
GAZİ ÜNİVERSİTESİ
Gazi
1926

GENETİK ALGORİTMA KULLANARAK BDD OPTİMİZASYONU

Ahmet Salih Dağ- 181180019

GENETİK ALGORİTMA DERSİ FİNAL PROJESİ
DOÇ. DR. ÜMİT ATİLA

GAZİ ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

OCAK 2024

İÇİNDEKİLER

ŞEKİLLER LİSTESİ

1. GİRİŞ

Bu raporda A.Kumar'ın makalesinde sunulan yöntem incelenecektir. Makale genetik algoritma kullanarak ikili karar diyagramı (BDD) boyutunu optimize etmeyi amaçlamaktadır. Genetik algoritma yöntemi ile hata ağacı analizi için geleneksel istatistiksel yöntemlerin ve boolean indirgemelerin kısıtlılıklarını aşmayı hedeflenmektedir.

Makalede sistem başarısızlığının olasılığını ve sıklığını değerlendirmek için yaygın olarak kabul edilen bir teknik olan Fault Tree Analysis'in (Hata Ağacı Analizi) kullanımını ve bu yöntemin sınırlamalarını ele almaktadır. Fault Tree yönteminin minimal kesim kümelerini hesaplama konusundaki kısıtlamaları, Binary Decision Diagram (BDD - İkili Karar Diyagramı) yöntemiyle aşılabilmektedir. BDD, ikili mantık temeline dayanan ve minimal kesim kümelerini bulmak için bu tür boolean ifadeleri çözme ihtiyacı olmayan nispeten yeni bir yaklaşımdır. BDD yöntemi, hata ağacını doğrudan analiz etmez, ancak ağacı bir BDD'ye dönüştürerek en üst olay için Boolean denklemi temsil eder ve bu nedenle daha iyi bir hesaplama verimliliğine sahiptir.

Makale, bu sorunları aşmak için farklı tekniklerin kullanıldığını ve bazı yöntemlerin yalnızca en önemli minimal kesim kümelerini ürettiğini belirtmektedir. Bu tekniklerden biri olan "culling" tekniği, belirli bir sıradaki (örneğin, 4 ve üzeri) kesim kümelerini ifadeden çıkarır veya siler; bu yöntem, yüksek düzeydeki kesim kümelerinin genellikle düşük olasılığa sahip olduğunu ve bu nedenle en üst olay olasılığına önemli bir katkı yapmadığını varsayar. Ancak, bu yöntemin dezavantajı, ortak nedenli hatalar devreye girdiğinde önemli hatalara neden olmasıdır. "Probabilistic culling" de uygulanabilir; bu durumda, olasılığı belirli bir eşik değerinin altında olan bir kesim kümesi tekrar göz ardı edilir.

Proje, ilk bölümde sistem güvenilirlik modellemesi için Fault Tree Analysis'in girişi ile başlar ve bu yöntemin önemi ve sınırlamaları ele alınır. Ardından, Binary Decision Diagram hakkında detaylı bir giriş sunulur. Sonraki bölümde, Shannon'ın if-then-else bağlantısı aracılığıyla BDD kullanarak fault tree analizi üzerine vurgu yapılır. GA optimizasyon

yönteminin avantajlarını göstermek amacıyla mevcut heuristiklerin dezavantajları vurgulanır. Bunu takiben, GA optimizasyon sürecine dair detaylı bir analiz gerçekleştirilir.

2. GEÇMİŞ ÇALIŞMALAR

Genetik algoritma, farklı optimizasyon problemlerini çözmek için uygulanabilen bir sezgisel algoritmadır. Genetik algoritmanın DD optimizasyonunda kullanılması ilk kez [1] makalesinde tartışılmıştır. Daha sonra, DD optimizasyonu için genetik algoritmalar birçok makalede geliştirilmiştir. Bazıları DD boyutunu optimize eder. [2] makalesinde 1-yolların sayısı optimize edilmiştir ve [3] makalesinde bu iki optimizasyon için bir yöntem önerilmiştir.

Karar diyagramları (DD), ayrık fonksiyonların temsili için kompakt bir veri yapılarıdır. Bryant kanonikliklerini 1986 yılında [4]'de gösterdi ve daha sonra ayrık fonksiyonların kullanıldığı birçok alanda uygulandı: donanım tasarımı, donanım testi, sinyal işleme, vb. Tasarlanan donanımın veya yapılan hesaplamaların karmaşıklığı Karar diyagramları diyagramın boyutuyla doğru orantılıdır. Karar diyagramlarının ana dezavantajı, boyutlarının diyagramda kullanılan değişkenlerin sırasına bağlı olmasıdır.

DD boyutunun optimizasyonu çok sık çözülen bir problemdir. DD optimizasyonu için algoritmalar genellikle iki kategoriye ayrılabilir: kesin algoritmalar ve sezgisel algoritmalar. Temel kesin algoritma, değişkenlerin tüm olası sıralamaları için diyagramlar oluşturan ve en iyisini seçen bir brute-force algoritmasıdır. Ancak, tüm kesin algoritmalar çok yavaş ve çok sayıda değişken içeren fonksiyonlar için uygunsuzdur.

İncelenen makalede güvenilirlik analizinde BDD'nin uygulanmasının Rauzy [5], Coudert ve Madre [6] tarafından başlatıldığını ve büyük hata ağaçlarını çözmek için BDD'nin araştırıldığını belirtir. Projenin amacı, BDD'nin boyutunu optimize etmek için Genetik Algoritma tabanlı yeni bir yöntemi uygulamaktır. BDD boyutunun optimize edilmesi, temel olayların en uygun sıralamasının seçilmesinden başka bir şey değildir. GA yönteminin BDD optimizasyonu için önemli olduğuna dair literatürde bazı raporlar bulunsa da bu çalışma, GA parametrelerini nasıl uyguladığımız konusunda diğer çalışmalardan farklıdır.

3. YÖNTEM

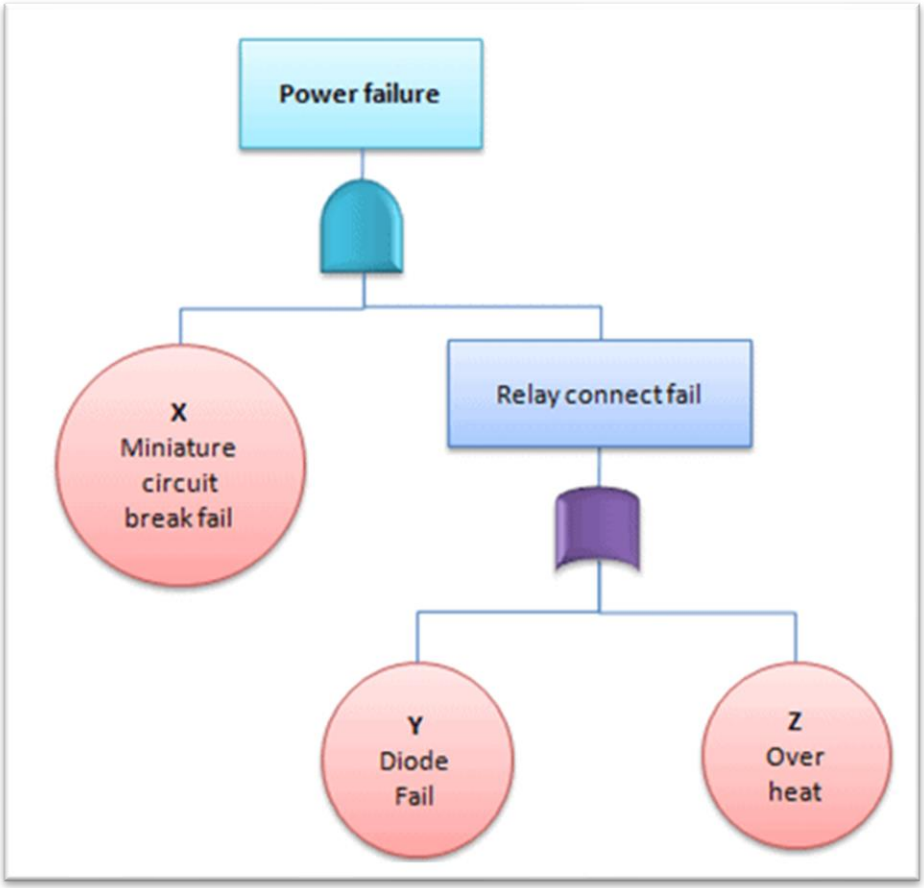
Bu bölümde makaledeki yaklaşımın değerlendirileceği ve makalede uygulanan tekniğin denenmesi sonucunda alınan sonuçların inceleneceği bölümdür. Yaklaşımları değerlendirmeye makalenin incelediği yöntemleri inceleyerek başlanacaktır.

3.1. Kavramlar

Makalede genetik algoritma kullanarak BDD boyutunu optimize etmeyi amaçlamaktadır. Makale hata ağacı analizi için geleneksel yöntemlerin kısıtlılıklarını genetik algoritma kullanarak aşmayı hedeflemektedir. Raporda bu hedefleri Python ortamında testi yapılacaktır. Makalenin hedeflerini daha yakından inceleyecek olursak hata ağacı analizi, ikili karar diyagramı ve denenen özel genetik algoritmalar yer almaktadır. Bunların ne olduğundan ve projede nasıl kullanıldıklarından kısaca bahsedelim.

3.1.1. Fault Tree Analizi

Fault Tree Analysis (Hata Ağacı Analizi), bir sistemde istenmeyen durumları değerlendirmek için kullanılan bir analitik tekniktir. Bu durumlar genellikle güvenlik veya güvenilirlik açısından kritik olan durumları içerir. Fault Tree Analysis, sistemdeki çeşitli paralel ve ardışık hataların kombinasyonlarını inceleyerek belirli bir istenmeyen olayın (genellikle en üst olay olarak adlandırılır) gerçekleşme olasılığını değerlendirir. Fault tree, bu olaya yol açan hataların grafik modelini içerir. Bu hatalar, bileşen donanım arızaları, insan hataları, yazılım hataları veya diğer önemli olaylar olabilir. Fault tree, temel olayların mantıksal ilişkilerini tasvir eder ve en üst olaya yol açan olayları içerir. Her fault tree, belirli bir sistem arıza moduna karşılık gelir ve bu nedenle yalnızca bu üst olaya katkıda bulunan hataları içerir.



Şekil 3.1 Fault Tree örneği [7]

Fault tree'nin temel özelliği, sonucun başarı veya başarısızlık olasılığının ikili bir olay olduğu kavramıdır. Fault tree, "gates" adı verilen bileşenlerden oluşan bir kompleks içerir. Bu gates, hata mantığının ağaç boyunca yukarı doğru hareketini ya engelleyen ya da izin veren işlevlere sahiptir. Gates, bir "üst" olayın meydana gelmesi için gereken olayların ilişkilerini gösterir. "Üst" olay, gate'in çıktısıdır; "alt" olaylar, gate'in "girişleri" dir. Gate sembolü, çıktı olayı için gereken giriş olaylarının ilişki türünü belirtir.

Fault Tree Nicel Değerlendirmeleri

Fault Tree üzerinde nitel ve nicel değerlendirmeler yapılabilir. Fault Tree, üst olaya neden olan olayların ve ilişkilerin nitel bir değerlendirmesini sunar. Fault tree oluşturulurken, üst olayın nedenleriyle ilgili önemli içgörüler elde edilir. Nitel değerlendirmeler, fault tree mantığını daha odaklı bilgi sağlayacak şekilde dönüştürür. En üst olayın minimal kesim kümeleri (MCS'ler) gibi nitel sonuçlar elde edilir. Bir MCS, en üst olayı tetikleyebilen temel olayların bir kombinasyonudur ve en küçük temel olay kombinasyonunu temsil eder. MCS'ler, temel olayların en üst olayı tetikleyebileceği tüm yolları temsil eder. MCS'lerin yapısı, önemli

bilgilerin elde edilmesini sağlar. MCS'ler, tek bir hatayı veya olayı tanımlar ve genellikle zayıf bağlantılara ve yükseltme önlemlerine odaklanır. Ayrıca, olaylarla ilgili özelliklere sahip olan bir MCS, bağlamsal bağımlı arızalara veya ortak nedenlere karşı duyarlılığı gösterir ve bu, bir redudansı yok edebilecek bir durumu ifade eder.

Fault Tree'nin Sınırlamaları

Fault tree'nin sınırlamaları arasında, tekrarlanan olaylar içeriyorsa elde edilen SOP (Sum-Of-Products) formunun minimal olmayabileceği ve bu durumda minimal kesim kümelerinin doğrudan elde edilememesi bulunmaktadır. Minimal kesim kümelerini elde etme görevi, mantık denklemleri çok sayıda kesim kümesi üretiyorsa hesaplama yoğunluğu yaratabilir. Büyük bir SOP ifadesi elde edildiğinde, bu ifadeyi minimal hale getirmek için gerekli karşılaştırmalar nedeniyle bilgisayar üzerinde uzun sürebilir. Ayrıca, her kapıdaki tüm mantıksal ifadeleri depolamak, bellek alanında yoğun taleplere neden olabilir. Ayrıca, fault tree'nin çok sayıda minimal kesim kümesi içeriyorsa, kesin en üst olay olasılığını hesaplamak için kapsamlı hesaplamalara ihtiyaç duyulabilir.

Yeni Yaklaşımlar

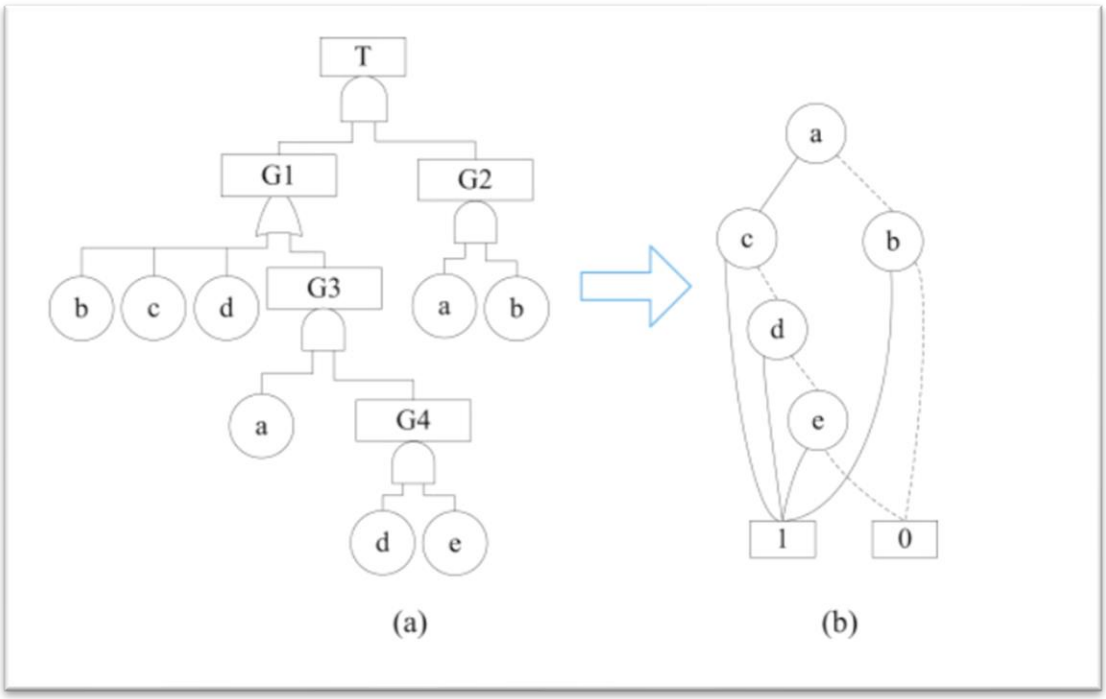
Bu sınırlamaları aşmak için çeşitli teknikler kullanılmaktadır. Örneğin, bazı teknikler sadece en önemli minimal kesim kümelerini üretir ve yüksek düzey kesim kümelerini ihmal eder. "Culling" olarak adlandırılan bir teknik, belirli bir düzenin (örneğin, 4 ve üzeri) kesim kümelerini ifadeden çıkarır veya siler. Ancak, yüksek düzey kesim kümelerinin düşük olasılığa sahip olduğu varsayımıyla yapılan bu işlemin ortak neden arızalarının dahil olduğu durumlarda hatalı sonuçlara yol açabileceği unutulmamalıdır.

Binary Decision Diagram (BDD) yöntemi, fault tree'leri analiz etmek için geleneksel tekniklere bir alternatif olarak ortaya çıkmıştır. BDD yöntemi, fault tree'yi doğrudan analiz etmek yerine ağacı bir BDD'ye dönüştürerek çalışır. Bu, en üst olayın Boolean denklemini temsil eder ve geleneksel yöntemlere göre daha etkili bir analiz sağlar.

3.1.2. İkili Karar Diyagramı

İkili Karar Diyagramı (BDD), sistemin başarı (0) veya başarısızlık (1) durumlarını temsil eden yönlendirilmiş, döngüsel olmayan bir grafiktir. 1 durumunda sonlanan tüm yollar, bir hata

ağacının kesme kümelerine karşılık gelir ve BDD, hesaplama verimliliği için alt grafik paylaşımını kullanır.



Şekil 3.2 Fault Tree (a) – BDD Dönüşüm Örneği [8]

Hata ağacındaki temel olayların sıralaması, ortaya çıkan BDD'nin boyutunu etkiler ve minimum kesim kümeleri elde etmek için minimum form esastır. Geleneksel sıralama gereksiz BDD'ye yol açabilirken, kapı olaylarını ve tekrarlanan olayları dikkate alan özel sıralama teknikleri minimum BDD üretebilir. Bununla birlikte, BDD analizine yönelik mevcut buluşsal yöntemlerin hata ağacı çizimine duyarlılık, sağlamlık eksikliği ve tekrarlanan olayların işlenmesindeki zorluklar gibi sınırlamaları vardır. İyi sıralama buluşsal yöntemi, olayın sistem arızasına olan katkısını yansıtmalı ve sağlam olmalı ve hata ağacı mantığından ve çizim yöntemlerinden bağımsız olmalıdır. Yapısal önem ölçüsü, bu kriterleri kısmen karşılayan bir buluşsal yöntemdir.

Makale, İkili Karar Diyagramlarını (BDD) kullanarak minimum kesim kümelerini bulmaya yönelik mevcut yöntemlerin sınırlamalarını tartışıyor ve minimum BDD boyutu için temel değişkenlerin sıralamasını optimize etmek için Genetik Algoritma (GA) kullanan yeni bir yaklaşım sunuyor. Büyük hata ağaçları için olası tüm değişken permütasyonları aramanın zorluğunu vurguluyor ve GA'nın bu soruna uygulanmasındaki üç önemli konuyu vurguluyor:

1. Değişken sıralamaların kromozomlar olarak uygun bir temsilinin seçilmesi,
2. İyiliği korumak için etkili bir çaprazlama prosedürünün uygulanması. Ebeveyn özellikleri aktarılması ve geçersiz yavruların önlenmesi
3. Optimizasyon için her nesilde en iyi sıralamanın seçilmesi.

3.2. Genetik Algoritma Kullanımı

GA'da kullanılan temel parametreler Seçim, Çaprazlama ve Mutasyondur. Popülasyon büyüklüğü başlangıçta bir boole fonksiyonunun sırasına veya bir hata ağacındaki temel olayların sayısına göre belirlenir. Ana amaç, temel olayların optimal sıralamasını belirleyerek BDD'nin boyutunu optimize etmektir.

N dereceli bir hata ağacı için karşılık gelen BDD, minimum N'den maksimum 2N-1'e kadar değişen düğümlere sahip olabilir. Amaç, her nesildeki en iyi düzeni bulmak ve onu korumaktır. Algoritma, arama sürecini belirli bir üretim sayısıyla sınırlandırır. Aşağıdaki akış şeması tüm GA uygulama sürecini özetlemektedir.

3.2.1. Selection

- Uygunluk Değerlerinin Hesaplanması: Uygunluk değerleri, tüm çözümler için BDD boyutunu optimize etme performanslarına göre hesaplanır.
- Rulet Çarkı Seçimi: Başlangıçta denendi, ancak doğasında olan stokastiklik nedeniyle istenmeyen tedirginlikler, nihai sonuçları beklenenden daha fazla etkiledi.
- Stokastik Kalan Rulet Çarkı Seçimi: Rastgelelik sorunlarını çözmek için algoritma, uygunluk değerlerine göre eşleştirme havuzuna dahil edilecek her çözümün tamsayı kopya sayısını belirler. Kalan kesirli kısım Rulet Çarkı seçimine tabi tutularak daha uygun çözümlerin en az bir veya daha fazla kopyasının eşleştirme havuzuna girmesi sağlanır. Bu süreç, Rulet Çarkı seçiminin doğasında olan stokastisiteyi ele alarak daha uygun çözümlerin havuza emilimini artırır.

3.2.2. Crossover

Geleneksel bir Genetik Algoritmada (GA), kromozom üzerinde rastgele bir kesim noktası belirlenir ve birinci ebeveynin kesim noktasından önceki genler ile ikinci ebeveynin kesim

noktasından sonraki genlerin birleştirilmesiyle bir çocuk üretilir. Ancak bu yaklaşım, geçersiz değişken sıraları üretebileceğinden, verilen durumda değişken sıralamasını optimize etmek için uygun değildir. Bunun yerine değişken sıralamalı çaprazlama kullanılır. Geleneksel GA'ya benzer şekilde, kesme noktasından önceki genler ilk ebeveynden gelir, ancak önemli fark, kesme noktasından sonraki genlerde yatmaktadır. İkinci ebeveyn, düzeni mümkün olduğu kadar koruyarak, kalan eksik genlere (değişkenlere) katkıda bulunur. Bu yöntemle iki çocuk üretiliyor. İkinci çocuk da, kesme noktasından önceki genleri ikinci ebeveynden, geri kalan genleri ise birinci ebeveynden alarak düzeni mümkün olduğunca koruyarak aynı prosedürü izler.

3.2.3. Mutasyon

Mutasyon, arama uzayının diğer alanlarını keşfetmek için popülasyona değişkenlik getirir. Çeşitli mutasyon operatörlerine sahip olan gerçek parametre GA'larından farklı olarak, bu durumda çevirme mutasyon operatörü kullanılır. Çevirme mutasyonu, rastgele seçilen bir sıralama kümesindeki iki öğenin yer değiştirmesini içerir. Çözümlerin yerel minimumlara takılıp kalmasını önlemek için mutasyon oluşma olasılığı düşük tutulur. Yüksek bir mutasyon oranı ayarlamak, algoritmanın optimal çözümlere yakın alanlardan kaçmasına neden olabilirken, düşük bir oran, yerel minimumlarda sıkışıp kalmaya yol açabilir.

3.2.4. Elitizm

Elitizm, her nesilden seçkin çözümlerin korunmasını içerir. Çaprazlama ve mutasyon elit çözümlerin kaybına yol açabileceğinden, her neslin en iyi sıralamasını saklamak için ayrı bir bellek konumu kullanılır. Bu sıralama daha sonra bir sonraki neslin en iyi sıralamasıyla karşılaştırılır ve üstün olan korunur. Bu süreç, en iyi sıralamanın nesiller arasında sürekli olarak karşılaştırılmasını ve nihai sonucun, tüm arama süreci boyunca elde edilen en iyi sıralama olmasını sağlar.

3.3. Genetik Algoritmanın Probleme Uygulanması

Raporda incelenen makalede fault tree analizi için kullanılan BDD yönteminin genetik algoritma ile optimizasyonunun nasıl olacağına değinilmiştir. Bu bölümde ise incelenen kavramlar ve önerilen çözüm yöntemleri Python ortamında uygulanacaktır.

BDD optimizasyonu hedefine ulaşmadan önce iki farklı Python projesi geliştirilmiştir. Bunlar Genetik algoritma ile yapılan optimizasyonun geliştirme aşamalarını bize daha yakından gösterecektir.

3.3.1. Genetik Algoritma Projesi

İlk proje genetik algoritma kullanarak sıralama problemine çözüm üreterek temel olayları sıralayan bir fonksiyon test edilmiştir. Bu kod sayesinde genetik algoritma temel fonksiyonları daha anlaşılır görülmüştür. Kodun genel akışı şu şekildedir.

Parametrelerin Belirlenmesi: Çeşitli genetik algoritma parametreleri, örneğin, popülasyon boyutu, jenerasyon sayısı, çaprazlama ve mutasyon olasılıkları gibi parametreler belirlenir.

Uygunluk Fonksiyonu (Fitness Function): Uygunluk fonksiyonu, her bireyin ne kadar iyi olduğunu değerlendirir. Bu örnekte, sıralama düzeltme maliyeti kullanılarak bir uygunluk değeri hesaplanır. Bu maliyet, temel olayların doğru sıraya yerleştirilmesi durumunda düşük, aksi takdirde yüksektir.

Birey Oluşturma Fonksiyonu: Her birey, temel olayların bir sıralamasını içerir. Bu sıralama, temel olayları içeren bir listeden rastgele seçilir.

Genetik Algoritma Tanımı: Genetik algoritmanın temel öğeleri, yaratıcı (creator), araç kutusu (toolbox) ve operatörler (çaprazlama, mutasyon, seçim) tanımlanır.

Ana Döngü: Belirli bir sayıda jenerasyon boyunca genetik algoritma çalıştırılır. Her jenerasyon, çaprazlama ve mutasyon operatörleri kullanılarak yeni bireyler üretilir. Uygunluk fonksiyonu ile değerlendirilirler ve en iyi bireyler seçilip bir sonraki jenerasyon oluşturulur. Her jenerasyonun en iyi bireyi ekrana yazdırılır.

3.3.2. Fault Tree Analizi Projesi

Genetik algoritma temelleri incelendikten sonra Fault Tree Analizi yapmak için bir yapı geliştirilmiştir. Önceden belirtildiği üzere hata ağacı, bir sistemdeki çeşitli olayların kombinasyonlarını gösteren bir yapıdır ve belirli bir olayın gerçekleşme olasılığını hesaplamak için kullanılır. Aşağıdaki şekilde kodun çıktısı yer almaktadır. 15 jenerasyon sonucunda ideal sıralamaya ulaşılmıştır.

```
Generation 11, Best Individual: [0, 1, 2, 3, 4, 5, 7, 6], Fitness: (2.0,)
Generation 12, Best Individual: [0, 1, 2, 3, 4, 5, 7, 6], Fitness: (2.0,)
Generation 13, Best Individual: [0, 1, 2, 3, 4, 5, 7, 6], Fitness: (2.0,)
Generation 14, Best Individual: [0, 1, 2, 3, 4, 5, 7, 6], Fitness: (2.0,)
Generation 15, Best Individual: [0, 1, 2, 3, 4, 5, 6, 7], Fitness: (0.0,)
```

Şekil 3.3 Kodun çıktısı

Kodun yapısı şu şekildedir:

Event Sınıfı:

- Event sınıfı, bir olayı temsil eder.
- Her olayın bir adı (name) ve bir olasılığı (probability) vardır.
- Olayın olasılığı, probability özelliği aracılığıyla erişilebilir.

Gate Sınıfı:

- Gate sınıfı, mantıksal bağlantıları temsil eder.
- Her kapı, bir mantıksal operatör (logic) ve girişleri (inputs) içerir.
- Mantıksal operatörler: 'AND', 'OR', 'NOT'
- Girişler, olaylar veya başka kapılar olabilir.

calculate_probability Fonksiyonu:

- Bu fonksiyon, bir olayın veya kapının gerçekleşme olasılığını hesaplar.
- Eğer giriş bir olay ise, olayın olasılığını döndürür.
- Eğer giriş bir kapı ise, mantıksal operatöre göre olasılığı hesaplar.
- 'AND' için minimum, 'OR' için 1 eksi çarpımı, 'NOT' için 1 eksi olasılık kullanılır.

Fault Tree Oluşturma ve Hesaplama:

- Örnek bir Fault Tree tanımlanır: A AND (B OR C)
- Bu ağaç, A, B, ve C adlı üç olayı içerir.
- calculate_probability fonksiyonu ile her olayın veya kapının olasılığı hesaplanır.
- Sonuçlar ekrana yazdırılır.

Örneğin, probability_A_AND_B_OR_C ifadesi, A AND (B OR C) mantığına göre hesaplanan olasılığı verir. Bu şekilde, Fault Tree analizi yapılarak sistemdeki belirli bir durumun olasılığı belirlenebilir.

3.3.3. GA ile BBD Optimizasyonu

Makalede asıl hedeflenen konu olan genetik algoritma ile fault tree optimizasyonu gerçekleştirilmiştir. Kodu incelemek gerekirse 2 sınıftan ve main bloğundan oluşmaktadır. Kodun yapısı şu şekildedir;

FaultTree Sınıfı:

- FaultTree sınıfı, hata ağacını temsil eder.
- İnitilize edilirken bir dizi olayı (events) alır.
- Bu sınıfın ana amacı, genetik algoritmayı çalıştırmak için bir fitness fonksiyonu sağlamaktır.

GeneticAlgorithm Sınıfı:

- GeneticAlgorithm sınıfı, genetik algoritmanın ana mantığını içerir.
- İnitilize edilirken popülasyon boyutu (population_size), nesil sayısı (generations), ve mutasyon oranı (mutation_rate) gibi parametreleri alır.
- initialize_population fonksiyonu, başlangıç popülasyonunu oluşturur. Her birey, olayların sıralamasını temsil eder.
- calculate_fitness fonksiyonu, popülasyonun fitness değerlerini hesaplar. Her bir bireyin fitness değeri, fitness_function fonksiyonu kullanılarak belirlenir.
- fitness_function fonksiyonu, hata ağacının bir tür fitness değerini üretir. Örneğin, burada sadece olayların sıralamasının toplamını kullanıyor.
- tournament_selection fonksiyonu, turnuva seçimini uygular.

- crossover fonksiyonu, çaprazlama işlemini gerçekleştirir. İki ebeveyn alır, rastgele bir noktada keser ve çocukları oluşturur.
- mutate fonksiyonu, mutasyon işlemini uygular. Rastgele iki nokta seçer ve yer değiştirir.
- run_genetic_algorithm fonksiyonu, genetik algoritmayı başlatır. Her nesilde turnuva seçimi, çaprazlama, mutasyon işlemleri yapılır ve en iyi birey bir sonraki nesle taşınır.

Kullanım:

- __main__ bloğunda bir FaultTree örneği oluşturulur.
- Bir GeneticAlgorithm örneği oluşturulur ve genetik algoritma çalıştırılır.
- En iyi sıralama ekrana yazdırılır.

Makalede önerilen yöntemin Python ortamında yazılan bu proje, genetik algoritma kullanarak bir hata ağacının olası olaylarını optimize etmeye yönelik bir temel uygulama sunmaktadır.

4. SONUÇLAR VE TARTIŞMA

Raporda A. Kumar'ın [9] "Optimization of Binary Decision Diagram using Genetic algorithm" makalesi incelenmiştir. Makalede hata ağacından minimum kesim kümelerini bulmak için kullanılan geniş ölçüde kullanılan bir boolean ifade çözme yönteminin, Binary Decision Diagram (BDD) yöntemi ile değiştirilmesini tartışılmaktadır. Makalede GA tabanlı bir çözüm sunulmuş ve bu çözüm Python ortamında kodlanarak incelenmiştir. Makalenin içeriği, geçmiş çalışmalar ve çözüm projesinin detayları rapora eklenmiştir.

Makalede boolean ifade çözme yöntemi, karşılaştırmalı olarak zayıf hesaplama verimliliği nedeniyle BDD yöntemi tercih edilmiştir. BDD yönteminden maksimum fayda elde etmek için temel olayların sıralamasının optimal olması gerektiği belirtiliyor. Bu optimal sıralama için en iyi seçimi yapmak için Genetik Algoritma yönteminin başarıyla uygulandığı ifade ediliyor. Bu optimizasyon yönteminde belirli bir sezgiye uymak zorunda olunmadığı vurgulanıyor.

İncelenen makale ve çözüm için denenen Python projesi sayesinde Genetik algoritmaları konusunda uygulamalı olarak geliştirmenin nasıl olacağı görülebilmektedir.

5. KAYNAKÇA

[1] R. Drechsler, B. Becker ve N. Göckel, «A genetic algorithm for variable,» *IEEE Proceedings Computers and Digital Techniques,* no. 6, 1996.

[2] M. Hilgemeier, N. Drechsler ve R. Drechsler, «Minimizing the number of one-paths in bdds by an evolutionary algoritm,» 2003.

[3] S. Shirinzadeh, M. Soeken ve R. Drechsler, «Multi-objective bdd optimization with evolutionary algorithms,» pp. 751-758, 2015.

[4] Bryant, «Graph-based algorithms for boolean function manipulation,» *IEEE Transactions on Computers,* pp. 677-691, 1986.

[5] Rauzy, «A new algorithms for fault tree analysis,» *Reliability Engineering and system safety,* pp. 203-11, 1993.

[6] Madre ve Coudert, «Implicit and incremental computation of primes and essential primes of Boolean functions,» %1 içinde *Proceedings of the 29thACM/IEEE DesignAutomation Conference DAC'92,* 1992.

[7] T. Hessing, «Fault Tree Analysis,» [Çevrimiçi]. Available: https://sixsigmastudyguide.com/fault-tree-analysis/. [Erişildi: 14 Ocak 2024].

[8] Y. Deng, H. Wangİ ve B. Guo, «BDD algorithms based on modularization for fault tree analysis,» *Progress in Nuclear Energy,* pp. 192-199, 2015.

[9] A. Kumar, S. Choudhary ve P. V. Varde, «Optimization of Binary Decision Diagram using Genetic algorithm,» *2010 2nd International Conference on Reliability, Safety and Hazard - Risk-Based Technologies and Physics-of-Failure Methods (ICRESH),* pp. 168-175, 2010.