

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

<b>1. Bevezető</b>	<b>3</b>
1.1. Áttekintés	3
1.2. Hozzájárulásaink	3
1.3. Bemeneti példányok	4
1.3.1. Barabási–Albert modell	4
1.3.2. Erdős–Rényi modell	4
1.3.3. Forest-fire modell	4
1.3.4. Watts–Strogatz modell	4
<b>2. Egycélú CNDP</b>	<b>6</b>
2.1. Páronkénti konnektivitás	6
2.2. Mohó algoritmus	7
2.2.1. Általánosan	7
2.2.2. Saját mohó algoritmus	7
2.3. Genetikus algoritmus	8
2.3.1. Általánosan	8
2.3.2. Saját genetikus algoritmus	8
2.4. GA, de okos inicializálással	13
<b>3. Kétcélú CNDP</b>	<b>14</b>
3.1. A CNDP-től a BOCNDP-ig	14
3.2. Kísérleti előkészítés	16

## 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.

### **1.3. Bemeneti példányok**

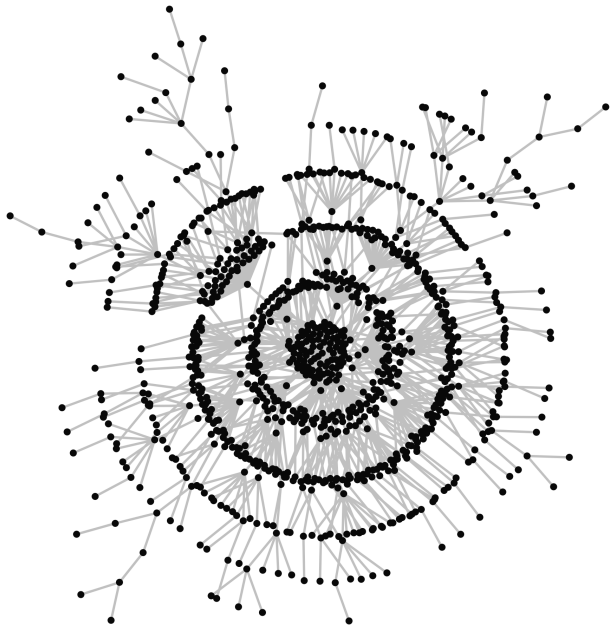
Benchmark tesztelés végett a Ventresca [2012] által javasolt gráfalmazt fogjuk használni, amelyben négy alapvető típus jelenik meg, mindegyik a maga jellegzetességeivel. A következőkben ezeket a modelleket szeretnénk röviden ismertetni.

#### **1.3.1. Barabási–Albert modell**

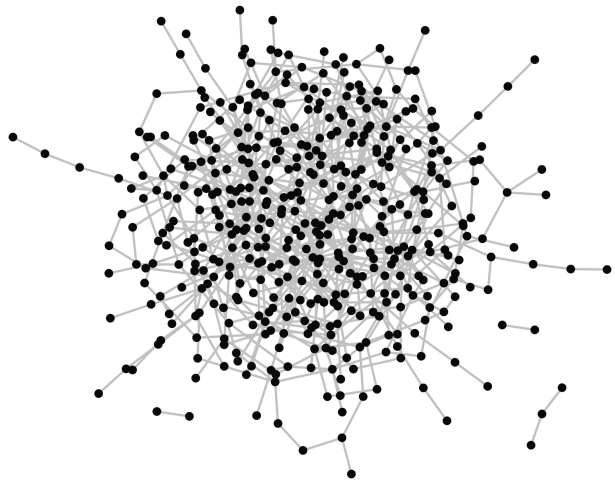
#### **1.3.2. Erdős–Rényi modell**

#### **1.3.3. Forest-fire modell**

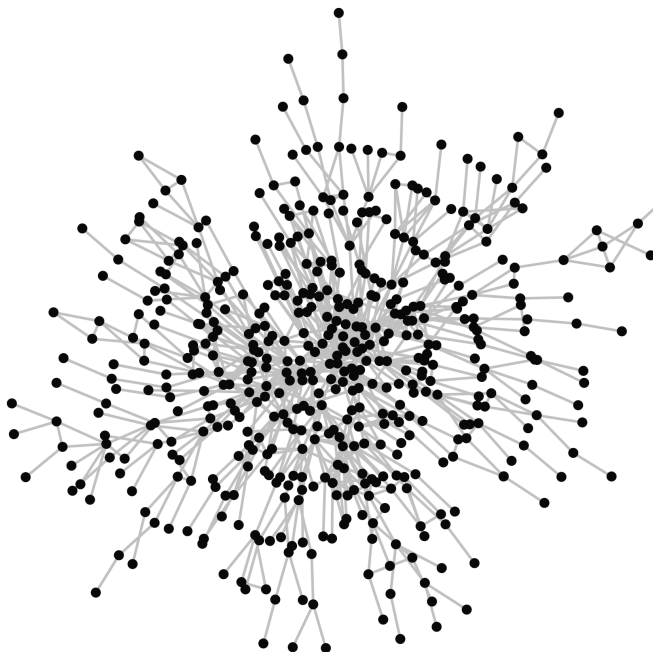
#### **1.3.4. Watts–Strogatz modell**



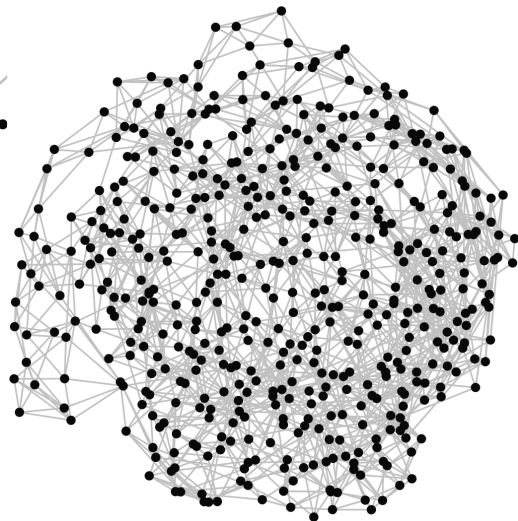
(a) Barabási–Albert típusú gráf, 1000 csomópont.



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



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



(d) Watts–Strogatz típusú gráf, 500 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:

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

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.

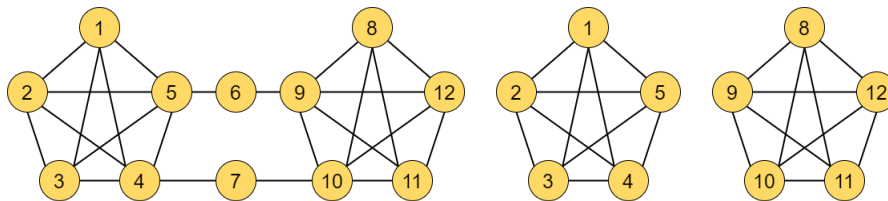
2.1. Listing. Páronkénti konnektivitás

```
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
```

#### Egy példa

A 2.1. ábrán látható gráfban, ha  $k = 2$  kritikus csomópontot kell azonosítanunk, akkor  $S = \{1, 2\}$  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 csúcislefedéséből.<sup>1</sup> Ez lesz a kezdeti  $S$  megoldásunk. A maradék csomópontok  $V \setminus S$  a gráf maximális független csúcshalmazát<sup>2</sup>  $MIS$  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 visszatesszük 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:    $S \leftarrow \text{VERTEX COVER}(G)$ 
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

```

---

**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.

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**

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 ismétljük, 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.

**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.

Egy új fitness függvényt vezetünk be egyed esetén 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

**Algorithm 2.2** Genetic Algorithm

---

```

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

```

---

következő képlettel írjuk le:

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

A képletben szereplő  $S^*$  jelenti az eddig talált legjobb megoldást. A  $\gamma$  egy változó, amely abban segít, hogy fenntartsuk a változatosságot a populáció egyedei között, megbüntetve azokat, amelyek túl közel vannak a legjobbhoz. A  $\gamma$  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  $\alpha$  pedig a képletben található változók egymás feletti fontosságát befolyásolja.

A  $\pi$  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  $\pi$  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  $S^*$ ,  $\gamma$  és  $\pi$  változók frissítését a 2.4 kódrészlet mutatja be.

**Algorithm 2.3** Random Solution

---

```

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

```

---

**Algorithm 2.4** Update  $S^*$ ,  $\gamma$  and  $\pi$  variables

---

```

1: function UPDATE( $N, P, S^*, \pi, \pi_{\min}, \pi_{\max}, \Delta\pi, \alpha$ )
2:    $avg \leftarrow 0$ 
3:   for  $i \leftarrow 1, N$  do
4:      $S \leftarrow P[i]$ 
5:      $avg \leftarrow avg + |S \cap S^*|$ 
6:   end for
7:    $avg \leftarrow \frac{avg}{N}$ 
8:    $\gamma \leftarrow \frac{\alpha \cdot f(S^*)}{avg}$ 
9:    $S \leftarrow P[0]$ 
10:  if  $f(S) < f(S^*)$  then
11:     $S^* \leftarrow S$ 
12:     $\pi \leftarrow \pi_{\min}$ 
13:  else
14:     $\pi \leftarrow \min(\pi + \Delta\pi, \pi_{\max})$ 
15:  end if
16:  return  $S^*, \gamma, \pi$ 
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 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

**Algorithm 2.5** Recombination Operator

---

```

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

```

---

valamilyen csekély valószínűséggel. A mutáció azért tartozik a nagy döntések halmazába, mert a mutáció révén fenntartjuk a populáció sokféleségét, és elkerüljük a korai konvergenciát.<sup>3</sup>

A populáció minden egyes új egyede esetén, a mutáció valószínűségét a  $\pi$  paraméter befolyásolja. Generálunk egy egyenletes eloszlású véletlen számot 1 és 100 között, és ha ez kisebb, mint  $\pi$ , 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 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  $N$  egyedet, és ezeket visszük tovább a következő iterációba. Genetikus algoritmusunk szelekciós szakaszát a 2.7 kódrészlet ismerteti.

---

3. Angolul: premature convergence.

## 2. FEJEZET: EGYCÉLÚ CNDP

---

### Algorithm 2.6 Mutation Operator

---

```

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

```

---



---

### Algorithm 2.7 Selection Operator

---

```

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

```

---

## 2.4. GA, de okos inicializálással

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 algoritmus, 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:    $P \leftarrow \emptyset$ 
3:   for  $i \leftarrow 1, N \cdot \frac{10}{100}$  do
4:      $P \leftarrow P \cup \{\text{GREEDY}(G, k)\}$ 
5:   end for
6:   while  $|P| < N$  do
7:      $P \leftarrow P \cup \{\text{RAND SOL}(k)\}$ 
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 nem egy függvényt fogunk optimalizálni, hanem kettőt. 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:

$$\max \quad |H|, \quad (3.1)$$

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

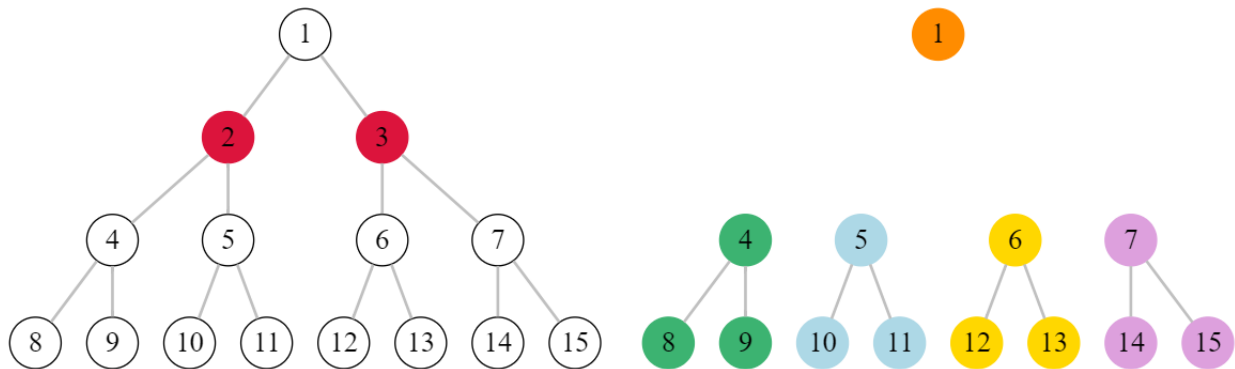
ahol  $H$ -val jelöljük a  $G[V \setminus S]$  feszített részgráf összefüggő komponenseinek a halmazát, és  $\text{var}(H)$  jelöli az összefüggő komponensek számosságának nem szabályos mintavételének a varianciáját. A  $H$  halmaz varianciáját a következő képlet segítségével számoljuk ki:

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

ahol  $n^* = \sum_{h \in H} |h|$  a  $G[V \setminus S]$  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:

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

3.1. Listing. A feszített részgráf összefüggő komponenseinek a száma

```
1 def connected_components(exclude=None):
    if exclude is None:
        exclude = {}
    S = set(exclude)
6 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))
8 num_of_components = len(components)
    num_of_nodes = subgraph.number_of_nodes()
    variance = 0
13 for component in components:
    cardinality = len(component)
    variance += (cardinality - num_of_nodes / num_of_components) ** 2
    variance /= num_of_components
```

### 3.2. Kísérleti előkészítés

Ebben a részben bemutatjuk a BOCNDP probléma megoldására javasolt genetikus algoritmusokat és ezek paraméterezéseit. A Platypus keretrendszert [Hadka, 2017] fogjuk használni.

**NSGAII** – Az NSGAII (Non-dominated Sorting Genetic Algorithm II) [Ventresca et al., 2018] 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*<sup>1</sup> 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*<sup>2</sup>, felállítva egy sorrendet az alpopulációk egyedei között, hogy az elszigetelt megoldásokat részesítse előnyben.

**EpsMOEA** – Az EpsMOEA (Epsilon Multi-Objective Evolutionary Algorithm) [Ventresca et al., 2018] egy egyensúlyi állapotú evolúciós algoritmus, amely  $\epsilon$ -dominancia archiválást használ a populáció sokszínűségének fenntartása végett.

**SPEA2** – A SPEA2 (Strength Pareto Evolutionary Algorithm 2) [Brownlee, 2011] feladata, hogy megtaláljon és fenntartsa egy frontnyi nem-dominált megoldást, ideális esetben egy halmaznyi Pareto-optimalis megoldást. Ennek elérése érdekében egy evolúciós eljárást használ - felhasználva a genetikai rekombinációs és mutációs operátorokat - a megoldástér felderítése végett, és egy szelekciós eljárást, amely fitness függvénye egy egyed dominánságának 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) [Li et al., 2017] alapötlete, hogy egy *bináris hipertér fogat 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

1. Angolul: non-dominated sorting.

2. Angolul: crowding distance.

### 3. FEJEZET: KÉTCÉLÚ CNDP

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, 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.

**EpsNSGAI** – Az EpsNSGAI (Epsilon NSGAI) [Ventresca et al., 2018] az NSGAI egy kibővített változata, amely  $\epsilon$ -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.
- Brownlee, J. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011.
- 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.
- 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.
- Li, W., Özcan, E., John, R., Drake, J. H., Neumann, A., és Wagner, M. A modified indicator-based evolutionary algorithm (mibea). In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1047–1054. IEEE, 2017.
- 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., 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.