ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

BÁNKI DONÁT GÉPÉSZ ÉS
BIZTONSÁGTECHNIKAI MÉRNÖKI KAR
Mechatronikai és Járműtechnikai Intézet

FUZZY HIBAFA ELJÁRÁS FEJLESZTÉSE
FUZZY FAULT TREE METHOD DEVELOPMENT

OE-BGK
2021.12.12

Hallgató neve: Végh Benjamin Bence
Hallgató törzskönyvi száma: T005735/FI12904/B

- ALÁÍRÁST TARTALMAZÓ LAP TÖRÖLVE -

- ALÁÍRÁST TARTALMAZÓ LAP TÖRÖLVE -

Table of Contents

1 BEVEZETÉS....6
2 HIBAFA ELEMZÉS....7
2.1 Hagyományos hibafa-elemzés....7
2.2 Bizonytalanság a hibafa-elmezés során....8
3 HALMAZOK ÉS LOGIKA....9
3.1 Hagyományos Halmazok és Logika....9
3.1.1 Halmazok....9
3.1.2 Logika....10
3.2 Fuzzy Számok....12
3.2.1 Fuzzy Halmazok....12
3.2.2 Fuzzy Tagsági Függvények....12
3.2.3 Alfa-vágás....14
3.2.4 Fuzzy Logika....16
4 FUZZY HIBAFA ANALÍZIS....17
4.1 Fuzzy Halmazelmélet a Hibafa analízisben....17
4.2 Tagsági függvények és Nyelvi Változók....17
4.3 Algebrai egyenletek fuzzy valószínűségekhez....18
4.4 Fuzzy fontossági és bizonytalansági mérőszámok....20
5 TREEZZY2 Szoftver....22
6 FUZZYFTAPY SOFTVER FEJLESZTÉS....24
6.1 Python nyelvi áttekintés....24
6.1.1 Változók....24
6.1.2 Funkciók....25
6.1.3 Osztályok....26
6.1.4 Kommentek, Doc-Stringek és F-Stringek....27
6.1.5 Elágazások....28
6.1.6 Ciklusok....29
6.1.7 Modulok Importálása....30
6.2 Szoftvertervezési szempontok....31
6.3 Mentési Fáljszerkezet....32
6.4 Szoftverkód és Dokumentáció....35
6.4.1 Szoftverfüggőségek....35
6.4.2 Modulszintű változók....35
6.4.3 Hiba Objektum Definíciók....36
6.4.4 ProbabilityTools osztály....37
6.4.5 AlphaLevelInterval osztály....38
6.4.6 TrapezoidalFuzzyNumber osztály....39
Constructor methods and Flags....39
Defuzzifikációs Stratégiák....40
Logikai műveletek....41
Alfa-vágás metódus....42
6.4.7 FuzzyFaultTree osztály....42
Inicializáció, Betoltés és Mentés....42
Verzió Ellenőrzés....44

Belső Számítások....45
Legfelső esemény Standard Közelítése....46
Legfelső esemény alfa-vágás számítása....47
Minimum Cut Sets Calculation....48
Fuzzy Fontossági Érték....50
Fuzzy Bizonytalansági Fontosság Mértéke....52
Hibafa manipulációs API....53
6.5 Konzolos Interfész....54
7 TESZTELÉS ÉS ÖSSZEHASONLÍTÁS....57
7.1 A tesztelés és az összehasonlítások módszertana....57
7.2 Teszt 1....58
7.2.1 Bemeneti Adatok....58
7.2.2 Eredmény....59
7.2.3 Összehasonlítás....61
7.3 Teszt 2....62
7.3.1 Bemeneti Adatok....62
7.3.2 Eredmények....63
7.3.3 Összehasonlítás....65
7.4 Teszt 3....66
7.4.1 Bemeneti Adatok....66
7.4.2 Eredmények....67
7.4.3 Összehasonlítás....69
7.5 Teszt 4....70
7.5.1 Bemeneti Adatok....70
7.5.2 Eredmények....71
7.5.3 Összehasonlítás....74
7.6 Lefutási idő....75
8 AZ FFTA LEHETSÉGES ALKALMAZÁSAI ÖNVEZETŐ AUTÓk TÉMAKÖRÉBEN....76
9 KONKLÚZIÓ....77
Irodalom Jegyzék....78

1 BEVEZETÉS

Az önvezető járművek elterjedésével egyre fontosabbá válik az alapos kockázatelemzés nemcsak az egyes hardver-, hanem a szoftverelemekre is, a neurális hálózati modellekre különösen a különböző forgalmi viszonyok és az egyes kereszteződések közötti általános teljesítményük felmérésére.

A hibafa-elemzést gyakran más módszerekkel együtt használják a komponensszintű megbízhatóság értékelésére és azon kulcsfontosságú összetevők azonosítására, amelyek a leginkább hozzájárulnak a teljes rendszerhibához. Feltételezi azonban, hogy minden komponensnek jól meghatározott meghibásodási valószínűsége van, ami gyakran nem így van, és ehelyett durva becsléseket kell használni, ami bizonytalanságot visz be a rendszermodellbe.

A Fuzzy Hibafa Analízist később azért fejlesztették ki, hogy számszerűsítsék ezeket a bizonytalanságokat, a fuzzy logikát a hagyományos hibafa-elemzésre alkalmazva, amely ígéretes eredményeket hozott a különböző tanulmányi területeken. Mindazonáltal nagyon kevés publikált kutatást tartalmaz, ha egyáltalán alkalmazzák az önvezető járművek területén.

Sajnos a Fuzzy Hibafa Analízis területén hiányzik a kutatás alapjául szolgálható szabadon elérhető szoftver. Az egyetlen ilyen példa, amelyhez az intézmény hozzáfér, a TREEZZY2 szoftver, amelyhez nem tartozik sem dokumentáció, sem forráskód, és beágyazott alkalmazásokban sem használható könnyen, így alkalmatlan az önvezető járművek területének nagy részén való alkalmazásra.

Ebből kifolyólag ebben a dolgozatban egy új számítógépes szoftver fejlesztése és dokumentálása vált nagy fontosságúva, amely a további fejlesztések és kutatások alapjául szolgálhat nyilt forráskódú licencel. Ezt a fejlesztést a népszerű Python programozási nyelven valósítja meg ez a szakdolgozat, beágyazott alkalmazásokba lehetővé téve.

Ez a dolgozat rövid áttekintést tartalmaz a hagyományos és a fuzzy hibafa elemzés területeiről, hogy bemutassa a felhasznált elméletet és logikát, valamint bemutatja és áttekinti a Python programozási nyelvet és szintaxisát, hogy segítsen megérteni a forráskódot, hogy akik használják, bővítik és javítják egy közös tudásalap birtokában jussanak.

2 HIBAFA ELEMZÉS

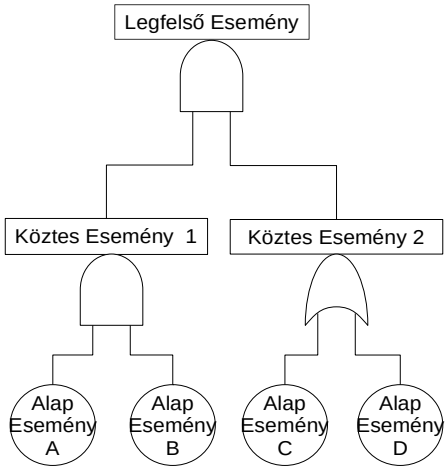
2.1 Hagyományos hibafa-elemzés

A hibafa-elemzés (angolul Fault Tree Analysis, röviden FTA) egy kvalitatív és kvantitatív elemzési módszer, amely logikai kapuk segítségével ábrázolja az alapvető események és a legfelső esemény (avagy top event) közötti logikai kapcsolatokat. [1]

A legfelső esemény az éppen elemzett esemény, például egy rendszerhiba vagy más nem kívánt esemény. Az alapesemény az a legkisebb elemzett esemény, amely valamilyen érdemi módon hozzájárul a legfelső eseményhez, például egy gépelem meghibásodása. Végül a logikai kapuk az alapesemények és a legfelső esemény közötti kapcsolatok megteremtésére szolgálnak. Egy logikai kapu számos eseményt vesz bemenetként, logikai műveletet (ÉS, VAGY, NEM) végez rajtuk, és egy köztes eseményt ad kimenetként, az utolsó ilyen köztes esemény a legfelső esemény.[1]

Ez a módszer bivalens (kétértékü avagy Boole) logikát használ, és bár a fának sok más fajta eleme is lehet, a legegyszerűbb formájaban csak az alapesemények, a köztes eseményeket és a felső esemény szerepelnek. Mivel az események bivalensek, vagy bekövetkeznek, vagy nem, a fa a mögöttes bivalens logika grafikus ábrázolásaként fogható fel, és így alkalmazható rajta Boole-algebra a Hibafa-elemzésére. [1]

Ábra 1: Szimbólumok az ÉS logikai kapunak (a.) és a VAGY logikai kapunak (b.)



Ábra 2: Egy Példa a Hibafa Felépítésére

A hibafák kvalitatív elemzése a Minimális Vágási Halmazok (angolul minimal cut sets) kiszámításával történik, amely a legfelső esemény bekövetkezését okozó alapesemények legkisebb kombinációja. Bármely hibafának véges számú minimális vágási halmazai vannak. [1]

A minimális vágási halmazok kiszámítása úgy történik, hogy először lefordítják a hibafát a vele egyenértékű Boole algebrai kifejezésre, majd a Boole-algebrával ezt átrendezve a szorzatösszek (avagy minterm vagy angolul sum of products, röviden SOP) alakra, ahol az összegek láncában minden szorzat egy vágási halmaz.

Miután a kvalitatív elemzést egy adott hibafán el lett végezve, kvantitatív elemzés végezhető. Az ilyen típusú elemzés úgy történik, hogy minden alapeseményhez hozzárendelnek egy becsült előfordulási valószínűséget. Ez után a hibafa felhasználható a legfelső esemény előfordulásának és az egyes alapesemények fontosságának becslésére. [1]

2.2 Bizonytalanság a hibafa-elmezés során

A hagyományos, bivalens logikát használó hibafa-elemzéshez az eseményeket pontosan meg kell határozni. Ennek elmulasztása, vagy akár a rosszul meghatározott események durva becsléseivel való számítás az elemzési eredmények bizonytalanságához vezethetnek. Ennek az az oka, hogy a kvantitatív elemzés során nagy mennyiségű összegyűjtött adat, egyes szakvélemények, szimuláció vagy egyéb módszerek alapján becsüljük meg az alapesemények bekövetkezésének valószínűségi eloszlását. Ezek mindegyikének megvannak a maga bizonytalanságai vagy szubjektivitásai, amelyek a legfelső esemény (angolul top event) bekövetkezési valószínűségének számítására kihatással lesznek. Ez a probléma azokban az esetekben súlyosbodik, amikor nem lehet elegendő mennyiségű adatot gyűjteni a bekövetkezési valószínűség számításához, például egy ritka alkatrész meghibásodás esetén. [2][3]

E bizonytalanságok leküzdésére vagy legalább számszerűsítésére számos módszert dolgoztak ki. Az egyik ilyen megközelítés a fuzzy halmazelmélet használata, gyakran nyelvi változókkal (angolul Linguistic Variables), hogy az alapesemény bekövezkezési valószínűségeinek becslésekből, a szakértői vélemények szubjektivitásából és a nagy mennyiségü adat hiányából adódó pontatlanságokat szamszerűsíteni lehessen. Ezt a megközelítést ígéretes eredményekkel alkalmazták az alapesemények bekövetkezési valószínűségeinek bizonytalanságának modellezésére, amik a hibafának a legfelső esemény bekövetkezési valószínűségében is számszerűsíthetően megjelennek és tovább analizálhatóak. [1][2][3]

3 HALMAZOK ÉS LOGIKA

3.1 Hagyományos Halmazok és Logika

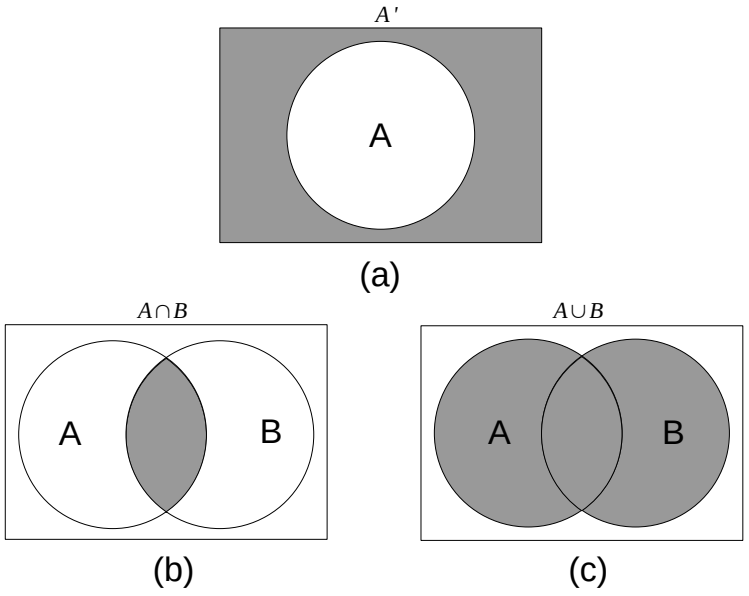
3.1.1 Halmazok

A halmaz a hagyományos értelemben olyan elemek gyűjteménye, amelyek lehetnek bármilyen matematikai objektumok vagy számok, akár más halmazok is. Azokat az elemeket, amelyek egy adott halmazon belül vannak, a halmaz tagjainak vagy elemeinek nevezzük.

A hagyományos halmazok bivalens logikát használnak a halmaztagság meghatározásakor, azaz egy adott objektum vagy szám a eleme a halmaznak, vagy nem eleme, nincs köztes állapot. [3]

Az objektum vagy szám egy adott halmazhoz való tagságát tesztelő matematikai függvényt a halmaz tagsági függvényének (angolul membership function) nevezzük.

A halmazoknak van néhány alapvető halmazművelete új halmazok létrehozásához a meglévőkből, például: Unió (Ábra 3.c) , Metszet (Ábra 3.b) és Komplemens (Ábra 3.a). Ezek alkotják a műveletek minimális halmazát amivel az összes hagyományos halmazlogika végrehajtható.



Ábra 3: Venn-Diagrammok (halmazábrák) a leggyakoribb halmazműveletekről

3.1.2 Logika

"A logika az érvelés módszereit és elveit tanulmányozza." [4]

A hagyományos logika propozíciókkal (állításokkal) dolgozik. Egy propozíció lehet igaz vagy hamis. Ez a propozícionális logika leírja a logikai változók és logikai primitívek (műveletek) kombinációinak kezelési módjait. A logikai primitívek és változók kombinációját logikai függvénynek nevezik. A logikai primitívek halmaza akkor teljes, ha bármely logikai függvény leírható véges számú logikai primitívvel a halmazból. ilyen propozíciós logika amely a logikai változók véges halmazán alapul, izomorf egy véges Boole-algebrával, és mindkét rendszer izomorf egy véges halmazelmélettel is. [4]

Propositional Logic		Boolean Algebra		Finite Set Theory	
Symbol	Name	Symbol	Name	Symbol	Name
∧	Konjunkció	· vagy &	ÉS	∩	Metszet
∨	Diszjunkció	+ vagy ||	VAGY	U	Unió
¬	Negáció	¯ vagy !	NEM	◌'	Komplemens

Táblázat 1: Egyenértékű logikai műveletek és elnevezéseik a Propozíciós logika, Boole-algebra és véges halmaz elméletben

A propozíciós logikát csak akkor mondjuk normál formájúnak, ha logikai változókat és azok tagadásait tartalmazza amik a ∧ és ∨ operátorokkal vannak összekapcsolva. Ezek a normál formák lehetnek konjunktívak vagy diszjunktívak, és mindkettő lehet kanonikus. A diszjunktív normálforma (DNF) az általános konjunktív-alkifejezések diszjunkciója, míg a konjunktív normálforma (CNF) az általános diszjunkciós részkifejezések konjunkciója. Mind a CNF, mind a DNF lehet kanonikus, ami azt jelenti, hogy minimalizálni kell az alkifejezések számát, miközben gondoskodik arról, hogy minden részkifejezés tartalmazza az összes logikai változót. A legfontosabb, hogy minden propozíciónak (logikai függvénynek) van egy ekvivalens konjunktív normálalakja és egy ekvivalens diszjunktív normálalakja. [5]

Példa a diszjunktív normál formára (DNF) :

$$(A \wedge B) \vee (C \wedge \neg B) \vee (\neg A \wedge \neg C \wedge D) \tag{3.1}$$

Példa a konjunktív normál formára (CNF) :

$$(\neg C \vee A) \wedge (D \vee B) \wedge (\neg A \vee D) \tag{3.2}$$

A Boole-algebrában a diszjunktív normálformát (DNF) szorzatösszegnek (angolul sum of products, röviden SOP) is nevezik, míg a konjunktív normálformát (CNF) összegek szorzatának (angolul product of sums, röviden POS) nevezik. Ez a két forma hasznos a logikai függvények egyszerűsítéséhez. A propozíciós logikához hasonlóan minden logikai függvénynek egyenértékű POS és SOP reprezentációja van.

A logikai algebra két alakjának szemléltetésére vegyük a következő logikai függvényt :

$$A + (D \cdot (B+C)) \cdot (B + (D \cdot C)) \tag{3.3}$$

Ennek egyenértékű egyszerűsített SOP alakja :

$$A + (B \cdot D) + (C \cdot D) \tag{3.4}$$

Míg az egyenértékü egyszerűsített POS alakja :

$$(A + B + C) \cdot (A + D) \tag{3.5}$$

3.2 Fuzzy Számok

3.2.1 Fuzzy Halmazok

"A fuzzy halmaz az objektumok egy osztálya, amelyek tagsági fokozatai folytonosak. Az ilyen halmazt a tagsági (karakterisztikus) függvénye jellemzi, amely minden objektumhoz egy nulla és egy közötti tagsági fokozatot rendel." [6]

A hagyományos halmazokban bivalens (avagy kétértékű) logikát használnak annak meghatározására, hogy egy objektum vagy szám egy adott halmaz tagja vagy sem. A hagyományos halmazok ilyen tagsági függvényei egyértelmű, igaz vagy hamis választ adnak arra a kérdésre, hogy „X tagja-e az Y halmaznak”, avagy 1-et és 0-t.

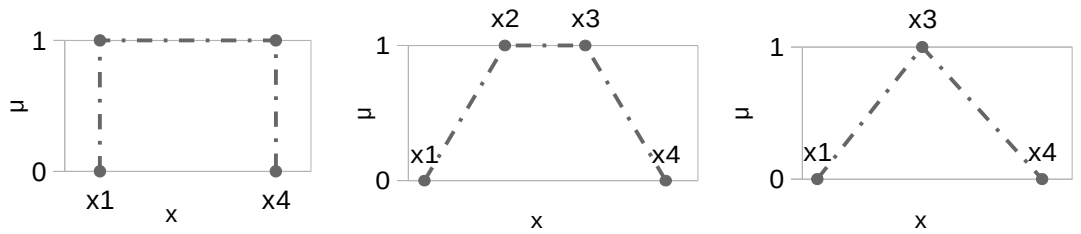
A hagyományos halmazokkal ellentétben a fuzzy halmazok tagsági függvényeket írnak le, amelyek segítségével minden objektum vagy szám tagságának mértéke ' μ_A ' kiszámítható, amely egy valós szám a $[0,1]$ zárt intervallumban. [6]

A fuzzy halmazok a hagyományos halmazokhoz hasonlóan viselkednek a halmazműveletek tekintetében. L. A. Zadeh bebizonyította, hogy a hagyományos halmazok alapvető azonosságai kiterjeszthetők a fuzzy halmazokra is, mint például De-Morgan törvényei és a disztributív törvények. [6]

3.2.2 Fuzzy Tagsági Függvények

Általában a fuzzy halmazokban a tagsági függvények háromszög vagy trapéz alakúak, még akkor is, ha más alakzatokat használnak. [3]

A fuzzy halmaz tagsági függvényével történő leírását **függőleges reprezentáció**nak (angolul Vertical Representation) nevezzük.



(a) Négyzetes (b) Trapéz (c) Háromszög

Ábra 4: Gyakori Fuzzy Halmazok Függőleges Reprezentációi

Legyen $x, x_1, x_2, x_3, x_4 \epsilon R$, egy fuzzy szám ' A ' leírható a $f_A(x)=\mu_A$ tagsági függvénnyel, ahol $\mu_A : R \rightarrow [0,1]$.

Függőleges reprezentációban a fuzzy halmazt a tagsági függvénye írja .

A trapéz fuzzy szám ' A ' ami a (Ábra 4.b)-ben látható a következő tagsági függvénnyel írható le:

$$f_A(x)=\begin{cases} \dfrac{x-x_1}{x_2-x_1} & for\ x_1 \le x \le x_2 \\ 1 & for\ x_2 \le x \le x_3 \\ \dfrac{x-x_4}{x_3-x_4} & for\ x_3 \le x \le x_4 \\ 0 & otherwise \end{cases} \tag{3.6}$$

Feltételezve, hogy $x_1 < x_2 < x_3 < x_4$ a trapéz fuzzy szám a következő képpen jelölhető $A=[x_1; x_2; x_3; x_4]$.

Egy háromszög alakú fuzzy szám, amint az a (Ábra 4.c) mutat, leírható egy ekvivalens trapéz fuzzy számmal, feltéve hogy $x_2=x_3$, így a következőképpen egyszerűsíthetjük az előző képletet:

$$f_A(x)=\begin{cases} \dfrac{x-x_1}{x_2-x_1} & for\ x_1 \le x \le x_2 \\ \dfrac{x-x_4}{x_2-x_4} & for\ x_2 \le x \le x_4 \\ 0 & otherwise \end{cases} \tag{3.7}$$

Feltételezve hogy $x_1 < x_2 < x_4$ a háromszög fuzzy szám a következőképpen jelölhető $A=[x_1; x_2; x_4]$.

Egy hagyományos halmaz tagsági függvénye közelíthető egy négyzet alakú fuzzy szám használatával, amint az a (Ábra 4.a) mutat, ez leírható egy ekvivalens trapéz fuzzy számmal, feltéve hogy $x_1 = x_2$, $x_3 = x_4$ és $x_1 < x_4$, így a következőképpen egyszerűsíthetjük az előző képletet:

$$f_A(x) = \begin{cases} 1 & for \quad x_1 \le x \le x_4 \\ 0 & otherwise \end{cases} \tag{3.8}$$

3.2.3 Alfa-vágás

Az alfa-vágás (angolul alpha-cut) metódus, más néven felbontási azonosság (angolul Resolution Identity), akkor alkalmazható fuzzy halmazokra, amikor hagyományos pont-szerű halmazra van szükség a képletek és a számítás egyszerűsítése érdekében.Az alfa-vágás módszerrel a fuzzy halmazokat diszkrét tagsági értékekre korlátozzuk. Az alfa-vágásokkal meghatározott fuzzy halmazt a halmaz **vízszintes reprezentációj**ának (angolul Horizontal Representation) nevezzük. [4]

Legyen ' A ' egy fuzzy halmaz $f_A(x)$ tagsági függvénnyel, ennek alfa-vágását (vagy alfa-szintű halmazát) ' A_α 'jelöljük, ahol $\alpha \ \epsilon \ [0,1]$ a vágás alfa-szintje, más néven megbízhatósági szint. Az alfa-szintű halmaza ' A '-nak a következőképpen definiálható:

$$A_\alpha = \{ \ x \epsilon X \ | \ f_A(x) \ge \alpha \ \} \tag{3.9}$$

Az erős alfa-vágás (vagy szigorú alfa-szintű halmaz), mint:

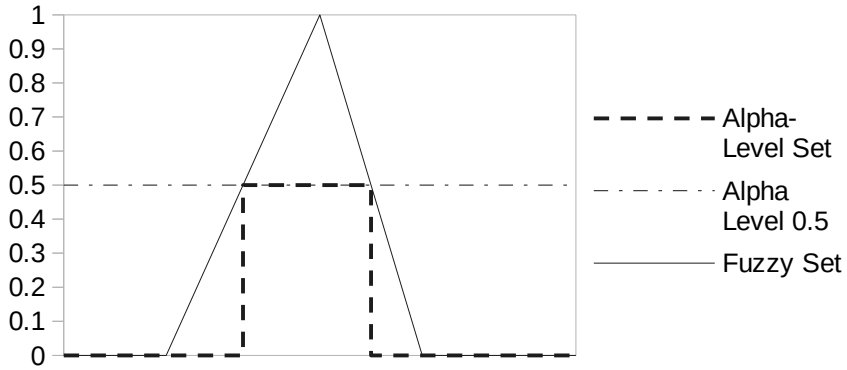
$$A_\alpha = \{ \ x \epsilon X \ | \ f_A(x) > \alpha \ \} \tag{3.10}$$

Ahol ' X ' az összes érték halmaza, amelyet ' x ' felvehet.

Más szavakkal, az alfa-vágás egy fuzzy halmazból létrehoz egy hagyományos halmazt, amelyet a fuzzy halmaz alfa-szintű halmazának neveznek, azáltal, hogy az elemek tagsági fokai ' μ ' által felvehető értékeket egy diszkrét bivalens halmazra korlátozza, amelyet matematikailag a következőképpen írnak le: $\mu \epsilon \{0, \alpha\}$.

A tagsági függvény a következőre módosul egy alfa-szintű halmaznál:

$$A_\alpha = \begin{cases} \mu_{A_\alpha} = \alpha \ \ if \ \ f_A(x) \ge \alpha \\ \mu_{A_\alpha} = 0 \qquad otherwise \end{cases} \tag{3.11}$$



Ábra 5: Az Alfa-Szintű halmaza egy háromszög fuzzy számnak a 0.5-ös alfa szinten

A fuzzy halmaz leírása az alfa-vágás segítségével a következő módon történik:

Legyen ' L ' az alfa-szintek halmaza a vágásoknak úgy, hogy $\alpha \epsilon L$. Legyen ' A ' egy fuzzy halmaz, amit az leírunk. Tegyük fel, hogy $X \epsilon R$, $x \epsilon X$ ahol ' X ' az intervallum, amelyen a Fuzzy halmazzal dolgozunk, és ' x ' egy pont az intervallumon belül. Az Alfa-szintű halmaz, ha folytonos a választott Alfa-szinten, úgy elírható mint $A_\alpha = [x_L \ , \ x_U]$, vagy ha nem folytonos az adott alfa-szinten leírható mint $A_\alpha = [x_{L(1)} \ , \ x_{U(1)}] \cup [x_{L(2)} \ , \ x_{U(2)}] \ ... \ [x_{L(i-1)} \ , \ x_{U(i-1)}] \cup [x_{L(i)} \ , \ x_{U(i)}]$, ahol $x_L \ , \ x_U$ az intervallum alsó és felső határa ahol $f_A(x) \geq \alpha$.

Egy konkrét példában tegyük fel, hogy leírjuk a fuzzy számot ' B ' és azt hogy $X = [0, 100]$, $L = [0, 0.5, 1]$. Az alfa-szintű halmazai minden alfa szinten a következők:

$$B_0 = [10, 80]$$
$$B_{0.5} = [10, 30] \cup [60, 80]$$
$$B_1 = \{15\} \cup [60, 70] \cup [75, 80]$$

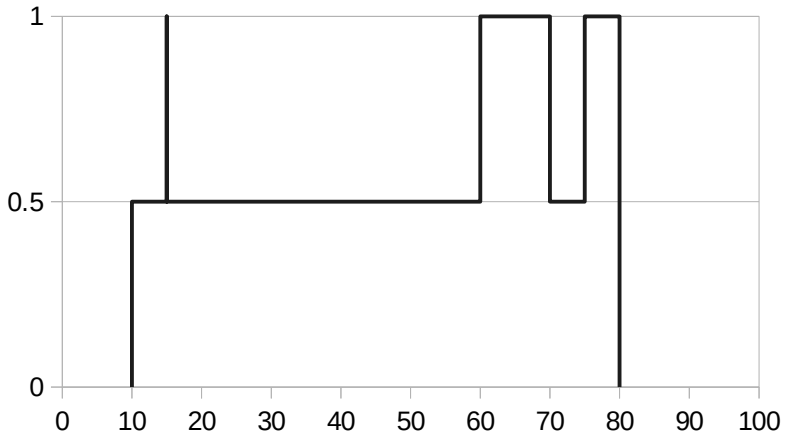


Figure 6: A fuzzy halmaz 'B' alfa-szintű halmazainak felső burkolata

3.2.4 Fuzzy Logika

A Fuzzy Logika esetében a Zadeh által leírt logikai műveletek teljes minimális halmaza {ÉS, VAGY, NEM}. [4]

A függőleges ábrázolásnál (angolul Vertical Representation) amely tagsági függvényeket használ a fuzzy halmazok leírására, ezek az alapvető logikai műveletek a következőképpen írhatók le:

Legyen ' A, B ' egy fuzzy szám $f_A(x), f_B(x)$ tagsági függvényekkel, ahol $x \in R$ és ' C ' a műveletek eredményeként kapott fuzzy szám, a logikai műveletek a következőképpen írhatók fel:

Komplemens (logikai NEM): $C = \neg A = 1 - f_A(x)$ vagy $C = \neg B = 1 - f_B(x)$ (3.12)

Metszet (logikai ÉS): $C = A \cap B = min(\, f_A(x)\, ,\, f_B(x)\,)$ (3.13)

Unió (logikai VAGY): $C = A \cup B = max(\, f_A(x)\, ,\, f_B(x)\,)$ (3.14)

A vízszintes reprezentáció (angolul Horizontal Representation) esetében, ahol a fuzzy halmazt az alfa-szintű halmazaival írjuk le, sokkal egyszerűbb, mivel ezek csak intervallumok uniói az egyes alfa-szinteken. Ezért hagyományos halmazlogikát lehet használni rajtuk, de ezt minden alfa-szinten meg kell tenni. Ennek megfelelően a logikai műveletek a következők:

Komplemens (logikai NEM): $C_\alpha = \neg A_\alpha$ minded alfa-szinten α az ' A ' halmaznak

Metszet (logikai ÉS): $C_\alpha = A_\alpha \cap B_\alpha$ (3.15)

minded alfa-szinten α az A, B halmaznak

Unió (logikai VAGY): $C_\alpha = A_\alpha \cup B_\alpha$ (3.16)

minded alfa-szinten α az A, B halmaznak

4 FUZZY HIBAFA ANALÍZIS

4.1 Fuzzy Halmazelmélet a Hibafa analízisben

A fuzzy hibafa elemzésben az események valószínűségét egy fuzzy számmal modellezik. A fuzzy halmazelméletet használva hasonló logikai kapukat használhatunk, mint a hagyományos hibafa-elemzésben a fa felépítéséhez. A köztes események és a legfelső esemény így szintén fuzzy számok lesznek, és az alapesemény-valószínűség bizonytalanságai a fán felfelé terjednek a legfelső eseményig. Így nem csak a legfelső esemény valószínűségét tudjuk megbecsülni, hanem a becslésben lévő bizonytalanságunkat is.

A legfelső esemény pont-szerű bekövetkezési valószínűség becsléseinek kiszámítása minden fuzzy szám alakzat esetén egyszerű. Viszont az értelmes elemzés elvégzéséhez bizonyos feltételezéseket kell tenni a mögöttes számítások és algebrai képletek egyszerűsítése érdekében.

4.2 Tagsági függvények és Nyelvi Változók

A fuzzy halmazok tagsági függvényei egységes alakúak minden eseménynél, általában trapéz vagy háromszög alakúak, ami leegyszerűsíti a logikai kapukkal való számítást, és csökkenti az egyes események alfa-vágásainak kiszámításának bonyolultságát.

Az alfa-szintű halmaz (avagy alfa-vágás) kiszámítható egy trapéz alakú fuzzy számra zárt intervallumként, amelynek alsó és felső korlátja a következőképpen irhatók le:

$$x_1 \le L \le x_2 \ , \ x_3 \le U \le x\,4 \tag{4.1}$$

$$\begin{aligned} L &= x_1 + \alpha * (x_2 - x_1) \\ U &= x_4 - \alpha * (x_4 - x_3) \end{aligned} \tag{4.2}$$

Ahol ' L , U ' az alfa-szint halmaz alsó és felső határa, x_1 , x_2 , x_3 , x_4 az trapéz alakú fuzzy szám kulcspontjai, és ' α ' az alfa-szint.

A háromszög alakú fuzzy számok ugyanazokkal a képletekkel számíthatók ki, mint a trapéz alakúak, azzal az eltéréssel, hogy $x_2 = x_3$.

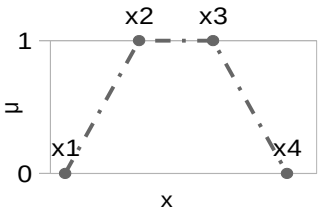


Figure 7: Tagsági funkciója egy trapéz fuzzy szám

Azonban az egyes alapesemények tagsági függvényének meghatározása időigényes lehet, és megköveteli a fuzzy halmazelmélet ismeretét és megértését. Így amikor nyers számokkal kell dolgozik, nehéz lehet elképzelni az egyik, másik fuzzy szám közötti különbséget. Emiatt nyelvi változókat (angolul Linguistic Variables) használnak, ahol a fuzzy számok egy csoportja ember által olvasható nevekhez van hozzárendelve az osztályozáshoz (például: Magas, Közepes, Alacsony) a terület szakértői által. Ez lehetővé teszi számunkra, hogy gyorsan és hatékonyan osztályozzuk az alapesemények bekövetkezési valószínűségét szakértői vélemények alapján, elfogadható bizonytalanság vagy szubjektivitás mellett. [3]

4.3 Algebrai egyenletek fuzzy valószínűségekhez

A hibafák kvantitatív elemzése során minden alapeseményhez hozzárendeljük annak bekövetkezési valószínűségét. A hagyományos hibafa analízisben ez egy pontbeli érték.

Ezeknek a valószínűségeknek a hibafa logikai kapukon való terjesztése valószínűség-elmélet segítségével történik, ami az [1]-ben van tárgyalva, még a [2]-ben pedig ezt a megközelítést alkalmazza fuzzy valószínűségi számokra.

Az események valószínűségeinek hibafán keresztüli számításához a következőképpen írhatjuk le a három leggyakoribb kapu függvényeit:

Legyen ' A , B ' két ismert esemény meyleg bekövetkezési valószínűsége F_A , F_B , nem negatív valószínűségeket feltételezve a logikai ÉS kapu a következőképpen írható fel:

$$F_{A\cap B}=F_A\times F_B \quad \text{általánosítva} \quad F^{\text{AND}} = \prod_{i=1}^{n} F_i \tag{4.3}$$

A NEM kapu vagy komplementer csak egyetlen valószínűséget fogad bemenetként, és a következőképpen írható le:

$$F^{NOT} = 1-F \tag{4.4}$$

Végül az [1]-ben leírt VAGY kapu a maxterm alakban van írva, míg [2]-ben leírt egy sokkal egyszerűbb minterm jelölés, és mivel mindkettő ekvivalens, az egyszerűbb minterm jelölést választva az általánosított alakj a következő képpen írható fel:

$$F^{\text{OR}} = 1 - \prod_{i=1}^{n} (1-F_i) \tag{4.5}$$

Ezeket a képleteket azonban pont-szerű valószínűségi számokra használjhatók, még a fuzzy hibafa analízisben ezek helyett fuzzy számok szerepelnek, így önmagukban a képletek nem használhatók.

Az egyszerű alakzatú, például trapéz vagy háromszög alakú fuzzy számokat azonban kulcspontjaik írják le, amelyek point-szerű valószínűségi számok. Még ezek a fuzzy számok alfa-vágása is két kulcspontot (az alsó és a felső korlát) írják le, amelyek szintén pont-szerű valószínűségi értékek. Lényegében a kulcspontokkal ábrázolt fuzzy számok felírhatók n-elemű vektorként, majd elemenkénti műveletekkel (összeadás, kivonás, szorzás) a fenti képletekkel a normál valószínűségi képleteket alkalmazhatjuk fuzzy számokra.

Például legyenek a bemenetek trapéz alakú fuzzy valószínűségi számok, amelyeknek 4 kulcspontjuk van $' x, y, z, k '$, a logikai kapu műveletei a következőképpen bővíthetők és írhatók fel:

$$F^{\mathrm{AND}} = \prod_{i=1}^{n} F_i = \prod_{i=1}^{n} \begin{bmatrix} x_i \\ y_i \\ z_i \\ k_i \end{bmatrix} = \begin{bmatrix} \prod_{i=1}^{n} x_i \\ \prod_{i=1}^{n} y_i \\ \prod_{i=1}^{n} z_i \\ \prod_{i=1}^{n} k_i \end{bmatrix} \tag{4.6}$$

$$F^{NOT} = 1 - F = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} x \\ y \\ z \\ k \end{bmatrix} = \begin{bmatrix} 1-x \\ 1-y \\ 1-z \\ 1-k \end{bmatrix} \tag{4.7}$$

$$F^{\mathrm{OR}} = 1 - \prod_{i=1}^{n} (1 - F_i) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \prod_{i=1}^{n} \begin{bmatrix} 1-x_i \\ 1-y_i \\ 1-z_i \\ 1-k_i \end{bmatrix} = \begin{bmatrix} 1 - \prod_{i=1}^{n} (1-x_i) \\ 1 - \prod_{i=1}^{n} (1-y_i) \\ 1 - \prod_{i=1}^{n} (1-z_i) \\ 1 - \prod_{i=1}^{n} (1-k_i) \end{bmatrix} \tag{4.8}$$

4.4 Fuzzy fontossági és bizonytalansági mérőszámok

A hibafa-elemzés egyik fontos felhasználási területe az olyan kritikus események azonosítása, amelyek a legnagyobb mértékben járulnak hozzá a legfelső esemény előfordulásához. Ez elengedhetetlen, ha biztonsági elemzésre használják. [7][13]-ben két különböző fontossági mérőszámot használtak, a fuzzy fontossági mérőszámot (angolul Fuzzy Importance Measure, röviden FIM) és a fuzzy bizonytalansági fontosság mérőszámot (angolul Fuzzy Uncertainty Importance Measure, röviden FUIM).

A FIM a Birnbaum fontosságának kiterjesztése, amelyet a hagyományos hibafa analízisnél használnak. Olyan kritikus alapesemények azonosítására szolgál, amelyek a legnagyobb mértékben járulnak hozzá a legfelső esemény előfordulásához. [7][13]

Tegyük fel, hogy van egy fuzzy hibafánk, amely ' n ' alapeseményt tartalmaz, és ki akarjuk számítani az ' i ' alapesemény FIM értékét, ahol ' q_i ' az alapesemény bekövetkezési valószínűségét leíró fuzzy szám. A legfelső esemény bekövetkezési valószínűségét ' Q ' egy olyan függvény számítja ki, amely az alapesemények és azok logikai kapcsolatait figyelembe veszi, az alábbi képlettel ezt helyettesítve:

$$Q = f(q_1 , q_2 , \dots , q_{i-1} , q_i , q_{i+1} , \dots , q_n) \tag{4.9}$$

A FIM kiszámítható úgy, hogy feltételezzük, hogy az alapesemény ' i ' bekövetkezett ($q_i = 1$), és ezzel a feltételezéssel számítjuk ki a legfelső eseményt ($Q_{q_i=1}$), majd feltételezzük, hogy az alapesemény nem következett be ($q_i = 0$), és ezzel a feltételezéssel újból kiszámítjuk a legfelső eseményt ($Q_{q_i=0}$). A különbség a két legfelső esemény között a Birnbaum fontosság, avagy a FIM. [7][13]

Azonban a legfelső esemény is egy fuzzy szám, ezért [7][13]-ben a szerzők egy egyszerű megközelítést javasoltak a különbség kiszámítására, euklideszi távolságok használatával. A fuzzy fontosság mértéke (FIM) kiszámítható a következő képlettel:

$$FIM_i = ED[Q_{q_i=1} , Q_{q_i=0}] \tag{4.10}$$

Ahol ' $ED[A,B]$ ' az euklideszi távolság a fuzzy halmazok ' A,B ' között és a következő képpen van definiálva:

$$ED[A,B] = \sum_{\alpha} \sqrt{(A_\alpha^L - B_\alpha^L)^2 + (A_\alpha^U - B_\alpha^U)^2} \tag{4.11}$$

Ahol ' A_α^L ' és ' A_α^U ' az alsó és felső határa a fuzzzy szám ' A ' alfa-vágásának az ' α ' alfa szinten, minden alfa-szinten.

A FUIM az alapesemény ' i ' bizonytalanságának hozzájárulását jelenti az legfelső esemény bizonytalanságához, ez használható annak meghatározására, hogy melyik alapeseményről kell több adatot gyűjteni, hogy a rendszer teljes bizonytalansága a legnagyobb mértékben csökkenjen.

A FUIM a FIM-hez hasonló módon számítható, azzal a különbséggel, hogy azt a feltételezést használjuk, hogy az alapeseménynek pont-szerű bekövetkezési valószínűsége van, nem pedig egy fuzzy szám. Ez lehet pontbeli vagy intervallum is attól függően, hogy milyen stratégiát használunk a fuzzy számból pont-szerű leképezésére. Ezzel a pont-szerű értéket majd kiszámítjuk a legfelsőbb esemény bekövetkezési valószínűségét ' Q_i ' . Ez azután összehasonlítjuk a legfelső esemény bekövetkezési valószínűségével ' Q ' e feltételezés nélkül a következő képlettel:

$$FUIM_i = ED[Q\ ,\ Q_i] \tag{4.12}$$

5 TREEZZY2 Szoftver

A **TREEZZY2** szoftvert a Palermo Egyetem Nukleáris Mérnöki Tanszéke fejlesztette ki Olaszországban, a munkát 2004 körül mutatták be. Ez egy grafikus felülettel ellátott számítógépes szoftver fuzzy hibafa vagy fuzzy eseményfa elemzés elvégzésére. Megközelítheti a legfelső eseményt, vagy használhatja az alfa-vágás módszert, hogy pontosabb eredményt kapjon hosszabb számítási idő árán. További funkciók közé tartozik a minimális vágási halmazok számítása, amely a legfontosabb számítások elvégzéséhez szükséges, a FIM és a FUIM számításához és rangsorolásához. [8]

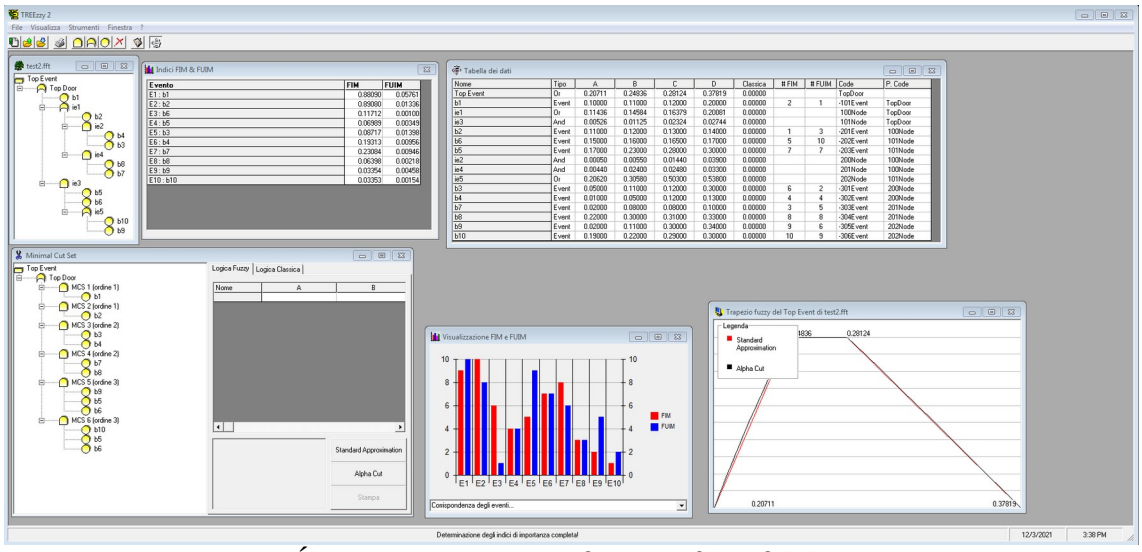
Sajnos a cikkből nem derül ki sok információ [8], mivel csak bemutatja a munkájukat, és nem adják meg a forráskódot sem. További bonyodalmat okoz, hogy olasz nyelvű a szoftver és nincs lehetőség arra, hogy angolra váltsák, ami szükségessé tette a fordítóprogramok és megfigyelésekre hagyatkozást a program működésének megértéséhez.

Ezenkívül a softver Visual Basic nyelven íródott, és működéséhez a **Microsoft Visual Basic 6 Service Pack 6**-ot telepíteni kell egy Windows gépre. A visual basic telepítés meghiúsulhat, ilyenkor egy külső **7zip** nevű programot kell használni a telepítő kicsomagolásához, az így létrejövő **.dll** és **.ocx** fájlokat manuálisan kell regisztrálni a Windows parancssorából (ügyelve, hogy a parancssor megnyitása rendszergazdai jogosultságokkal történjen) a **regsvr32** paranccsal.

A szoftverben viszonylag kevés beállítási lehetőség van, főleg, hogy hány számjegy pontosságot jelenít meg, az alapesemények alakja (trapéz vagy háromszög) és végül a jelölés, amely lehet decimális (0.15) vagy tudományos jelölés (1.5e-1) .

Érdekes megjegyzés a szoftver működésével kapcsolatban, hogy a legfelső esemény számítása normál módon történik, míg a FIM és a FUIM minimális vágási halmazokat igényel. Figyelmeztető üzenet jelenik meg, ha a minimális vágási halmazokat még nem lettek kiszámítva a fontossági mérték számításának kiválasztásakor.

A szoftver hiányossága, hogy nincs kézenfekvő módja annak, hogy ugyanazt az alapeseményt több helyen használjuk, többször kell deklarálni különböző néven, ami miatt a FIM és a FUIM ilyenkor használhatatlanná válik.



Ábra 8: TREEZZY2 softver grafikus felülete

6 FUZZYFTAPY SOFTVER FEJLESZTÉS

6.1 Python nyelvi áttekintés

A Python egy magas szintű, objektumorientált, fordított nyelv amely dinamikus szemantikával és dinamikus gépeléssel. A könnyen olvasható szintaxisra összpontosít, és támogatja a moduláris alkalmazások létrehozásához szükséges csomagokat és modulokat. [10]

Olvashatósága, valamint a népszerű keretrendszereket és könyvtárakat fejlesztő programozók és tudósok nagy közössége miatt népszerű a tudományos és mérnöki területeken.

Ez a rész az alapvető programozási szintaxis általános áttekintése a kifejlesztett szoftverben használt szintaxisból példákkal. Az egyes szoftverrészekben használt trükköket és szintaxist a (6.4 fejezet) ismerteti, ahol először fordul elő a forráskódban.

6.1.1 Változók

A Pythonban a változók deklarálása a következő szintaxissal történik:

```
varName = 2
varNameAswell = varName

anotherVar: dict = {'a': 1}
```

Kódrészlet 1: Példa a változó deklarációra

Az így deklarált változók abba a névtérbe kerülnek, amelyben deklaráltak, a fenti példában mindkét változó a globális névtérbe kerül.

A Python egy dinamikusan tipizált nyelv, ami azt jelenti, hogy a változótípus-deklarációk szükségtelenek, mivel a rendszer a változók hozzárendelésekor következtet a tipusukra minden esetben. [10]

Fontos szem előtt tartani, hogy a Pythonban minden egy objektum, ide tartoznak a számok, listák, függvénydefiníciók és osztálydefiníciók is, a változók pusztán az objektumra mutató elnevezett mutatók (pointer). Ennek megvan az a mellékhatása, hogy a fenti példát használva, amikor megváltoztatjuk a **varNameAswell** értékét, akkor a **anotherVar** vele együtt fog változni, ez olyan esetekben történik, amikor a változó egy összetett objektumra mutat, például egy szótárra (dictionary), ebben az esetben az objektum másolatának készítéséhez a hívása **copy()** vagy **deepcopy()** metódus hívása szükséges az objektum független másolatának létrehozásához. [10]

Végül a Python lehetővé teszi a változók típusainak megadása (avagy "type-hinting") amely a fenti **anotherVar** változóban található szintaxis használatával, ami hasznos a fejlesztés során, mivel a modern kódszerkesztők "type-hint"-eket jelenítenek meg a függvényhívások során, hogy segítsenek a fejlesztőknek eldönteni, milyen argumentumokat használjanak, és statikus típusellenőrző szoftverrel is használható. Általánosan bevált gyakorlat a függvény argumentumainak típusainak megadása a deklaráció során, mivel az egy másik rétegként működik a dokumentáció. [10]

6.1.2 Funkciók

A Pythonban a kódblokkok (loopok, függvénydeklarációk stb.) elválasztása szóközök behúzásával történik a kapcsos zárójelek használata helyett. Ez viszont a programkód vizuális szerkezetét pontosabban ábrázolja szemantikai struktúrájában. A következő példa:

```
def some_function(positionalArgument: int, keyWordArgument: bool = True):
        aVar = 2
        print("Hello world!")
        print("This is a function")
        print(positionalArgument)
        return 42

some_function(3)
some_function(3, keyWordArgument = False)
```

Snippet 2: Példa funkció deklarációra

A fenti példában egy függvényt két különböző fajta argumentum deklarál, pozicionális (positional) és kulcsszó (key-word).

Pozíciónális argumentumok szükségesek, és ezek megadásának elmulasztása hibát okoz, sorrendjük számít, és megadandó a függvény meghívásakor, ahogy fentebb látható a " **some_function(2)** " szintaxissal. [10]

A Kulcs-szó argumentumok szigorúan a pozíciónális argumentumok után jönnek a függvény deklarálása során, ezek deklarálása valamilyen alapértelmezett érték hozzárendelésével ugyanúgy történik, mint a változók deklarálásakor, a fenti példában a kulcsszó argumentum hozzáadásának szintaxisa "**keyWordArgument = True**", ez beállítja az alapértelmezett értéket, ha nincs megadva, és a függvényhívások során történő megadása a következő szintaxissal történik "**some_function(3, keyWordArgument = False)**". [10]

Végül a függvényeknek mindig van visszatérési értéke, ha nincs kifejezetten megadva, a függvény a **None** objektumot adja visszatérési értéknek, a függvény visszatérési

értékének meghatározásához a **return** kulcsszó használatával történik, amit a visszatérési érték követ. Figyelembe, hogy ez azonnal a függvény kilépését okozza. [10]

Érdemes megjegyezni, hogy a függvények használhatják a **yield** kulcsszót a **return** helyett hogy egy generátor funkciót hozzunk létre, a generátorok visszatérési érték adása után nem lépnek ki, hanem a **yield** sorában szüneteltetik lefutásukat, így később folytatható a végrehajtásuk. [10]

6.1.3 Osztályok

Az új objektumosztályok meghatározása a Pythonban a következő szintaxissal történik:

```
class NameOfClass:

        clsVar = 2

        def __init__(self, argument: int, kwArgument: bool = False):
                self.locVar = argument
                self.locVarKw = kwArgument

        def display_all(self):
                print(self.clsVar)
                print(self.locVar)
                print(self.locVarKw)

        @staticmethod
        def test():
                print("Hello World!")

class Inheritor(NameOfClass, OtherParent):
        pass

newInstance = NameOfClass(12)  # instancing
newInstance.display()          # calling a method
print(NameOfClass.clsVar)      # accessing an attribute
```

Snippet 3: Példa az objektum osztályokra

A fenti példában a **NameOfClass** osztály nem örököl más osztályoktól, így a zárójel elhagyható. Az osztályon belül definiált függvényeket metódusoknak nevezzük, az első automatikusan bevitt argumentum a methódusoknak az objektum hivatkozása (amelyen a metódus működik), és konvencióként **self** névvel jelöljük. [10]

A dupla aláhúzással kezdődő és végződő metódusok (más néven "dunder methods") speciálisak (mágikus metódusoknak nevezik), számos ilyen metódus létezik, amelyeket az osztályok definiálhatnak a funkcionalitásuk megváltoztatására, de a leggyakoribb az **__init__** , amely az osztály példányosítása után hívják meg az osztály inicializálásának befejezéséhez. [10]

Az objektumban definiált változókat attribútumoknak nevezzük, és az **object.attribute** szintaxissal vagy konkrét példában **newInstance.locVar** segítségével érhetők el. Az objektumokhoz több attribútum is hozzáadható, és értékeik felülbírálhatók futás közben, mint bármely változó. A metódusok ugyanúgy érhetők el, mint az attribútumok pontozott szintaxis használatával. [10]

Az **__init__**-en kívül deklarált attribútomok, mint például a **clsVar** , osztályszintű attribútumok, amelyekhez minden példány hozzáfér, és még az osztály példányosítása nélkül is elérhetők. Amikor egy objektum megpróbálja felülbírálni az osztályszintű attribútumait helyette egy új, azonos nevű attribútumot kap, és mivel az objektumok először a helyi attribútumokban keresnek, mielőtt az osztályszintű attribútumokat keresnék, szükség esetén példányonként felülbírálhatók. [10]

Az osztályok örökölhetnek attribútumokat és metódusokat más osztályoktól, a Python pedig támogatja a többszörös öröklődést, ami azt jelenti, hogy egyetlen osztály több osztályból is örökölhet, amint az az **Inheritor** [10] példában látható.

Végül az **@staticmethod** szintaxist dekorátornak nevezzük. A dekorátorok kívül esnek ennek az áttekintésnek a hatókörén, de érdemes megjegyezni, hogy ez a konkrét metódus lehetővé teszi a fent leírt metódus meghívását anélkül, hogy osztályt kellene példányosítani, mivel nem igényel objektum hivatkozást. [10]

6.1.4 Kommentek, Doc-Stringek és F-Stringek

Megjegyzések a # karakterrel illeszthetők be, az ugyanabban a sorban utána lévő bármi megjegyzésként fog kezelni. [10]

Többsoros megjegyzések írhatók hármas idézőjelekkel **"""Példa megjegyzés"""** ez csak egy többsoros karakterlánc, amelyet a Python értelmező figyelmen kívül hagy, ha nincs más kódrészlet ugyanabban a sorban, mint az elején vagy a végén abból. [10]

Ha egy függvény, metódus vagy osztály meghatározása során az első sor egy karakterlánc (string), akkor ezt doc-stringnek nevezzük, és ez az adott függvény, metódus vagy osztály dokumentációjaként működik, amelyet a kódszerkesztők gyakran megjelenítenek, hogy segítsék a fejlesztőket. Egy példa a doc-stringek írására minden használati esetre a következő [10]:

```
def func():
        """This is a function documentation"""

class TestClass:
        """This is a class documentation"""

        def test_method(self):
                """This is a method documentation"""
```

Kódrészlet 4: Példa a Doc-string szintaxisra

Végül az f-stringek a karakterláncok egy speciális típusa, amelyet az **f"..."** szintaxis jelöl. Az F-stringek lehetővé teszik változók és függvényhívások karakterláncokba ágyazását, és azok visszatérési értékeinek beépítését a szövegbe. Az f-string belül a kapcsos kapcsos zárójelek között lévő dolgok, például a **{varName}** kódként értelmeződnek és végrehajtódnak, változók esetén az értékeket veszik, míg a függvények visszatérési értékeit veszik, és karakterláncokká alakítják át, amelyek beágyazódnak a f-string. [10]

Példa az f-stringre:

```
f"Hello {name} ! Today is {get_day()} ."
```

Kódrészlet 5: Példa az f-string szintaxisra

6.1.5 Elágazások

A Pythonban az **if, elif, else** kulcsszavak alkotják a számítógépes kód vezérlési folyamatának elágazó végrehajtását. A legjobban egy példa írja le:

```
if value < 1:
        print("runs if the expression evaluates true")
elif value < 2:
        print("runs if the first expression evaluates false, but this one true")
else:
        print("runs if none of the if/elif branches run")
```

Kódrészlet 6: Példa az elágazás szintaxisra

A fenti példa alapján az **if** kulcsszó létrehozza az elágazást, az utána lévő kifejezésnek logikai (Igaz vagy Hamis) értékre kell kiértékelődnie hogy végrehajtódjon. [10]

Az **elif** utasítás ugyanúgy működik, mint az **if** azzal a kivétellel, hogy egy **if** blokkot kell követnie, és csak akkor fut le ha az előtte lévő **if** blokk nem futott le és ha kiértékeli a saját függvényét Igazra, lényegében egy "**else if**" blokk.[10]

Az **else** blokknak követnie kell egy **if** vagy **elif** blokkot, és ez az utolsó blokk, amely akkor fut, ha az előzőek nem futottak le. [10]

A programvégrehajtás elágazásának egy másik módja a **try** kulcsszónak használata, **except, else, finally**. Ez túlmutat a dolgozat keretein, de érdemes megjegyezni, hogy egy kódrészlet sikeres végrehajtásának vagy hibás működésének ellenőrzésére, valamint konkrét hibák elkapására vagy figyelmen kívül hagyására szolgál. [10]

6.1.6 Ciklusok

A pythonban kétféle ciklus létezik, az egyik feltételes, a másik pedig iteratív. A **while** kulcsszó ciklus fut, miközben a feltétel kifejezés Igazra értékelődik ki és minden ciklusnál tesztelődik, egyébként kilép a ciklusból. [10]

A **for** ciklus viszont másként működik, mint a többi programozási nyelvben. Ahelyett, hogy a ciklus növelne egy számlálót, iterálható objektumokat (angolul iterable) fogyaszt el. Minden ciklusban egy elemet vesz ki az iterálható objektumból és hozzárendeli egy változóhoz, amellyel a ciklus során dolgozni lehet. Iterálható bármely objektum, amely lista, szótár, karakterlánc, generátor vagy bármilyen más összetett objektumosztály ami definiálja a megfelelő mágikus metódusokat, hogy iterálhatóak legyenek. [10]

A ciklusoknak két kulcsszó is van a működésük további finomítására. A **continue** kulcsszó a következő ciklusra ugrik (a ciklus elejére), míg a **break** kulcsszó kilép a ciklusból. [10]

Végül a ciklusokat követheti egy opcionális **else** blokk, amely akkor és csak akkor fut le, ha a ciklus nem egy **break** kulcsszó miatt lép ki. [10]

```
x = 0

while x < 5:
        x += 1

for i in range(10):
        if i == 5:
                break
        elif i == 2:
                Continue

else:
        print("woops")
```

Kódrészlet 7: Példa ciklusok szintaxisára

A fenti példában a **range(10)** beépített függvény létrehoz egy generátort, amely pontosan 10 egész számot ad vissza **0**-tól kezdve, **1**-gyel növelve. Általában ezt a pontos szintaxist használjuk, ha a ciklusnak egy bizonyos számszor kell lefutnia.

6.1.7 Modulok Importálása

Az importálás az **import** kulcsszóval történik. Az importálás a megadott python-fájlt az aktív könyvtárba tölti be, egy telepített vagy beépített modult teljes egészében. [10]

A Python import modulja kívül esik a dolgozat hatókörén.

6.2 Szoftvertervezési szempontok

A szoftver elnevezése **fuzzyftapy** , ami a Fuzzy-FaultTreeAnalysis-Python névből jön.

Ahogy korábban a (4.2 fejezetben) tárgyaltuk, a leggyakoribb fuzzy számok trapéz vagy háromszög alakúak, a háromszög alakzat pedig a trapéz egy speciális esete. Ebben a szoftveres megvalósításban kizárólag a trapéz fuzzy számok kerülnek. Erőfeszítéseket kell tenni annak érdekében, hogy a jövőben szükség szerint más alakú fuzzy számokra is kiterjeszthető legyen.

Két megvalósításra választott logikai kaputípus az **ÉS** kapu, **VAGY** kapu. A NEM kapu kimarad, mivel nem használták a szakirodalomban, illetve az eredmények ellenőrzésére használt szoftverben. A kapuk az alapesemények logikai összekapcsolására szolgálnak.

A hibafa kvalitatív elemzéséhez le kell vezetni a minimális vágási halmazokat. Ez megtehető, ha feltételezzük, hogy az alapesemények bivalens logikát használnak, majd Boole-algebrát használunk a (3.1.2 fejezetben) említett szorzatösszeg (SOP) megjelenítéséhez.

A kvantitatív elemzés elvégzéséhez a fuzzy fontosság (FIM) és a fuzzy bizonytalanság fontossága (FUIM) mértékeket a (4.4 fejezetben) leírtak szerint hajtjuk végre.

A program parancs-sorból vezérelhetőként lesz fejlesztve, amivel lecsökken a fejlesztési időt, a későbbiekben pedig grafikus felhasználói felületet lehet hozzáadni, hogy a szoftver mások számára is használható legyen.

A szoftverben használandó hibafa betöltéséhez a szöveges fájlok használatának egyszerű megközelítését kell alkalmazni. Ez kiküszöböli az adatok betöltéséhez szükséges összetett felhasználói felületek programozását, mivel azok szövegszerkesztővel írhatók, feltéve, hogy a fájlszerkezet elég egyszerű.

Végül az elemzés eredményei egy JSON-szövegfájlba kerülnek későbbi elemzés céljából, vagy közvetlenül a konzolra nyomtatódnak JSON formában, a felhasználói preferenciáktól függően.

6.3 Mentési Fáljszerkezet

A Pythonban a szótár (dictionary) az objektumkapcsolatokat, sőt maguk az objektumok megjelenítésének natív módja. A szótárak lemezen fájlként való mentéséhez a JavaScript Object Notation (JSON) formátumot használják, mivel a python eszközöket biztosít a szótárak és a JSON-dokumentumok közötti natív konvertáláshoz.

A szótár (dictionary) és a JSON formátum kulcs-érték párokban tárolja az adatokat. A kulcs egyedi, és általában egy karakterlánc, amivel adatokat tudunk lekérni. Az érték bármilyen tetszőleges adat lehet egy szótárban, sőt más szótárak is lehetnek, de a JSON-ban néhány típusra korlátozódik, nevezetesen karakterláncra, egész számra, lebegőpontos számra és egyebek, legfőképpen más JSON-dokumentumok is lehetnek. Ez utóbbi funkció lehetővé teszi JSON-dokumentumok beágyazását egy JSON-dokumentumba egy kulcs alatt, és lehetővé teszi, hogy összetett adatstruktúrákat és objektum-kapcsolatokat reprezentáljon.

Fontos megjegyzés a JSON-struktúrában, hogy a kulcs-érték párokat és a listabejegyzéseket vessző választja el egymástól, és ami a legfontosabb, az utolsó ilyen elem vagy kulcs-érték pár nem tartalmazhat végződhet vesszővel, különben hiba keletkezik.

```
{
        "key1" : {
                "key11" : 12,
                "key12" : "embedded"
        },
        "key2" : 3.14
}
```

Kódrészlet 8: Példa JSON struktúrára

A JSON formátumot használva a fastruktúra reprezentációjához szükség van a metaadatok, az alapesemények és a logikai kapuk be-/kimeneteinek tárolására. A teljes szerkezet a következő:

```
{
"metadata" : { ... },
"base-events" : { ... },
"logic-gates" : { ... }
}
```

Kódrészlet 9: A mentési fájl struktúrája

A **"metadata"** rész tartalmazza a szoftver verzióját és az alapesemények típusait.

A **"base-events"** részben kerülnek alapesemények tárolásra, amelyek neveinek egyedinekek kell lenniük, mivel kulcsként használják, míg az értékeik a kulcspontok listája lesz. A következő egy példa erre:

```
"base-events" : {
        "example trapezoidal base event" : [0.12 , 0.15 , 0.17 , 0.20],
        ...
}
```

Kódrészlet 10: Alap esemény deklaráció a bemeneti JSON fájlban

A **"logic-gates"** rész tartalmazza a hibafa elemzésben használt összes logikai kaput. Ebben a részben a kulcsok a közbenső események (a logikai művelet kimenete) nevei. Minden logikai kapu tartalmaz egy másik JSON dokumentumot, amelyben két kulcs van: **"type"** és **"inputs".** A **"type"** határozza meg a kapu által végrehajtott logikai műveletet, míg az **"inputs"** felsorolja az összes alapesemény vagy köztes esemény kulcsát, amelyet a kapu bemenetként használ. Egy példa ennek a struktúrájára:

```
"logic-gates" : {
        "top-event" : {"type": "and", "inputs": ["base event 1", "intermediate 1"]},
          "intermediate 1": {"type": "or", "inputs": ["base event 2", "intermediate 2"]},
            "intermediate 2": {"type": "and", "inputs": ["base event 3", "base event 4"]}
}
```

Kódrészlet 11: logikai kapu deklaráció a bemeneti JSON fájlban

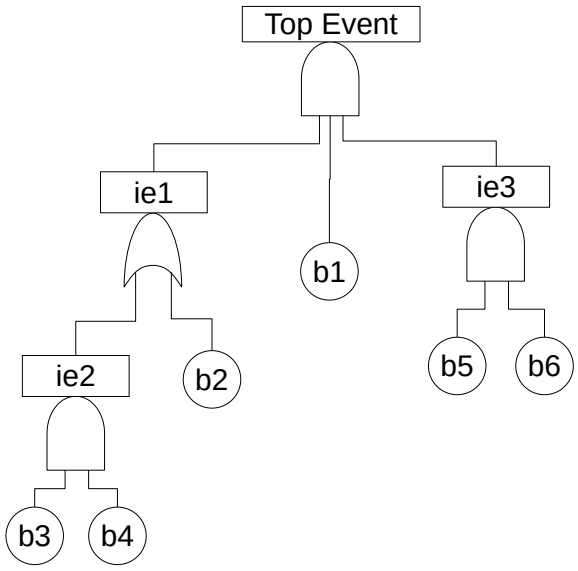
A fenti példában a bekezdés behúzása nem szolgál más célt, csak a fa szerkezetének megjelenítését egyértelműbbé tenni.

Egy érvényes bemeneti fájl konkrét példája a következő:

```
{
  "metadata": {
    "version": "0.0.1",
    "base-event-shape": "trapezoidal"
  },
  "base-events": {
    "b1": [0.1, 0.11, 0.12, 0.2],
    "b2": [0.21, 0.29, 0.31, 0.4],
    "b3": [0.05, 0.11, 0.12, 0.3],
    "b4": [0.3, 0.35, 0.48, 0.49],
    "b5": [0.17, 0.23, 0.28, 0.3],
    "b6": [0.22, 0.3, 0.3, 0.33]
  },
  "logic-gates": {
    "top-event": {"type": "and", "inputs": ["b1", "ie1", "ie3"]},
      "ie1": {"type": "or", "inputs": ["b2", "ie2"]},
        "ie2": {"type": "and", "inputs": ["b3", "b4"]},
      "ie3": {"type": "and", "inputs": ["b5", "b6"]}
  }
}
```

Kódrészlet 12: Érvényes bemeneti fájl konkrét példája

Melyik a következő hibafa reprezentációja:



Ábra 9: A 12. kódrészletben szereplő bemenet fájl hibafa ábrája

6.4 Szoftverkód és Dokumentáció

6.4.1 Szoftverfüggőségek

```
import os
import itertools
import json
import math
```

Kódrészlet 13: Python modulok importálása

Maga a szoftver kifejezetten Python 3.9.7-re íródott, a későbbi verzióknak probléma nélkül kell működniük, és a Python 3.6+ vagy későbbi verziójának is működniük kell.

Az egyetlen opcionális külső függőség a **matplotlib** (a 3.4.3-as verzió, de a többi verziónak probléma nélkül kell működnie), ami grafikonok rajzolására és megjelenítésére használatos. Ha ezeket a funkciókat nem használják (például amikor a szoftver be van ágyazva), akkor nem szükséges telepíteni.

6.4.2 Modulszintű változók

```
__PRECISION__ = 12
__version__ = "0.0.1"
__MINIMUM_VERSION__ = "0.0.1"
```

Kódrészlet 14: Modulszintű változók és értékeik

Amikor egy Python-fájlban deklarálunk változókat, nem pedig függvény- vagy osztálydefiníción belül, akkor a modul globális névterébe helyezzük azt, ami csak akkor egyezik meg a globális névterével, ha a Python-fájlt közvetlenül futtatjuk, és nem importáljuk mint függőség.

A **__PRECISION__** egy egész szám, amely meghatározza a számok pontosságát (hány tizedes számjeggyel számít). Minden számítás a **round(x , __PRECISION__)** függvényt használja, hogy elkerülje a lebegőpontos pontossági hibákat az összehasonlítások és logikai műveletek során, ahol **x** a szám amely **__PRECISION__** tizedesjegyre van kerekítve.

A **__version__** egy karakterlánc, amely a szoftver aktuális verziószámát tartalmazza. A verzió pontokkal elválasztott egész számokból épül fel (például: "0.1.1"), a verzióellenőrzés során a bal szélső szám a legjelentősebb, míg a jobb oldali szám a legkisebb jelentőségű (például "0.1.1"). kisebb, mint "1.0.0"). Hibafa mentési fájlok kompatibilitásának meghatározására van halsználva.

A **__MINIMUM_VERSION__** ugyanaz a szerepe, mint a **__version__**, azonban a hibafa mentési fájlok minimálisan elfogadott verziójának jelölésére szolgál.

A **__licence__** hordozza a Python-kód szerzői jogi megjegyzését. (Permisszív MIT)

6.4.3 Hiba Objektum Definíciók

```
class FaultTreeLoadError(Exception):
   """An issue has occurred during file loading."""

class FaultTreeError(Exception):
   """An issue has occoured during some operation of the Fault Tree."""

class ComputationOrderingError(Exception):
   """An issue has occurred during the calculation of computation order."""

class VersionError(Exception):
   """An issue has occurred during version checking."""

class FuzzyNumberError(Exception):
   """An issue has occoured during the running of fuzzy number class operations."""
```

Kódrészlet 15: Source Kód a hiba objektum definícióknak

A szoftverben definiált hibák (avagy kivételek) egyszerűen a Python alap **Exception** osztályának átnevezései, azért készültek, hogy hiba esetén a nevük leíróbb és könnyebben megfogható legyen, elválasztva a beépített kivételeket a maga szoftver által generált kivételektől. Példa a felhasználási esetre az érvénytelen fájl betöltésekor keletkező kivétel elkapása, és hibaüzenet megjelenítése a felhasználónak az egész alkalmazás leállítása nélkül. A kivételek elfogadnak egy opcionális karakterláncot is a fellépő hiba magyarázatára. A kivételeket jelezni lehet a **raise** kulcsszó használatával.

A **FaultTreeLoadError** akkor jelenik meg, ha a hibafa fájlból való betültésekor hiba keletkezett.

A **FaultTreeError** akkor jelenik meg, ha a hibafa valamely művelete valamilyen okból hibába ütközik.

A **ComputationOrderingError** akkor jelenik meg, ha egy **FuzzyFaultTree** objektum megpróbálja kiszámítani a logikai kapuk kiszámításának sorrendjét a bemeneti események alapján, de ez nem sikerül belátható időn belül.

A **VersionError** akkor jelenik meg, ha a verzióellenőrzés sikertelen a hibafa fájlból történő betöltésekor.

A **FuzzyNumberError** akkor jelenik meg, ha hiba történik egy fuzzy szám osztály metódusával.

6.4.4 ProbabilityTools osztály

```
class ProbabilityTools:

    @staticmethod
    def logical_and(probabilities: list[float]):
        return round(math.prod(probabilities), __PRECISION__)

    @staticmethod
    def logical_or(probabilities: list[float]):
        return round(1 - math.prod([1-p for p in probabilities]), __PRECISION__)

    @staticmethod
    def logical_not(probability):
        return round(1 - probability, __PRECISION__)
```

Kódrészlet 16: Source kód a ProbabilityTools osztályának, dokumentáció és hibakeresés nélkül

Ez egy konténerosztály olyan függvények tárolására, amelyek egyetlen logikai műveletet hajtanak végre valószínűségi értékek listáján, amelyek lebegő típusok (törtszámok) a 0 és 1 intervallumban. A felhasznált használt képleteket a (4.3. fejezet) ismerteti.

A használt **math.product()** beépített függvény egy számlistát vesz mint bemenet vesz fel, és minden elemet összeszorozva ad egy számot kimenetként.

Az **[1-p for p in probabilities]** kód egy listaértelmezés (list comprehension), amely egy új listát épít fel úgy, hogy végrehajtja az ' $1-p$ ' műveletet a megadott valószínűségi értékek listájának minden elemén.

6.4.5 AlphaLevelInterval osztály

```
class AlphaLevelInterval:
    def __init__(self, lower: float, upper: float, alphaLevel: float):
        self.lower = lower
        self.upper = upper
        self.alphaLevel = alphaLevel

    def __repr__(self):
        return f"""AlphaLevelInterval(lower = {self.lower}, upper = {self.upper}, alphaLevel =
{self.alphaLevel})"""

    def to_list(self):
        return [self.lower, self.upper, self.alphaLevel]

    @staticmethod
    def logical_and(terms: list):
        lower = ProbabilityTools.logical_and([t.lower for t in terms])
        upper = ProbabilityTools.logical_and([t.upper for t in terms])
        return AlphaLevelInterval(lower, upper, terms[0].alphaLevel)

    @staticmethod
    def logical_or(terms: list):
        lower = ProbabilityTools.logical_or([t.lower for t in terms])
        upper = ProbabilityTools.logical_or([t.upper for t in terms])
        return AlphaLevelInterval(lower, upper, terms[0].alphaLevel)
```

Kódrészlet 17: Source kód az AlphaLevelInterval osztályának, kommentek és hibakeresés nélkül

Ez elsősorban egy adattároló osztály a fuzzy számok alfa-vágásainak eredményeihez, és mely metódusokat is biztosít az azonos osztályba tartozó objektumok pontonkénti logikai műveleteinek (és, vagy) elvégzéséhez, a (4.3. fejezet) képleteinek felhasználásával.

6.4.6 TrapezoidalFuzzyNumber osztály

Ez elsősorban egy konténerosztály, amely a trapéz alakú fuzzy számo kulcspontjait tárolja, ahogyan azt a (3.2.2 fejezetben) tárgyaltuk.

Constructor methods and Flags

```
class TrapezoidalFuzzyNumber:
    _CRISP_STRATEGY: str = None
    _IMPLEMENTED_CRISP_STRATEGIES = ["interval-x2x3", "fully-available",
"fully-unavailable"]

    def __init__(self, x1: float, x2: float, x3: float, x4: float):
        self.x1 = float(x1)
        self.x2 = float(x2)
        self.x3 = float(x3)
        self.x4 = float(x4)

    @staticmethod
    def from_list(terms: list):
        return TrapezoidalFuzzyNumber(terms[0], terms[1], terms[2], terms[3])

    @staticmethod
    def from_list_triangular_with_errorfactor(terms: list):
        x1 = round(terms[0] / terms[1], __PRECISION__)
        x2 = x3 = terms[0]
        x4 = round(terms[0] * terms[1], __PRECISION__)
        return TrapezoidalFuzzyNumber(x1, x2, x3, x4)

    @staticmethod
    def from_list_triangular(terms: list):
        return TrapezoidalFuzzyNumber(terms[0], terms[1], terms[1], terms[2])
```

Kódrészlet 18: Source kód a TrapezoidalFuzzyNumber osztálynak, komment és hiba keresés nélkül

Az osztály több konstruktor metódust ad. A **from_list** létrehozza a fuzzy számot a kulcspontjainak listájából. A **from_list_triangular** metódus egy háromszög alakú fuzzy szám kulcspontjaiból állítja össze az objektumot. A **from_list_triangular_errorfactor** metódus a [7][13]-ben használt pont-mediánérték és hibatényező számokból állítja össze az objektumot.

A **_CRISP_STRATEGY** osztályváltozó, ha másra van állítva, mint a **None** , akkor defuzzifikációs stratégiát használ az objektum, hogy tiszta kimenetet hozzon létre mind az alfa-vágásokhoz, mind a logikai műveletekhez. A megvalósított stratégiák listája a **_IMPLEMENTED_CRISP_STRATEGIES** osztályváltozóban található, és a neveinek egyedieknek kell lenniük.

Defuzzifikációs Stratégiák

```
def apply_crisp_strategy(self):
    if self._CRISP_STRATEGY is None:
        return self

    elif self._CRISP_STRATEGY == "fully-available":
        return TrapezoidalFuzzyNumber(0, 0, 0, 0)

    elif self._CRISP_STRATEGY == "fully-unavailable":
        return TrapezoidalFuzzyNumber(1, 1, 1, 1)

    elif self._CRISP_STRATEGY == "interval-x2x3":
        return TrapezoidalFuzzyNumber(self.x2, self.x2, self.x3, self.x3)

def set_crispness_to_fully_avalable(self):
    self._CRISP_STRATEGY = "fully-available"

def set_crispness_to_fully_unavalable(self):
    self._CRISP_STRATEGY = "fully-unavailable"

def set_crispness_to_interval_x2x3(self):
    self._CRISP_STRATEGY = "interval-x2x3"

def reset_crispness(self):
    self._CRISP_STRATEGY = None
```

Kódrészlet 19: Source kód a TrapezoidalFuzzyNumber osztály defuzzifikáció metódusai, kommentek és hibakeresés nélkül

Az elemzés során defuzzifikációra van szükség a fuzzy fontosság és a fuzzy bizonytalansági fontosság mérőszámainak meghatározásához, mivel a stratégia alkalmazása után az osztály új példányát adja vissza, triviális új stratégiákat ebbe a módszerbe integrálni. A következő stratégiákat valósítottuk meg, amelyektől a FIM és FUIM elemzés megvalósítása függött (a 4.4 fejezetből):

fully-available pontonkénti számot hoz létre 0 valószínűséggel, FIM számítás ' $q_i = 0$ '

fully-unavailable egy pontonkénti számot hoz létre 1 valószínűséggel, FIM számítás ' $q_i = 1$ '

interval-x2x3 négyzet alakú fuzzy számot hoz létre, amely a FUIM kiszámításához szükséges

További stratégiatípusok hozzáadását egy új **elif** ág hajtja végre, amely magát a kódot valósítja meg, majd az új egyedi nevet a stratégiának hozzáadjuk a **_IMPLEMENTED_CRISP_STRATEGY** osztályváltozóhoz listabejegyzésként.

Kényelmi módszerek is rendelkezésre állnak a defuzzification stratégia módosításához és visszaállításához, hogy olvashatóbbá tegyék a használatát, és elkerüljék a hibákat a **_CRISP_STRATEGY** változó közvetlen módosítása során.

Logikai műveletek

```
@staticmethod
def logical_and(terms: list):
    terms = [t.apply_crisp_strategy() for t in terms]

    x1 = ProbabilityTools.logical_and([t.x1 for t in terms])
    x2 = ProbabilityTools.logical_and([t.x2 for t in terms])
    x3 = ProbabilityTools.logical_and([t.x3 for t in terms])
    x4 = ProbabilityTools.logical_and([t.x4 for t in terms])

    return TrapezoidalFuzzyNumber(x1, x2, x3, x4)

@staticmethod
def logical_or(terms: list):
    terms = [t.apply_crisp_strategy() for t in terms]

    x1 = ProbabilityTools.logical_or([t.x1 for t in terms])
    x2 = ProbabilityTools.logical_or([t.x2 for t in terms])
    x3 = ProbabilityTools.logical_or([t.x3 for t in terms])
    x4 = ProbabilityTools.logical_or([t.x4 for t in terms])

    return TrapezoidalFuzzyNumber(x1, x2, x3, x4)
```

Kódrészlet 20: Source kód a TrapezoidalFuzzyNumber osztály logikai műveleteihez, kommentek és hibakeresés nélkül

A TrapezoidalFuzzyNumber objektumokon végzett logikai **ÉS** és **VAGY** műveletek előkészítéséhez két kényelmi módszer áll rendelkezésre, amelyek a fuzzy számok kulcspontjain pontonként hajthatódnak végre. ezek végrehajtása előtt minden egyes rendelkezésre bocsátott **TrapezoidalFuzzyNumber** objektumot a megfelelő defuzzifikációs módszerrel dolgozunk fel, az **apply_crisp_trategy** metódussal, ha egy stratégia volt definiálva.

Alfa-vágás metódus

```
def alphacut(self, alphaLevel: float):
    if not self._CRISP_STRATEGY is None:
        self = self.apply_crisp_strategy()

    lower = round(self.x1 + alphaLevel * (self.x2 - self.x1), __PRECISION__)
    upper = round(self.x4 - alphaLevel * (self.x4 - self.x3), __PRECISION__)

    return AlphaLevelInterval(lower, upper, alphaLevel)
```

Kódrészlet 21: Source kód a TrapezoidalFuzzyNumber osztály alfa vágási műveletéhez, kommentek és hibakeresés nélkül

Az alfa-vágást úgy hajuk végre, hogy először a defuzzifikációs stratégiát alkalmazuk, ha definiálva van, majd a (4.2 fejezetben) vázolt módszerrel végrehajtjuk az alfa-vágást, majd az **AlphaLevelInterval** új példánya mint visszatérési érték tárolja az eredményt.

6.4.7 FuzzyFaultTree osztály

Inicializáció, Betoltés és Mentés

A mentés és a betöltés egy JSON-fájlba történik, amelynek szerkezete a (6.3 fejezetben) leírtak szerint történik.

A Python fájlműveletei úgy történnek, hogy közvetlenül megnyitunk egy fájlt az **open()** beépített funkcióval, amely két argumentumot vesz fel: a fájl elérési útját és a megnyitási módot, amely egy karakterlánc (alapértelmezett **"r"**). A fájlokat azonban manuálisan be kell zárni az olvasás vagy írás végén. Ennek érdekében a **with** kulcsszó segít létrehozni egy kontextuskezelőt, amely automatikusan bezárja a megnyitott fájlt, ha a kódblokk lefut vagy hibába ütközik, míg az **as** kulcsszót új változónév hozzárendeléséhez használható.

Például a következő két szintaxis egyenértékű ebben a használati esetben:

open(file) as f

f = open(file)

```
class FuzzyFaultTree:
    _MAX_SEARCH_DEPTH = 1000
    _METADATA_STRUCTURE = {"version": 'str',
                           "base-event-shape": ["trapezoidal", "triangular", "triangular-errorfactor"]
                           }
    _LOGIC_GATE_STRUCTURE = {"type": ["and", "or"],
                             "inputs": 'list'
                             }

    def __init__(self, baseEventShape: str = 'trapezoidal', topEventGateType: str = 'and'):
        self.metadata = {'version': __version__, 'base-event-shape': baseEventShape}
        self.baseEvents = {}
        self.logicGates = {'top-event': {'type': topEventGateType, 'inputs': []}}

    @staticmethod
    def load_from_file(loadPath : str):
        treeDict = None
        with open(loadPath) as f:
            treeDict = json.load(f)

        if not self.version_check(treeDict['metadata']['version']):
            raise FaultTreeLoadError()

        metadata = treeDict['metadata']
        baseEvents = treeDict['base-events']

        for name, event in baseEvents.items():
            if metadata['base-event-shape'] == "trapezoidal":
                baseEvents[name] = TrapezoidalFuzzyNumber.from_list(event)
            elif metadata['base-event-shape'] == "triangular-errorfactor":
                baseEvents[name] =
TrapezoidalFuzzyNumber.from_list_triangular_with_errorfactor(event)
            elif metadata['base-event-shape'] == "triangular":
                baseEvents[name] = TrapezoidalFuzzyNumber.from_list_triangular(event)

        logicGates = treeDict['logic-gates']

        fft = FuzzyFaultTree()
        fft.metadata = metadata
        fft.baseEvents = baseEvents
        fft.logicGates = logicGates

        return fft

    def save_to_file(self, savePath: str):
        md = self.metadata.copy()
        be = {n: self.baseEvents[n].to_list() for n in self.baseEvents.keys()}
        lg = self.logicGates.copy()

        with open(savePath, 'w') as f:
            json.dump({'metadata': md, 'base-events': be, 'logic-gates': lg}, f, indent=4)
```

Kódrészlet 22: Source kód a FuzzyFaultTree osztály konstruktor metódusai, kommentek és hibakeresés nélkül

Verzió Ellenőrzés

```
    @staticmethod
    def version_check(version: str):
        minVersionParts = __MINIMUM_VERSION__.split('.')
        thisVersionParts = version.split('.')
        maxVersionParts = __version__.split('.')

        versionNumberLength = max([len(vnum) for vnum in minVersionParts + maxVersionParts
+ thisVersionParts])

        minVersion = ''.join([part.ljust(versionNumberLength, '0')
for part in minVersionParts])
        thisVersion = ''.join([part.ljust(versionNumberLength, '0')
for part in thisVersionParts])
        maxVersion = ''.join([part.ljust(versionNumberLength, '0')
for part in maxVersionParts])

        return minVersion <= thisVersion <= maxVersion
```

Kódrészlet 23: Source kód a FuzzyFaultTree osztály konstruktor metódusai, kommentek és hibakeresés nélkül

A verzióellenőrzés célja az inkompatibilis verziók kiszűrése, ha a programkód megváltozott vagy ha új metódusokat adtunk hozzá a **__version__** modulszintű változót módosítani kell, hogy ezt a fejlesztést tükrözze. Néhány változtatás azonban inkompatibilitást válthat ki régebbi verziókkal szemben, például ha a bemeneti fájl szerkezete megváltozik, ebben az esetben a **__MINIMUM_VERSION__** frissítését igényeli.

A verzióellenőrzés az egyszerű karakterlánc-összehasonlítással történik. A verziókat először a pontokkal elválasztott alkotórészekre osztjuk, majd minden résznél megszámláljuk a benne lévő karakterek számát, amivel az egyes részeket a egyenlő hosszúságúra fel kell tölteni úgy, hogy nullákat adnuk hozzá a bal oldalához. Ez a lépés fontos az összehasonlítások megfelelő működéséhez.

A karakterlánc-összehasonlítások működése miatt a pythonban betűket és számokat is elfogad a verziókban.

Érdemes megjegyezni, hogy a **versionNumberLength** számítása során az összeadás (+) jelet használjuk a listák összefűzéséhez, Pythonban ez támogatott, és a művelet eredménye egy új lista lesz, amely tartalmazza az összeadott listák összes elemét. Ez hasonlóan működik, mintha minden egyes listából minden elemet egy üres listához adnánk.

Belső Számítások

```
   def _calculate_dependent_logic_gates(self, gateName: str):
      search = [i for i in self.logicGates[gateName]["inputs"] if i in self.logicGates]
      dependancies = search.copy()

      for _ in range(self._MAX_SEARCH_DEPTH):
         if search == []:
            return dependancies
         found = [i for i in self.logicGates[search[0]]["inputs"] if i in self.logicGates]

         dependancies.extend(found)
         search.extend(found)
         search.pop(0)
      else:
         raise FaultTreeError(f"Could not calculate dependencies of '{logicGate}' :
MAX_SEARCH_DEPTH reached without success.")

   def _calculate_computation_order(self):
      processed = list(self.baseEvents.keys())
      computeOrder = [list(self.baseEvents.keys())]
      allEventKeys = []
      allEventKeys.extend(self.baseEvents.keys())
      allEventKeys.extend(self.logicGates.keys())

      for _ in range(self._MAX_SEARCH_DEPTH):
         if len(processed) == len(allEventKeys):
            break

         currentComputeOrder = []
         currentProcessed = []

         for name, gate in self.logicGates.items():
            if name in processed:
               continue
            elif not all([i in allEventKeys for i in gate['inputs']]):
               raise ComputationOrderingError(f"Gate '{name}' has inputs that are neither another
gate or a base event")

            elif all([i in processed for i in gate['inputs']]):
               currentProcessed.append(name)
               currentComputeOrder.append(name)
               continue

         computeOrder.append(currentComputeOrder)
         processed.extend(currentProcessed)
      else:
         raise ComputationOrderError("Maximum iteration count reached.")

      return computeOrder
```

Kódrészlet 24: Source kód a FuzzyFaultTree osztály belső számítási metódusainak, kommentek és hibakeresés nélkül

Ezek a metódusok a **FuzzyFaultTree** osztály működéséhez elengedhetetlenek, és általános metódus amit kifele is elérhetővé teszünk. Azonban ez nem egy privát metódus így ez ha kell az osztályon kívül is elérhető ha szükséges.

A **_calculate_computation_order** metódus határozza meg a logikai kapu számítási műveleteinek sorrendjét. Használható annak kiszámítására is, hogy mely logikai kapuk számíthatók párhuzamosan egymással. Ez egy iteratív módszer, amely először hozzáadja az alapeseményeket egy listához, mivel azok nem függenek semmitől, majd egy folyamatos ciklusban, amely legfeljebb **_MAX_SEARCH_DEPTH** számú alkalommal fut le, hozzáadjuk ezen listához azokat az eseményeket, amelyek bemenetei már mind szerepelnek ebben a listában.

A **_calculate_dependent_logic_gates** metódus megkeresi az összes logikai kaput amelyik függ egy bizonyos szülő logikai kaputól. Ezt használja az **API**-ban egy logikai kapu törlésekor, hogy eltávolítsa az összes többi logikai kaput is a függési láncban. Ez egy iteratív metódus, amely legfeljebb **_MAX_SEARCH_DEPTH** számú alkalommal fut le, mielőtt hibát jelez.

Legfelső esemény Standard Közelítése

```
    def calculate_top_event(self, useMinCutSets: bool = False):
        if useMinCutSets:
            self = self.calculate_minimum_cut_sets(newTree=True)

        computeOrder = self._calculate_computation_order()
        computed = {}

        for depth, layerEvents in enumerate(computeOrder):
            for event in layerEvents:
                if depth == 0:
                    computed[event] = self.baseEvents[event]
                elif self.logicGates[event]['type'] == "and":
                    computed[event] = TrapezoidalFuzzyNumber.logical_and([computed[e]
for e in self.logicGates[event]['inputs']])
                elif self.logicGates[event]['type'] == "or":
                    computed[event] = TrapezoidalFuzzyNumber.logical_or([computed[e]
for e in self.logicGates[event]['inputs']])
                else:
                    raise Exception("logic gate neither 'and' nor 'or' ")

        return computed['top-event']
```

Kódrészlet 25: Source kód a FuzzyFaultTree osztály legfelső esemény approximáció metódusa, kommentek és hibakeresés nélkül

A legfelső esemény fuzzy szám kulcspontonként kerül kiszámításra a (4.3 fejezetben) leírtak szerint

A metódus általában a hibafán operál, de a **useMinCutSets** argumentum, ha igazra van állítva, kiszámítja a hibafa minimális vágási halmazait és létrehoz egy új FuzzyFaultTree objektumot amely ezeket tartalmazza mint logikai kapuk, majd lecseréli a saját referenciáját erre az új fára.

A metódusnak először ki kell számítania a logikai kapuk kiszámítási sorrendjét, és ezt a **_calculate_computation_order** metódussal teszi meg, amelynek eredményeit azután egy beágyazott hurokban használja fel az egyes logikai kapukimenetek kiszámításához, amelyeket a egy szótárban tárol.

Legfelső esemény alfa-vágás számítása

```
    def calculate_top_event_alphacut(self, alphaLevel: float, useMinCutSets: bool = False):
        if useMinCutSets:
            self = self.calculate_minimum_cut_sets(newTree=True)

        computeOrder = self._calculate_computation_order()
        computedCuts = {}

        for layer in computeOrder:
            for event in layer:
                if event in self.baseEvents:
                    computedCuts[event] = self.baseEvents[event].alphacut(alphaLevel)
                else:
                    gate = self.logicGates[event]
                    if gate['type'] == "and":
                        computedCuts[event] = AlphaLevelInterval.logical_and([computedCuts[k]
for k in gate['inputs']])
                    elif gate['type'] == "or":
                        computedCuts[event] = AlphaLevelInterval.logical_or([computedCuts[k]
for k in gate['inputs']])
                    else:
                        raise Exception("gate type is neither 'and' , 'or' ")

        return computedCuts['top-event']
```

Kódrészlet 26: Source kód a FuzzyFaultTree osztály legfelső esemény alfa-vágásai metódusa, kommentek és hibakeresés nélkül

A legfelső esemény kiszámításának alfa-vágás módszere ugyanúgy történik, mint a **calculate_top_event** módszer, azzal a fő különbséggel, hogy kiszámítja az alapesemény alfa-vágását, amelyeket aztán a logikai kapukkal összekapcsol és számít.

A **useMinCutSets** bemenet ha igazra van állítva a legfelső eseményt egy fuzzy hibafa segítségével számítja ki, amelynek logikai kapuit a minimális vágási halmazok helyettesítik, hasonlóan a **calculate_top_event** metódusában leírtakhoz.

Minimum Cut Sets Calculation

```
    def calculate_minimum_cut_sets(self, newTree: bool = False, analyzedGate='top-event',
_topCall = True):
        gateType = self.logicGates[analyzedGate]["type"]
        inputs = self.logicGates[analyzedGate]["inputs"]

        visited = []

        for i in inputs:
            if i in self.baseEvents:
                visited.append([i])
            else:
                visited.append(self.calculate_minimum_cut_sets(analyzedGate=i, _topCall = False))

        if gateType == "and":
            ret = list(itertools.product(*visited))
        elif gateType == "or":
            ret = list(itertools.chain(*visited))

        if not _topCall:
            answer = ret
        else:
            def flatten(L, outerLayer = False):
                if not (isinstance(L, list) or isinstance(L, tuple)):
                    return [L]
                else:
                    if outerLayer:
                        return list(itertools.chain([flatten(x) for x in L]))
                    else:
                        return list(itertools.chain(*[flatten(x) for x in L]))

            flt = flatten(ret, outerLayer=True)

            if not newTree:
                answer = flt
            else:
                ft = FuzzyFaultTree(self.metadata["base-event-shape"], "or")
                ft.baseEvents = self.baseEvents.copy()
                ft.metadata = self.metadata.copy()

                for i, inputs in enumerate(flt):
                    ft.add_logic_gate('top-event', f'MCS-{i+1}', 'and', inputs)

                answer = ft

        return answer
```

Kódrészlet 27: Source kód a FuzzyFaultTree osztály konstruktor metódusai, kommentek és hibakeresés nélkül

A minimális vágáskészleteket a (2.1. és a 3.1.2 fejezetek) tárgyalják. Ez a módszer rekurzív mélység-első gráfjárási algoritmust használ ezek kiszámításához. A **_topCall** bemenet kifejezetten a külső hívás és a rekurzív hívás elkülönítésére szolgál, ezért erősen ajánlott, hogy ne használja semmi más.

Először is, a módszer a fuzzy hibafán járja át a mélységig először, a legfelső eseménytől kezdve, majd a logikai kapukon keresztül halad lefele a hibafán. Ha találkozik egy logikai kapuval, akkor rekurzívan újra hívja magát, amíg nem találkozik egy olyan logikai kapuval, amelynek az összes bemenetei alapesemények. Ezután a logikai kapu típusától függően két dolog egyikét hajtja végre.

A **VAGY** művelet a legegyszerűbb, egyetlen listába láncolja a bemeneteket, mivel bármelyik előfordulása okozza a közbenső eseményt (a logikai kapu kimenetét). Például a logikai kapunak **D, F, K, J** alapeseményei vannak, majd a kapun áthaladva először létrehoz egy listát a következőképpen:

[[D], [F], [K, J]]

Ezután kicsomagolja a listát (a ***variable** szintaxissal) az **itertools.chain** függvény bemenetébe külon-külon, amelyeket végül egyetlen listába fűzi mint: **[D, F, [K, J]]**

A Python kicsomagolás viselkedésének szemléltetésére a következő két kódrészlet illusztrálja amelyek funkcionálisan egyenértékűek:

function(*[[A], [B, C]])

function([A], [B, C])

Az **ÉS** művelet azonban más megközelítést igényel. Itt az **itertools.product** beépített függvényt használjuk, amely megadja az argumentumként átadott összes iterálható descartes-szorzatát. A fenti példával a **[[D], [F], [K, J]]** listák listáját véve a descartes-szorzatai ebben az esetben: **[(D, F, K), (D, F, J)]**.

Ha ez megtörtént, az eredmény egy mélyen beágyazott lista lesz, például:

[A, [B, [C, D] , [E, [F]], G], H]

Ebből az n-dimenziós mélyen beágyazott listából 2 dimenziós listát kell alkotni, lényegében lapítani kell. Ezt a **flatten_list** rekurzív függvénnyel valósítja meg a szükséges mélységig, jelen esetben 2. Ennek a művelet eredménye a következő:

[A, [B, C, D, E, F, G], H]

Figyelembe kell venni, hogy a **flatten_list** függvény egy másik függvényen belül van definiálva. Ez a Pythonban megengedett, és csekély mértékben lassítja ap rogram működését.

Végül a **newTree** argumentum megváltoztatja a visszatérési formátumot az alapeseménynevek listájáról egy új fuzzy hibafára, amely ugyanazokkal az adatokkal rendelkezik mint az eredeti, de a logikai kapuit a minimális vágási halmazok helyettesítik az **SOP** (szorzatösszeg) reprezentációjában.

Fuzzy Fontossági Érték

```
    def calculate_fim(self, alphaLevels=[0.0], useMinCutSets: bool = False):
        if useMinCutSets:
            self = self.calculate_minimum_cut_sets(newTree=True)

        fim = {}

        for eventName in self.baseEvents:
            currentFim = 0
            for level in alphaLevels:
                qi = []

                self.baseEvents[eventName].set_crispness_to_fully_avalable()
                qi.append(self.calculate_top_event_alphacut(level))

                self.baseEvents[eventName].set_crispness_to_fully_unavalable()
                qi.append(self.calculate_top_event_alphacut(level))

                self.baseEvents[eventName].reset_crispness()

                currentFim += round(math.sqrt((qi[0].lower - qi[1].lower)**2 +
(qi[0].upper - qi[1].upper)**2), __PRECISION__)

            fim[eventName] = round(currentFim, __PRECISION__)

        fimRanks = {n: {"rank": None, "value": v} for n, v in fim.items()}

        lastNumber = None
        offset = 1
        for rank, name in enumerate([k for k, _ in sorted(fim.items(), key=lambda item: item[1],
reverse=True)]):
            if fimRanks[name]["value"] == lastNumber: offset -= 1
            fimRanks[name]["rank"] = rank + offset
            lastNumber = fimRanks[name]["value"]

        return fimRanks
```

Kódrészlet 28: Source kód a FuzzyFaultTree osztály FIM számítási metódusa, kommentek és hibakeresés nélkül

Ez a módszer a (4.4 fejezetben) tárgyalt Fuzzy Fontossági Mérték (FIM) számítását valósítja meg.

A **useMinCutSets** ha Igazra van állítva egy fuzzy hibafát fog használni, amelynek logikai kapuit a minimális vágáshalmazok helyettesítik a számítás során, hasonlóan a **calculate_top_event** metódusban leírtakhoz.

A metódus tartalmaz egy beágyazott ciklust, amely minden esemény FIM-számát úgy számítja ki, hogy kiszámítja a legfelső esemény alfa-vágását, amikor az alapesemény teljesen elérhetőre van beállítva, majd teljesen elérhetetlen. Ezután kiszámítja a két

alfa-vágás közötti euklideszi távolságot, és összegzik a reszeredményeket minden alfa-szint esetében.

Miután minden alapeseménynél kiszámoltuk a FIM-et, azt rangsorolni kell. A rangsorolás a FIM-számok csökkenő sorrendbe rendezésével történik, de mivel az eredményeket szótárban tárolják, ezt nehéz elvégezni.

A " **fimRanks = {n: {"rank": None, "value": v} for n, v in fim.items()}** " kód egy listaértelmezés (list comprehension), amellyel új szótár hozható létre, amely nem csak az értékeket, hanem az egyes alapesemények rangsorolását is tartalmazza.

A szótár bejegyzéseinek rendezéséhez a beépített **sorted** metódust használjuk, amely bemenetként egy iterálható objektumot fogad és rendezi azt, opcionálisan ez a függvény bemenetként egy másik függvényt is fogad, amellyel a **key** kulcs-szó argumentumba ami alapján a rendezést elvégezheti. Jelen esetben egy lambda függvény " **lambda item: item[1]** ". Ennek a megközelítésnek a használatához a rendezett listára van szükség amit a **sorted()** beépíttet funkcióval lehetséges, a kulcs-érték párok szótárból listába történő átrendezéséhez pedig a **dictionary.items()** metódus használható. Annak érdekében, hogy a kapott lista csökkenő sorrendben jelenjen meg, a **reversed** kulcsszó argumentumot aIigaz értékre kell állítani.

Végül végigléptetünk ezen a rendezett listát, figyelmen kívül hagyva az értéket, és csak a nevet veszünk egy listaértelmezés segítségével, amely névlistát hoz létre a kulcs-érték párokból. A rangok megszerzéséhez a beépített **enumerate** metódus a lista elemeinek indexét adja vissza a listaelem mellett, ami lényegében a rang.

A **lastNumber** és az **offset** változók segítenek ha a a rangsorolási ciklusban kettő vagy több azonos szám jelenik meg, akkor ugyanazt a rangot rendeli hozzá minden egyenlő számhoz. Ez nagymértékben függ a **__PRECISION__** attribútumtól, ellentétben a **TREEZZY2** kóddal, amely úgy tűnik, hogy a saját numerikus pontosságán működik, és csak a kijelzett értékeket kerekíti.

Fuzzy Bizonytalansági Fontosság Mértéke

```
    def calculate_fuim(self, alphaLevels=[0.0], defuzzingMethod='interval-x2x3', useMinCutSets:
bool = False):
        if useMinCutSets:
            self = self.calculate_minimum_cut_sets(newTree=True)

        fuim = {}
        q = [self.calculate_top_event_alphacut(level) for level in alphaLevels]

        for eventName in self.baseEvents:
            self.baseEvents[eventName]._CRISP_STRATEGY = defuzzingMethod
            qi = [self.calculate_top_event_alphacut(level) for level in alphaLevels]
            self.baseEvents[eventName].reset_crispness()

            euclideanDistances = [math.sqrt((q[x].lower - qi[x].lower)**2 +
(q[x].upper - qi[x].upper)**2) for x in range(len(alphaLevels))]
            fuim[eventName] = round(sum(euclideanDistances), __PRECISION__)

        fuimRanks = {n: {"rank": None, "value": v} for n, v in fuim.items()}

        lastNumber = None
        offset = 1
        for rank, name in enumerate([k for k, _ in sorted(fuim.items(), key=lambda item: item[1],
reverse=True)]):
            if fuimRanks[name]["value"] == lastNumber: offset -= 1
            fuimRanks[name]["rank"] = rank + offset
            lastNumber = fuimRanks[name]["value"]

        return fuimRanks
```

Kódrészlet 29: Source kód a FuzzyFaultTree osztály FUIM számítási metódusa, kommentek és hibakeresés nélkül

Ez a módszer a (4.4 fejezetben) tárgyalt Fuzzy Bizonytalansági Fontosság Mérték (FIUM) számítást valósítja meg.

Ez a metódus funkcionálisan megegyezik a **calculate_fim** metódussal, azzal a különbséggel, hogy a legfelső esemény alfa-vágása először a normál módon kerül kiszámításra, majd minden alapesemény defuzzifikálásra kerül egy bizonyos, a **defuzzingMethod** argumentummal átadott stratégiával.

Vegye figyelembe, hogy mivel egyetlen felhasznált irodalom sem említette, hogy milyen stratégiát használnak a defuzzifikáláshoz, hogy megkapják a FUIM kiszámításához használt pont-szerű értéket, ez opcionális argumentum a későbbi fejlesztéshez ezen a területen.

Hibafa manipulációs API

```
def api_add_logic_gate(self, parentGate: str, name: str, gateType: str, inputs: list = []):
    if not name in self.logicGates[parentGate]["inputs"]:
        self.logicGates[parentGate]["inputs"].append(name)

    self.logicGates[name] = {"type": gateType, "inputs": inputs}

def api_remove_logic_gate(self, gateName: str):
    for gate in self.logicGates:
        if gateName in self.logicGates[gate]["inputs"]:
            self.logicGates[gate]["inputs"].remove(gateName)

    dependancies = [gateName]
    dependancies.extend(self._calculate_dependent_logic_gates(gateName))
    for d in dependancies:
        del self.logicGates[d]

def api_add_base_event(self, name: str, fuzzyNumber):
    self.baseEvents[name] = fuzzyNumber

def api_remove_base_event(self, name: str):
    for gate in self.logicGates:
        try:
            self.logicGates[gate]["inputs"].remove(name)
        except ValueError:
            pass

    del self.baseEvents[name]

def api_assign_base_event_to_gate_input(self, eventName: str, gateName: str):
    if not eventName in self.logicGates[gateName]["inputs"]:
        self.logicGates[gateName]["inputs"].append(eventName)

def api_remove_input_from_gate(self, inputName: str, gateName: str):
    if inputName in self.baseEvents:
        try:
            self.logicGates[gateName]["inputs"].remove(inputName)
        except ValueError:
            pass

    elif inputName in self.logicGates:
        self.api_remove_logic_gate(inputName)
```

Kódrészlet 30: Source kód a FuzzyFaultTree osztály API metódusainak, kommentek és hibakeresés nélkül

Ezek a módszerek egy API-t (alkalmazás programozási interfész) valósítanak meg a fuzzy hibafával való interakcióhoz, főként az alapesemények és logikai kapuk hozzáadásához és kezeléséhez. Ez segíti a beágyazott alkalmazásokat. Minden metódus **api_**-val kezdődik, hogy megkülönböztesse őket a többi metódustól.

Az **api_add_logic_gate** egyszerűen hozzáad egy logikai kaput a hibafához, amihez szükséges argumentum a **name** ami a logikai kapu neve és a **parentGate**, ami annak a logikai kapunak a neve, amelyhez bemenetként hozzáadódik, opcionálisan a bemenetei adhatók meg az **inputs** kulcsszó argumentummal. Az új logikai kapu nevének egyedinek kell lennie a fában (mind az alapeseményekben, mind a logikai kapukban).

Az **api_remove_logic_gate** metódus eltávolítja a megadott logikai kaput a **name** nevvel az összes bemenetről, majd megkeresi a hibafában az összes olyan logikai kaput, amelytől függ, és azokat is törli, hogy a hibafa logikai kapu része teljesen kapcsolatban maradjon.

A **api_add_base_event** egyszerűen egy új alapeseményt ad a hibafába anélkül hogy bármilyen logikai kapuhoz hozzárendelné.

A **api_remove_base_event** kitöröl egy alapeseményt a hibafából, és mindeggyik logikai kapu bemenetéről is eltávolítja.

A **api_assign_base_event_to** hozzáad egy meglévő alapeseményt egy logikai kapu bemenetéhez.

A **api_remove_input_from_gate** eltávolít egy bemenetet a megadott nevü logikai kapuról. Abban az esetben, ha az eltávolítani kívánt bemenet egy másik logikai kapu a **api_remove_logic_gate** methódust hívja rajta hogy a hibafából eltávolítsa azt véglegesen.

6.5 Konzolos Interfész

A szoftver önálló működéséhez a kifejlesztett interfész parancssori (vagy terminál) alapú. A szoftverforrásfájl futtatásához szükséges Python 3.6+ és opcionálisan a **matplotlib** csomag, ha a vizualizációt használják. A program a következőképpen hívható meg a konzolról:

python fuzzyftapy.py <OPTIONS>

Ahol az **<OPTION>** a konzolos opciókkal van helyettesítve..

A következő konzolos opciók érhetőek el ezen dolgozat irásakor:

-h vagy **--help** : Az angol nyelgű program segítség kinyomtatása konzolra, amiben le vannak irva az összes opciók és hozzájuk tartozó súgó.

-i vagy **--input-file** : megadja a bemeneti JSON fájl helyét (nem használható a **-gui** opcióval). Alkalmazására példa:

python fuzzyftapy.py -i "C:/the folder/input.json"

-gui : A grafikus kezelőfelületet elindítja. (NINCS IMPLEMENTÁLVA)

-l vagy **--alpha-levels** : A számítások alfa-szintjeit megadja, nem használható a **-nl** optióval. Példa a használatára:

python fuzzyftapy.py -i "C:/the folder/input.json" -l 0.0 0.2 0.4 0.8 1.0

-nl vagy **--number-of-alpha-levels** : A számításokban használt alfa-szinteket adja meg úgy hogy a 0 és 1 es intervallumot a megadott számú eggyenlő részre ossza fel (minimum 2), nem használható a **-l** opcióval. Alkalmazására példa:

python fuzzyftapy.py -i "C:/the folder/input.json" -nl 7

-o vagy **--output-file** : A számításik eredményeket JSON formában a megadott fájlba írja (csak akkor ha a fájl nem létezik), ennek hiányában a számítási eredmények a konzolra lesznek kinyomtatva. Példa alkalmazására:

python fuzzyftapy.py -i "C:/the folder/input.json" -o "C:/the folder/output.json"

-teap vagy **--calc-top-approx** : Legfelső esemény standard approximáció számítása. Példa alkalmazására:

python fuzzyftapy.py -i "C:/the folder/input.json" -teap

-tecut vagy **--calc-top-cuts** : Legfelső esemény alfa-vágás számítása a megadott alfa-szinteken. Példa alkalmazására:

python fuzzyftapy.py -i "C:/the folder/input.json" -tecut

-mcs vagy **--minimum-cut-sets** : A minimális vágási halmazok kiszámítása (de nem használása a számításokban). Példa alkalmazására:

python fuzzyftapy.py -i "C:/the folder/input.json" -mcs

-umcs vagy **--use-minimum-cut-sets** : A minimális vágási halmazok kiszámítása és használása az összes többi számítás elvégzésére. Példa alkalmazására:

python fuzzyftapy.py -i "C:/the folder/input.json" -umcs

-im vagy **--calc-importance** : A FIM és FUIM mutatók és rangsorok számítása az adott alfa szinteken. Példa alkalmazására:

python fuzzyftapy.py -i "C:/the folder/input.json" -im

-p vagy **--precision** : Tizedesjegy pontosság beállítása. Példa alkalmazására:

python fuzzyftapy.py -i "C:/the folder/input.json" -p 6

-v vagy **--visualize** : Vizualizálja a legfelső eseményt illetve a FIM és FUIM rankokat grafikonokon. Példa alkalmazására:

python fuzzyftapy.py -i "C:/the folder/input.json" -teap -im -v

-t vagy **--time-execution** : A program egyes számításainak lefutási idejének becslése. Példa alkalmazására:

python fuzzyftapy.py -i "C:/the folder/input.json" -mcs -t

--licence : Licence konzolra kinyomtatása.

python fuzzyftapy.py --licence

7 TESZTELÉS ÉS ÖSSZEHASONLÍTÁS

7.1 A tesztelés és az összehasonlítások módszertana

A kifejlesztett program teszteléséhez a **TREEZZY2** számítógépes programot használjuk és az összes tesztelt hibafát reprodukáljuk mind a kifejlesztett programban, mind a **TREEZZY2** számítógépes szoftverben.

A bemeneti esemény fuzzy valószínûségeit táblázatban adjuk meg, és az eredményül kapott fuzzy fuzzy számot, a FIM-et és a FUIM-et szintén táblázatban hasonlítjuk össze, és szemléltetjük a fa szerkezetét. Minden számítás 6 számjegy pontossággal történik, ahol csak lehetséges, 10 számjegy pontossággal, ha szükséges a kerekítéssel kapcsolatos különbségek elkerülése érdekében.

Az összehasonlítás főbb pontjai a következők:

- Legfelsőbb esemény standard közelítés
- Fuzzy Fontossági Mérőszám (FIM) és rangsorolása
- Fuzzy Bizonytalansági Fontosság Mérőszám (FIM) és rangsorolása
- Legfelsőbb esemény Alfa-vágás megközelítés (számításokhoz az alfaszintek [0, 1])
- Minimális vágási halmazok

Vegye figyelembe, hogy az összes szám és eseményfa csak kitalált, hogy tesztelje és összehasonlítsa mindkét program eredményeit különböző bemeneti készletek mellett.

A kifejlesztett **fuzzyftapy** szoftver egyes eredményeihez használt pontos opciók, amelyek az eredmények összehasonlító táblázataiban is megjelennek, a következők:

fuzzyftapy: **python fuzzyftapy.py -i input.json -teap -tecut -im -p 6 -l 0 1**

fuzzyftapy with mcs:
python fuzzyftapy.py -i input.json -teap -tecut -im -umcs -p 6 -l 0 1

fuzzyftapy alpha = 0: **python fuzzyftapy.py -i input.json -teap -tecut -im -p 6 -l 0**

Speciális esetekben, amikor a számok túl kicsik, a decimális pontosság 10 számjegyre nő a **TREEZZY2**-ben és a fuzzyftapy-ban a **-p 10** jelzővel.

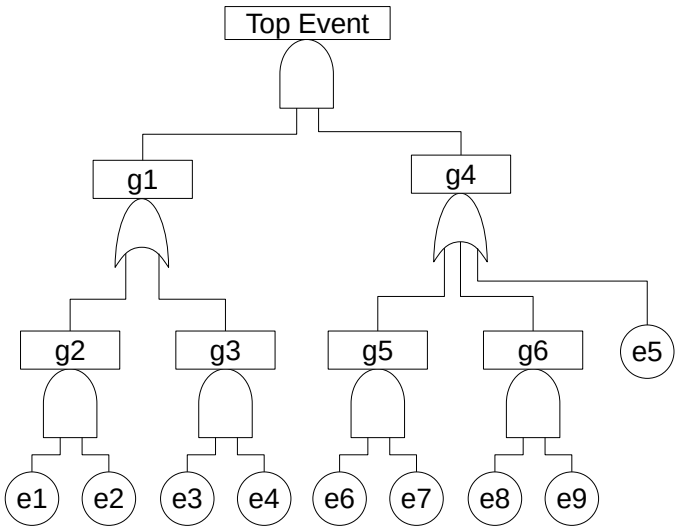
A legfelső esemény grafikonjának megrajzolása az **-nl 10** opcióval történik, hogy lecserélje a pontos alfa szintet (**-l 0 1**), és ugyanazt a rajzot kapja, mint a **TREEZZY2**-ben, amely alapértelmezés szerint 10 alfa szintet használ. .

7.2 Teszt 1

7.2.1 Bemeneti Adatok

Base Event Name	x1	x2	x3	x4
e1	0.15684	0.15998	0.161789	0.171034
e2	0.05684	0.08143	0.099785	0.120013
e3	0.31498	0.31649	0.319564	0.319719
e4	0.123456	0.135847	0.164891	0.235881
e5	0.005698	0.016234	0.121041	0.123052
e6	0.015893	0.019753	0.023495	0.032479
e7	0.002358	0.007985	0.123798	0.145287
e8	0.079521	0.099002	0.179811	0.498552
e9	0.225889	0.2522	0.314412	0.337008

Táblázat 2: Trapéz alapesemény kulcspontjai az 1. teszthez



Ábra 10: Hibafa rajza az 1. teszthez

7.2.2 Eredmény

Top Event Apprximate	x1	x2	x3	x4
TREEZZY2	0.001120	0.002271	0.011772	0.025848
fuzzyftapy	0.001120	0.002271	0.011771	0.025848
fuzzyftapy with mcs	0.001133	0.002314	0.012374	0.028110

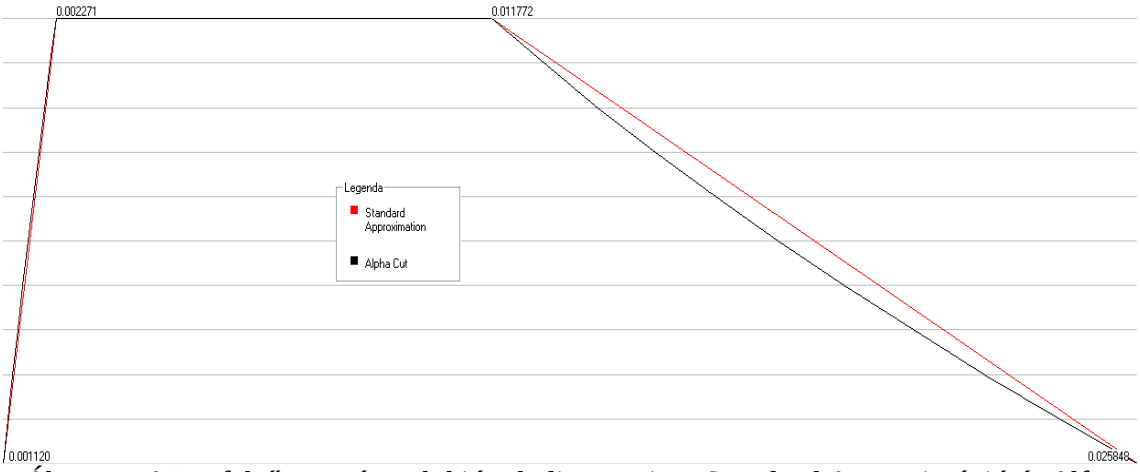
Táblázat 3: Trapéz legfelső esemény standard approximációja 1. teszthez

Event Name	FIM				FIM ranking			
	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0
e1	0.0344	0.047086	0.052506	0.030412	6	7	7	6
e2	0.048837	0.070715	0.078457	0.043449	4	4	4	4
e3	0.068129	0.091954	0.098197	0.063333	3	3	3	3
e4	0.091758	0.14199	0.150297	0.086069	2	2	2	2
e5	0.092834	0.174749	0.191715	0.091001	1	1	1	1
e6	0.013522	0.017001	0.02194	0.010008	8	8	8	8
e7	0.003028	0.004046	0.005064	0.002355	9	9	9	9
e8	0.031764	0.052983	0.059138	0.02974	7	6	6	7
e9	0.046867	0.053244	0.060459	0.041246	5	5	5	5

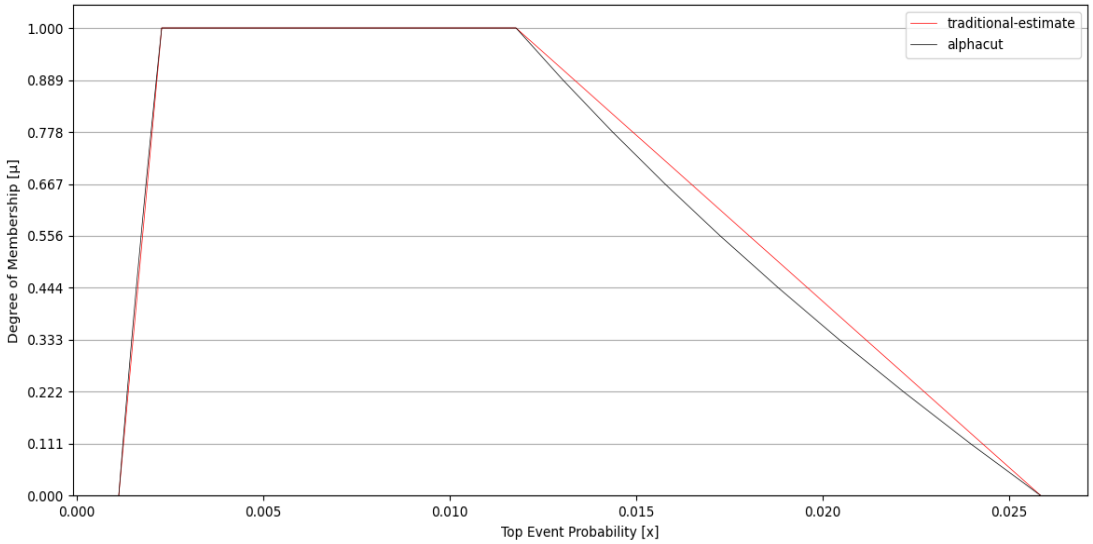
Táblázat 4: FIM számok és rangjai az alapeseményeknek 1. teszthez

Event Name	FUIM				FUIM ranking			
	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0
e1	0.00032	0.00028	0.000321	0.00028	6	6	6	6
e2	0.001483	0.000879	0.001003	0.000879	5	4	4	4
e3	0.00013	1.00E-05	1.10E-05	1.00E-05	9	9	9	9
e4	0.007144	0.006088	0.00661	0.006088	2	2	2	2
e5	0.005239	0.000515	0.000536	0.000515	3	5	5	5
e6	0.000137	8.90E-05	0.000123	8.90E-05	8	7	7	7
e7	0.000233	4.80E-05	6.50E-05	4.80E-05	7	8	8	8
e8	0.009999	0.008851	0.010145	0.008851	1	1	1	1
e9	0.002551	0.000933	0.001067	0.000933	4	3	3	3

Táblázat 5: FUIM számok és rangjai az alapeseményeknek 1. teszthez



Ábra 11: A Legfelső esemény alakjának diagramja a Standard Approximáció és Alfa-vágás módszerrel a TREEZY2 szoftverből az 1. teszthez



Ábra 12: A Legfelső esemény alakjának diagramja a Standard Approximáció és Alfa-vágás módszerrel a FUZZYFTAPY szoftverből az 1. teszthez

```
{
        “mcs-TREEZZY2”: [     ["e1", "e2", "e5"], ["e3", "e4", "e5"],
                              ["e1", "e2", "e6", "e7"], ["e1", "e2", "e8", "e9"],
                              ["e3", "e4", "e8", "e9"], ["e3", "e4", "e6", "e7"]
                        ]
        ”mcs-FUZZYFTAPY”: [ ["e1", "e2", "e5"], ["e3", "e4", "e5"],
                              ["e1", "e2", "e6", "e7"], ["e1", "e2", "e8", "e9"],
                              ["e3", "e4", "e8", "e9"], ["e3", "e4", "e6", "e7"]
                            ]
}
```

Kódrész 31: 1. teszt A FUZZYFTAPY és TREEZZY2 minimális vágási halmazai, a formátum (JSON) és a sorrend módosult az olvashatóság érdekében

7.2.3 Összehasonlítás

A (3. Táblán) látható szabványos közelítést használó Legfelső esemény kulcspontokat összehasonlítjuk a TREEZZY2 és a FUZZYFTAPY között, ami azt mutatja, hogy mindegyik pontosan ugyanaz, kivéve az **x2**-t, amely kerekítési hibának tűnik. A minimális vágáskészletek használata a FUZZYFTAPY legfelső eseményének kiszámításához összességében magasabb becsléseket eredményez minden kulcspontjához.

Mind a Fuzzy Importance, mind a Uncertainty Importance mérőszámok eltérő számszerű eredményeket adnak, amint az a (4. táblázatban és 5. táblázatban) látható, azonban a FIM-rangsor mindkét szoftver előrejelzésében hasonló volt, a az eredmény minden változatánál a FUZZYFTAPY-ban teszteltek esetén, csak az **e1** és az **e7** eseményeknél volt eltérés. A FUIM rangsorok azonban kisebb eltéréseket mutattak a két szoftver között, miközben a FUZZYFTAPY összes opciója ugyanazt az eredményt hozta.

A FIM-eredmények, amelyek különböző alfa-szinteket használnak a számításaik elvégzéséhez, egyértelmű különbséget mutatnak, mivel a nagyobb számú alfa-vágás magasabb összértéket eredményez. Ez a várható viselkedés, mivel a képlet minden alfa szinten összeadja a mértéket, de a TREEZZY2-nek nincs ilyen opciója, ami arra utalna, hogy más algoritmust használnak benne.

A (11. ábra és 12. ábra) az alfa-vágási módszer diagramjait mutatja a csúcsesemény számításánál. Mivel a TREEZZY2 nem tartalmaz ismert numerikus kimenetet, csak egy grafikont, ezért csak alakban hasonlítható össze, ami azt mutatja, hogy az alfa vágási közelítés hasonló inflexiót követ.

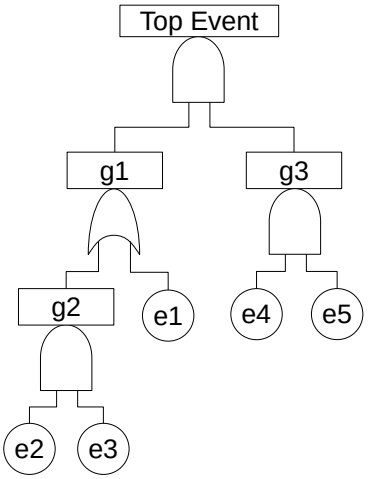
Végül a (31 Kódrészben) látható minimális vágáskészletek azt mutatják, hogy mindkét szoftver pontosan ugyanazt a minimális vágáskészletet számította ki, esetleg más sorrendben.

7.3 Teszt 2

7.3.1 Bemeneti Adatok

Base Event Name	x1	x2	x3	x4
e1	0.015531	0.025998	0.298410	0.310005
e2	0.110035	0.153320	0.154220	0.192235
e3	0.095513	0.155200	0.155200	0.160000
e4	0.200563	0.210056	0.219535	0.225753
e5	0.005698	0.016234	0.121041	0.123052

Táblázat 6: Trapéz alapesemény kulcspontjai az 2. teszthez



Ábra 13: Hibafa rajza a 2. teszthez

7.3.2 Eredmények

Top Event Approximate	x1	x2	x3	x4
TREEZZY2	0.000030	0.000168	0.008376	0.009201
fuzzyftapy	0.000030	0.000168	0.008376	0.009201
fuzzyftapy with mcs	0.000030	0.000170	0.008561	0.009459

Táblázat 7: Trapéz legfelső esemény standard approximációja 2. teszthez

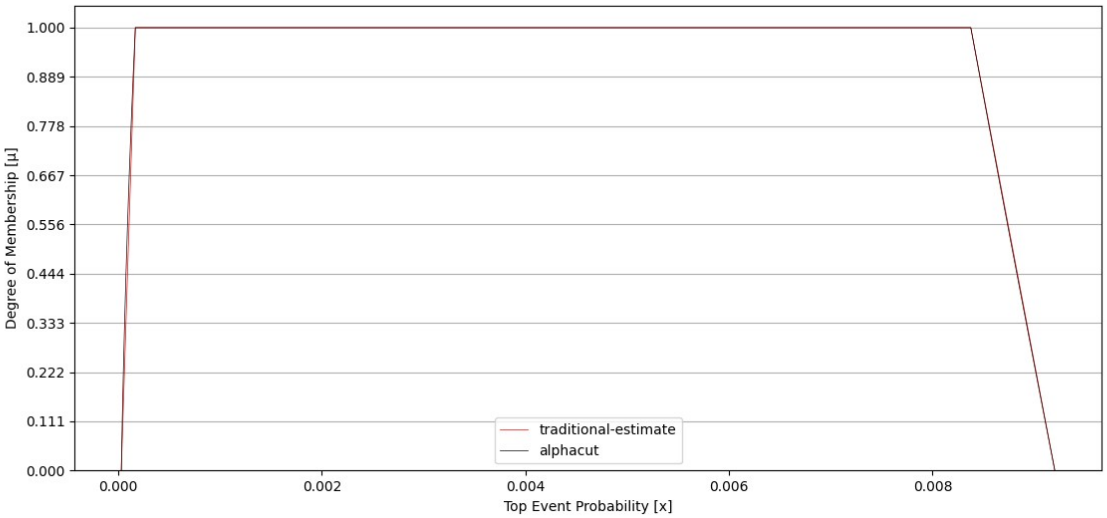
Event Name	FIM				FIM ranking			
	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0
e1	0.027756	0.053099	0.054553	0.026949	3	3	3	3
e2	0.004406	0.006006	0.008533	0.003068	5	5	5	5
e3	0.005294	0.006606	0.009394	0.003686	4	4	4	4
e4	0.041787	0.078918	0.080708	0.040758	2	2	2	2
e5	0.076442	0.144921	0.147811	0.074956	1	1	1	1

Táblázat 8: FIM számok és rangjai az alapeseményeknek 2. teszthez

Event Name	FUIM				FUIM ranking			
	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0
e1	0.004098	0.000312	0.000322	0.000312	2	1	1	1
e2	0.000173	0.000116	0.000168	0.000116	4	4	3	4
e3	0.000069	0.000019	0.000026	0.000019	5	5	5	5
e4	0.004790	0.000253	0.000260	0.000253	3	2	2	2
e5	0.004279	0.000159	0.000164	0.000159	1	3	4	3

Táblázat 9: FUIM számok és rangjai az alapeseményeknek 2. teszthez

Ábra 14: A Legfelső esemény alakjának diagramja a Standard Approximáció és Alfa-vágás módszerrel a TREEZY2 szoftverből az 2. teszthez



Ábra 15: A Legfelső esemény alakjának diagramja a Standard Approximáció és Alfa-vágás módszerrel a FUZZYFTAPY szoftverből az 2. teszthez

```
{
        “mcs-TREEZZY2”: [ ["e1", "e4", "e5"], ["e2", "e3", "e4", "e5"] ],
        ”mcs-FUZZYFTAPY”: [ ["e1", "e4", "e5"], ["e2", "e3", "e4", "e5"] ]
}
```

Kódrész 32: 2. teszt A FUZZYFTAPY és TREEZZY2 minimális vágási halmazai, a formátum (JSON) és a sorrend módosult az olvashatóság érdekében

7.3.3 Összehasonlítás

A (7. Táblázatban) bemutatott, tradicionális közelítést használó Top Event kulcspontok azt mutatják, hogy mind a TREEZZY2, mind a FUZZYFTAPY pontosan ugyanarra a következtetésre jut. Ennek a FUZZYFTAPY-ban történő kiszámításához a minimális vágási halmazokat használva összességében magasabb becslést kapunk a legfontosabb esemény minden kulcspontjához.

Mind a FIM, mind a FUIM eltérő számszerű eredményeket adnak, amint az a (8. Táblázatban és 9. Táblázatban) látható, azonban a FIM-rangsor pontosan úgy alakult, ahogy azt mindkét szoftver megjósolta, a paraméterek minden változatát a FUZZYFTAPY-ban tesztelték. A FUIM rangsorai azonban kisebb eltéréseket mutattak, nevezetesen az **e1** és **e5** alapesemények rangjait a legtöbb esetben felcserélték a TREEZZY2-vel összehasonlítva, a FUZZYFTAPY minimális vágási készleteinek használata az **e2** alapeseményt is eltérítette.

Míg a (14 ábra és 15. ábra) a két szoftver közötti standard közelítést és alfa vágásokat egyaránt ábrázoló legfelső eseménygrafikonok olyan grafikont mutatnak, amely túl kevés változást mutatnak az elemzéshez, mivel a TREEZZY2 nem teszi lehetővé a grafikon nagyítását, ennek ellenére hasonló inflexiókat mutat. (a fekete vonal). Bár ez az eredmény nem meggyőző, mivel a funkciók mérete túl kicsi.

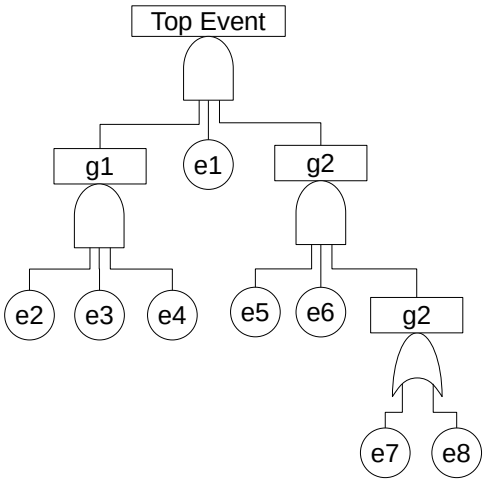
Végül a (32. Kódrészben) látható minimális vágás halmazok azt mutatják, hogy mindkét szoftver pontosan ugyanazon halmazokat számította ki.

7.4 Teszt 3

7.4.1 Bemeneti Adatok

Base Event Name	x1	x2	x4
e1	0.019985	0.035122	0.042238
e2	0.059841	0.061238	0.071002
e3	0.156991	0.235120	0.272563
e4	0.100251	0.120000	0.156847
e5	0.110000	0.130000	0.130000
e6	0.051235	0.068945	0.071462
e7	0.156234	0.195312	0.256843
e8	0.258952	0.279856	0.315689

Táblázat 10: Trapéz alapesemény kulcspontjai a 3. teszthez



Ábra 16: Hibafa rajza a 3. teszthez

7.4.2 Eredmények

Top Event Approximate	x1	x2	x4
TREEZZY2	3.98E-08	2.29E-07	5.85E-07
fuzzyftapy	3.98E-08	2.29E-07	5.85E-07
fuzzyftapy with mcs	4.41E-08	2.58E-07	6.82E-07

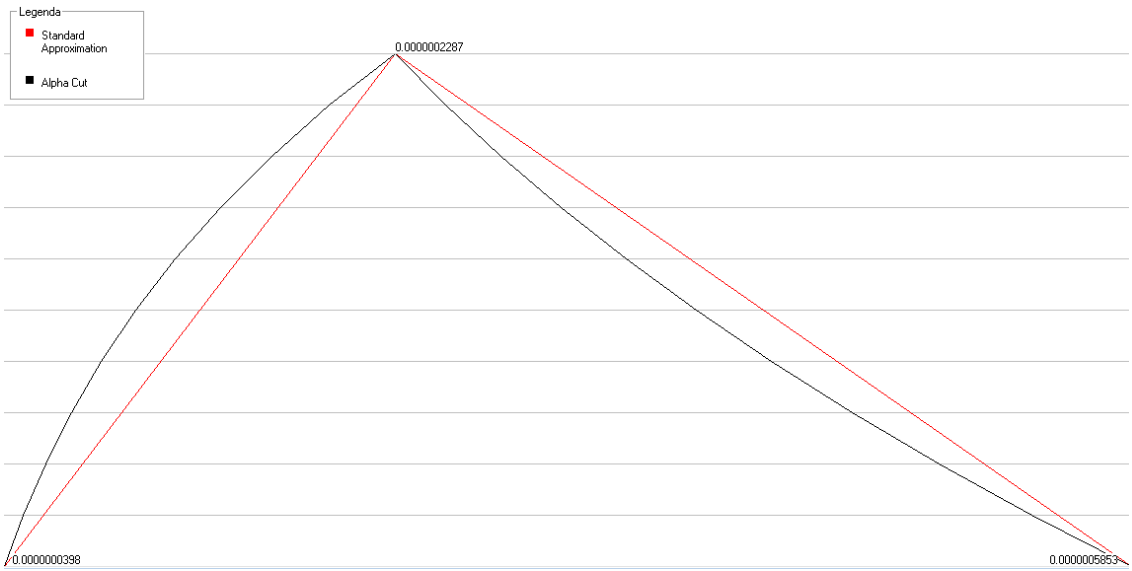
Táblázat 11: Trapéz legfelső esemény standard approximációja 3. teszthez

Event Name	FIM				FIM ranking			
	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0
e1	1.61447E-05	2.32097E-05	2.67007E-05	1.40003E-05	1	1	1	1
e2	9.60430E-06	1.35526E-05	1.56009E-05	8.27080E-06	2	2	2	2
e3	2.50190E-06	3.53820E-06	4.07210E-06	2.16250E-06	6	6	6	6
e4	4.34770E-06	6.44830E-06	7.41550E-06	3.75300E-06	5	5	5	5
e5	5.24560E-06	7.00520E-06	8.07250E-06	4.51720E-06	4	4	4	4
e6	9.54240E-06	1.29191E-05	1.48822E-05	8.22770E-06	3	3	3	3
e7	1.19110E-06	1.37280E-06	1.96500E-06	8.18900E-07	8	7	7	8
e8	1.19110E-06	1.50870E-06	1.96500E-06	8.89700E-07	7	7	8	7

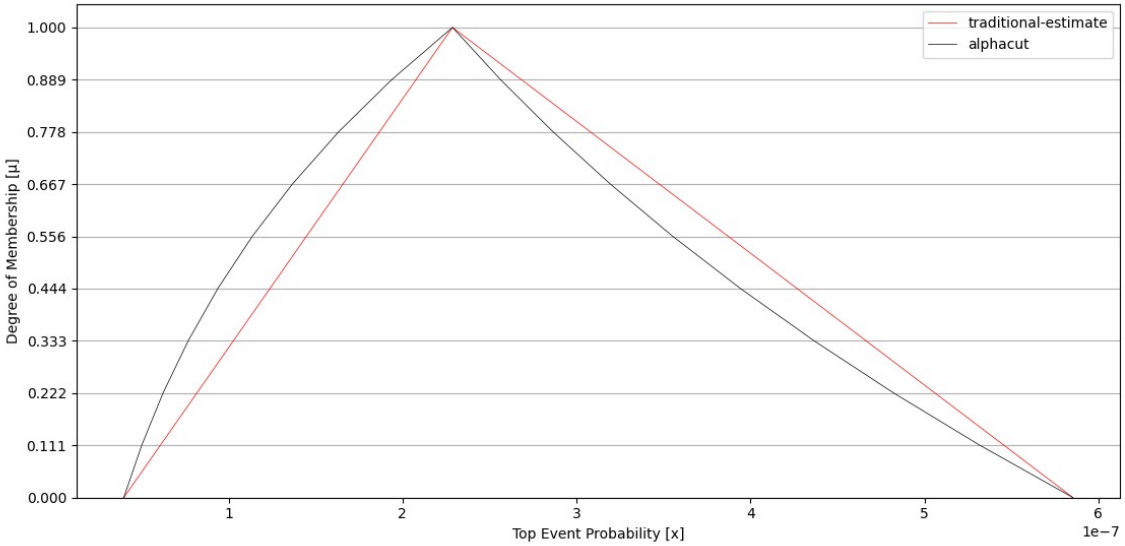
Táblázat 12: FIM számok és rangjai az alapeseményeknek 3. teszthez

Event Name	FUIM				FUIM ranking			
	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0
e1	1.34300E-07	1.03100E-07	1.19500E-07	1.03100E-07	2	2	2	2
e2	8.17000E-08	8.04000E-08	9.38000E-08	8.04000E-08	4	4	4	4
e3	1.08900E-07	8.28000E-08	9.61000E-08	8.28000E-08	3	3	3	3
e4	1.49000E-07	1.37700E-07	1.60400E-07	1.37700E-07	1	1	1	1
e5	1.57000E-08	7.20000E-09	8.00000E-09	7.20000E-09	8	8	8	8
e6	4.58000E-08	2.47000E-08	2.84000E-08	2.47000E-08	6	7	7	7
e7	6.93000E-08	5.02000E-08	7.34000E-08	5.02000E-08	5	5	5	5
e8	4.00000E-08	3.18000E-08	4.28000E-08	3.18000E-08	7	6	6	6

Táblázat 13: FUIM számok és rangjai az alapeseményeknek 3. teszthez



Ábra 17: A Legfelső esemény alakjának diagramja a Standard Approximáció és Alfa-vágás módszerrel a TREEZY2 szoftverből az 3. teszthez



Ábra 18: A Legfelső esemény alakjának diagramja a Standard Approximáció és Alfa-vágás módszerrel a FUZZYFTAPY szoftverből az 2. teszthez

```
{
        “mcs-TREEZY2”: [
                                [“e2”, “e3”, “e4”, “e5”, “e6”, “e7”, “e1”],
                                ["e2", "e3", "e4", "e5", "e6", "e8", "e1"]
                        ],
        ”mcs-FUZZYFTAPY”: [
                                ["e2", "e3", "e4", "e5", "e6", "e7", "e1"],
                                ["e2", "e3", "e4", "e5", "e6", "e8", "e1"]
                        ]
}
```

Kódrész 33: 3. teszt A FUZZYFTAPY és TREEZZY2 minimális vágási halmazai, a formátum (JSON) és a sorrend módosult az olvashatóság érdekében

7.4.3 Összehasonlítás

A (11. Táblázatban) látható legfontosabb események kulcspontjai pontosan megegyeznek a TREEZZY2 és a FUZZYFTAPY becslései között. A minimális vágáshalmazok közelítésben történő ismételt használata azt eredményezi, hogy a becslés magasabb becslést ad.

A (12.Táblázat és 13. Táblázat)-ban a FIM és FUIM számok nagymértékben különböznek a TREEZZY2 és a FUZZYFTAPY előrejelzései között. A FIM-rangsorok eközben ugyanazokat az előrejelzéseket mutatják, kivéve az **e7** és **e8** eseményeket, amelyek a FUZZYFTAPY opciói között különböznek, míg a FUIM rangsorok hasonló mintát követnek, az **e6** és az **e8** különbözik a két szoftver között, de összességében ugyanazok maradnak, függetlenül az opcióktól.

A (17 ábrán és 18 ábrán) látható legfelső esemény Alfa-vágási grafikonok alakjukban pontosan megegyeznek a két szoftver között.

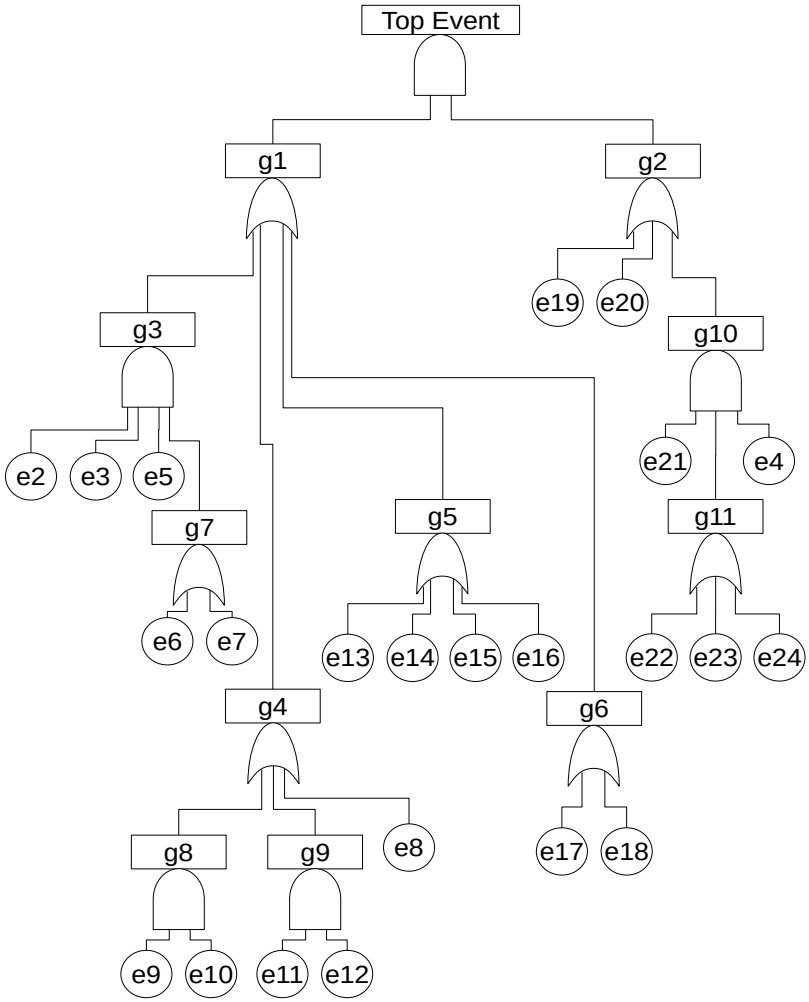
Végül a (33. Kódrészben) látható minimális vágáskészletek azt mutatják, hogy mindkét szoftver pontosan ugyanazt a minimális vágáskészletet számította ki.

7.5 Teszt 4

7.5.1 Bemeneti Adatok

Base Event Name	x1	x2	x3	x4
e1	0.100	0.110	0.122	0.200
e2	0.215	0.294	0.300	0.351
e3	0.051	0.119	0.122	0.371
e4	0.311	0.354	0.485	0.499
e5	0.170	0.230	0.289	0.325
e6	0.220	0.300	0.300	0.339
e7	0.108	0.114	0.121	0.222
e8	0.219	0.291	0.316	0.451
e9	0.051	0.115	0.124	0.358
e10	0.350	0.351	0.480	0.496
e11	0.171	0.230	0.284	0.371
e12	0.229	0.351	0.351	0.371
e13	0.110	0.118	0.122	0.205
e14	0.210	0.295	0.310	0.400
e15	0.051	0.113	0.127	0.328
e16	0.301	0.350	0.489	0.490
e17	0.179	0.233	0.289	0.352
e18	0.221	0.302	0.310	0.330
e19	0.081	0.115	0.123	0.211
e20	0.210	0.295	0.316	0.401
e21	0.054	0.117	0.125	0.313
e22	0.311	0.351	0.481	0.490
e23	0.107	0.232	0.287	0.308
e24	0.221	0.234	0.330	0.337

Táblázat 14: Trapéz alapesemény kulcspontjai a 3. teszthez



Ábra 19: Hibafa rajza a 4. teszthez

7.5.2 Eredmények

Top Event Approximate	x1	x2	x3	x4
TREEZZY2	0.021874	0.037964	0.048164	0.113625
fuzzyftapy	0.021874	0.037964	0.048164	0.113626
fuzzyftapy with mcs	0.039942	0.085578	0.123440	0.369692

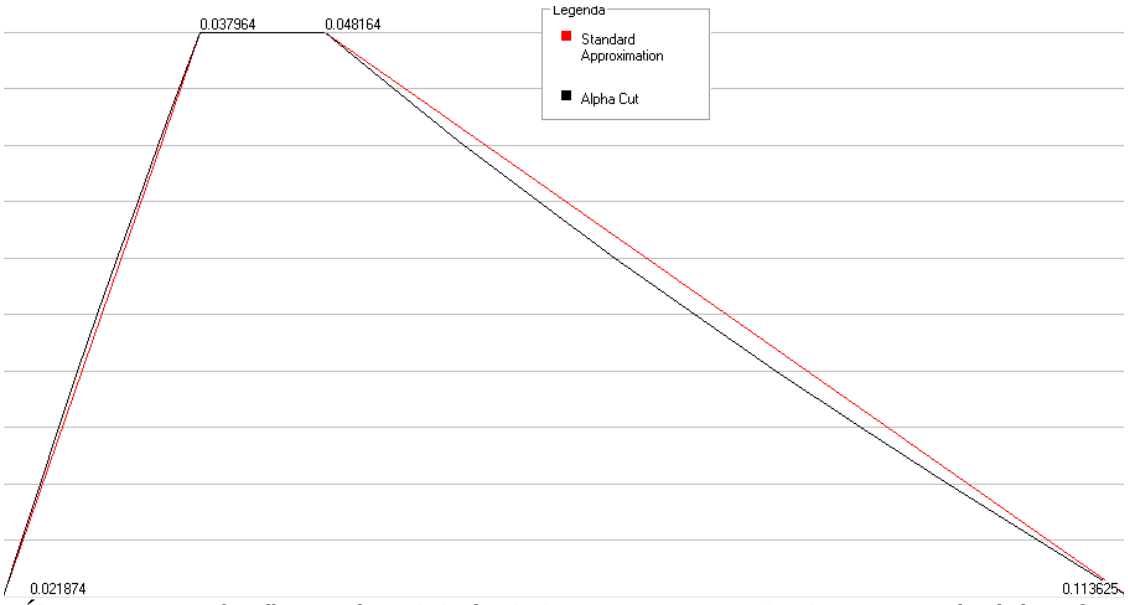
Táblázat 15: Trapéz legfelső esemény standard approximációja 4. teszthez

Event Name	FIM				FIM ranking			
	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0	TREZZY2	fuzzyftapy	fuzzyftapy with mcs	fuzzyftapy alpha = 0.0
e1	0.117836	1.133165	1.850544	0.608784	1	1	1	1
e2	0.000392	0.000266	0.007674	0.000190	22	22	22	22
e3	0.001424	0.000380	0.008709	0.000191	19	20	20	21
e4	0.011010	0.029818	0.166286	0.022106	7	5	12	5
e5	0.000503	0.000293	0.008317	0.000205	21	21	21	20
e6	0.000190	0.000159	0.004887	0.000107	24	23	23	23
e7	0.000380	0.000132	0.004885	0.000091	23	24	24	24
e8	0.014188	0.019092	0.175885	0.009765	6	7	6	7
e9	0.010674	0.005684	0.080702	0.002907	8	16	13	16
e10	0.002819	0.002225	0.044452	0.001413	18	19	19	19
e11	0.004093	0.004512	0.062214	0.002001	14	17	17	17
e12	0.001314	0.003550	0.055838	0.001747	20	18	18	18
e13	0.007450	0.015390	0.171337	0.007982	12	12	11	12
e14	0.009893	0.018738	0.175072	0.009410	9	8	7	8
e15	0.019149	0.015414	0.173193	0.008016	5	11	10	11
e16	0.007978	0.021909	0.177062	0.010774	11	6	5	6
e17	0.009139	0.017711	0.174179	0.008950	10	10	8	10
e18	0.003365	0.018594	0.174000	0.009208	16	9	9	9
e19	0.031769	0.218600	0.616996	0.119428	4	3	3	3
e20	0.036793	0.278676	0.666225	0.152712	3	2	2	2
e21	0.036967	0.064303	0.324788	0.036329	2	4	4	4
e22	0.004522	0.009072	0.073078	0.006623	13	13	14	13
e23	0.003676	0.006775	0.072099	0.004883	15	15	16	15
e24	0.003205	0.007066	0.072256	0.005103	17	14	15	14

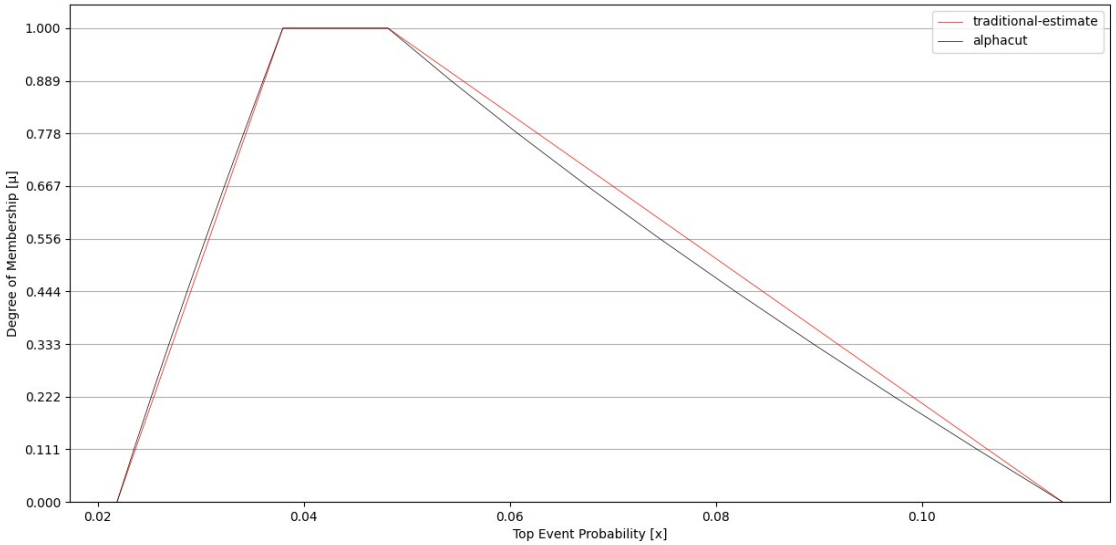
Táblázat 16: FIM számok és rangjai az alapeseményeknek 4. teszthez

Event Name	FUIM				FUIM ranking			
	TREZZY2	fuzzyftapy	futtyftapy with mcs	fuzzyftapy alpha = 0.0	TREZZY2	fuzzyftapy	futtyftapy with mcs	fuzzyftapy alpha = 0.0
e1	0.910042	0.044368	0.125199	0.044368	1	1	1	1
e2	0.006726	0.000010	0.000344	0.000010	20	21	22	21
e3	0.006365	0.000045	0.001589	0.000045	21	18	14	18
e4	0.130400	0.000317	0.001836	0.000317	5	13	13	13
e5	0.007262	0.000008	0.000263	0.000008	19	22	23	23
e6	0.004211	0.000005	0.000165	0.000005	22	23	24	24
e7	0.004208	0.000010	0.000426	0.000010	23	21	20	22
e8	0.101540	0.000963	0.014030	0.000963	7	6	6	6
e9	0.049476	0.000469	0.011750	0.000469	16	10	7	10
e10	0.035967	0.000023	0.000575	0.000023	18	20	18	20
e11	0.037012	0.000147	0.003265	0.000147	17	14	11	14
e12	0.037012	0.000137	0.000958	0.000137	17	15	17	15
e13	0.097603	0.000336	0.008392	0.000336	12	12	9	12
e14	0.100706	0.000816	0.009526	0.000816	8	7	8	7
e15	0.099545	0.001028	0.020660	0.001028	11	5	5	5
e16	0.102184	0.000432	0.001430	0.000432	6	11	15	11
e17	0.099930	0.000509	0.006602	0.000509	9	9	10	9
e18	0.099577	0.000648	0.003108	0.000648	10	8	12	8
e19	0.321984	0.009266	0.033645	0.009266	3	3	4	3
e20	0.360316	0.012981	0.034461	0.012981	2	2	3	2
e21	0.196490	0.006634	0.040243	0.006634	4	4	2	4
e22	0.057281	0.000066	0.000526	0.000066	13	17	19	17
e23	0.056342	0.000120	0.001231	0.000120	15	16	16	16
e24	0.056490	0.000037	0.000402	0.000037	14	19	21	19

Táblázat 17: FUIM számok és rangjai az alapeseményeknek 4. teszthez



Ábra 20: A Legfelső esemény alakjának diagramja a Standard Approximáció és Alfa-vágás módszerrel a TREEZY2 szoftverből az 4. teszthez



Ábra 21: A Legfelső esemény alakjának diagramja a Standard Approximáció és Alfa-vágás módszerrel a TREEZY2 szoftverből az 4. teszthez

```
{ "mcs": [ ["e1", "e8", "e19"], ["e1", "e8", "e20"],
   ["e1", "e13", "e19"], ["e1", "e13", "e20"], ["e1", "e14", "e19"],
   ["e1", "e14", "e20"], ["e1", "e15", "e19"], ["e1", "e15", "e20"],
   ["e1", "e16", "e19"], ["e1", "e16", "e20"], ["e1", "e17", "e19"],
   ["e1", "e17", "e20"], ["e1", "e18", "e19"], ["e1", "e18", "e20"],
   ["e1", "e11", "e12", "e19"], ["e1", "e11", "e12", "e20"],
   ["e1", "e9", "e10", "e19"], ["e1", "e9", "e10", "e20"],
   ["e1", "e15", "e21", "e23", "e4"], ["e1", "e15", "e21", "e24", "e4"],
   ["e1", "e13", "e21", "e23", "e4"], ["e1", "e13", "e21", "e24", "e4"],
   ["e1", "e16", "e21", "e22", "e4"], ["e1", "e16", "e21", "e23", "e4"],
   ["e1", "e16", "e21", "e24", "e4"], ["e1", "e8", "e21", "e24", "e4"],
   ["e1", "e8", "e21", "e22", "e4"], ["e1", "e17", "e21", "e22", "e4"],
   ["e1", "e17", "e21", "e23", "e4"], ["e1", "e17", "e21", "e24", "e4"],
   ["e1", "e14", "e21", "e22", "e4"], ["e1", "e14", "e21", "e23", "e4"],
   ["e1", "e18", "e21", "e22", "e4"], ["e1", "e18", "e21", "e23", "e4"],
   ["e1", "e18", "e21", "e24", "e4"], ["e1", "e14", "e21", "e24", "e4"],
   ["e1", "e8", "e21", "e23", "e4"], ["e1", "e13", "e21", "e22", "e4"],
   ["e1", "e15", "e21", "e22", "e4"], ["e1", "e11", "e12", "e21", "e24", "e4"],
   ["e1", "e11", "e12", "e21", "e22", "e4"], ["e1", "e11", "e12", "e21", "e23", "e4"],
   ["e1", "e9", "e10", "e21", "e22", "e4"], ["e1", "e9", "e10", "e21", "e23", "e4"],
   ["e1", "e9", "e10", "e21", "e24", "e4"], ["e1", "e2", "e3", "e5", "e7", "e19"],
   ["e1", "e2", "e3", "e5", "e7", "e20"], ["e1", "e2", "e3", "e5", "e6", "e19"],
   ["e1", "e2", "e3", "e5", "e6", "e20"],
   ["e1", "e2", "e3", "e5", "e7", "e21", "e24", "e4"],
   ["e1", "e2", "e3", "e5", "e7", "e21", "e22", "e4"],
   ["e1", "e2", "e3", "e5", "e7", "e21", "e23", "e4"],
   ["e1", "e2", "e3", "e5", "e6", "e21", "e22", "e4"],
   ["e1", "e2", "e3", "e5", "e6", "e21", "e23", "e4"],
   ["e1", "e2", "e3", "e5", "e6", "e21", "e24", "e4"] ]}
```


Kódrész 34: 4. teszt FUZZYFTAPY és TREEZZY2 minimális vágási halmazai, a formátum (JSON) és a sorrend módosult az olvashatóság érdekében

7.5.3 Összehasonlítás

A (15. Táblázatban) látható hogy a legfelső események kulcspontjai pontosan megegyeznek a TREEZZY2 és a FUZZYFTAPY előrejelzései között. A minimális vágáshalmazok közelítésben történő ismételt használata azt eredményezi, hogy a becslés magasabb becslést ad.

A (16 and 17. Táblázatokban) a FIM- és FUIM-számok nagymértékben különböznek a TREEZZY2 és FUZZYFTAPY előrejelzései között, különösen az a tény, hogy a TREEZZY2-ben az **e1** alapesemény FIM-száma egy nagyságrenddel alacsonyabb, mint a FUZZYFTAPY-ban. Míg a FUZZYFTAPY 1-nél nagyobb számokat képes előállítani a FIM és FUIM számítás során, addig a TREEZZY2 szoftver nem. A rangsorok ismét hasonlóak, jelentős különbségek vannak a két szoftver között, és néhány kisebb különbség a FUZZYFTAPY opciók között.

A (20. és 21. ábrákon) látható Legfelső esemény Alfa-vágási grafikonok alakjukban pontosan megegyeznek a két szoftver között.

A Snippet (34. Kódrészben) látható minimális vágás halmazok megegyeztek a FUZZYFTAPY szoftverrel is, de a rövidség kedvéért kimaradt.

7.6 Lefutási idő

	test 1	test 1 using MCS	test 4	test 4 using MCS
top event approximate	0.000173 s	0.000197 s	0.000269 s	0.001297 s
top event alphacut	0.000351 s	0.000503 s	0.000640 s	0.002705 s
FIM	0.003500 s	0.003834 s	0.020930 s	0.068483 s
FUIM	0.005197 s	0.005707 s	0.030690 s	0.101906 s
MCS	0.000053 s	0.000066 s	0.000236 s	0.000000 s
total	0.009274 s	0.010307 s	0.052529 s	0.174391 s

Táblázat 18: A FUZZYFTAPY egyszeri futtatási végrehajtási ideje tesztadatok használatával

A TREEZZY2 szoftvert nem lehetett összehasonlítani, mivel a használat és a tesztelés során virtuális gépben futott, azonban a 4. teszt számításai során a minimális vágás halmazok meghatározása több mint egy percet vett igénybe, sőt a standard approximációja a legfelső eseménynek is másodpercekig tartott.

Ezzel szemben a FUZZYFTAPY szoftvernek még a bonyolultabb 4. teszt adatkészletben is a másodperc töredéke volt az összes számítás elvégzéséhez. Ezek a számok erősen hardverfüggőek, és eseti alapon kell értékelni őket. Ezenkívül ezek egyetlen futtatási számok, egy átfogóbb teszt átlagolva sok futtatást jelentene ugyanazon az adatkészleten. Ez megmutatja a valós idejű alkalmazásokban való felhasználás lehetőségét a további optimalizálás után.

Érdemes megjegyezni, hogy ezeket a számokat Python 3.9+ verzión kaptuk meg egy Ryzen 1700x processzoron futva.

8 AZ FFTA LEHETSÉGES ALKALMAZÁSAI ÖNVEZETŐ AUTÓk TÉMAKÖRÉBEN

Mivel az autóipar a teljesen autonóm önvezető járművek felé halad, és a fogyasztóknak értékesített új autók bizonyos szintű autonómiával rendelkeznek, legyen szó akár sebességtartó automatika, sávtartó vagy félautonóm vezetés formájában, egyre fontosabbá válik a számszerűsithetően felmérni az önvezetés biztonságát és kockázatait. Mivel viszonylag kevés az önvezető autó, és néhány éve vezették be a forgalomba, viszonylag kevés balesetből lehet pontos statisztikákat levonni a biztonságukról. Nem csak ez, hanem az ütközés bekövetkezte után is ki kell vizsgálni annak okát hogy megállapíthato legyen, hogy az autonóm jármű volt-e a hibás (és melyik része), vagy emberi hibát követett el a jármű vezetője, vagy a baleset más résztvevője.

A Fuzzy Logkát bizonyos mértékben alkalmazták az önvezető járművek autonóm rendszereinek egyes részein, kevés kutatási cikk foglalkozott az autonóm járműalkatrészek kockázatértékelésében a hibafa alkalmazásokkal, nevezetesen [11]. És csak egy szakirodalmat tártunk fel, amely a hibafák használatát tárgyalta az önvezető járművek neurális hálózatának (AI) robusztusságának becslésére extrém körülmények között, nevezetesen [12].

Míg akad egy pár tanulmány ami a Fuzzy Hibafa Analízist sikeresen alkalmazta bizonyos alkalmazásokban, amelyekben nagy volt a bizonytalanság vagy a meghibásodás valószínűségének becslésére [1][2][3], viszont egyetlen tanulmány sem alkalmazta ezt a megközelítést önvezető járművekre.

Kevés irodalom van, amelyből ihletet meríthetnénk, és a téma mélyreható elemzése túlmutat e dolgozat keretein. Bár van néhány hely, ahol ezt lehet alkalmazni.

Az egyik ilyen terület az önvezető autóalkatrészek meghibásodásának kockázatértékelése. A komponensek ebben az esetben egyaránt lehetnek hardverek (érzékelők, elektronika, kommunikáció) vagy szoftverek (jelfelismerés, akadályérzékelés stb.).

Egy másik ilyen terület a kereszteződések neurális hálózatok általi tanulhatóságának meghatározása, elemzése. Az egyes kereszteződéseket sok szempont szerint elemezhetők, mint például a balesetek gyakorisága, a sávok száma, a gyalogosforgalom, a kanyarodó sávok stb. olyan számításokkal, mint a FIM és a FUIM, hogy az erőfeszítéseket az adatgyűjtésre és a neurális hálózati modellek teljesítményének javítására a meghatározott kritikus kereszteződésekre öszpontosuljon.

9 KONKLÚZIÓ

A hagyományos hibafa analízisre alkalmazott fuzzy módszertan segíthet modellezni és számszerűsíteni a bizonytalanságokat a legfelső esemény közelítésében, segítve a mérnököket annak meghatározásában, hogy mely alapeseményeket kell megvizsgálni, hogy a fuzzy fontossági mérőszámmal (FIM) csökkentsük a legfelső esemény előfordulását, ugyanakkor megmutatja, mely alapeseményeknek a fuzzy bizonytalanság fontossági mérőszámával (FUIM) járulnak hozzá a legnagyobb mértékben a rendszer teljes bizonytalanságához.

Ez a dolgozat áttekintést nyújtott a fuzzy hibafa elemzés területéről, valamint egy rövid összefoglalót Python programozási nyelvnek, hogy segítse a megértést és a további fejlesztést ezen a területen.

A TREEZZY2 szoftvert alapul véve egy új szoftvert FUZZYFTAPY néven kedvező licenccel és annyi dokumentációval, amennyire csak szükséges, a lehető legkevesebb függőséget felhasználva fejlesztettük ki, hogy beágyazható legyen más szoftverekbe belső vagy külső könyvtárként.

A tesztelés arra a következtetésre jutott, hogy a FUZZYFTAPY ugyanazt a legfelső eseményt becsléseket számít ki standard közelítéssel, hasonló legfelső eseményt az alfa vágási módszerrel és pontosan ugyanazokat a minimális vágás halmazokat, mint a TREEZZY2 szoftver. A FIM és a FUIM számok azonban nagyon eltérőek voltak, ami arra utal, hogy a TREEZZY2 más számítási módszert használ, mint az irodalom. A FIM és a FUIM rangsorolása viszont hasonló volt mindkét szoftver esetében kevés eltéréssel, ami a fontos mérőszám, bár a különbségek miatt további tesztelést igényelnek.

Nem található irodalom a fuzzy hibafa analízis alkalmazásáról és kevés a hibafa-elemzést tárgyaló szakirodalom önvezető járművek témakörben, így ez a kutatás új irányvonala.

A kifejlesztett szoftveren további teszteléseket és validálást kell végezni az eredmények helyességének biztosítása érdekében, optimalizálni kell a számítási sebesség javítása érdekében, és meg kell vizsgálni a defuzzifikáció megfelelő módját a FUIM számításához, mivel a szakirodalomban nem tették közzé milyen módot alkalmaztak.

Az elkészített szoftver és a szakdolgozat angol és magyar nyelvre lefordítva a következő Github tárhelyen található (aláírások tartalmazó lapokat törölve a biztonság érdekében):

https://github.com/AzureDVBB/fuzzyftapy

Irodalom Jegyzék

[1] VESELY, William E., et al. *Fault tree handbook*. Nuclear Regulatory Commission Washington DC, 1981, pp. 34 36 58-59 79 93-97 174-180.

[2] ZHENG, Xiaoxia, et al. Drive system reliability analysis of wind turbine based on fuzzy fault tree. In: *2016 35th Chinese Control Conference (CCC)*. IEEE, 2016. p. 6761-6765, pp. 1-2.

[3] CASAMIRRA, Maddalena, et al. Safety analyses of potential exposure in medical irradiation plants by Fuzzy Fault Tree. 2014, pp. 1-2 4.

[4] Fuzzy Set Theory, KRUSE, Rudolf; MOEWES, Christian. Fuzzy Systems. 2012, [viewed date: 11 December 2021]. Available From: <http://fuzzy.cs.ovgu.de/wiki/uploads/Lehre.FS1213/fs2012_ch02_fst.pdf>

[5] PAHL, Peter J.; DAMRATH, Rudolf. *Mathematical foundations of computational engineering: a handbook*. Springer Science & Business Media, 2001. pp. 15

[6] ZADEH, Lotfi A. Fuzzy sets and information granularity. *Advances in fuzzy set theory and applications*, 1979, 11: 3-18.

[7] GUIMARÃES, Antonio CF; EBECKEN, Nelson FF. FuzzyFTA: a fuzzy fault tree system for uncertainty analysis. *Annals of Nuclear Energy*, 1999, 26.6: 523-532. pp. 2-4

[8] BELLINI, Salvatore; CASAMIRRA, Maddalena; CASTIGLIA, Francesco. TREEZZY2, a Fuzzy Logic Computer Code for Fault Tree and Event Tree Analyses. In: *Probabilistic Safety Assessment and Management*. Springer, London, 2004. p. 3577-3582.

[9] Python Software Foundation, [viewed date: 11 December 2021]. Available From: <https://www.python.org/doc/essays/blurb/>

[10] VANROSSUM, Guido; DRAKE, Fred L. *The python language reference*. Amsterdam, Netherlands: Python Software Foundation, 2010.

[11] DAS, Plaban. *Risk analysis of autonomous vehicle and its safety impact on mixed traffic stream*. Rowan University, 2018.

[12] ZHANG, Jin, et al. Analyzing Influence of Robustness of Neural Networks on the Safety of Autonomous Vehicles. In: *31th European Safety and Reliability Conference (Forthcoming)*. 2021.

[13] SURESH, P. V.; BABAR, A. K.; RAJ, V. Venkat. Uncertainty in fault tree analysis: A fuzzy approach. *Fuzzy sets and Systems*, 1996, 83.2: 135-141.