200
FF250	250	514	50
FF500	500	828	110
FF1000	1000	1817	150
FF2000	2000	3413	200
WS250	250	1246	70
WS500	500	1496	125
WS1000	1000	4996	200
WS2000	1500	4498	265



(a) Barabási–Albert típusú gráf, 1000 csomópont.

(b) Erdős–Rényi típusú gráf, 466 csomópont.

(c) Forest-fire típusú gráf, 500 csomópont.

(d) Watts–Strogatz típusú gráf, 250 csomópont.

1.1. ábra. A bemeneti példányok négy különböző modellje.

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 jóság függvény 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 fitnesz függvény implementációját a 2.1. kódrészlet szemlélteti Python-ban.

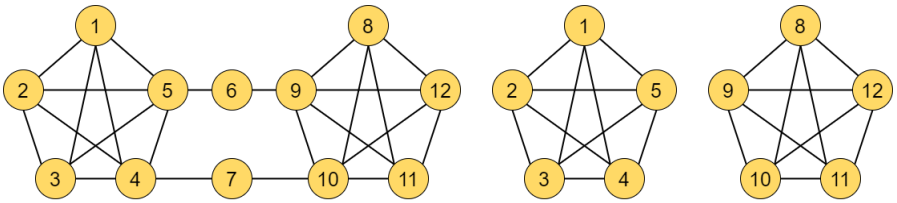
2.1. Listing. Páronkénti konnektivitás

```
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
```

Egy példa

A 2.1. ábrán látható gráfban, ha $k = 2$ kritikus csomópontot kell azonosítanunk, akkor $S = \{6, 7\}$ eredményezi az optimális megoldást. A $G[V \setminus S]$ feszített részgráf két, egyenként öt csomópontból álló összefüggő komponensre esik szét, vagyis $|H| = 2$. Így a 2.1 jóság függvény a következőképpen számolódik:

$$f(S) = \frac{5 - (5 - 1)}{2} + \frac{5 - (5 - 1)}{2} = 20.$$



2.1. ábra. Példa egy kis méretű gráfra (bal oldalt), amely a 6. és 7. csomópontok törlése után szétesik két összefüggő komponensre (jobb oldalt).

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 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 a megoldásunkban több, mint k csomópont lesz, ezért mohón elkezdünk kivenni csomópontokat S-ből, majd ezeket hozzáadni *MIS*-hez, 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 *MIS*-be. Ezzel az eljárással garantáljuk, hogy a mohó algoritmusunk különböző megoldásokat fog adni többszöri futtatások esetén. A CNDP esetén a mohó algoritmust a 2.1 kódrészlet szemlélteti.

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

Algorithm 2.1 Greedy CNP

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

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 fitnesz vagy jóság függvény mondja meg.

A genetikus algoritmus mindig egy populációnyi megoldással dolgozik. A populációba egyedek tartoznak, amelyek egyenként 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

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 fitnesz 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. A CNDP esetén a genetikus algoritmust a 2.2 kódrészlet szemlélteti.

Reprezentáció

Egy egyedet egy halmaznyi paraméter (változó) jellemez, amelyeket úgy nevezünk, hogy *gének*. A gének összessége alkotja a *kromoszómát*, amely a probléma egy lehetséges megoldását kódolja, amit a genetikus algoritmus próbál megoldani. A CNDP esetén minden gén a bemeneti gráf egy különböző csomópontja, a kromoszóma pedig a gráf csomóponthalmazának egy részhalmaza. Például, ha a G bemeneti gráf a $V = \{1, 2, \ldots, 9\}$ csomópontokból áll, és $k = 5$ nódust akarunk törölni, akkor a kromoszómát egy öt elemű halmazzal fogjuk reprezentálni: $\{g_1, g_2, g_3, g_4, g_5\}$.

Algorithm 2.2 Genetic Algorithm

```
1: function GA(G, k, N, π_min, π_max, Δπ, α, t_max)
2:     t ← 0
3:     INIT(N, P, S*, γ, π)
4:     while t < t_{t_max} do
5:         P' ← CROSSOVER(k, N, P)
6:         P' ← MUTATION(k, N, P', π)
7:         P ← SELECTION(N, P, P')
8:         S*, γ, π = UPDATE(N, P, S*, π, π_min, π_max, Δπ, α)
9:         t ← t + 1
10:    end while
11:    return P
12: end function
```

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 2.3 kódrészlet.

Algorithm 2.3 Random Solution

```
1: function RAND SOL(k)
2:     S ← V
3:     while |S| > k do
4:         elem ← SELECT(S)
5:         S ← S \ {elem}
6:     end while
7:     return S
8: end function
```

Egy új fitnesz függvényt vezetünk be 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 fitnesz 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 és a legjobb egyed közötti átlagos hasonlóságot fejezi ki. Az α pedig a képletben található változók egymás feletti fontosságát befolyásolja.

A π paraméter a mutáció valószínűségét fejezi ki egy egyed esetén. Ezt kezdetben $\pi_{\min}$-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 $\pi_{\min}$-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.

Az S^*, γ és π változók frissítését a 2.4 kódrészlet mutatja be.

Algorithm 2.4 Update S^*, γ and π variables

```
1:  function UPDATE(N, P, S*, π, π_min, π_max, Δπ, α)
2:      avg ← 0
3:      for i ← 1, N do
4:          S ← P[i]
5:          avg ← avg + |S ∩ S*|
6:      end for
7:      avg ← avg / N
8:      γ ← α · f(S*) / avg
9:      S ← P[0]
10:     if f(S) < f(S*) then
11:         S* ← S
12:         π ← π_min
13:     else
14:         π ← min(π + Δπ, π_max)
15:     end if
16:     return S*, γ, π
17: end function
```

Reprodukció

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.

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. A reprodukciós folyamatot a 2.5 kódrészlet szemlélteti.

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 fitnesz értékkel rendelkező egyedet is szülőnek. Ez lelassítja a populáció uniformizálódásának folyama-

Algorithm 2.5 Recombination Operator

```
1: function CROSSOVER(k, N, P)
2:     P' ← ∅
3:     for i ← 1, N do
4:         S_1 ← SELECT(P)
5:         S_2 ← SELECT(P)
6:         S' ← S_1 ∪ S_2
7:         if |S'| = k then
8:             P' ← P' ∪ {S'}
9:         else
10:            S' ← RANDOM SAMPLE(S', k)        ▷ Take k random elements from S'
11:            P' ← P' ∪ {S'}
12:        end if
13:    end for
14:    return P'
15: end function
```

tát, 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 gráfból. Ezért a kevésbé jó fitnesz 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 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 kitörlünk 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. A 2.6 kódrészlet a mutáció műveletét hívatott bemutatni.

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

3. Angolul: premature convergence.

Algorithm 2.6 Mutation Operator

```
1: function MUTATION(k, N, P, π)
2:     P' ← ∅
3:     for i ← 1, N do
4:         r ← RAND INT(1, N)
5:         if r ≤ π then
6:             S' ← P[i]
7:             n_g ← RAND INT(0, k)                ▷ Number of genes to mutate
8:             for j ← 1, n_g do
9:                 elem ← SELECT(S')
10:                S' ← S' \ {elem}
11:            end for
12:            MIS ← V \ S'
13:            while |S'| < k do
14:                elem ← SELECT(MIS)
15:                S' ← S' ∪ {elem}
16:            end while
17:            P' ← P' ∪ {S'}
18:        else
19:            S ← P[i]
20:            P' ← P' ∪ {S}
21:        end if
22:    end for
23:    return P'
24: end function
```

újonnan létrejött P' populációkat, és rendezzük az egyedeket a 2.3 fitnesz függvény alapján. Növekvő sorrendbe rendezzük őket, mivel nem szabad elfelejtenünk, hogy célunk végső soron a páronkénti konnektivitás minimalizálása. Ezután kiválasztjuk az első N egyedet, és ezeket visszük tovább a következő iterációba. Genetikus algoritmusunk szelekciós szakaszát a 2.7 kódrészlet ismerteti.

Algorithm 2.7 Selection Operator

```
1: function SELECTION(N, P, P')
2:     P ← P ∪ P'
3:     SORT(P)                    ▷ Sort individuals by fitness function in ASC order
4:     return P[:N]               ▷ Take best N solutions
5: end function
```

2.4. Memetikus algoritmus

Ahhoz, hogy ne teljesen véletlen megoldásokból induljunk ki a 2.2 kódrészlettel szemléltetett genetikus algoritmus esetén, ezért a kezdeti populáció egy részét a 2.1 algoritmus segítségével fogjuk kigenerálni. Ugyan a populáció inicializálása így több időt fog igénybe venni, de a megoldások egy része a bemeneti gráf struktúráját figyelembe véve lesznek meghatározva. Ezt szemlélteti a 2.8 algortimus, amely a kezdeti populáció 10%-át okosan generálja ki, a maradék 90%-át pedig véletlenül, felhasználva a 2.3 algoritmust.

Algorithm 2.8 Smart Initialization

```
1: function SMART INIT(G, k, N)
2:     P ← ∅
3:     for i ← 1, N · 10/100 do
4:         P ← P ∪ {GREEDY(G, k)}
5:     end for
6:     while |P| < N do
7:         P ← P ∪ {RAND SOL(k)}
8:     end while
9:     return P
10: end function
```

3. fejezet

Kétcélú CNDP

3.1. A CNDP-től a BOCNDP-ig

Az egycélú CNDP-től úgy jutunk el a kétcélú CNDP-ig, hogy ebben az esetben két függvényt fogunk optimalizálni. Míg a CNDP esetén a 2.1 képlettel leírt függvény minimalizálása volt a feladat, addig a BOCNDP esetén két célfüggvényünk van, amelyeket optimalizálni szeretnénk k csomópont kitörlése után a G gráfból:

1. Maximalizálni szeretnénk az összefüggő komponensek számát.
2. Minimalizálni szeretnénk az összefüggő komponensek számosságának a varianciáját.

Ennek érdekében a következő két célfüggvényt vezetjük be:

$$\max \quad |H|, \tag{3.1}$$

$$\min \quad var(H), \tag{3.2}$$

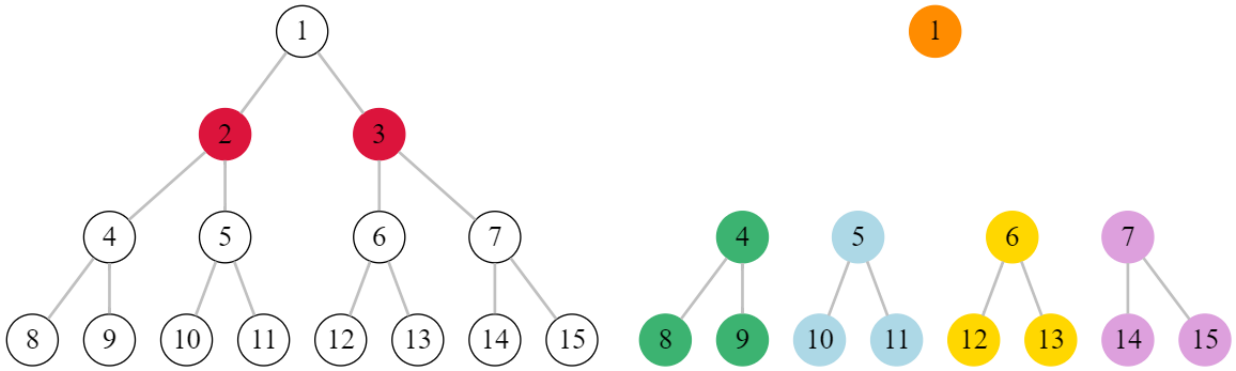
ahol H-val jelöljük a $G\left[V \setminus S\right]$ feszített részgráf összefüggő komponenseinek a halmazát, és $var(H)$ jelöli az összefüggő komponensek számosságának nem szabályos mintavételének a varianciáját. A H halmaz varianciáját a következő képlet segítségével számoljuk ki:

$$\frac{1}{|H|} \sum_{h \in H} \left(|h| - \frac{n^*}{|H|} \right)^2, \tag{3.3}$$

ahol $n^* = \sum_{h \in H} |h|$ a $G\left[V \setminus S\right]$ feszített részgráf csomópontjainak a száma.

A 3.1 és a 3.3 képletekkel leírt problémát úgy ismerjük az irodalomban [Ventresca et al., 2018], mint **BOCNDP**. A CNDP is ugyanerre a problémára nyújt megoldást azáltal, hogy ezt a két függvényt egyesíti a 2.1 függvényben, melynek minimalizálása (lásd a 2.2 egyenlet) maximalizálni fogja a komponensek számát, amelyekre szétesik az eredeti gráf, de ugyanakkor minimalizálja is a komponensek közötti varianciát.

A H halmaz számosságának meghatározását a 3.1. kódrészlet mutatja be Python-ban, míg a 3.3 képlet implementációját a 3.2. kódrészlet.



3.1. ábra. Példa egy kis méretű gráfra (bal oldalt), amely a 2. és 3. csomópontok (piros színnel emeltük ki ezeket) törlése után szétesik öt összefüggő komponensre (jobb oldalt), amelyeket különböző színekkel jelöltünk meg a könnyebb láthatóság kedvéért.

Egy példa

A 3.1 ábrán látható gráfban, ha $k = 2$ kritikus csomópontot kell azonosítanunk, akkor $S = \{2, 3\}$ eredményezi az optimális megoldást. A $G[V \setminus S]$ feszített részgráf szétesik egy egy csomópontból álló, és négy három csomópontból álló komponensre, vagyis $|H| = 5$. Így a 3.3 képlettel leírt komponensek közötti variancia a következőképpen számolható ki:

$$var(H) = \frac{1}{5} \cdot \left[\left(1 - \frac{13}{5}\right)^2 + 4 \cdot \left(3 - \frac{13}{5}\right)^2 \right] = \frac{16}{25} = 0.64.$$

3.1. Listing. A feszített részgráf összefüggő komponenseinek a száma

```
def connected_components(exclude=None):
    if exclude is None:
        exclude = {}

    S = set(exclude)
    subgraph = networkx.subgraph_view(G, filter_node=lambda n: n not in S)
    return networkx.number_connected_components(subgraph)
```

3.2. Listing. Az összefüggő komponensek számosságának a varianciája

```
def cardinality_variance(exclude=None):
    if exclude is None:
        exclude = {}

    S = set(exclude)
    subgraph = networkx.subgraph_view(G, filter_node=lambda n: n not in S)
    components = list(networkx.connected_components(subgraph))
    num_of_components = len(components)
    num_of_nodes = subgraph.number_of_nodes()
    variance = 0

    for component in components:
        cardinality = len(component)
        variance += (cardinality - num_of_nodes / num_of_components) ** 2

    variance /= num_of_components
    return variance
```

3.2. Kísérleti előkészítés

Ebben a részben bemutatjuk a BOCNDP probléma megoldására javasolt genetikus algoritmusokat és ezek paraméterezéseit. Az algoritmusokat nem mi implementáljuk, hanem a Platypus keretrendszer [Hadka, 2017] által biztosított osztályokat fogjuk használni.

NSGAII – Az NSGAII (Non-dominated Sorting Genetic Algorithm II) [Deb et al., 2002] az egyik legnépszerűbb többcélú optimalizáló algoritmus, amely az NSGA továbbfejlesztett változata. Az NSGAII a megszokott rekombinációs és mutációs genetikai operátorokon kívül, amelyek új egyedek létrehozásáért felelősek, két másik különleges mechanizmust használ a következő generáció populációjának létrehozásához: *nem-dominált rendezés*[1] révén a populációt alpopulációkra osztja valamilyen dominancia által meghatározott sorrend alapján (pl. Pareto, Nash vagy Berge dominancia), és kiszámítja az alpopulációk egyedei közötti *tömörülési távolságot*[2], felállítva egy sorrendet az alpopulációk egyedei között, hogy az elszigetelt megoldásokat részesítse előnyben.

EpsMOEA – Az EpsMOEA (Epsilon Multi-Objective Evolutionary Algorithm) [Deb et al., 2003] egy egyensúlyi állapotú evolúciós algoritmus, amely ϵ-dominancia archiválást használ a populáció sokszínűségének fenntartása végett.

SPEA2 – A SPEA2 (Strength Pareto Evolutionary Algorithm 2) [Zitzler et al., 2001] feladata, hogy megtaláljon és fenntartson egy frontnyi nem-dominált megoldást, ideális esetben egy halmaznyi Pareto-optimális megoldást. Ennek elérése érdekében egy evolúciós eljárást használ - felhasználva a genetikai rekombinációs és mutációs operátorokat - a megoldástér felderítése végett, és egy szelekciós eljárást, amely fitnesz függvénye egy egyed dominált ságának és a becsült Pareto front zsúfoltságának a kombinációja. A nem-dominált megoldások halmazáról egy archívum van karbantartva, amely különbözik az evolúciós eljárásban használt megoldások populációjától, biztosítva ezáltal egy elitista kiválasztást.

IBEA – Az IBEA (Indicator Based Evolutionary Algorithm) [Zitzler és Künzli, 2004] alapötlete, hogy egy *bináris hipertérfogat indikátort* használ a szelekciós eljárás szakaszában, amikor elválik, hogy mely egyedek fognak tovább élni, a következő generáció alapjául szolgálva.

PAES – A PAES (Pareto Archived Evolution Strategy) [Knowles és Corne, 1999] egy többcélú optimalizáló, amely két fő céllal lett kifejlesztve. Az elsődleges cél, hogy szigorúan lokális keresésre korlátozódik: a jelenlegi megoldást csak kis mértékben változtatja (mutáció), ezáltal eljutva a jelenlegi megoldástól egy szomszédos megoldásig. Ez a folyamat jelentős mértékben megkülönbözteti más többcélú optimalizáló genetikus algoritmustól (pl. NSGAII, SPEA2, IBEA), amelyek egy populációnyi megoldással dolgoznak, és ezen egyedek segítségével történik meg a keresztezés és kiválasztás. A második cél, hogy az algoritmus egy valódi Pareto optimalizáló kell, hogy legyen, minden nem-dominált megoldást egyformán kezelve. Mindkét cél elérése azonban elég problémás,

1. Angolul: non-dominated sorting.
2. Angolul: crowding distance.

mert az esetek többségében, amikor egy pár megoldást összehasonlítunk, akkor egyik sem fogja dominálni a másikat. A PAES ezt úgy oldja meg, hogy karbantart egy archívumot a nem-dominált megoldások halmazáról, amely révén felbecsüli az új megoldás jóságát.

EpsNSGAII – Az EpsNSGAII (Epsilon NSGAII) [Kollat és Reed, 2005] az NSGAII egy kibővített változata, amely ϵ-dominancia archiválást használ. Továbbá, véletlenszerű újraindítás jellemzi, biztosítva ezáltal egy változatosabb megoldáshalmazt.

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.

Arulselvan, A., Commander, C. W., Elefteriadou, L., és Pardalos, P. M. Detecting critical nodes in sparse graphs. *Computers & Operations Research*, 36(7):2193–2200, 2009.

Aspnes, J., Chang, K., és Yampolskiy, A. Inoculation strategies for victims of viruses and the sum-of-squares partition problem. *Journal of Computer and System Sciences*, 72(6):1077–1093, 2006.

Boginski, V. és Commander, C. W. Identifying critical nodes in protein-protein interaction networks. In *Clustering challenges in biological networks*, pages 153–167. World Scientific, 2009.

Deb, K., Pratap, A., Agarwal, S., és Meyarivan, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

Deb, K., Mohan, M., és Mishra, S. Towards a quick computation of well-spread pareto-optimal solutions. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 222–236. Springer, 2003.

Hadka, D. Platypus: A free and open source python library for multiobjective optimization. *Available on Github, vol. https://github.com/Project-Platypus/Platypus*, 2017.

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.

Kempe, D., Kleinberg, J., és Tardos, É. Influential nodes in a diffusion model for social networks. In *International Colloquium on Automata, Languages, and Programming*, pages 1127–1138. Springer, 2005.

Knowles, J. és Corne, D. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 1, pages 98–105. IEEE, 1999.

Kollat, J. B. és Reed, P. M. The value of online adaptive search: a performance comparison of nsgaii, ε-nsgaii and εmoea. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 386–398. Springer, 2005.

Nguyen, D. T., Shen, Y., és Thai, M. T. Detecting critical nodes in interdependent power networks for vulnerability assessment. *IEEE Transactions on Smart Grid*, 4(1):151–159, 2013.

Tomaino, V., Arulselvan, A., Veltri, P., és Pardalos, P. M. Studying connectivity properties in human protein–protein interaction network in cancer pathway. In *Data Mining for Biomarker Discovery*, pages 187–197. Springer, 2012.

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.

Ventresca, M. és Aleman, D. Evaluation of strategies to mitigate contagion spread using social network characteristics. *Social Networks*, 35(1):75–88, 2013.

Ventresca, M. és Aleman, D. A randomized algorithm with local search for containment of pandemic disease spread. *Computers & operations research*, 48:11–19, 2014.

Ventresca, M., Harrison, K. R., és Ombuki-Berman, B. M. The bi-objective critical node detection

problem. *European Journal of Operational Research*, 265(3):895–908, 2018.

Zitzler, E. és Künzli, S. Indicator-based selection in multiobjective search. In *International conference on parallel problem solving from nature*, pages 832–842. Springer, 2004.

Zitzler, E., Laumanns, M., és Thiele, L. Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report*, 103, 2001.