TANSZÉKVEZETŐ

SZAKDOLGOZAT FELADAT

Hajdú Zsombor László
Mérnökinformatikus hallgató részére

Hálózati topológia bővítése regionális hibák védelmére

Az optikai kábelekből kiépített nagyméretű távközlési hálózatok igen sérülékenyek különféle meghibásodásokra. A probléma egyszerűbb eseteinek kezelésére -amikor egy-egy link vagy csomópont hibásodik meg- már léteznek jól bevált módszerek és protokollok. Regionális katasztrófák azonban (ilyen lehet például egy árvíz, földrengés, vagy hurrikán) egyszerre több hálózati elemben is kárt tehetnek, az így keletkező hibák pedig nehezebben kezelhetőek.

A regionális hibák ellen való védekezésnek az egyik módja a hálózat új összeköttetésekkel való bővítése. Az új, robosztusabb topológiával rendelkező hálózatnak egy esetleges katasztrófa után is biztosítania kell a megmaradó központok közötti zavartalan kommunikációt. Az új élek pontos útvonalainak kiválasztása azonban nem egyértelmű.

A hallgató feladatának a következőkre kell kiterjednie:

- Végezzen irodalomkutatást a regionális hibák elleni védekezési módszerek területén.
- Implementáljon több topológia bővítést javasló algoritmust.
- Vizsgálja meg ezen algoritmusok teljesítményét, végezzen összehasonlítást és értékelje az eredményeket.
- Biztosítsa az elosztott működést, amely egy időben több hálózatra, és külön méretű katasztrófára kiszámolja az optimális bővítést.
- Készítsen részletes dokumentációt és teszteket az alkalmazáshoz.

Tanszéki konzulens: Dr. Tapolcai János

Budapest, 2020. szeptember 27.

Dr. Magyar Gábor
egyetemi docens
tanszékvezető

Budapest Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék
H-1117 Budapest, Magyar tudósok krt 2. I.B.220
Telefon: 463-4187. Fax: 463-3107
www.tmit.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Hálózati topológia bővítése regionális hibák védelmére

Szakdolgozat

Készítette
Hajdú Zsombor László

Konzulens
dr. Tapolcai János
dr. Pašić Alija

2020. december 11.

Tartalomjegyzék

Kivonat **i**

Abstract **ii**

1. Bevezetés **1**

2. Irodalomkutatás **4**

3. Modellek, matematikai háttér **6**

3.1. A hálózat és hibák modellezése 6

3.2. Demonstráció egy egyszerű példán keresztül 8

3.2.1. Dupla élek esete 10

3.3. Az általánosított probléma 11

3.4. Algoritmusok 14

3.4.1. Heurisztikus algoritmus 14

3.4.2. A probléma egészértékű lineáris programja (ILP) 15

3.5. Geometriai részfeladatok 16

3.5.1. Particionálás 17

3.5.2. Hizlalás 17

3.5.3. Legrövidebb utak akadályok közt 18

4. Implementáció **19**

4.1. Tervezési megfontolások 19

4.2. A optimalizálás főbb lépései 20

4.3. A kód struktúrája 21

4.3.1. Használt Python könyvtárak 21

4.3.2. A használt C++ könyvtárak 25

4.4. Elosztott működés 26

4.4.1. Architektúra 27

4.4.2. Felhasznált elemek 28

4.4.3. Az online optimalizáló rendszer működése 29

5. Eredmények kiértékelése **31**

5.1. Egy valós indiai hálózat 32

5.2. Teljesítmény 33

5.3. Az eredmények összegzése 36

6. Összegzés **38**

7. További fejlesztési lehetőségek **40**

Köszönetnyilvánítás **41**

Irodalomjegyzék **42**

HALLGATÓI NYILATKOZAT

Alulírott *Hajdú Zsombor László*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2020. december 11.

Hajdú Zsombor László
hallgató

Kivonat

Manapság az életünk egyre inkább internet-alapúvá válik, így a telekommunikációs hálózatok védelmének fontossága egyre relevánsabb lesz. A hálózatban történő kimaradások gyakran regionális szintű katasztrófák következményei, mint például egy földrengés, áradás, hurrikán vagy akár bombatámadás. A hálózatok kiterjedése miatt sokszor rengeteg olyan felhasználót is érint egy-egy hiba hatása, akik egyébként fizikailag messze vannak a katasztrófától. A dolgozatomban következő kérdésre keresem a választ: Milyen módszerrel lehet olyan hálózatokat tervezni, amelyek ellenállónak regionális hibáknak, vagyis miként garantálható, hogy egy katasztrófa ne tudjon részekre szakítani egy hálózatot?

Esetünkben a problémát új összeköttetések kiépítésével oldjuk meg. Ekkor felmerül a kérdés, hogy az új optikai kábeleknek mely csomópontok közt, milyen útvonalon kell haladniuk, hogy optimális legyen a bővítés, persze amellett, hogy a hálózat „bombabiztosságát" (azaz azt, hogy a közvetlenül nem érintett felhasználók ne maradjanak szolgáltatás nélkül) is garantálni lehessen.

A szakdolgozatomban kitérek a problémát eddig (részben) tárgyaló szakirodalomra és az alkalmazott módszerekre, majd felépítem a matematikai modellt, aminek segítségével a probléma formalizálható lesz: a hálózatot egy irányítatlan geometrikus gráfként ábrázoljuk, majd számítógépes geometriai módszerekkel meghatározunk bizonyos „veszélyzónákat", amiket, ha katasztrófa (például egy bizonyos zónába eső epicentrumú földrengés) érne, a hálózat több részre esne szét. Ezután a feladat részekre bontásával és geometriai algoritmusok segítségével keressük az így keletkező komponensek közötti új élek legolcsóbb útvonalait.

Azután több lehetséges algoritmus dolgozok ki a leginkább költséghatékony hálózat bővítés kiválasztására. Az új élekkel kiegészített, bővített hálózat már adott méretű regionális katasztrófáktól védett lesz, akárhol is történjenek azok. A különböző módszereket implementáltam, majd a hatékonyságukat összevetve elemeztem őket. A gördülékeny munkához és teszteléshez egy elosztottan működni képes rendszert is felépítettem.

Abstract

Our modern life is becoming more and more Internet-based; thus, the importance of protecting telecommunication networks is getting more and more relevant. Internet outages are often caused by natural catastrophes, such as earthquakes, floods, hurricanes, or even bombing attacks. Due to the size of these networks, many users are affected by a given failure, even though they are physically far from the disaster area. In my thesis, focus on the following question: How can we create regional failure resilient networks, or in other words, how can one guarantee that a given disaster will not disconnect the network?

In this case, we solve the problem by building brand-new fiber optic links. This, of course, raises the question that between which end nodes and on what routes must the cables be built to guarantee the optimality. In other words, how to extend the network with links to make it "bombproof" (meaning that the users not directly affected by a disaster will still be able to use the services).

In this thesis, I overview the related work and the algorithmic techniques that can be applied. Then I define the mathematical model to formalize a network augmentation problem. The input is an undirected geometrical graph representing the network topology. Based on this, we identify "danger zones" using computational geometric methods, which if struck by a disaster (e.g., an earthquake with its epicenter in the zone), the network would fall into disconnected components. After this, we search for the shortest routes that the new edges can use to connect the components with the deconstruction of the task into smaller problems and using geometric algorithms.

Next, we present several heuristic algorithms to choose the set of edges as the minimum cost extension. The new network, extended with the freshly computed edges, will be protected against natural disasters, wherever they may strike. I have implemented these different methods and analyzed them, comparing their effectiveness. I have also bulit a distributed system for smooth operation and testing.

1. fejezet

Bevezetés

Ma már nehezen találunk olyan embert, akinek mindennapi élete ne függene valamilyen mértékben az internettől. Az világméretű hálózat maga több ezer kilométernyi optikai kábelből épül fel, amik hol tengerek fenekén, hol a földbe ásva, hol pedig felfüggesztve kötik össze a – szintén tekintélyes számban jelen lévő – kommunikációt végző központokat és felhasználókat egymással. Egy ekkora rendszerben azonban gyakoriak a kimaradások is, és a hibák elleni védekezés, azok kijavítása, és a hálózat megfelelő működésének biztosítása egy tipikus mérnöki feladat.

Az említett hálózati kimaradások 20%-át okozzák előre eltervezett karbantartások és tesztelési műveletek. A maradék 80% két részre bomlik: 70% szimpla, különálló felszerelés (azaz egy konkrét kábel, router, stb.) meghibásodásának számlájára róható, míg 30% katasztrófák által okozott többszörös meghibásodás [16, 9]. Példának találjuk a 2011-es japán földrengést, ahol tengeralatti optikai kábelek sérültek meg, és így körülbelül 1500 telekommunikációs csomópontot ért meghibásodás, vagy a 2018-as attikai tüzeket Görögországban, ahol a környező területeken nem volt lehetőség a kommunikációra, és a mentőcsapatok között is korlátokba ütköztek a kapcsolatteremtési kísérletek.

Annak ellenére, hogy a regionális katasztrófák milyen számottevő részét okozzák hálózati hibáknak, és hogy intuitívan ezek ellen a hibák ellen könnyebb lenne tervezni (hiszen nem véletlenszerűen lépnek fel a hálózatban, hanem összefüggő területekre koncentrálódnak), jóval kevesebb megoldást látunk a gyakorlatban a biztosításukra, mint az egyszerű meghibásodások esetében.

A hálózatot kiépítő, illetve üzemeltető cégeknek több opciójuk van regionális hibák elleni védekezésre. Egy triviális megoldás a már meglévő összeköttetések és csomópontok további védelemmel való ellátása. Ez bevett gyakorlat például az óceán fenekén haladó kábeleknél, ahol az élővilág[1] károsíthatja a vonalakat [42]. Legna-

[1]Egy érdekesség: vannak példák cápák által megrongált vonalakra, lásd [22].

gyobb előnye, hogy a bővítés „házon belül” tud maradni, ez a módszer azonban az esetek többségében nem nyújt teljes védelmet. A fix útvonalon haladó összeköttetések ugyan úgy veszélyes területeken haladnak, és ha mégis meghibásodás történik a kábel megerősítése kidobott pénz volt. A hálózat kiegészítésére két további opció van:

- Már meglévő, ún. „sötét” kábelek megvásárlása és beüzemelése.
- Teljesen új összeköttetések kiépítése.

Az előbbi megoldás abból a tekintetből lehet jó, hogy a kábelek már le vannak fektetve, így gyorsabb a beüzemelésük azzal szemben, ha a szolgáltató maga építené ki az összeköttetéseket. Azonban ezen a sötét kábeleknek már van egy meghatározott útvonala, így nem feltétlenül tudják biztosítani a teljes hálózat – értve ezalatt a hálózat összes csomópontjának – védelmét, a fix adatsebességeket nem is említve. Ezért az utolsó opcióval foglalkoztunk, ami a teljesen új összeköttetések kiépítését jelenti.

Ez a feladat már pusztán a geometriai részek (útkeresés, veszélyes területek identifikálása) miatt is roppant izgalmas, így a munka során kizárólag erre koncentráltunk. Természetesen a módszerek tetszőlegesen kombinálhatóak egymással, és egy későbbi munkának elképzelhetően témája lesz többféle megoldás beépítése a modellbe. Ezekről a lehetőségekről részletesebben beszélek a 7. fejezetben.

A szakdolgozatomban folytatom a regionális hibákat feldolgozó témámat, amit még egy éve, témalaboratóriumon kezdtem el. A munkából azóta egy TDK dolgozat[2], illetve egy IEEE INFOCOM 2021 konferenciára elfogadott cikk [35] is született. A korábbi munka során egy heurisztikus algoritmust használtunk a hálózathoz bővítésként alkalmazott élek számítására, azonban ennél az eredeti heurisztikánál jobbnak bizonyult egy ILP-t alkalmazó megoldás. Több algoritmust is kipróbáltunk, illetve született egy új, akár elosztottan is működni képes keretrendszer, aminek segítségével könnyebb a modell kibővítése, más módszerek próbálgatása.

A szakdolgozat felépítése innentől a következő: Egy rövid irodalomkutatással kezdem a 2. fejezetben, ahol kitérek a regionális hibákat tárgyaló, eddig megjelent cikkekre. Több modellel, ötlettel, és megoldással találkozhatunk, azonban egyik eddigi írás sem foglalkozik kifejezetten új élek kiépítésével abból a célból, hogy a hálózat regionális katasztrófákkal szembeni védettségét biztosítsák.

Az szakirodalom után a feladat matematikai modellének megépítésével folytatom a 3. fejezetben: A hálózatokat gráfokként értelmezzük, majd különböző számítógépes geometriai módszerekkel meghatározzuk a modell által definiált feladat

[2]A dolgozat elérhető a TDK portálon: http://tdk.bme.hu/VIK.

megoldásához szükséges többi részelemet is. Ezek után egy-két egyszerűbb példán keresztül demonstrálom az algoritmusok működését.

A matematikai modell után a 4. fejezetben beszélek az implementációról, a tervezési megfontolásokról, és indoklom a választott eszközöket, valamint a kivitelezést. Ez a rész egy beszámoló a kódról, illetve annak egy dióhéjba csomagolt, magas szintű dokumentációja.

Végül az algoritmusok működését és teljesítményét fogom demonstrálni a 5. fejezetben, ahol konkrét, valós hálózatokra való futtatások eredményeit vizsgálom. Mélységében bemutatom az algoritmusok sajátosságait is egy példán keresztül, illetve több gráfra, mennyiségében is összevetem a két módszert egymással.

2. fejezet

Irodalomkutatás

A dolgozat témájához lazán kapcsolódik az egyszerű hibák kérdésköre, persze itt az egyszerű hibák alatt a nem regionális katasztrófák okozta hibákat értem. A rendelkezésre álló szakirodalom jelentős hányada ilyen típusú hibákat boncolgat. A cikkek nagy része az egyszeres él- vagy csomópont-kieséseken túl azonos hibacsoportba[1] tartozó, nagy valószínűséggel együtt meghibásodó elemek kiesésével foglalkozik. Több ok miatt lehet különböző hálózati elemeket azonos csoportba sorolni, például egy kábelen osztozik több összeköttetés, vagy földrajzilag is közel vannak egymáshoz. A terület egy extenzíven kutatott része a hálózattervezésnek [38, 43, 32, 41], rengeteg megoldással már évek óta találkozhatunk is különféle forgalomirányító protokollokban (főleg kapcsolat-állapot, mint például az *OSPF* vagy az *IS-IS*) implementálva.

A közelmúltban kezdett egyre nagyobb figyelemnek örvendeni a regionális hibák elleni védekezés – pontosabban a nem védettség, és a hálózatok sérülékenysége – [28, 12]. Például egy 2013-ban megjelent cikk [13] a telekommunikációs hálózatok védtelenségére hívja fel a figyelmet, hiszen ezeknek a hálózatoknak a zavartalan működése már a mindennapjainkban elvárt, meghibásodásuk pedig mindenki számára komoly problémákat okozhat.

[20] is kiemeli a regionális hibák elleni védelem fontosságát, és egy érdekes, tenger alatti kábelek fektetésével foglakozó optimalizációs problémát vet fel, ahol egy beálló katasztrófa költségeit minimalizálják. A nem katasztrófa által okozott hibákat, azaz az egyszerű meghibásodásokat egyszerűen egy, a „társadalmi és üzemeltetői költségeket is figyelembe vevő" úton lefektetett plusz tartalék kábellel kezelik. Konkrét esetnek van felhozva a Földközi-tenger hibabiztossá tétele az általuk bemutatott lineáris programozást használó megoldással.

[1] lásd: SLRG-k, *Shared Link Risk Groups*

Több írás foglalkozik behatóan azzal is, hogy milyen matematikai eszközökkel érdemes modellezni a regionális katasztrófákat [21, 33, 3, 15]. Ezen munkák többsége azt feltételezi, hogy a regionális hibának valamilyen síkbeli alakja van. Ez legtöbbször nemes egyszerűséggel egy kör [24]. Erre a modellre az angol irodalomban „*circural disk failure*" néven bukkanhatunk rá.

Az optimális megoldások témáját néhány cikk, mint például [15] is taglalja, ahol a linkek idő alatti terhelése is szerepet játszik. A feladat, és hozzá hasonló problémák nehézségével [11] foglalkozik, míg [29, 27] a késleltetést, valamint a protokollokból adódó hibák kezelését helyezi előtérbe különböző hálózattervezési módszereket alkalmazva.

Vannak munkák, amik az érintett hálózati elemek adott osztályai szerint kezelik a problémát. Míg [23, 30, 10] a szárazföldi összeköttetéseket vizsgálja, találunk a víz alatti kábelekkel foglalkozó cikkeket is [5, 39].

Egy további érdekes megközelítés az ún. geodiverz forgalomirányítás kiterjesztése, amikkel több cikkben is foglalkoznak [7, 8, 2]. Ez a módszer azon alapszik, hogy a kapcsolat kialakításakor használt utaktól megkövetelik, hogy fizikailag kellő távolságra legyenek egymástól. Például [8] folyamokkal közelíti meg a problémát, míg [2] pont-párok közötti elérés biztosításával foglalkozik.

A topológiák k-összefüggőségéről említést tesz egy másik [31] cikk, ami kiemeli, hogy bár legtöbbször az összefüggőség van metrikaként használva, a hibák mégis sokszor egy területen lépnek fel, ezért valamilyen lokalitást kell figyelembe venni a vizsgálatokkor. Ezt „régió alapú összefüggőségnek" nevezik, és bemutatják, hogy a RAÖ-val való számolás sok esetben jobb eredményeket produkál, mint az egyszerű régió alapú hibamodellek. Említést tesznek még különböző megkötésekről, illetve a probléma NP-nehézségéről.

Kicsit másfelől megközelítve, [4] és [40] forgalomirányítás, valamint szoftver szempontból áll a problémához, míg [1] egy sztochasztikus modellt ajánl fel a hálózat hibabiztosításának megoldására.

Ahogy már említettem, itt azt az esetet vizsgálom, amikor a meglévő hálózati infrastruktúrán nem változtatunk, és a bővítés is csak és kizárólag új élekkel megengedett. Így biztosítható a legjobban (ugyanakkor sajnos a legdrágábban) az egész hálózatra kiterjedő védelem.

3. fejezet

Modellek, matematikai háttér

3.1. A hálózat és hibák modellezése

A probléma formálisabb kezelésének érdekében a hálózatunkat egy

$$G = (V, E)$$

összefüggő, irányítatlan, geometrikus gráfként tudjuk elképzelni, ahol v_i és e_j

$$v_i \in V, |V| = n, i = 1 \dots n$$

$$e_j \in E, |E| = m, j = 1 \dots m$$

a hálózat csúcsai, illetve élei. A gráf nevében a geometrikus azt jelenti, hogy csúcspontjait a 2 dimenziós síkon egyértelműen meg tudjuk feleltetni pontoknak (koordinátáknak), azaz v_i-ket a következőképpen kezelhetjük:

$$v_i \longmapsto v_i^p, \quad v_i^p \in \mathbb{R}^2$$

ahol i nem más, mint az index (vagy a csomópont sorszáma/neve), v_i^p pedig az adott csomóponthoz tartozó koordináta. Az éleket – bár a valóságban legtöbbször nem tökéletes egyeneseket követnek – egyszerűen az adott él két végpontja között húzott egyenként képzeljük el a geometriai értelemben:

$$e_j = \{v_a, v_b\} \longmapsto (v_a^p, v_b^p) \text{ szakasz}$$

A modell persze kicsit egyszerűsít a valóságon azzal, hogy csak egyenes éleket engedünk meg csomópontok közt, azonban nagyobb kiterjedésű hálózatokra, mint

például a gerinchálózatok, ez a feltevés jó közelítéssel a legtöbb esetben fent áll. *(Megjegyzés: A fejlesztés során kipróbáltam görbe éleket sok egyenesre felbontva modellezni, ez azonban a komplexitás szempontjából nem volt előnyös.)*

Az egyszerűség kedvéért a kör alakú hiba modelljét adaptáljuk a feladat megoldása során. Tehát a katasztrófákat

$$C \longmapsto (p_c, r)$$

módon írjuk fel, ahol p_c az epicentrum, és r a katasztrófa kiterjedése – itt a kör sugara. Ez a módszer túlbecsüli egy katasztrófa által sújtott terület nagyságát, hiszen a valóságban legtöbbször nem egy tökéletes körben helyezkednek el a károsodott területek.

Most, hogy a kiinduló elemeinket (a hálózat éleit és csomópontjait, valamint a katasztrófákat) felírtuk, továbbhaladhatunk a feladat problémájának formalizálására. Először definiáljuk, hogy mikor károsít egy-egy katasztrófa egy adott hálózati elemet.

Definíció 1. Egy C katasztrófa akkor **talál el** egy v_i^p koordinátákkal rendelkező v_i csúcsot, ha $v_i^p \in C$ (tehát a csúcs a kör területén belül található). Az eltalált csúcsok halmazát V_c-vel jelöljük. ▪

Ennek analógiájára:

Definíció 2. Egy C katasztrófa akkor **talál el** egy $e_j = \{v_a, v_b\}$ élet, ha annak két végpontja között húzott (v_a^p, v_b^p) szakasz elmetszi C-t. Az eltalált élek halmazát E_c-vel jelöljük. ▪

Ezek a katasztrófák tehát hibákat okoznak, ha eltalálnak egy csúcsot vagy élet (tehát $V_c \neq \varnothing \vee E_c \neq \varnothing$), azonban mi csak egy bizonyos részükre vagyunk kíváncsiak. Szükséges definiálnunk tehát, hogy mikor *kritikus* egy hiba a hálózatnak: ha a maradék csúcsok és élek nem összefüggő gráfot alkotnak.

Definíció 3. Egy C hiba akkor **kritikus hiba**, ha $V_c = (V \setminus V_c, E \setminus E_c)$ módon definiált maradék gráf több komponensből áll. ▪

A feladat tehát innen adott: Az összes elképzelhető C katasztrófa ellen védenünk kell a hálózatot.

Probléma 1 ***Általános hálózat-bővítési probléma:*** *Adott $G = (V, E)$ geometrikus irányítatlan gráf. Keressük azt az E_U élhalmazt, amelynek összköltsége minimális, és vele kiegészítve G-t a sík tetszőleges p_c pontjába eső $C = (p_c, r)$ katasztrófa*

ellen védett lesz a hálózat, azaz C katasztrófa $G = (V, E \bigcup E_U)$ gráfban nem tud kritikus hibát okozni.

Fontos megemlíteni egy-két feltevést/kikötést is, amikkel élnünk:

- Az új élek költsége egyenesen arányos a hosszukkal
- Az új élek akár tetszőleges görbéket is követhetnek
- Az élek ugyan olyan nagy, és minden esetben elegendő sávszélességet nyújtanak

Ezen kívül persze azt is fontos tisztázni, hogy a katasztrófa sújtotta területekkel ez a modellen belül nem foglalkozunk. A megoldás során arra koncentrálunk, hogy egymástól távoli, katasztrófa által nem sújtott csomópontok zavartalanul tudjanak kommunikálni egymással, és csak új élek behúzását „engedjük meg". Végül a további tárgyalás egyszerűsítése érdekében még két definíciót bevezetünk:

Definíció 4. Egy $C = (p_c, r)$ katasztrófa esetén p_c pont akkor **veszélyes pont**, ha V_c gráf nem összefüggő. ▪

Ezeket a pontokat a következőképp, „veszélyzónákba" osztályozzuk:

Definíció 5. Z_i **veszélyzóna** *(Danger Zone)*, ha $\forall p_z \in Z_i$ pont esetén $C = (p_z, r)$ katasztrófa után maradó V_c maradék gráf ekvivalens, és Z_i nem bővíthető további pontokkal. ▪

Tehát a sík pontjait az alapján pakoljuk szét, hogy milyen módon tesznek kárt a gráfban.

3.2. Demonstráció egy egyszerű példán keresztül

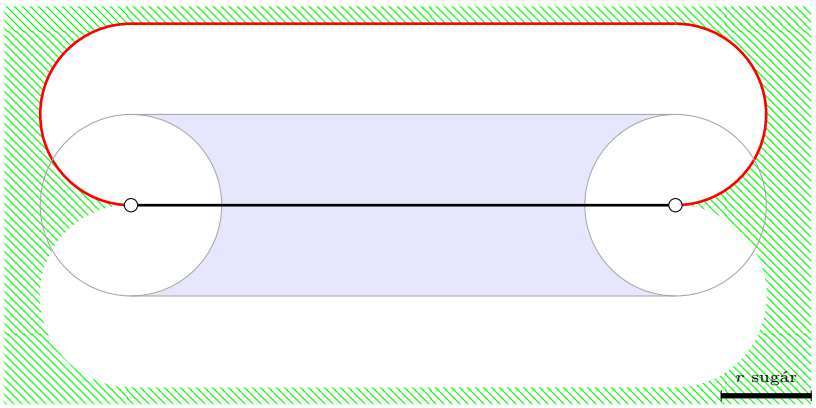
A fentebb definiált, Általános hálózat-bővítési probléma első ránézésre bonyolultnak tűnhet. Elvégre adott r paraméter mellett a hálózat bármely pontjához legfeljebb r távolságra fekvő pontról meg kell vizsgálni, hogy kritikus hibát okoz-e, majd ezeket a pontokat valamilyen módon rendszerezni kell, és kitalálni egy útvonalat a védelmükre. Az alábbiakban azonban bemutatom, hogy a probléma megközelítésére létezik egy intuitív, és ráadásul hatékony megoldás is.

Elsőként tekintsünk egy 2 csúcsból és az őket összekötő élből álló, triviális hálózatot, ahol a csúcsok kellően távol helyezkednek el egymástól (azt, hogy miért van szükség erre a távolságra, a következő szekcióban meglátjuk). Ebben az esetben a következő gondolatmenettel határozhatjuk meg a veszélyes pontokat:

1. Egy adott v_i^p pont körüli, legfeljebb r távolságban lévő összes pont v_i csúcs kiesését eredményezi. Azonban csak egy pontot kivéve a hálózatból a definíció szerint nem történik kritikus hiba, hiszen a megmaradó, izolált pont önmagában egy összefüggő gráfot fog alkotni. $\longrightarrow$ A pontok körüli r sugárban nem történik kritikus hiba.

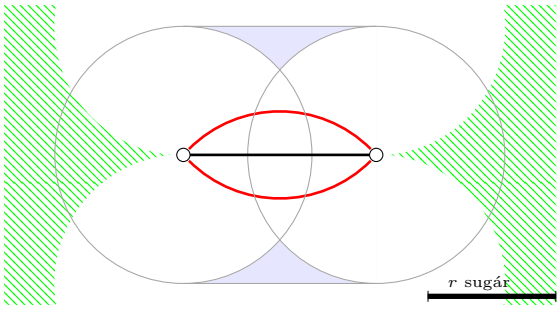
2. Ha a hálózat bármely elemétől nagyobb, mint r távolságra kerül egy katasztrófa csúcspontja, akkor egyik elemet sem találja el, tehát a hálózat sehol nem sérül és biztos összefüggő marad. $\longrightarrow$ A hálózattól r távolságra nem történik kritikus hiba

3. Ha egyébként a két ponttól nagyobb mint r távolságra, de az élhez közelebb, mint r távolságra történik a katasztrófa, akkor csak az él esik ki, és a két pont izolálódik. $\longrightarrow$ Az éltől r-nél közelebbi, de a csúcsoktól r-nél távolabbi pontokban történik a kritikus hiba.



3.1. ábra. Egy egyszerű példa.
A veszélyzóna kékkel van jelölve.

A 3.1-es ábrán jól látszik, hogy néz ki az erre a hálózatra megállapított egyetlen veszélyzóna: az él köré rajzolt „virsliből” kivesszük a két végénél található, csúcspontok köré rajzolt köröket. Az így kapott alakzat lesz az (összefüggő) veszélyzóna. Az ábrán pirossal már fel van tüntetve a megoldás is: A veszélyzónától r távolságra vezetett görbe biztosítja, hogy a hálózat ellenálljon a kritikus hibáknak. Ez érthető, hiszen így a veszélyzónának nem lesz olyan pontja, ami mindkét élnek egyszerre van az r távolságú "sugarában", azaz az adott katasztrófák esetén az egyik él garantáltan meg fog maradni.



3.2. ábra. Rövid él

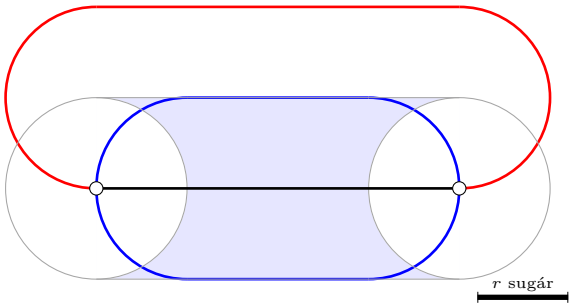
3.2.1. Dupla élek esete

Az előbbi példában ki kellett kötni, hogy a két csúcs kellő távolságra legyen egymástól. Erre azért volt szükség, mivel vannak esetek, amikor nem egyetlen egy új él behúzása nyújtja a legjobb megoldást. Ezt demonstrálom a következő példán:

A 3.2-es ábrán látszik, hogy alakul át a probléma, hogy ha a két kör már metszi is egymást. Ekkor jelentősen lecsökken az előző fejezetben végigfuttatott gondolatmenet alapján kapott terület, és bár egy veszélyzónának számít, nem lesz összefüggő.

Ekkor a két izolált rész egyszerűbb külön veszélyzónaként kezelni, hiszen ha egyszerre akarnánk kikerülni mindkettőt, sokkal nagyobb költséggel kellene számolnunk, mintha a zónát két részre bontva külön-külön, sokkal rövidebb útvonalakon kerülnénk ki.

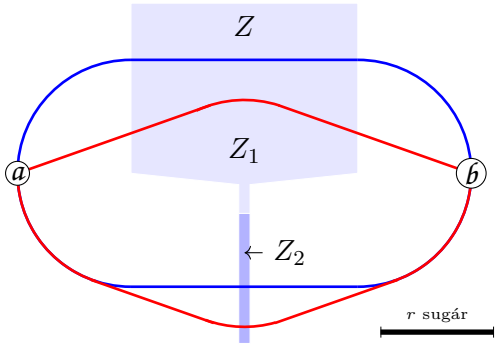
Az, hogy mikor érdemes a veszélyzónát két részre vágni ebben az egyszerű esetben triviális. A 3.3 ábra mutatja, hogy ha már a körök nem metszik egymást, a megoldás költsége $2|e| + r^2\pi$ lesz, ahol $|e|$ az él hossza. Ez akkor lesz egyenlő az egyszeres él költségével, amikor $|e| = 4r$.



3.3. ábra. $4r$ hosszú él

A veszélyzónák kettévágása általában egyszerűen csak a két végpont közötti egyenes mentén történve ad optimális eredményt, de az egzaktság kedvéért meg kell említeni az általános esetet is. Egy tetszőleges alakú kikerülendő területnél nem feltétlenül az él mentén való szeparáció nyújtja a legjobb megoldást. A 3.4 ábra segít

például elképzelni egy olyan esetet, ahol az él mentén történő kettévágás nem a legjobb dupla éles megoldást generálja. A tesztelés során azonban sehol nem alakult ki egy olyan veszélyzóna-konfiguráció, ami ilyen nehezen kezelhető alakzatokat eredményezett volna, ezért –ha egyáltalán szóba jöhet a dupla éles megoldás, $4r$ távolságnál egymáshoz közelebb lévő pontok esetén– a kettévágásokat az algoritmus egyszerűen az él mentén végzi el.



3.4. ábra. Egy lehetséges alakzat, ahol az ab egyenes mentén történő kettévágás nem az optimális eredményt nyújtja.

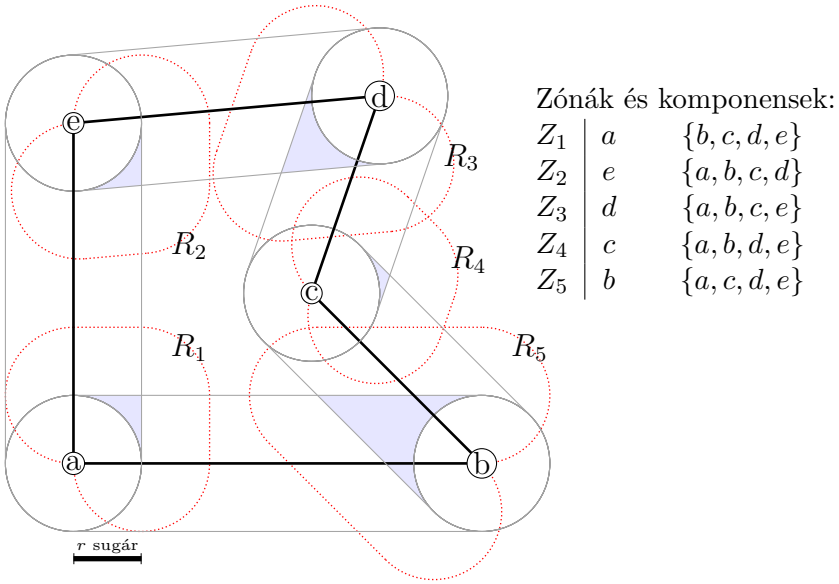
3.3. Az általánosított probléma

Ahogy a példákból már láttuk, a veszélyzónákat viszonylag egyszerű módszerrel meg tudjuk határozni: ha minden egyes csúcs köré rajzolunk egy r sugarú kört, és minden éltől r távolságra húzunk párhuzamos szakaszokat, akkor az egész síkot fel tudjuk bontani jól elkülönülő lapokra. Ezeket a köröket és párhuzamosokat a továbbiakban *görbének* fogjuk nevezni.

Az így létrejött síklapok közül mindegyikre igaz lesz az, hogy minden pontjukba eső, r sugarú katasztrófa ugyan úgy bontja szét a hálózatot, mivel a hálózati elemektől csak r távolságra húztuk a felbontó görbéket. Úgy is lehet tekinteni erre, hogy az adott hálózati elem vagy kiesik vagy nem, hiszen a görbék „binárisan osztályozzák” a síkrészeket. Ez összesen maximum $(n+m)^2$ darab konfigurációt tud eredményezni.

A 3.5 ábrán látszik, hogy egy egyszerűbb hálózat esetén mit fog eredményezni a felbontás. A különböző lapokat könnyű osztályozni, hiszen csak egy pontot kell a belső pontjaik közül mintavételezni ahhoz, hogy el tudjuk dönteni, mely hálózati elemek fognak kiesni. Az ábrán a kritikus területek (veszélyzónák) kékkel vannak jelölve.

Miután az osztályzással végeztünk, a következő feladatunk a veszélyzónákat r távolságra kerülő görbék megtalálása. Ehhez az egyszerűség kedvéért a zónákat



3.5. ábra. Egy egyszerű példa hálózat. A veszélyzónák kékre vannak színezve, és a felhizlalt régiók piros szaggatott vonallal vannak határolva.

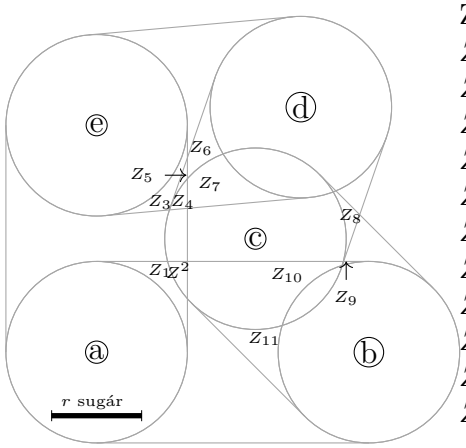
„felhizlaljuk”, aminek az eredményeként az ábrán piros szaggatott vonallal jelölt síkidomokat kapjuk. Ezeket a síkidomokat kerülve az eredeti zónákat r távolsággal fogjuk kerülni, azaz biztos a legrövidebb, még megfelelő utakon fogunk haladni.

Fontos megjegyeznünk, hogy különböző r értékekre különböző felbontásokat, és így különböző konfigurációkat fogunk kapni a síkon, így minden új r esetében ezt újra ki kell számolnunk. Nagyobb r értékekre bonyolultabb lesz a felosztás, mivel több metszéspont keletkezik az alakzatok között. Ebből kifolyólag több lap lesz potenciálisan veszélyzóna, ahogy az a 3.6 ábrán is látható.

Mivel a felbontásokban után előfordulhatnak olyan veszélyzónák is, ahol a hálózat több, mint két komponensre esik szét, a veszélyzónák alapján kialakítunk „vágásokat”. Ezek nem mások, mint két nem üres diszjunkt részhalmazai a komponenseknek, amik együttesen tartalmaznak minden komponenset. Úgy is gondolhatunk erre, mintha „két oldalra válogatnánk” a fennmaradó komponenseket.

Definíció 6. Egy G gráfot Z_i veszélyzóna k db komponensre bontja szét, a komponensek halmaza: $H = \{h_1 \dots h_k\}$. Ekkor Z_i-hez tartozó C **vágás** egy $C = S_a|S_b$ pár, ahol $S_a \subset H, S_b \subset H, S_a \neq \varnothing, S_b \neq \varnothing, S_a \cup S_b = H, \text{és} S_a \cap S_b = \varnothing$ ▪

Erre azért van szükség, mivel egy adott él maximum két komponenst tud összekötni, így a megoldásban gondolnunk kell azokra az esetekre is, amikor 3 vagy több részt kell újra összekötnünk, így gondoskodva arról, hogy egyik komponens se maradjon izoláltan a javítás után. Ha egy veszélyzóna több vágást eredményez, például



Zónák és komponensek:

Z_1	a	$\{b, c, d, e\}$
Z_2	a b	$\{d, e\}$
Z_3	e	$\{a, b, c, d\}$
Z_4	e d	$\{a, b\}$
Z_5	e d	$\{a, b, c\}$
Z_6	d	$\{a, b, c, e\}$
Z_7	d	$\{a, b, e\}$
Z_8	c	$\{a, b, d, e\}$
Z_9	b	$\{a, c, d, e\}$
Z_{10}	b	$\{a, d, e\}$
Z_{11}	b	$\{a, c, d, e\}$

3.6. ábra. Az előbbi példa egy nagyobb sugárral számolva. Figyeljük meg, hogy itt már megjelennek olyan veszélyzónák is (Z_4, Z_5), amik több komponensre bontják a hálózatot.

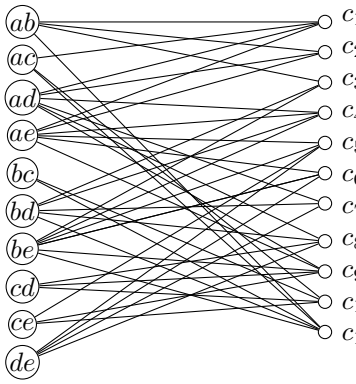
három (A, B, C) komponenst, akkor ezeket 3 db vágásra, azaz $A|BC$-re, $B|AC$-ra, és $C|AB$-re bontjuk szét. Általánosan t különálló komponens esetén összesen $2^{t-1} - 1$ db vágást fog egy veszélyzóna eredményezni.

Miután ezt a lépést az összes veszélyzónára elvégeztük, rendelkezésünkre állnak $c_1 \dots c_l$ vágásaink. Azokat a vágásokat persze összevonhatjuk, amiknél ugyan az az eredményezett két csúcshalmaz-pár.

Ezek után a teljes problématér „felderítéséhez" definiálunk egy (V_{pp}, V_v, E_j) páros gráfot, amit később az algoritmusok is kiindulási pontként fognak alkalmazni. Ebben a páros gráfban a V_{pp}-ben található csomópontok a gráfba behúzható összes plusz élt reprezentálják (tehát pont-párokat, és a közöttük futó potenciális éleket tartalmaznak), $V_v = \{c_1 \dots c_l\}$ pedig a vágásokat jelölő csúcsok halmaza. Egy $x \in V_{pp}$ és egy $y \in V_v$ csúcs között futó élt akkor húzunk be, hogyha az a bizonyos $x = (a_x, b_x)$ él képes összekapcsolni a y csúcsban tárol vágás két részét. Például a 3.7 ábrán c_4 vágás $\{a, b\}, \{d, e\}$ csúcshalmazokból áll, tehát javító él lehet számára a külön halmazokban található csúcsok közötti bármely él, így ad, ae, bd és be.

Fontos figyelni a terminológiára, ugyanis bár V_{pp} elemei az optimalizáláshoz használt páros gráfban *csúcsok*, azonban az eredeti topológiában *csúcs-párok között húzható éle*ket jelentenek.

Ezzel azonban mindenünk megvan ahhoz, hogy megkezdjük az élek kiválasztását.



Vágás	Veszélyzónák	Halmazok	
c_1	Z_1	a	$\{b, c, d, e\}$
c_2	Z_2	a	$\{b, d, e\}$
c_3	Z_2, Z_{10}	b	$\{a, d, e\}$
c_4	Z_2, Z_4	$\{a, b\}$	$\{d, e\}$
c_5	Z_3, Z_5	e	$\{a, b, c, d\}$
c_6	Z_4	e	$\{a, b, d\}$
c_7	Z_4, Z_7	d	$\{a, b, e\}$
c_8	Z_5, Z_6	d	$\{a, b, c, e\}$
c_9	Z_5	$\{e, d\}$	$\{a, b, c\}$
c_{10}	Z_8	c	$\{a, b, d, e\}$
c_{11}	Z_9, Z_{11}	b	$\{a, c, d, e\}$

3.7. ábra. Az optimalizáláshoz felépített páros gráf

3.4. Algoritmusok

A fentebb ismertetett páros gráfból több módszerrel lehet kiválogatni éleket. Első intuíciók egy mohó elven működő heurisztikus módszer, ahol bizonyos szempontok alapján egymás után válogatunk be éleket a végső megoldásba. Azonban a gráfban tárolt információk alapján akár optimális megoldást is el tudunk érni, ha egy lineáris programozást használó módszert alkalmazunk. Ebben a szekcióban bemutatom a két algoritmust:

3.4.1. Heurisztikus algoritmus

A heurisztikus algoritmus egyszerűen mindig az adott lépésben legjobbnak látszó élet fogja beválogatni a megoldáshalmazba. A legjobbnak látszó élet egy választófüggvénnyel határozzuk meg. Maga a függvény például egyszerűen az út hossza, átlagos hossz védett vágásonként, vagy a vágások száma lehet. Az algoritmus minden iteráció során a legjobb élet (tehát a páros gráfban bal oldalt, V_{pp}-ben lévő csúcsot) válogatja a megoldáshalmazba, majd kitörli az élet az összes jobb oldali szomszédjával együtt, hiszen azokat a vágásokat felesleges lenne még egyszer védeni. Ezután az bal oldalon újraszámolja a költségeket, majd a választó-függvény alapján újra választ (azaz kiszámolja az értéket minden lehetséges élre, majd a minimumot/maximumot veszi). Ezt az eljárást pedig addig folytatja, amíg le nem védtük az összes vágást.

Az algoritmus tehát formálisan, tetszőleges $choose(x, c_x)$ választó függvény esetén:

Inicializálás Legyen a beválogatott V_{pp}-beli csúcsok halmaza H_x (kezdetben $\varnothing$). Rendeljük hozzá c_x a költséget minden $x = \{a, b\} \in V_{pp}$ csúcsra a következő

módon:

$$c_x = \text{route}(a, b, r, Z_H)$$

Ahol a Z_H minden olyan veszélyzóna, amit x csúcs bal oldali szomszédjai tárolnak (azaz a csúcs által védett vágásokat alkotó veszélyzónák), és a „route" függvény a csúcsban tárolt $a - b$ pontok közötti legrövidebb út, ami minden Z_H-beli zónát legalább r távolságra kerül.

1. lépés Válasszuk ki azt az $x \in V_{pp}$ csúcsot, ami a $choose(x, c_x)$ választó-függvény alapján a legjobb választás! legyen ez a csúcs x_{best}

2. lépés Tároljuk el x_{best}-et a beválogatott H_x csúcspontok halmazába, és töröljük a gráfból x_{best}-et, és minden $N(x_{best})$ szomszédos csúcsát.

4. lépés Ha van még le nem védett vágás, számoljuk újra a bennmaradt csúcsok c_x költségeit a *route* függvénnyel, majd folytassuk az 1. lépésnél, egyébként STOP.

Eredményül kapjuk H_x -et, a heurisztika által kiválasztott élek halmazát.

3.4.2. A probléma egészértékű lineáris programja (ILP)

Észrevehető, hogy a heurisztika figyelmen kívül hagyja azokat az eseteket, amikor egy éllel nem az összes szomszédos vágást akarjuk védeni. Ezért sokszor H_x halmaz nem az optimális bővítésben található élhalmazt adja eredményéül.

ILP (Integer Linear Programming - Egészértékű Lineáris Programozás) segítségével azonban le tudunk írni a páros gráfban talált információk alapján egy olyan feladatot, ami a fentebb említett lehetőségeket se hagyja ki a keresési térből, és így az optimális megoldással tud szolgálni. Az LP problémákban valamilyen függvényt minimalizálunk vagy maximalizálunk bizonyos korlátozó tényezők (kényszerek) mentén. A hálózatbővítési problémát a páros gráfból a következőképp írhatjuk fel:

ILP

Az alapötlet az, hogy minden egyes lehetségesen behúzható élhez hozzárendelünk egy bináris (beválogatjuk/nem válogatjuk be) változót. Itt a lehetséges éleket úgy kell érteni, hogy egy adott $v_x \in V_{pp}$ csúcsot összes $N(v_x)$ szomszédjának kombinációjával végigszámoljuk, és hozzárendeljük a költséget az adott kombináció által reprezentált bináris változóhoz. Tehát ha egy ab él védheti c_1, c_2 és c_3 vágásokat, egy változó lesz csak a c_1-et védő variáns, egy változó a csak c_2-t, stb. egészen a mindhármat védőig. figyeljük meg, hogy itt exponenciális számú változóról van szó, ezért nem minden esetben fog komplexitás szempontjából a legjobb lenni a megoldás.

Jelöljük

- S^x-el $N(v_x)$ összes nem üres részhalmazát
- s-el egy tetszőleges elemet S^x-ből
- z_s-el egy s-hez tartozó bináris változót
- C_s-el egy s-hez tartozó költséget

Ekkor a célfüggvényünk:

$$\text{minimalizálni:} \sum_{\forall v_x \in V_{nn}} (\sum_{\forall s \in S^x} C_s * z_s)$$

A kényszerek pedig:

$$\sum_{s \in S | c \in Z_s} z_s \geq 1 \qquad \forall v_c \in V_v$$

Tehát minden vágáshoz kell, hogy tartozzon védő él.

Az ILP probléma duálisát alkalmazó heurisztika (DLPH)

A teljes ILP feladat mindig optimális, de nem mindig elég gyors megoldással tud szolgálni. Ezért a feladat duálisából kiindulva előállunk egy másik, LP alapú heurisztikával is.
Ezen esetben a célfüggvényünk:

$$\text{maximalizálni:} \sum_{v_c \in V_v} y_i$$

A kényszerek pedig:

$$\sum_{\forall v_c \in V_v | c \in Z_s} y_i \leq C_s \qquad \forall s \in S^x, \forall v_x \in V_{nn}$$

Azaz itt sokkal több kényszerrel dolgoznánk, azonban mivel S_x lényegében egy hatványhalmaz, a heurisztikában azzal az egyszerűsítéssel élhetünk, hogy nem minden elemét használjuk fel. Ekkor például csak a legbővebb halmazokat bevéve is jó eredményeket kaphatunk, ahogy a későbbiekben is látjuk majd.

3.5. Geometriai részfeladatok

A fő feladat megoldásának leírása után ki kell térniük az elvégzendő részfeladatokra is. Alapvetően három problémával kell megküzdenünk:

- A sík lapjainak meghatározása, veszélyzónák identifikálása
- A veszélyzónák r -el való megnövelése
- Az útkeresés a veszélyzónák (akadályok) közt.

3.5.1. Particionálás

Probléma 2 *Adott C halmaz, ami síkbeli görbéket tartalmaz (esetünkben egyeneseket és köríveket). Keressük $A(C)$-t, ami a sík a görbék által meghatározott, diszjunkt lapokra való felosztása.*

A felosztások a számítógépes geometriában sok helyen jelen vannak, és rengeteg felhasználási módjuk van.

Általános esetben egyébként nem feltétlenül x-monoton görbékkel kell dolgozni, ami azt jelenti, hogy nem biztos, hogy a görbék függvénnyel leírhatóak, illetve saját magukat is metszhetik. Az elrendezés meghatározása a következő módon történik: A C-beli görbéket először szétbontjuk x-monoton görbékre, és ezeknek az új görbéknek a halmazát C'-nek nevezzük el. Az x-monoton görbéknek már nem lehet önmagukkal metszéspontja. A következő lépés a C'-beli görbék metszéspontjainak meghatározása, és a metszéspontok mentén való szétbontása új görbék C'' halmazára. Ebből a halmazból már könnyen meghatározhatunk egy síkgráfot, aminek a csúcsai a metszéspontok, és az élei pedig a C''-beli görbék.

Könnyen látható, hogy $A(C) = A(C'')$, így a kapott síkgráfból könnyen meghatározhatjuk a felosztást. A fő ötlet az, hogy minden élet két „fél-él”-re bontunk, hasonlóképpen amikor egy irányítatlan gráfból irányítottat képzünk. Minden fél-éltől balra fog elhelyezkedni a lap, amit határol, és a fél-éleken egy lap mentén körbejárva egyszerre a határoló csúcspontokat is megkapjuk, ezzel meghatározva egy síkidomot[1].

3.5.2. Hizlalás

Probléma 3 *Adott Z régió a síkon, illetve $r \in \mathbb{R}$. Keressük Z^* régiót, amit a következőképpen határozunk meg:*

$$Z^* = Z \cup \{p | p \in \mathbb{R}^2, \textit{plegfeljebb } r \textit{ távolságra van } Z\textit{-től}\}$$

Ez a részfeladat ismételten fontos számunkra mivel ha a veszélyes területeket védeni akarjuk, minimum r távolságra kell a kerülőutakat kiépíteni, az előbb már tárgyalt okok miatt.

[1]További információ a felosztásokról: https://doc.cgal.org/latest/Arrangement_on_surface_2/index.html

A feladat egy jól ismert területre mutat, igen nagy motivációja volt a kutatásnak főleg fúró és marógépek útjának kiszámítása miatt. A fúrófejek is rendelkeznek egy r paraméterrel, hiszen nem 0 átmérőjűek a bitek, és a munkadarabot megformázó gép útját úgy kell megtervezni, hogy csak a szükséges részek legyenek eltávolítva/kilyukasztva.

Adott régiók r-el „meghizlalt" változatait akarjuk kiszámolni, hogy utána az új alakzatokat használva legrövidebb utakat tudjunk számolni. Sajnos általánosan a probléma nehezen megoldható feladat (értsd: NP-nehéz) [18]. Esetünkben azonban, amikor a kérdéses régiókat csak egyenes szakaszok valamint körívek határolhatják, már egyszerűbb dolgunk van, hiszen egyenes szakaszoknak és köríveknek is egyenesek és körívek határolják a „meghizlalt" területét [26]. Erre a verzióra már létezik hatékony, polinom idő alatt futó algoritmus.

3.5.3. Legrövidebb utak akadályok közt

Probléma 4 *Adottak $Z_1^+ \dots Z_k^+$ régiók a síkon (esetünkben ezek már a felhizlalt régiók), és a, b pontok, amiket nem tartalmaz egyik Z_i^+ régió sem. Keressük azt a görbét, amely összeköti a-t b-vel, minimális költségű, valamint nem metszi el egyik Z_i^+ régiót sem.*

Szerencsénkre a feladatunk lényegi részének, azaz a legrövidebb utak keresésének és kiszámításának léteznek rendkívül gyors és könnyen kezelhető algoritmikus megoldásai. Rengeteget kutatott terület, hiszen a legrövidebb útvonalak tervezésétől a robotmunkások raktáráruházakban való mozgásáig számos területen felhasználhatóak az algoritmusok.

A legtöbb munkában [14, 19] poligonok által meghatározott kikerülendő területekkel találkozunk, azonban nekünk körívek is határolhatják a területeket. Egy frissebb munka [6] azonban ún. „spilnegon"-okat feltételez. Ekkor a poligon e élei kicserélhetőek a két csúcs közt futó e' görbékkel, ahol e és e' egy konvex területet zár közre. Megoldhatjuk így is, de egyszerűben a köríveket tudjuk n oldalú sokszögekkel is közelíteni.

Maga a probléma megoldható a polinom idő alatt a geometriai Dijkstra algoritmussal, aminek futási ideje a legrosszabb esetben $O(n \cdot \log(n))$, ahol n Z_i, $i = 1 \dots k$ poligonokban az összes csúcspont száma.

4. fejezet

Implementáció

A feladat leírása után a következő lépés persze adja magát: meg kell oldani. A fentebb már definiált feladatok többségére (így az útválasztás, hizlalás) szerencsére már vannak létező megoldásaink, azonban a matematikai modelleket (így a veszélyzóna, vágás, stb.) aligha találjuk meg bármilyen könyvtárban, így azokat nekem kellett megvalósítani. Ezekről, a tervezésnél figyelembe vett szempontokról, a kód struktúrájáról és a felépített elosztott megoldásról számol be ez a fejezet. A kód néhány példa gráffal és futási eredményekkel együtt elérhető a `hajduzs/network_extension`[1] github repository-ban.

4.1. Tervezési megfontolások

Az ötlet újnak számít, így munka közben sokszor kellett a modelleken módosítani, finomítani. Ezért a tervezésnél fő szempont volt a bővíthetőség, hogy ne okozzon túl sok fejfájást új elemekkel kiegészíteni a kódot, vagy akár teljes elemeket kicserélni a megoldásban. Ezért, és a feladat természete (mivel az alapprobléma megoldásához nem egy komplex rendszert kellett megépíteni, hanem egy algoritmust implementálni) az egyszerű kezelhetőség, átláthatóság és gyors fejlesztés miatt a kód nagy részét Pythonban írtam meg, azonban a komolyabb számítási kapacitást igénylő feladatok megoldását C++-ban implementáltam. Ennek a döntésnek indoklásáról pár bekezdéssel később részletesebben is beszámolok.

A munkám eredményeként megszületett program gerincét alapvetően egy parancssorból indítható *szkript* alkotja. Ezt a programot „csomagoltam be” az elosztott megoldásba is, amiről a fejezet végén számolok be. Ahol indokolt volt, ott természetesen alkalmaztam az OOP elveket, illetve bizonyos részfeladatokban tervezési mintákat is. A magasabb, architekturális szinten lévő tervezésnél egyértelmű volt,

[1] https://github.com/hajduzs/network_extension

hogy a „csővezetékek és szűrők”[2] mintát alkalmazzam, hiszen egymást követő, jól definiált lépések sorozatával oldjuk meg a hálózat bővítésének problémáját.

Felelősségem volt még az eredmények részletes kimentése, így statisztikák a futásról, algoritmusokról, algoritmus futásidejéről, a használt szerkezetekben tárolt elemek számáról és egyéb paramétereiről, és természetesen az új élekről is. Ezek nagy részét a Python által biztosított `logging` modul segítségével, illetve a saját osztályokkal oldottam meg, amik képesek a XML/JSON formátumokban exportálni.

Az elosztott működés biztosításának érdekében a program részeit úgy kellett megterveznem, hogy csak minimális mértékben függjenek egymástól. Mivel több gráfra és r-re is egyidőben futtathatónak kellett lennie a programnak, a modulok és osztályok közötti egyszerű interface biztosítása kiemelkedően fontos volt.

4.2. A optimalizálás főbb lépései

A kód lényegi részének lefutását – lévén, hogy a Csővezeték mintát követi – egyszerűen, pontokba szedve leírhatjuk:

1. **Gráf beolvasása:** A kiinduló állományt beolvassuk, felépítjük a topológiát a benne talált információk alapján

2. **A sík particionálása:** A topológia alapján létrehozzuk a görbéket, és elkészítjük a sík particonálását

3. **Veszélyzónák meghatározása:** A particionálás során létrejött lapokból kiválogatjuk a veszélyzónának minősülő területeket

4. **Vágások generálása:** A veszélyzónákból pontokat mintavételezve megvizsgáljuk a lehetséges katasztrófa okozta károkat, és ezek alapján a veszélyzónákból elkészítjük a vágásokat

5. **Kiinduló páros gráf létrehozása:** A vágásokból és a lehetséges javító élekből felépítjük a probléma megoldásához szükséges páros gráfot

6. **A probléma megoldása:** Egy (vagy több) algoritmust futtatunk, kiszámítjuk az új utakat

7. **Kimenet mentése:** Az eredményül kapott élhalmazt, paramétereket, statisztikákat, ábrákat, stb. mentjük.

[2]Lásd: Szoftvertechnikák jegyzet

Ez persze egy konkrét topológia és egy konkrét r esetére vonatkozik. Kötegelt/-párhuzamos végrehajtásra is lehetőség van egy támogató szkript segítségével (ez a kényelmesség kedvéért szintén a parancssorból indítható). Ez a támogató szkript magába foglalja a csővezetéket, a futási konfigurációkat automatikusan elosztja és ütemezi a rendelkezésre álló erőforrások függvényében, és gondoskodik a kimenet megfelelő mentéséről.

4.3. A kód struktúrája

Mivel –ahogy előbb említettem– a fejlesztés során gyakran előfordult, hogy új paraméterekre voltunk kíváncsiak a kimeneti oldalon, illetve az algoritmuson gyakran kellett módosítani/finomítgatni, a Python nyelv volt a legmegfelelőbb választás a program legnagyobb részének (Így a teljes vezérlés, osztályok, adatok beolvasása és kiírása, logolás, stb.) megírására.

A számításigényesebb, bonyolult részfeladatok megoldására a Python azonban önmagában sokszor nem –így esetünkben sem– nyújt megfelelő megoldást, már csak az interpretált voltja miatt sem (ekkor a futásidővel keletkeznek komoly gondok). Ezért a particionálás, hizlalás és útkeresés komplexebb problémáit C++ nyelven implementáltam.

Az így megírt kódot azonban Pythonból kevésbé elegáns külön folyamatként elindítani. Azonban szerencsénkre rendelkezésre áll több könyvtár, aminek segítségével ún. megosztott könyvtárba *(shared library)* fordított C++ kódot gyárilag biztosított elemekkel tesz elérhetővé az eredeti kód számára. Erre a feladatra a *ctypes*[3] könyvtárat választottam, mivel a legtöbb forrás ezt jelölte meg a leggyorsabbnak, és a leginkább megfelelőnek egy-egy egyszerű modul összekötésére.

Az útszámítás és particionálás problémáját a Pythonban megoldva a `pathfinder` és a `partition` modulokban találjuk, *.so* fájlokban tárolt függvényhívások mögé elrejtve. A kódot így teljes egészében modulokba/csomagokba lehetett szervezni felelősségük, illetve a modellben elfoglalt szerepük alapján. Ezeknek rövid leírása a 4.1 táblázatban található.

4.3.1. Használt Python könyvtárak

networkx

A `networkx`[4] egy gráfok létrehozására, és kezelésére született Python csomag. Egy standardizált interfészt és implementációt biztosít, aminek segítségével renge-

[3] https://docs.python.org/3/library/ctypes.html
[4] https://networkx.org/

Csomag	Felelősség
`geometry`	A geometriai műveletekért felelős osztályok, műveletek
`geometry.partition`	A sík felosztását végző c++ könyvtár, és a köré írt Python wrapper kód
`geometry.pathfinder`	A hizlalást és legrövidebb utak keresését végző c++ könyvtár, és a köré írt Python wrapper kód
`geometry.structures`	A geometriai műveletek támogató segédstruktúrák, adatszerkezetek
`measurement`	A statisztikák kimentéséhez, ábrák generálásához szükséges kódok
`netext`	A hálózatbővítési feladathoz szükséges struktúrák, eljárások nagyobb gyűjtő modulja
`netext.exceptions`	Saját kivételek és kezelésük az egyszerűbb működés és hibakeresés végett
`netext.networkmodels`	A feladat megoldásához szükséges modellelemek, így például a **`DangerZone`**, **`Cut`** osztályok, illetve a **`Pipeline`** osztály is, ami tárolja a megoldás során keletkező szerkezeteket, és függvényeket biztosít az egyszerű kezelésükre.
`netext.solver`	A feladat megoldását végző algoritmusok, és a hozzájuk szükséges segédfüggvények

4.1. táblázat. A főbb csomagok, amikbe a kód van szervezve

teg alkalmazás számára jó választás lesz. Sok bementi gráfleíró formátumot képes kezelni, illetve rengeteg gráfokon alkalmazott algoritmust (például teljes párosítás, legrövidebb utak) is natív módon támogatva rendelkezésünkre bocsájt.

A **`Topology`** osztály is a networkx modult használja alapul, ahol így a plusz információkat (amikre például a statisztikák miatt van szükségünk) elég csak egy dicitonary-ként összecsomagolnunk az alap gráffal, a példának hozott 4.1 ábrán látható módon.

```
@staticmethod
def get_topology_from_json(data):
    g = nx.Graph()
    cs = {}

    # get nodes
    for node in data['nodes']:
        g.add_node(node['id'])
        cs[node['id']] = Point(*node['coords'])

    # get edges
    for edge in data['edges']:
        g.add_edge(edge['from'], edge['to'])

    # construct topology
    t = Topology()
    t.graph = g
    t.data = {
        'name': data['name'],
        'scale_factor': data['scale_factor'] if 'scale_factor' in data else 1
    }
    t.coords = cs
    return t
```

4.1. ábra. A **`Topology`** példányokat létrehozó statikus függvény. A `networkx` alkalmazása lehetővé teszi a gráf egyszerű kezelését.

Python-MIP

A `python-mip`[5] gondoskodik a 3.4.2-ben leírt probléma megoldásáról. A csomag tartalmaz több, nyílt forráskódú lineáris programozás problémát megoldó alkalmazást tesz elérhetővé, többek közt a *c* nyelven megírt CLP *(COIN-OR Linear Programming Solver)*, CBC *(COIN-OR Branch-and-Cut solver)*, valamint a manapság legjobbnak tartott Gurobi MIP-t is. Ezek mind alacsonyabb szinten vannak implementálva, és a `python-mip` modul a `cffi`[6] interfészt alkalmazva biztosítja a használatukat, így a lehető legjobb gyorsaságot érhetjük el anélkül, hogy feladnánk a Python nyújtotta kényelmet. A 4.2 ábrán látható kódrészletben látszik, hogy miként történik az ILP feladat felírása és megoldása a csomag segítségével.

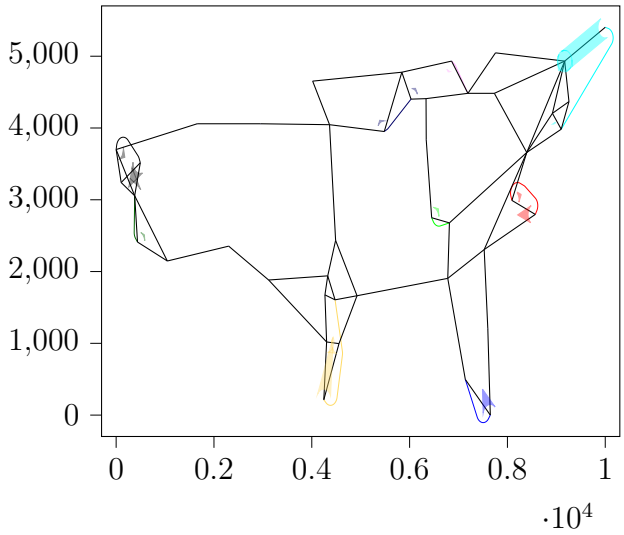
[5] https://www.python-mip.com/
[6] https://cffi.readthedocs.io/en/latest/

```python
def full_lp_calc(pl: Pipeline) -> (mip.Model, dict):
    logging.debug('Beginning FIRST LP method')

    # set up base path planner
    pp = PathPlanner()
    pp.setR(pl.r)
    for dz in pl.dzl:
        pp.addDangerZone(dz.polygon.convert_to_str())

    # set up model
    model = mip.Model()
    model.verbose = 0

    cs = dict()
    rhs = dict()

    # iterate over the bipartite graph
    number_of_lhs = len(pl.bpd.graph.nodes) - len(pl.bpd.cut_list)
    i = 0
    c_index = 0
    for n, d in pl.bpd.graph.nodes(data=True):

        if i < number_of_lhs:
            nodes = n.split("/")
            pi = pl.topology.coords[nodes[0]]
            pg = pl.topology.coords[nodes[1]]

            nc = [v for v in pl.bpd.graph.neighbors(n)]
            p_cuts = itertools.chain.from_iterable(
                itertools.combinations(nc, r) for r in range(1, len(nc) + 1)
            )

            for cuts in p_cuts:
                ids = set()
                for c in cuts:
                    ids.update(pl.bpd.return_ids_for_cut(c))
                    if c in rhs:
                        rhs[c].add(c_index)
                    else:
                        rhs[c] = set()
                        rhs[c].add(c_index)
                ids = list(ids)
                if len(ids) == 0:
                    continue

                pp_cost, pp_path = calculate_path(pi, pg, pl.r, pl.dzl, ids, pp)
                cs[c_index] = (
                    model.add_var(var_type=mip.BINARY),
                    pp_cost, pp_path, nodes, list(cuts)
                )
                c_index += 1

        else:
            model += mip.xsum([cs[k][0] for k in rhs[n]]) >= 1
        i += 1

    model.objective = mip.minimize(mip.xsum([c[0] * c[1] for _, c in cs.items()]))
    model.optimize()

    chosen = []
    for n, v in cs.items():
        if v[0].x == 1:
            chosen.append((v[3], v[2], v[1], v[4]))
    return chosen
```

4.2. ábra. A `mip` csomagot használó ILP megvalósítás

matplotlib/tikzplotlib

A `matplotlib`[7] egy széles körben elterjed Python csomag, aminek segítségével sokféle (így akár statikus, akár animált vagy interaktív) ábrát lehet készíteni. A kimenet generálásánál fontos volt, hogy az eredmény látható is legyen, és a csomag lehetővé tette már a fejlesztés közben könnyen ellenőrizni az algoritmus lépéseit és az eredményeket. Ezen A `matplotlib` csomagra épül a `tikzplotlib` is, amivel játszi könnyedséggel lehet statikus *tikz* ábrákat generálni. A dolgozatba bekerült ábrák is mind ezen csomag segítségével lettek generálva, mint például 4.3 ábra is.



4.3. ábra. Egy tikzplotlib által generált, egyszerűsített, hibakeresést segítő ábra egy amerikai gerinchálózatról

4.3.2. A használt C++ könyvtárak

Bár a nyelv népszerűsége miatt szinte minden feladatra találunk Python modulokat, esetünkben a 3.5 szekcióban leírt geometriai feladatokat már nem találjuk meg kényelmesen létrehozott, mindössze egy `pip` parancs segítségével letölthető modulokban. C++ implementációkat azonban bőven találunk, hiszen a legtöbb grafikai motor[8] is ezen a nyelven van implementálva.

CGAL

A `cgal`[9] egy szoftver projekt, amely könnyű hozzáférést biztosít a hatékony és megbízható geometriai algoritmusokhoz C ++ könyvtár formájában. A CGAL-t

[7] https://matplotlib.org/
[8] pl. Unity, Unreal Engine, CryEngine, Clausewitz engine, stb.
[9] https://www.cgal.org/

számos, geometriai számítást igénylő területen alkalmazzák, például földrajzi információs rendszerek, orvosi képalkotás, számítógépes grafika és robotika terén.

A könyvtárban megtaláljuk az Arrangements[10] nevű csomagot, aminek segítségével a sík particionálását kényelmesen el tudjuk végezni. A csomag lehetőséget biztosít tetszőleges görbék síkra illesztésére, kezeli a metszéspontok kiszámítását, majd a kialakult elrendezés bármely paraméterének könnyű lekérdezésére is lehetőséget nyújt.

SIG

A `sig`[11], egy Marcello Kallmann által fejlesztett C++ osztálycsomag, ami modern, grafikai alkalmazások fejlesztését teszi lehetővé hatékony algoritmusokkal. Kevésbé ismert, főleg akadémiai közegben alkalmazzák a kutatók a saját projektjeik megvalósítására. A csomag része egy vizuális gráf osztály, ami rengeteg műveletet tesz elérhetővé számunkra, és a „hizlalás” problémáját is automatikusan kezeli. Ennek segítségével egyszerre oldjuk meg a hizlalást az útkeresés feladatával.

A 4.4. árbán az útszámítást megvalósító kódrészlet látható. A számítás során az alakzatok hizlalása sajnos nem tökéletes, mivel lebegőpontos számokat használ az implementáció. Ezért – az ebből adódó hibákat kiküszöbölendő – rekurzívan óvatosan növelnünk kell egy tolerancia paramétert, amíg értelmes eredményt nem kapunk.

4.4. Elosztott működés

Feladat volt továbbá az is, hogy a programot elosztottan is lehessen futtatni, azaz hogy egy időben több gráfra és több paraméterre is számolni lehessen megoldásokat. E mögött a specifikáció mögött megint csak az volt a motiváció, hogy gyakran kellettek új tesztek és eredmények a modell változtatása miatt, és a magunk mindössze 1 darab virtuális gépének számítási kapacitása sajnos szűk keresztmetszetet jelentett ebben.

Természetesen egy számítógépen is lehet biztosítani több bemenetre a párhuzamos végrehajtást, például a Python `asyncio`[12] moduljával, ez azonban csak az adott gépen található magok számáig skálázódik elégségesen. Az igazán elegáns megoldás egy akár felhőben is futtatható rendszer, ami több számítógép számítási kapacitását tudja egyszerre kihasználni. Ezért az elosztott működést a `docker swarm` segítségével valósítottam meg. A következő részben erről számolok be.

[10] https://doc.cgal.org/latest/Arrangement_on_surface_2/index.html
[11] https://bitbucket.org/mkallmann/sig/wiki/Home
[12] https://docs.python.org/3/library/asyncio.html

```
void CalculatePath()
{
  GsVisGraph _vg;                 // The visual graph we use for path planning
  float dang = GS_TORAD(3);       // Circle resolution parameter of the visual graph
  _vg.build(obstacles, R - _epsilon, dang);
  GsPolygon detour;
  bool found = _vg.shortest_path(_pi, _pg, detour, &_cost);

  if(found){
    int n = detour.size();
    int size = 3 * n;
    for (int i = 0; i < n; i++){ size += strlen(detour[i].x) + strlen(detour[i].y); }

    delete[] _path;
    _path = new char[size*2];

    int j = 0;
    for (int i = 0; i < n; i++){ j += sprintf(_path + j, "%.5f %.5f  ", detour[i].x, detour[i].y); }

  }
  else{
      if(_retries < 5){
        _epsilon += R * 0.01;     //increase epsilon
        _retries++;
        CalculatePath();          //try again
      }
      else{
          std::cout << "path find not successful" << std::endl;
          delete[] _path;
        _path = new char[2] {'-', '\0'};
        _cost = -1;
      }
    }
}
```

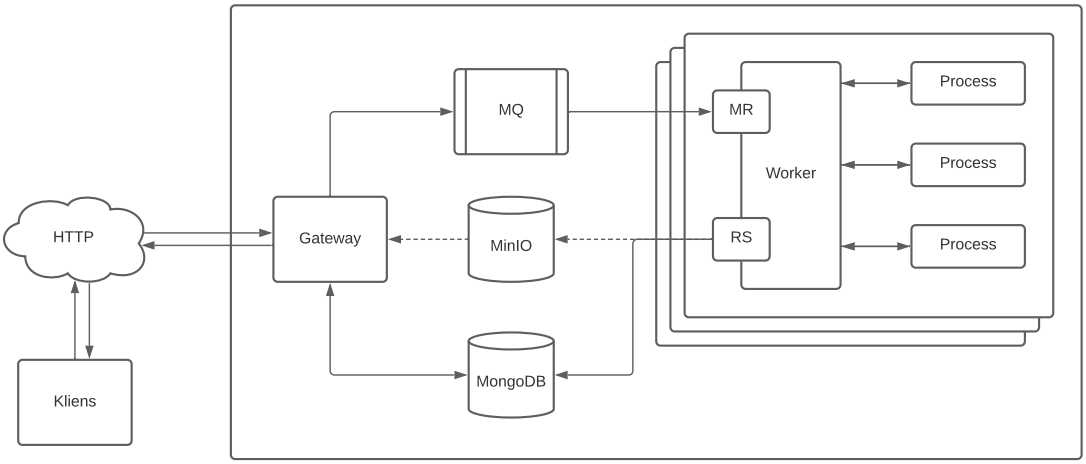
4.4. ábra. A legrövidebb utakat számoló C++ kódrészlet. A lebegőpontos számábrázolásból eredő hibák miatt egy tolerancia-paramétert (epsilon) alkalmazunk.

4.4.1. Architektúra

Az architektúra tervezésénél több szempontot is figyelembe kellett venni. Elsőként azt, hogy a virtuális gépeken/felhőben futó rendszer lehetőleg minél kisebb terhelést jelentsen a platform számára. Ez több szempontból előnyös, főként azért, mert így egy esetleges külső szolgáltatónál való elhelyezésnél nem kell annyit erőforrásért fizetni. Másodszor azt, hogy az eredmények azonnal megjelenjenek a mi oldalunkon, ne kelljen külön ssh/sftp kapcsolatot létesíteni egy szerverrel azért, hogy a futási eredményeket le tudjuk kérdezni, harmadszor pedig azt, hogy módosítás után is újra könnyen frissíthető és működésre bírható legyen.

A 4.5 ábrán látható a rendszer magas szintű „tervrajza”: Egy kliens applikáción, HTTP kommunikáció segítségével érjük el a `docker-swarm`-ban elhelyezett rendszert. Egy Gateway dolgozza fel a kéréseinket, és egy üzenetsoron keresztül juttatja el azokat a számítást végző worker-ek felé. A worker-ek a meessage queue (MQ)-ből kiveszik a feladatokat, elvégzik őket, majd az eredményeiket feltöltik az adatbázisokba. A kliens egy egyedi ID segítségével tudja fix időközönként megkér-

dezni a gateway-től, hogy megvannak-e az eredmények, és ezen keresztül is tudja lekérni őket.



4.5. ábra. Az docker-swarm alapon elosztottan is működni képes, algoritmus köré épülő rendszer architektúrája

4.4.2. Felhasznált elemek

Ebben a részben röviden bemutatom a használt technológiákat, és ismertetem a helyüket és feladataikat a rendszeren belül. Az elemek ezen módon való kiválasztását (pl. a két külön adatbázist, stb.) a későbbi könnyű bővíthetőség, és az erőforrások optimálisabb kihasználása indokolta.

Docker-swarm

A `docker-swarm`[13] segítségével lehetőség van több gép összekapcsolására, és alkalmazások egyszerű kihelyezésére a swarm módban futó docker host-okra. Alkalmazásával egyszerűen tudunk konténerekbe szervezett elemekből rendszert építeni. *A rendszert egy docker swarm-ba telepítjük, az elemek konténerekbe való szerezésével.*

Node.js

A `node.js`[14] egy aszinkron, eseményalapú `javascript`[15] környezet, ami optimalizálva van a konkurens kérések kezelésére. Könnyen fejleszthetőek ezért benne jól skálázódó, elosztott rendszerben működő applikációk. *A Gateway és Worker konténereinken használjuk őket a külvilággal, és az adatbázisokkal való kommunikációra.*

[13] https://docs.docker.com/engine/swarm/
[14] https://nodejs.org/en/
[15] https://www.javascript.com/

RabbitMQ

A `rabbitmq`[16] egy könnyű, elosztott rendszerekben széles körben alkalmazott üzenetközvetítő rendszer. Aszinkron is működik, nyílt forráskódú, így a mi rendszerünkbe is tökéletes. *A gateway és a worker-ek között közvetíti a job-okat.*

MongoDB

A `mongodb`[17] egy nyílt forráskódú dokumentumorientált adatbázis szoftver. Több mint 10 nyelven biztosít driver API-kat, amik segítségével egyszerűvé válik az adatkezelés. *A rendszerben a job-ok ID-jeit és állapotait tároljuk benne.*

MinIO

A `minio`[18] egy finomra hangolt, nagy teljesítményű, szintén nyílt forráskódú object storage rendszer. Széles körben elterjedt skálázhatósága és arányos erőforrásigénye miatt. *A mi rendszerünkben az algoritmus kimeneteként kapott eredményeket, logokat stb. tárolja.*

4.4.3. Az online optimalizáló rendszer működése

Az elemek bemutatása után a rendszer működésének leírása maradt csak hátra. A következőkben ismertetem egy átlagos futtatás lefolyását, ahol akár több gráf és paraméter is van:

A kéréseket még kliensoldalon szétválasztjuk egymástól, és egyenként továbbítjuk a gateway felé. Amikor a gateway megkapja a gráf-r-paraméter leírókat `job.json` fájlokba csomagolva, létrehoz egy-egy hozzájuk tartozó bejegyzést a MongoDB-ben, ahonnan ID-ket kap vissza minden job-hoz. Az ID-ket egyrészt visszaküldi a kliensnek, illetve kiegészíti velük a `job.json` fájlokat, és elhelyezi őket a Message Queue-ba.

A MQ-ból a worker node-ok veszik ki a job-okat, majd elkezdenek rajtuk dolgozni. Minden főbb lépés során frissítik a kérésekhez tartozó MongoDB bejegyzést is. Ha megvan az eredmény, a worker létrehoz egy bejegyzést a MinIO adatbázisban az adott ID-vel, amiben minden futás után keletkezett információ benne van. Ha valamiért meghiúsul egy futás, a MongoDB-ben „error” státuszú lesz a hozzá tartozó bejegyzés.

[16] https://www.rabbitmq.com/
[17] https://www.mongodb.com/
[18] https://min.io/

A futás során a kliens bizonyos időközönként megkérdezi (azaz *polling*-olja) a gateway-t a visszakapott job ID-vel az adott konfiguráció futásának állapotáról. A gateway mindig lekérdezi a MongoDB-ből az állapotot. Ha ez:

1. **`error:`** Értesíti a klienset a meghiúsulásról a hibaüzenettel együtt, majd kitörli a job bejegyzést az adatbázisból.
2. **`success:`** Visszaküldi az eredményeket a kliensnek, majd kitörli a job bejegyzést az adatbázisból.
3. **`akármi más:`** Visszaküldi a kliensnek az állapotot

A folyamat végén a kliensoldalon a elvégezzük a megkapott adatok kicsomagolását, kiértékelését, ábrák generálását stb. Ismételten amiatt, hogy a rendszeren belüli kapacitás legnagyobb része a konkrét feladatására megoldására kell hogy koncentrálódjon, a kevésbé számításigényes részeket (mint például statisztikák készítése egy egyszerű szövegfájl feldolgozása után, vagy ábrák generálása) pedig a kliens is el tudja végezni.

5. fejezet

Eredmények kiértékelése

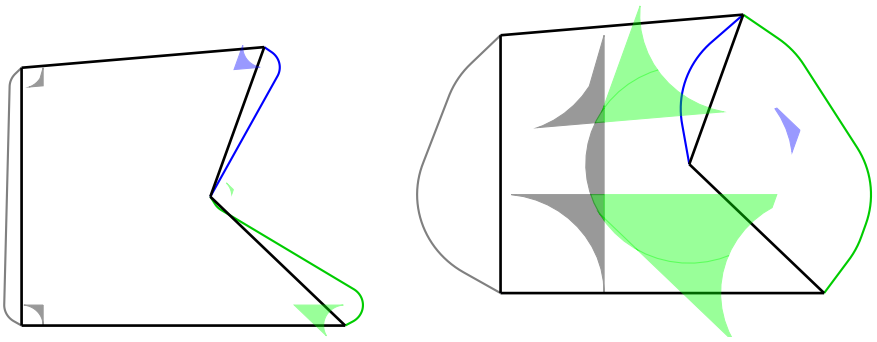
Az algoritmusokat a The Internet Topology Zoo[1] oldalról szerzett, változó méretű és bonyolultságú, valós hálózatok segítségével teszteltem. A gráfok főként országos méretű gerinchálózatokat reprezentálnak, és változatos topológiai jellemzőkkel bírnak.

A tapasztalatok alapján összességében elmondható, hogy a bővítés költsége (tehát a modellben a bővítésben használt összes kábel aggregált hossza) főleg a gráf topológiai tulajdonságaitól függ. Megfigyelhető, hogy olyan csúcsok, amiken sok irányba van kimenő élük –főleg a hálózat geometriai értelemben vett középpontjában– ritkán igényelik egy új él bekötését, azonban a hálózat szélén elhelyezkedőeknél ez sokszor válik szükségessé.

Nagy számításbeli komplexitás keletkezik akkor, ha egy nem egyenletes fokszám eloszlású gráfban sűrűbb csúcsokat tartalmazó területet ér meghibásodás. Ekkor egy katasztrófa sok olyan kisebb veszélyzónát eredményez, amik több komponensre bontják a hálózatot, Mivel a vágások száma exponenciális függvénye a komponenseknek adott veszélyzóna esetén, a páros gráfban lévő összeköttetések száma (így a problématér) jelentősen megnövekszik. Szerencsére azonban az ilyen topológiák ritkák.

Ezen kívül az r paraméter étékének változásával drasztikusan változni tudnak az eredményül kapott útvonalak is. Az 5.1 ábrán ez jól látszik, már a korábban példaként felhozott egyszerű, 5 csúcsú hálózatra is: Míg első esetben kicsi r (60km) értéknél a veszélyzónák is kicsik, és egyszerűen kezelhetőek lesznek, nagyobb (200km) kiterjedésű katasztrófa elleni védekezés estén a partícionálásnál sokkal bonyolultabb elrendezés keletkezik.

[1] http://www.topology-zoo.org



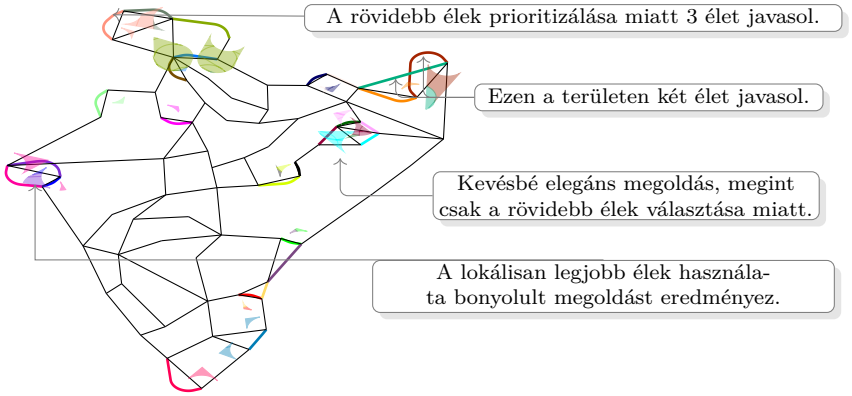
(a) $r = 60$ **(b)** $r = 200$

5.1. ábra. A példa 5-csúcsú hálózat bővítse két különböző r paraméter esetére számolva. Az adott színre színezett zónákat a megfelelő színű élek kerülik.

5.1. Egy valós indiai hálózat

Ebben a szekcióban tárgyalom a heurisztikus megoldás és az ILP közötti fő különbségeket, egy indiai gerinchálózatra kapott eredményüket összehasonlítva. Adott esetünkben a hálózatot $r = 80km$ méretű katasztrófák ellen akartuk védetté tenni. Ez nagyjából valós méret, ugyanis a közelmúltban történtek hasonló méretű természeti csapások, például az Uttarakhand-i árvizek[2], vagy trópusi ciklonok[3].

A 5.2 és 5.3 ábrákon lehet látni a heurisztika és az ILP megoldás által adott eredményeket. Összesen 87 veszélyzóna, és belőlük 49 vágás keletkezett. A magyarázatot az ábrákon található, érdekesebb területekre mutató szövegdobozok segítik. Először figyeljük meg az 5.2 ábrát!

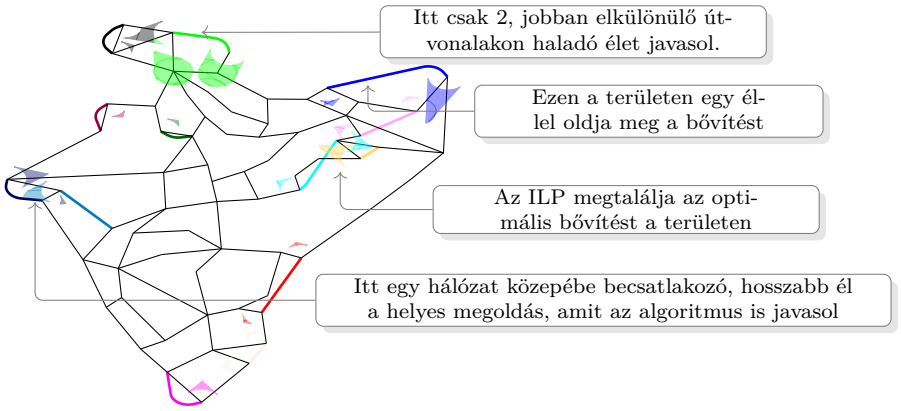


5.2. ábra. A heurisztika által adott megoldás

[2] https://reliefweb.int/sites/reliefweb.int/files/resources/IND_UttarandLocationmap_130628.pdf

[3] https://reliefweb.int/sites/reliefweb.int/files/resources/Sitrep-Cyclone-AMPHAN-3.pdf

Ezen jól látszanak a heurisztika hiányosságai: az alkalmazott heurisztika a legkisebb költség (így legrövidebb hossz) alapján válogat, így sok helyen jelennek meg indokolatlanul bonyolultnak tűnő javaslatok is. Az algoritmus egyébként összesen 27 élt ad eredményül, amiknek 5058km a hosszuk, azaz 25% plusz költséggel kell számolni.



5.3. ábra. Az LP által adott megoldás

Az LP megoldás ezzel szemben 13 új összeköttetés kiépítését javasolja, amiknek összesen $3425km$ a hosszuk, azaz összesen 17% plusz kábel lefektetése szükséges a hálózat hibabiztosításának érdekében. Az élek útvonalán is látszik, hogy a megoldás sokkal „okosabb" mint a mohó elven működő heurisztika.

Mindkét megoldásnál megfigyelhetünk azonban meglévő élek mellett szorosan haladó új összeköttetéseket is. Ennek a modellben tett kikötés –azaz csak csomópontba engedélyezett becsatlakozás– az oka.

5.2. Teljesítmény

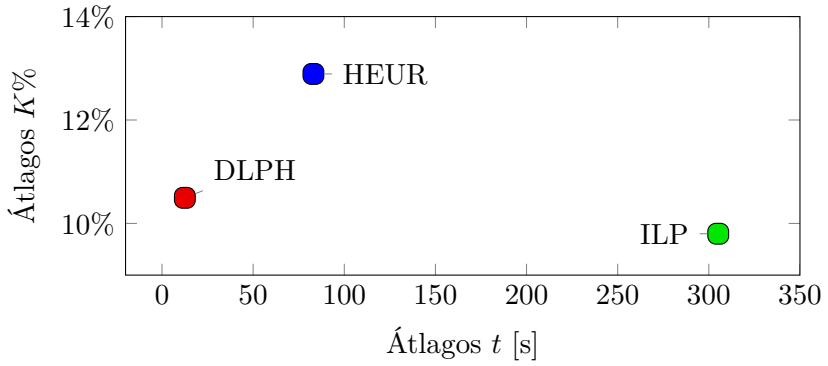
Az algoritmusok teljesítmények összehasonlítása érdekében kiválasztottam még 11 különböző topológiát is, és két különböző, reálisnak mondható r értékre (40 és 80 km-re) futtattam teszteket[4]. Az eredmények a 5.2 és 5.3-ban találhatóak, és néhány érték megtalálható grafikonon ábrázolva is alább. A használt jelöléseket a 5.1 táblázat magyarázza.

A táblázatokban megfigyelhetjük, hogy az élek összköltsége a hálózatban már meglévő élekhez képest az ILP megoldás esetén igen olcsó, maximum 17%, 9.4%-os átlaggal. A heurisztikus megoldás az optimálishoz hasonló teljesítményt nyújt, ugyanakkor a bonyolultabb felállások során alulteljesít, például a legnagyobb eltérés az optimális megoldástól (GAP) 63%, ami több mint másfélszer annyi kábelt jelent!

[4]Konfiguráció: Inter Core i7-7700 HQ, 8GB ram, Ubuntu 20.04 LTS

Érdemes megjegyezni, hogy a GAP a DLPH esetén a táblázatban feltüntetett esetekben mindig 0 volt (tehát optimális megoldást hozott), így helytakarékosság miatt nem tüntettem fel.

Észrevehetjük azt is, hogy a nagyobb hálózatokra több veszélyzóna és így vágás keletkezik, ezzel nagyobb komplexitást eredményezve Érdekes továbbá, hogy a teljes költség nem függ a zónák és vágások számától, de még az új élek számától (amik a vizsgált gráfokban 2 és 14 közé jöttek ki) sem.

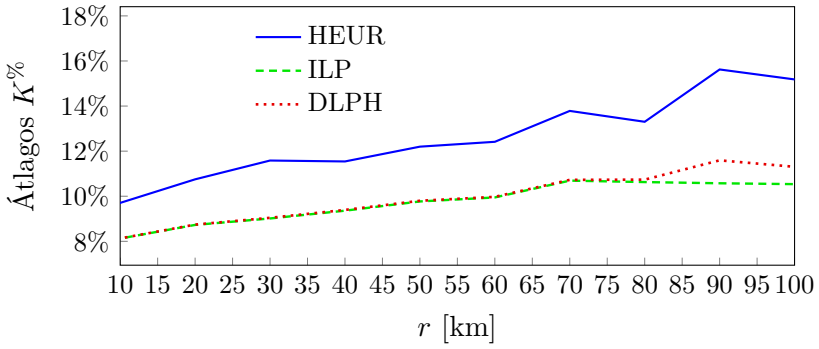


5.4. ábra. A futási idő és az összköltség közötti átlagos tradeoff

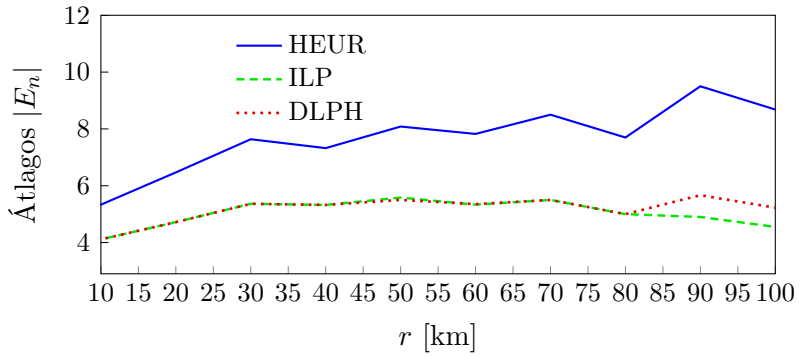
5.4 ábra mutatja a trade-off-ot a különböző módszerek közt. Ez (és a többi hasonló ábra is) átlagolt eredménye körülbelül 120 különböző konfigurációra történt futásnak, ahol 5-től 100 kilométerig teszteltem a hálózatokra az algoritmusokat. Az ILP megoldás a nagy problématér miatt viszonylag nagy futási idővel rendelkezik. A DLPH jobbnak bizonyult a kiinduló heurisztikánál is, ami nem meglepő, hiszen kevesebbszer kell benne elvégezni a futási idő szempontjából kritikus útköltségeket.

Az 5.5 ábrán megfigyelhető, hogy –nagy meglepetésünkre– az új élek összköltsége növekedik az egyre nagyobb kiterjedésű kiterjedésű katasztrófákkal is. A heurisztikánál megfigyelhetünk töréseket a vonalban, amiket az algoritmus sajátosságai (mindig a lokális optimum kiválasztása) okoznak bizonyos topológiákra. További érdekesség, hogy az ILP és a DLPH görbéje egész sokáig ($r = 80km$ környékéig) együtt haladnak. Ebből látszik, hogy a DLPH is képest optimális eredményeket hozni, azonban bonyolultabb esetekre, ahol sok egymást metsző területtel partícionálunk nem mindig találja meg a legjobb megoldást.

Az ILP megoldás egyébként 5.5a-en láthatóan jól skálázódik r növekedésével. A a 5.2 és 5.3 táblázatok ezt alátámasztják: r 40 km-el való növelésénél az összköltség sokszor alig emelkedik 1%-ot. Ez arra enged következtetni, hogy a hálózatoknak van egy „fix” bővítési költsége, amiket topológiai jellemzők határoznak meg. Ezt elképzelhetjük úgy is, mintha $r = 0$ esetére oldanánk meg a feladatot, azaz csak



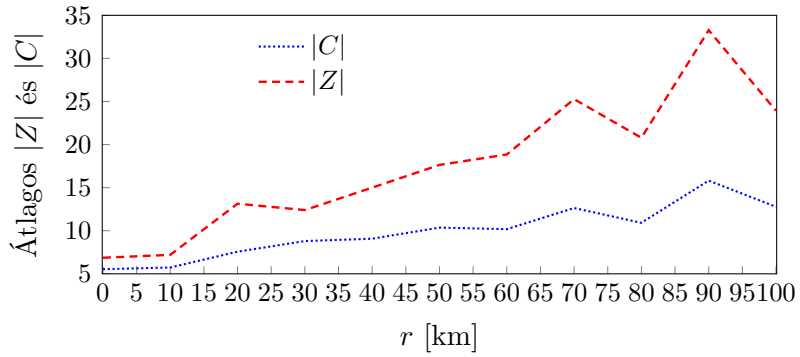
(a) A megoldások átlagos költsége



(b) A megoldásokban hasznél élek átlagos száma

5.5. ábra. További teljesítménymutatók r függvényében

kétszeresen pont- és él-összefüggővé akarnánk tenni a hálózatot. A 5.5b ábrán látszik, hogy a szükséges élek száma csak kismértékben nő r-el.



5.6. ábra. A zónák és vágások számának növekedése r függvényében

Érdekes megemlíteni, hogy a tesztelés során dupla éles megoldásokat jóformán csak a kiinduló heurisztika javasolt (bár itt sem számottevő mennyiséget, 15-t az összesen 1169 új élből). A DLPH mindössze 1 darabot a maga összes 616 éléből, míg az ILP minden veszélyzónát egy szimpla összeköttetéssel védett. Bár matematikailag

érdekes a nem egyszerű élek esete, a gyakorlat azt mutatja, hogy a való életből szerzett példákon sokszor elhanyagolhatóak.

5.3. Az eredmények összegzése

A tesztelés rávilágított arra, hogy az eredmények nagyban függenek a bementi gráf bizonyos topológiai sajátosságaitól. A módszerek azonban jól skálázódnak r szerint, azonban bonyolultabb veszélyzóna-konfigurációk esetén jelentősen megnő a komplexitás. A DLPH megoldás jelenti a kulcsot ennek kiküszöbölésére.

A továbbiak pár pontban összefoglalva:

- Az olyan csomópontok, amik különféle irányokba kimenő kapocslatokkal rendelkeznek (tehát például a hálózat közepén elhelyezkedő csúcsok) kevés esetben igényelnek plusz bekötő éleket. A csak 1 vagy 2 kapcsolattal rendelkező csomópontok –legtöbbször a hálózat szélén– gyakran védelmet igényelnek, hiszen egy katasztrófa könnyen izolálja őket.

- Gyakran keletkezik nagy problématér és így számításbeli komplexitás, ha magas fokszámú csomópontokban gazdag régiót talál el egy katasztrófa a vágások exponenciális száma miatt.

- Ha az offline komplexitás nem okoz gondot, az ILP módszer nyújtja a legjobb választást a minden esetben optimális eredménnyel, bár a futási idő az exponenciális problématér miatt jelentősen megnőhet. A DLPH nyújtja a legjobb trade-off-ot a futási idő és az eredmények optimalitása közt, ha valamilyen oknál fogva online akarjuk alkalmazni az algoritmust.

Jelölés	Jelentése
$\|V\|$	A gráf csúcsainak száma
$\|E\|$	A gráf éleinek száma
$\|Z\|$	A létrejövő veszélyzónák száma
$\|C\|$	A generált vágások száma
$\|\text{Eq}\|$	Az LP problémában felírt kényszerek száma
$K^{\%}$	A bővítés költsége a hálózatban már meglévő élekhez össz. hosszához képest
$\|E_n\|$	A megoldásban lévő élek száma
GAP	Eltérés az optimális megoldás által meghatározott alsó korláttól (csak a heurisztika esetén)
t	Futási idő

5.1. táblázat. Magyarázat az ábrákon és táblázatokban található értékekhez.

Topológia				$r = 40$km										
						ILP				DLPH	HEUR			
Név	$\|V\|$	$\|E\|$	teljes élhossz (km)	$\|Z\|$	$\|C\|$	$\|\text{Eq}\|$	$K^{\%}$	$\|E_n\|$	t (s)	t (s)	GAP	$K^{\%}$	$\|E_n\|$	t (s)
Gridnet (USA) [17]	9	20	42635	12	6	108	10	2	4.90	3.71	4.85	10	4	1.02
GlobalCenter (USA) [17]	9	36	57165	5	3	27	4.07	3	4.61	4.51	0	4.07	3	0.19
Pan-European [34]	16	22	6321	4	4	66	16	4	3.97	3.63	0	16	4	0.19
Electric Lightwave [17]	20	30	23698	10	6	148	5.43	4	4.43	3.82	22	6.66	6	1.38
COST (EU) [34]	22	45	24475	5	5	115	12	4	4.02	4.17	18	15	5	0.55
US nationwide [34]	24	42	27218	6	6	153	6.17	4	3.69	3.59	29	8.02	6	0.70
France [25]	25	45	11487	58	17	2251	9.61	7	558	40	39	12	11	115
Darkstrand (USA) [17]	28	31	14411	5	5	145	8.02	3	3.65	3.50	49	11	5	0.58
Nobel (EU) [25]	28	41	16864	5	5	145	11	4	4.28	4.44	0	11	4	0.60
Pioro (random) [25]	40	89	24202	9	8	340	4.92	6	4.66	4.00	17	5.79	7	1.63
Telia (USA) [37]	44	65	30295	26	16	1078	13	11	27	10	31	18	15	77
Tata (India) [36]	70	97	20047	40	27	6468	13	14	226	31	37	18	21	445

5.2. táblázat. Futási eredmények $r = 40km$ esetén

Topológia				$r = 40$km										
						ILP				DLPH	HEUR			
Név	$\|V\|$	$\|E\|$	teljes élhossz (km)	$\|Z\|$	$\|C\|$	$\|\text{Eq}\|$	$K^{\%}$	$\|E_n\|$	t (s)	t (s)	GAP	$K^{\%}$	$\|E_n\|$	t (s)
Gridnet (USA) [17]	9	20	42635	10	5	67	10	2	3.65	3.20	35	14	4	0.82
GlobalCenter (USA) [17]	9	36	57165	7	4	38	5.08	3	4.48	4.21	1.47	5.16	4	0.29
Pan-European [34]	16	22	6321	6	5	100	22	4	3.87	3.81	6.55	24	5	0.59
Electric Lightwave [17]	20	30	23698	11	8	304	5.86	5	6.53	4.47	34	7.87	7	3.02
COST (EU) [34]	22	45	24475	5	5	115	13	4	3.71	3.79	17	15	5	0.53
US nationwide [34]	24	42	27218	6	6	153	6.51	4	4.06	3.74	27	8.32	6	0.66
France [25]	25	45	11487	116	35	59332	14	8	55128	104	63	23	18	764
Darkstrand (USA) [17]	28	31	14411	6	6	205	9.23	3	3.66	3.52	38	12	5	0.65
Nobel (EU) [25]	28	41	16864	4	4	114	12	4	3.74	4.53	0	12	4	0.53
Pioro (random) [25]	40	89	24202	16	11	655	6.20	6	10	6.24	14	7.08	7	5.04
Telia (USA) [37]	44	65	30295	34	17	1783	14	9	55	13	41	20	15	108
Tata (India) [36]	70	97	20047	87	49	56149	17	13	5290	71	41	25	27	1096

5.3. táblázat. Futási eredmények $r = 80km$ esetén

6. fejezet

Összegzés

Ebben a dolgozatban hálózatok regionális hibák ellen való védettségével foglalkoztam. A kapcsolódó szakirodalom áttanulmányozása volt az első lépés, ami során megismerkedhettem a probléma modellezési lehetőségeivel, már létező, hasonló megoldásokkal és néhány, témát érintő problémával is.

A rendelkezésre álló információkat alapul véve ezután a feladat könnyen formalizálhatóvá vált. A matematika nyelvén a hálózatokat irányítatlan gráfokként írtuk le, a katasztrófákra pedig a kör alakú hibamodellt alkalmaztuk. A modellezést követően a feladatot –lépéseire bontása után– több algoritmust használva oldottuk meg, majd a további részfeladatok megoldását a számítógépes geometria eszköztárából kölcsönzött módszerekkel biztosítottuk.

A modelleket, algoritmusokat elsődlegesen Python nyelven implementáltam, a nagyobb számítási kapacitást igénylő részeket leszámítva, amik C++ könyvtárakba kerültek. Egy elosztott rendszert is terveztem illetve felépítettem, a megoldó program minél gyorsabb és gördülékenyebb futtatása érdekében.

Az algoritmusok által kapott eredmények elemezésekor a következőket tapasztaltam a használt modellekről, illetve algoritmusok sajátosságairól. A heurisztikákkal optimális vagy közel optimális megoldásokat tudtam elérni. A mohó módszerek legnagyobb hibája az, hogy csak az adott lépésben, és nem globálisan optimális éleket válogatják be a megoldásba. Ennél sokkal jobb, ha „előre gondolkozva” egyből a teljes problémateret vizsgáljuk. A modell sajátosságaiból kifolyólag (csak a pont-pont összeköttetések engedélyezése) azonban bizonyos esetekben az optimális megoldásban is feltűnhetnek kissé kontra-intuitív megoldások.

Az ILP algoritmus – a többihez hasonlóan – jól skálázódik, bizonyos speciális topológiák eseteit leszámítva (csomópontban sűrű területek, és ott katasztrófa hatására létrejövő sok komponens). A nehezebb esetekben is elérhetünk azonban optimálishoz igen közeli megoldásokat a DLPH heurisztikával.

Összességében elmondhatom, hogy ennek a komplex feladatnak a megoldása során rengeteget tanultam, és nagyon örülök, hogy viszonylag korán sikerült olyan témával találkoznom, amely amellett, hogy felkeltette az érdeklődésemet, még izgalmas kihívásokkal is szembeállított.

7. fejezet

További fejlesztési lehetőségek

A jelenlegi megoldásnak a legnagyobb hátránya az, hogy az összeköttetéseket csak és kizárólag meglévő csomópontok között engedi meg lefektetni. Ez sok esetben (például olyan gráfoknál, amikben a csomópontok kifejezetten távol helyezkednek el egymástól) sajnos életszerűtlen. Ezért egy nagy, és a későbbiek során mindenképpen kiaknázandó fejlesztési lehetőség, hogy akár megmaradt élekbe vagy él-részekbe, tetszőleges helyeken is engedélyezni lehessen a becsatlakozást. Ez a valóságban is egyszerűbben megvalósítható, hiszen például szárazföldi kábelek esetén nem kell annyit ásni, és a becsatlakozás egyszerű jelismétlőkkel könnyen kivitelezhető, az eredmény pedig lecsökkenő összköltség.

A gyakorlati alkalmazhatósághoz egy másik ponton is szükséges finomítani a modell: Gondoljunk csak bele, hogy milyen költségekkel jár egy város közepén földbe ásni egy kábelt, és milyenekkel egy lakatlan területen fekvő mezőn átvezetni egyet. Azzal, hogy az útköltség-számításban egyenlő költséget feltételezünk, ez elveszik a modellünkben. Érdekes irányt jelenthet, hogy ennek az információnak a bevitelével finomítsuk a modellt: Egy saját költségfüggvénnyel, amit a sík bizonyos területeire definiálunk. Ezzel szimulálni lehet városokat, hegyeket, stb. ahol lényegesen költségesebb az új útvonalak kiépítése.

Sokszor a katasztrófa típusától is függ, hogy mik és mekkorák pontosan az érintett területek. Például egy árvíznek útját állhatja egy hegygerinc, vagy egy rakétatámadást a megfelelő védelmi rendszerekkel meg lehet előzni. Érdemes lehet a különböző terep-jellemzőket is beépíteni a modellbe, illetve számszerűsíteni, hogy adott területet mekkora valószínűséggel sújt katasztrófa, hogy egy teljesen új optimalizálási problémát kapjunk.

Végül a meglévő matematikai modellben maradva, új megoldási módszerek és heurisztikák kigondolása, implementálása és tesztelése is lehetséges továbbfejlesztési irány. Mindezt az elosztott megoldás finomhangolása és esetleg kibővítése mellett.

Köszönetnyilvánítás

Szeretném hálámat kifejezni konzulenseimnek, dr. Tapolcai Jánosnak, és dr. Pašić Alijának, akik ismeretségünk kezdete óta mentoráltak, illetve mindig nagy türelemmel és megértéssel viszonyultak hozzám és segítettek kezelni a problémáimat. Rengeteg köszönettel tartozom nekik, nélkülük nehezen tudtam volna elképzelni az elmúlt 2 évem az egyetemen. Ezen kívül szeretném megköszönni szüleimnek, nagyszüleimnek és nagynénémnek a sok finom házi kosztot, amik segítettek bennem tartani a lelket a munka során.

Irodalomjegyzék

[1] Anuj Agrawal – Vimal Bhatia – Shashi Prakash: Network and risk modeling for disaster survivability analysis of backbone optical communication networks. *Journal of Lightwave Technology*, 37. évf. (2019) 10. sz., 2352–2362. p.

[2] M. Waqar Ashraf – Sevia M. Idrus – Farabi Iqbal – Rizwan Aslam Butt: On spatially disjoint lightpaths in optical networks. *Photonic Network Communications*, 36. évf. (2018. Aug) 1. sz., 11–25. p. ISSN 1572-8188.

[3] Muhammad Ashraf – Sevia Idrus – Farabi Iqbal – Rizwan Butt – Muhammad Faheem: Disaster-resilient optical network survivability: a comprehensive survey. In *Photonics* (konferenciaanyag), 4. köt. 2018, Multidisciplinary Digital Publishing Institute, 35. p.

[4] Yoshinari Awaji – Hideaki Furukawa – Sugang Xu – Masaki Shiraiwa – Naoya Wada – Takehiro Tsuritani: Resilient optical network technologies for catastrophic disasters. *Journal of Optical Communications and Networking*, 9. évf. (2017) 6. sz., A280–A289. p.

[5] Cong Cao – Moshe Zukerman – Weiwei Wu – Jonathan H Manton – Bill Moran: Survivable topology design of submarine networks. *Journal of Lightwave Technology*, 31. évf. (2013) 5. sz., 715–730. p.

[6] Danny Z Chen – Haitao Wang: Computing shortest paths among curved obstacles in the plane. *ACM Transactions on Algorithms (TALG)*, 11. évf. (2015) 4. sz., 26. p.

[7] Yufei Cheng – Deep Medhi – James P. G. Sterbenz: Geodiverse routing with path delay and skew requirement under area-based challenges. *Networks*, 66. évf. (2015) 4. sz., 335–346. p. ISSN 1097-0037.

[8] A. de Sousa – D. Santos – P. Monteiro: Determination of the minimum cost pair of D-geodiverse paths. In *The 2017 International Conference on Design of Reliable Communication Networks (DRCN 2017)* (konferenciaanyag). Munich, 2017. 8-10 March.

[9] Ferhat Dikbiyik – Massimo Tornatore – Biswanath Mukherjee: Minimizing the risk from disaster failures in optical backbone networks. *J. Lightwave Technol.*, 32. évf. (2014. Sep) 18. sz., 3175–3183. p.
URL http://jlt.osa.org/abstract.cfm?URI=jlt-32-18-3175.

[10] Ferhat Dikbiyik – Massimo Tornatore – Biswanath Mukherjee: Minimizing the risk from disaster failures in optical backbone networks. *Journal of Lightwave Technology*, 32. évf. (2014) 18. sz., 3175–3183. p.

[11] B. Elshqeirat – S. Soh – S. Rai – M. Lazarescu: Topology design with minimal cost subject to network reliability constraint. *IEEE Transactions on Reliability*, 64. évf. (2015. March) 1. sz., 118–131. p. ISSN 0018-9529.

[12] Teresa Gomes – János Tapolcai – Christian Esposito – David Hutchison – Fernando Kuipers – Jacek Rak – Amaro De Sousa – Athanasios Iossifides – Rui Travanca – Joao André és mások: A survey of strategies for communication networks to protect against large-scale natural disasters. In *8th International Workshop on Resilient Networks Design and Modeling (RNDM 2016)* (konferenciaanyag). 2016, IEEE, 11–22. p.

[13] M Farhan Habib – Massimo Tornatore – Ferhat Dikbiyik – Biswanath Mukherjee: Disaster survivability in optical communication networks. *Computer Communications*, 36. évf. (2013) 6. sz., 630–644. p.

[14] John Hershberger – Subhash Suri: An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28. évf. (1999) 6. sz., 2215–2256. p.

[15] Farabi Iqbal – Fernando Kuipers: On centrality-related disaster vulnerability of network regions. In *2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)* (konferenciaanyag). 2017, IEEE, 1–6. p.

[16] Shrinivasa Kini – Srinivasan Ramasubramanian – Amund Kvalbein – Audun Fosselie Hansen: Fast recovery from dual-link or single-node failures in ip networks using tunneling. *IEEE/ACM Trans. Netw.*, 18. évf. (2010. december) 6. sz., 1988–1999. p. ISSN 1063-6692. URL http://dx.doi.org/10.1109/TNET.2010.2055887. 12 p.

[17] S. Knight – H.X. Nguyen – N. Falkner – R. Bowden – M. Roughan: The internet topology zoo. *Selected Areas in Communications, IEEE Journal on*, 29. évf. (2011. october) 9. sz., 1765 –1775. p. ISSN 0733-8716.

[18] In-Kwon Lee – Myung-Soo Kim – Gershon Elber: Planar curve offset based on circle approximation. *Computer-Aided Design*, 28. évf. (1996) 8. sz., 617–630. p.

[19] Joseph SB Mitchell és mások: Geometric shortest paths and network optimization. *Handbook of computational geometry*, 334. évf. (2000), 633–702. p.

[20] Dawson Ladislaus Msongaleli – Ferhat Dikbiyik – Moshe Zukerman – Biswanath Mukherjee: Disaster-aware submarine fiber-optic cable deployment for mesh networks. *Journal of Lightwave Technology*, 34. évf. (2016) 18. sz., 4293–4303. p.

[21] Biswanath Mukherjee – M Habib – Ferhat Dikbiyik: Network adaptability from disaster disruptions and cascading failures. *Communications Magazine, IEEE*, 52. évf. (2014) 5. sz., 230–238. p.

[22] Melissa Cristina Márquez, 2020.
URL https://www.forbes.com/sites/melissacristinamarquez/2020/07/20/our-underwater-world-is-full-of-cables-that-are-sometimes-attacked-by-sharks.

[23] Sebastian Neumayer – Alon Efrat – Eytan Modiano: Geographic max-flow and min-cut under a circular disk failure model. *Computer Networks*, 77. évf. (2015), 117–127. p.

[24] Sebastian Neumayer – Gil Zussman – Reuven Cohen – Eytan Modiano: Assessing the vulnerability of the fiber infrastructure to disasters. *Networking, IEEE/ACM Transactions on*, 19. évf. (2011) 6. sz., 1610–1623. p.

[25] S. Orlowski – M. Pióro – A. Tomaszewski – R. Wessäly: SNDlib 1.0–Survivable Network Design Library. In *Proc. of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium* (konferenciaanyag). 2007. April. URL http://www.zib.de/orlowski/

Paper / OrlowskiPioroTomaszewskiWessaely2007-SNDlib-INOC . pdf . gz. http://sndlib.zib.de, extended version accepted in Networks, 2009.

[26] H Persson: Nc machining of arbitrarily shaped pockets. *Computer-Aided Design*, 10. évf. (1978) 3. sz., 169–174. p.

[27] Y. Prieto – J. E. Pezoa – N. Boettcher – S. K. Sobarzo: Increasing network reliability to correlated failures through optimal multiculture design. In *CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies* (konferenciaanyag). 2017. Oct, 1–6. p.

[28] Jacek Rak – David Hutchison – Eusebi Calle – Teresa Gomes – Matthias Gunkel – Paul Smith – Janos Tapolcai – Sofie Verbrugge – Lena Wosinska: Recodis: Resilient communication services protecting end-user applications from disaster-based failures. In *18th International Conference on Transparent Optical Networks (ICTON 2016)* (konferenciaanyag). 2016, IEEE, 1–4. p.

[29] F. Robledo – P. Romero – M. Saravia: On the interplay between topological network design and diameter constrained reliability. In *2016 12th International Conference on the Design of Reliable Communication Networks (DRCN)* (konferenciaanyag). 2016. March, 106–108. p.

[30] Hiroshi Saito: Spatial design of physical network robust against earthquakes. *Journal of Lightwave Technology*, 33. évf. (2015) 2. sz., 443–458. p.

[31] Arunabha Sen – Sudheendra Murthy – Sujogya Banerjee: Region-based connectivity-a new paradigm for design of fault-tolerant networks. In *2009 International Conference on High Performance Switching and Routing* (konferenciaanyag). 2009, IEEE, 1–7. p.

[32] Arun Somani: *Survivability and traffic grooming in WDM optical networks*. 2006, Cambridge University Press.

[33] Rodrigo Souza Couto – Stefano Secci – Miguel Mitre Campista – Kosmalski Costa – Luis Maciel: Network design requirements for disaster resilience in iaas clouds. *Communications Magazine, IEEE*, 52. évf. (2014) 10. sz., 52–58. p.

[34] J. Tapolcai – L. Rónyai – B. Vass – L. Gyimóthi: Fast enumeration of regional link failures caused by disasters with limited size. *IEEE/ACM Transactions on Networking*, 2020., 1–14. p.

[35] János Tapolcai – Zsombor László Hajdú – Alija Pašić – Pin-Han Ho – Lajos Rónyai: On network topology augmentation for global connectivity under regional failures. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications* (konferenciaanyag). Vancouver, Canada, 2021. május.

[36] Tata communications, India & neighbouring countries network. URL https://www.tatacommunications.com/map/.

[37] Telia carrier map USA. URL https://www.teliacarrier.com/our-network.html.

[38] S. Verbrugge – D. Colle – P. Demeester – R. Huelsermann – M. Jaeger: General availability model for multilayer transport networks. In *Proc. DRCN* (konferenciaanyag). Lacco Ameno, Italy, 2005. 16-19.

[39] Weiwei Wu – Bill Moran – Jonathan H Manton – Moshe Zukerman: Topology design of undersea cables considering survivability under major disasters. In *2009 International Conference on Advanced Information Networking and Applications Workshops* (konferenciaanyag). 2009, IEEE, 1154–1159. p.

[40] An Xie – Xiaoliang Wang – Wei Wang – Sanglu Lu: Designing a disaster-resilient network with software defined networking. In *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)* (konferenciaanyag). 2014, IEEE, 135–140. p.

[41] Song Yang – Stojan Trajanovski – Fernando Kuipers: Availability-based path selection and network vulnerability assessment. *Wiley Networks*, 66. évf. (2015) 4. sz., 306–319. p.

[42] Jianan Zhang – Eytan Modiano – David Hay: Enhancing network robustness via shielding. *IEEE/ACM Transactions on Networking (TON)*, 25. évf. (2017) 4. sz., 2209–2222. p.

[43] Dongyun Zhou – Subramaniam Subramaniam: Survivability in optical networks. *IEEE Network*, 14. évf. (2000) 6. sz., 16–23. p.