

Graf Teorisi Eğitiminde Yeni Bir Araç: Z3 ve Keçeci Dizilimi ile Hamilton Probleminin İnteraktif Keşfi

Mehmet Keçeci

mkececi@yaani.com

ORCID:  <https://orcid.org/0000-0001-9937-9839>

Received: 15.08.2025

Abstract: Graf teorisi, bilgisayar bilimleri ve matematik eğitiminin temel taşlarından birini oluşturmakla birlikte, Hamilton döngüsü gibi NP-tam problemlerin soyut doğası, öğrenciler için önemli bir anlama zorluğu teşkil etmektedir. Bu tür problemlerin sezgisel olarak kavranması, genellikle karmaşık ve standart dışı graf yapılarının analitik olarak incelenmesini gerektirir. Geleneksel öğretim metotları, bu karmaşıklığı etkili bir şekilde aktarmada yetersiz kalabilmekte ve öğrencilerin konuya olan ilgisini azaltabilmektedir. Bu çalışmada, söz konusu pedagojik zorluklara yanıt olarak, Hamilton döngüsü probleminin keşfi ve analizi için tasarlanmış yeni bir birleşik hesaplama ve görsel bir araç sunulmaktadır. Geliştirilen bu çerçeve, iki temel teknolojiyi bir araya getirmektedir: Microsoft Research tarafından geliştirilen yüksek performanslı bir SMT çözücüsü olan Z3 (Z3-Solver) ve tarafımızca geliştirilen, sıralı yapılar için optimize edilmiş yenilikçi bir graf yerleşim algoritması olan Keçeci Dizilimi (Layout). Yaklaşımımızın temelinde, Hamilton döngüsü problemini algoritmik bir çözüm arayışından ziyade bir dizi mantıksal kısıtın tatmin edilmesi problemi olarak yeniden formüle etmek yatmaktadır. Bu bildirimsel model, Z3 çözücüsüne sunularak bir grafın Hamilton döngüsü içerip içermediği otomatik olarak kanıtlanmaktadır. Bu yöntem, geleneksel deneme-yanılma veya kaba kuvvet (brute-force) algoritmalarına kıyasla hem daha verimli hem de sonucun doğruluğunu matematiksel olarak garanti eden bir yapı sunar. Z3, bir çözüm bulduğunda (SAT), bu çözümü temsil eden bir model üretir; çözüm olmadığında ise (UNSAT), döngünün var olmadığını kesin olarak bildirir. Çalışmanın eğitime yönelik asıl yeniliği ise Z3 tarafından elde edilen bu soyut mantıksal sonuçların görselleştirilmesi aşamasında ortaya çıkmaktadır. Standart graf yerleşim algoritmaları (örn. Fruchterman-Reingold), genellikle estetik açıdan optimize edilmiş olsalar da sıralı bir yolun veya döngünün takibini zorlaştırabilen karmaşık ve öngörülemez çıktılar üretirler. Buna karşın, önerdiğimiz Keçeci Dizilimi, düğümleri ana bir eksen boyunca sıralı bir zikzak deseniyle konumlandırarak grafın içsel yol yapısını net bir şekilde ortaya koyar. Z3'ün bulunduğu Hamilton döngüsü bu düzen üzerine vurgulandığında, öğrenciler döngünün grafın tamamını nasıl kat ettiğini, her düğüme tam olarak bir kez nasıl uğradığını ve başlangıç noktasına nasıl döndüğünü adım adım ve sezgisel olarak takip edebilirler. Bu görsel netlik, Dodekahedral graf (Hamiltonian) ve Herschel graf (Hamiltonian olmayan) gibi klasik örnekler üzerinde test edilmiş ve kanıtlanmıştır. Sonuç olarak, bu bütünleşmiş araç, NP-tam problemler gibi zorlu konuların öğretiminde soyut mantık ile somut görsel kanıt arasında bir köprü kurarak, öğrenci anlama ve katılımını artırma potansiyeline sahip güçlü bir pedagojik kaynak olarak öne çıkmaktadır.

Keywords:

Hamilton Döngüsü, Z3 Teorem Kanıtlayıcısı, Z3-Solver, Z3, SMT Çözücü, SMT Solver, Kısıt Memnuniyet Problemi, CSP, Graf Teorisi, Graf Yerleşim Algoritmaları, Keçeci Dizilimi, Keçeci Yerleşimi, Keçeci Layout, NP-Tam Problemler, Hesaplama Mantık, Otomatik Akıl Yürütme, Veri Görselleştirme.

I. Graf Teorisi

1.1. Graf Teorisinin Eğitimdeki Süregelen Meydan Okuması

Graf teorisi, düğümler (vertices) ve bu düğümleri birleştiren kenarlardan (edges) oluşan yapıları inceleyen bir matematik ve bilgisayar bilimleri dalı olarak, modern teknolojinin ve bilimin temel yapı taşlarından birini oluşturmaktadır. Sosyal ağların dinamiklerinden, lojistik optimizasyonuna, biyoinformatikteki moleküler etkileşimlerden, internetin altyapısal topolojisine kadar sayısız alanda uygulama alanı bulan graflar, soyut ilişkileri somut ve analiz edilebilir modellere dönüştürme gücüne sahiptir. Bu temel önemi nedeniyle, graf teorisi, bilgisayar bilimleri, mühendislik ve matematik lisans programlarının vazgeçilmez bir bileşeni haline gelmiştir. Ancak, teorinin temelindeki zarafet ve güce rağmen, pedagojik uygulaması önemli zorlukları da beraberinde getirmektedir.

Bu zorlukların temelinde, graf teorisinin doğasında var olan soyutlama yatmaktadır. Düğümler ve kenarlar, fiziksel nesnelerden ziyade kavramsal varlıklar ve aralarındaki ilişkilerdir. Öğrenciler için, bu soyut yapıları zihinsel olarak canlandırmak ve üzerlerindeki algoritmik süreçleri takip etmek, özellikle problemin ölçeği büyüdüğünde ve graf yapısı karmaşıklaştığında, bilişsel olarak oldukça meşakkatli bir hâl alabilmektedir. Geleneksel tahta ve tebeşir metotları veya statik diyagramlar, grafların dinamik ve ilişkisel doğasını aktarmada çoğu zaman yetersiz kalır. Bu durum, öğrencilerin temel algoritmaları (örneğin, en kısa yol veya minimum kapsayan ağaç) anlamasını engellemekle kalmaz, aynı zamanda onları daha ileri ve teorik olarak daha derin konulardan da uzaklaştırabilir. Dolayısıyla, bu soyut kavramlar ile öğrencinin sezgisel anlayışı arasında bir köprü kurabilecek yenilikçi pedagojik araçlara âcil bir ihtiyaç bulunmaktadır.

1.2. Hamilton Döngüsü Problemi: Kanonik bir NP-Tam Meydan Okuma

Graf teorisi eğitiminde karşılaşılan zorlukları en iyi temsil eden örneklerden biri, şüphesiz Hamilton döngüsü problemidir. İrlandalı matematikçi William Rowan Hamilton tarafından 1857'de ortaya atılan bir oyunla popülerleşen bu problem, basit bir soru sorar: Bir graf üzerinde, her düğüme tam olarak bir kez uğrayıp başlangıç düğümüne geri dönen bir kapalı yol (döngü) mevcut mudur? Bu basit tanımına rağmen, Hamilton döngüsü problemi, hesaplama teorisinin en temel ve en zorlu sınıflandırmalarından biri olan NP-tam (NP-complete, NP: Nondeterministic polynomial time Problem) [1, 2] problemler sınıfına aittir.

NP-tam sınıflandırması, bir problemin çözümünü doğrulamanın (eğer bir çözüm sunulursa) hesaplamalı olarak kolay (polinomsal zamanda), ancak bir çözüm bulmanın bilinen en verimli algoritmalarla bile son derece zor (üstel zamanda) olduğu anlamına gelir. Düğüm sayısı arttıkça, olası tüm yolları deneyerek bir Hamilton döngüsü bulmaya çalışmanın mâliyeti astronomik bir şekilde artar. Bu "hesaplamalı uçurum", Hamilton döngüsü problemini, öğrencilere algoritma verimliliği, hesaplama karmaşıklığı ve teorik bilgisayar bilimlerinin sınırları gibi temel kavramları öğretmek için mükemmel bir vaka analizine dönüştürür. Problem, aynı zamanda, seyyar satıcı problemi (Traveling Salesman Problem - TSP) gibi pratik optimizasyon problemlerinin de temelini oluşturur ve bu yönüyle teorik bilginin pratik uygulamalarla olan bağını göstermek için de değerli bir örnektir.

1.3. Geleneksel Yaklaşımlar ve Pedagojik Sınırlılıkları

Hamilton döngüsü probleminin öğretiminde geleneksel olarak birkaç yaklaşım benimsenmektedir. Bunlardan ilki, Dodekahedral (Dodecahedral, On iki yüzlü) graf (Hamiltonian) veya Petersen graf (non-Hamiltonian) gibi küçük ve iyi bilinen örnekler üzerinde manuel analiz yapmaktır. Bu yöntem, problemin temel mantığını tanıtmak için faydalı olsa da öğrenciler daha karmaşık veya daha önce görmedikleri bir graf ile karşılaştıklarında genelleme yapmakta zorlanırlar. İkinci yaklaşım, backtracking gibi kaba kuvvet arama algoritmalarını öğretmektir. Bu algoritmalar, problemin hesaplamalı zorluğunu göstermede etkili olsa da büyük graflar için pratik değildir ve bir döngünün *neden var olmadığını* anlamak için sezgisel bir içgörü sunmazlar.

Bu noktada, görselleştirme [3, 4] bir çözüm gibi görünse de standart graf görselleştirme araçları da kendi sınırlılıklarını beraberinde getirir. Fruchterman-Reingold gibi kuvvete dayalı (force-directed) [5] yerleşim algoritmaları, genellikle estetik açıdan hoş ve simetrik görünen çizimler üretirler. Bu algoritmalar, düğümleri birbirine bağlı olanları çeken (yaylar gibi) ve olmayanları iten (elektiriksel kuvvetler gibi) bir fiziksel benzeşim temelinde konumlandırır. Bu yöntem, grafın kümelenme yapısını veya merkezi düğümleri göstermede başarılı olabilir; ancak, belirli bir yolu veya döngüyü takip etme görevini, yâni "yol okunabilirliğini" (path readability) ciddi şekilde zorlaştırır. Düğümlerin konumları öngörülemezdir ve bir döngüyü oluşturan kenarlar, grafın diğer kenarları arasında görsel bir karmaşa (visual clutter) içinde kaybolabilir. Dolayısıyla, öğrencinin bir Hamilton döngüsünün varlığını veya yokluğunu görsel olarak doğrulaması neredeyse imkânsız hâle gelir.

1.4. Bir Paradigma Değişimi: Kısıt Tabanlı Çözümleme ve Z3

Bu çalışmada önerdiğimiz yaklaşım, Hamilton döngüsü problemini çözmek için geleneksel algoritmik (imperative) paradigmadan uzaklaşarak, bildirimsel (deklaratif, declarative) bir paradigma benimsemektedir. "Bir döngüyü nasıl bulurum?" sorusunu sormak yerine, "Bir Hamilton döngüsü hangi özelliklere sâhip olmalıdır?" sorusunu soruyoruz. Bu yaklaşımın temelinde, Microsoft Research tarafından geliştirilen yüksek performanslı bir SMT (Satisfiability Modulo Theories: Karşılanabilirlik (tatmin edilebilirlik) Modülü Teorileri) çözücüsü olan Z3 [6–9] yatmaktadır.

SMT çözücüler [6–9], karmaşık mantıksal formüllerin belirli kısıtlar altında tatmin edilip edilemeyeceğini belirleyen güçlü araçlardır. Biz, Hamilton döngüsünün tanımını bir dizi matematiksel kısıta çevirerek Z3'e sunuyoruz. Bu kısıtlar temel olarak şunları ifâde eder:

1. Her düğüm, döngüdeki bir ve yalnızca bir pozisyonda yer almalıdır.
2. Döngüdeki her pozisyonda bir ve yalnızca bir düğüm bulunmalıdır.
3. Döngüde ardışık pozisyonlarda yer alan iki düğüm, orijinal graf üzerinde birbirine komşu olmalıdır (yani aralarında bir kenar bulunmalıdır).

Bu kurallar Z3'e verildiğinde, Z3, bu kısıtları aynı anda sağlayan bir atama olup olmadığını bulmak için gelişmiş sembolik akıl yürütme tekniklerini kullanır. Eğer böyle bir çözüm varsa, Z3 "tatmin edilebilir" (sat) sonucunu döndürür ve bu çözümü kanıtlayan bir model sunar. Eğer matematiksel olarak bir çözümün imkânsız olduğuna karar verirse, "tatmin edilemez" (unsat) sonucunu döndürür. Bu, bir döngünün var olmadığına dair kesin bir kanıt sunar ki bu, geleneksel arama algoritmalarının sağlayamadığı bir özelliktir. Bu deklaratif yaklaşım, öğrencilere problemi farklı bir soyutlama seviyesinde düşünme ve çözümü, onu inşâ etme adımlarından ziyâde, özellikler kümesi olarak modelleme yeteneği kazandırır.

1.4. Görselleştirmenin Kritik Rolü: Estetik Düzenlerin Ötesinde

Z3'ün sağladığı sat veya unsat cevabı, matematiksel olarak kesin olsa da pedagojik olarak tek başına yeterli değildir. Bir öğrencinin, özellikle bir döngü bulunduğunda, bu soyut çözümü somut olarak görmesi ve anlaması kritik öneme sâhiptir. Daha önce de belirtildiği gibi, standart yerleşim algoritmaları bu noktada yetersiz kalmaktadır. Bir yolun görsel tâkibi, düğümlerin mantıksal veya sıralı bir düzende konumlandırılmasını gerektirir. Kuvvete dayalı algoritmaların organik ve bâzen kaotik çıktıları, bu bilişsel gereksinimi karşılayamaz.

Bu sorunu çözmek için, bu çalışmanın ikinci temel yeniliği olan **Keçeci Dizilimi**'ni (Keçeci Yerleşimi, Keçeci Layout) [10–39] öneriyoruz. Bu yerleşim algoritması, estetik simetriden ziyade "okunabilirliği" önceliklendirir. Algoritma, grafin düğümlerini, genellikle sıralı bir kimliğe (ID) göre, belirlenmiş bir ana eksen (örneğin, soldan sağa) boyunca dizer. Ancak düğümleri tek bir çizgi üzerine yerleştirmek yerine, kenar karmaşasını azaltmak ve görsel ayrımı artırmak için ikincil bir eksen (örneğin, dikey eksen) boyunca alternatif bir zikzak (zigzag veya paralel yapılabilir) deseniyle hafifçe saptırır. Bu zikzaklı düzenleme, her düğümün genel konumunu korurken, kenarların üst üste binmesini azaltır ve özellikle sıralı yolların görsel olarak takip edilmesini son derece kolaylaştırır.

1.5. Önerilen Çözüm: Keçeci Dizilimi ile Bütünleşik Bir Çerçeve

Bu makalede, yukarıda açıklanan iki temel bileşeni bir araya getiren bütünleşik bir hesaplamalı ve görsel çerçeve sunulmaktadır. Sistemimiz, herhangi bir grafi girdi olarak alır, Z3'ü kullanarak bir Hamilton döngüsünün varlığını test eder ve sonuçları Keçeci Dizilimi'ni [10–39] kullanarak görselleştirir. Eğer bir döngü bulunursa, bu döngüyü oluşturan kenarlar, grafin geri kalanından ayırt edici bir renkte ve stilde vurgulanır. Keçeci Dizilimi'nin öngörülebilir ve sıralı yapısı sayesinde, öğrenci bu vurgulanmış yolu kolayca takip edebilir, döngünün her düğümden geçtiğini ve başlangıç noktasına döndüğünü net bir şekilde görebilir. Bir döngü bulunmadığında ise, sistem, Z3'ün sağladığı kesin unsat sonucunu bildirir ve öğrenciye grafin bu özelliğe sahip olmadığına dair güvenilir bir geri bildirim sunar.

Bu yaklaşım, öğrencilere sadece Hamilton döngüsü probleminin ne olduğunu değil, aynı zamanda onun hesaplamalı doğasını, mantıksal modellemenin gücünü ve etkili bir görselleştirmenin soyut bir kanıtı nasıl sezgisel bir anlayışa dönüştürebileceğini de öğretir. Bu araç, zorlu NP-tam problemlerin öğretiminde, teorik derinlik ile pratik keşif arasında bir denge kurarak, öğrencilerin hem yetkinliğini hem de motivasyonunu artırma potansiyeline sahiptir.

II. Hamilton Döngüsünün Testi ve Garfı

```
# pip install -U kececilayout==0.3.7
```

Python modülü güncellendikçe kodların kullanımında yıkıcı değişikliklerden dolayı verilen sürüm numarası kodun çalışmasını garanti eder. Kodu bu sürümünde test ettikten sonra güncelleyebilirsiniz fakat bu yapılan güncelleme kodun çalışmasını garanti etmez.

```
# -*- coding: utf-8 -*-
"""
```

Bu betik, Z3 SMT çözücüsünü kullanarak bir graftaki Hamilton döngüsünü bulur ve görselleştirir.
Bu sürüm, hem mantıksal olarak doğru hem de çok daha verimli bir Z3 modellemesi kullanır.

```
"""
```

```
from typing import Dict, List, Optional
import networkx as nx
import matplotlib.pyplot as plt
from z3 import Solver, Int, And, Distinct, sat, ModelRef, Implies, Or

def graf_gecerli_mi(graf: Dict[int, List[int]]) -> bool:
    """
    Graf yapısını doğrular:
    - Tüm düğümlerin bağlı olduğunu (izole alt graflar olmadığını) kontrol eder.
    - Komşuluk listelerinin simetrik olmasını sağlar (A, B'nin komşusu ise B de A'nın komşusu olmalıdır).
    """
    if not graf:
        return False

    # Komşuluk listelerinin simetrisini kontrol et
    for dugum, komsular in graf.items():
        for komsu in komsular:
            if dugum not in graf.get(komsu, []):
                print(f"Hata: {dugum} ve {komsu} arasında simetrik olmayan kenar")
                return False

    # Bağlılığı kontrol et (BFS kullanarak)
    ziyaret_edilenler = set()
    kuyruk = [next(iter(graf.keys()))]

    while kuyruk:
        mevcut = kuyruk.pop(0)
        if mevcut not in ziyaret_edilenler:
            ziyaret_edilenler.add(mevcut)
            kuyruk.extend([n for n in graf[mevcut] if n not in ziyaret_edilenler])

    if len(ziyaret_edilenler) != len(graf):
        print("Hata: Graf tam olarak bağlı değil")
        return False

    return True

def hamilton_kisitlari_olustur_hizli(graf: Dict[int, List[int]]) -> Optional[Solver]:
    """
    Bir Hamilton döngüsü bulmak için Z3'e yönelik verimli kısıtlar oluşturur.
    Bu model, her pozisyonda hangi düğümün olduğunu tanımlar.
    Graf geçersizse None döndürür.
    """
    if not graf_gecerli_mi(graf):
        return None

    dugum_sayisi = len(graf)
    tum_dugumler = list(graf.keys())
    cozucu = Solver()
```

```

yol = [Int(f'yol_{i}') for i in range(dugum_sayisi)]

cozucu.add([And(y >= 0, y < dugum_sayisi) for y in yol])
cozucu.add(Distinct(yol))

for i in range(dugum_sayisi):
    sonraki_pozisyon = (i + 1) % dugum_sayisi
    for d in tum_dugumler:
        komsuluk_kisiti = Or([yol[sonraki_pozisyon] == komsu for komsu in graf[d]])
        cozucu.add(Implies(yol[i] == d, komsuluk_kisiti))

cozucu.add(yol[0] == tum_dugumler[0])
return cozucu

def donguyu_gorsellestir(graf: Dict[int, List[int]], model: ModelRef) -> None:
    """
    Grafı görselleştirir ve bulunan Hamilton döngüsünü NetworkX ile vurgular.
    """
    G = nx.from_dict_of_lists(graf)

    dugum_sayisi = len(graf)
    dongu_yolu = [model.eval(Int(f'yol_{i}')).as_long() for i in range(dugum_sayisi)]
    dongu_yolu.append(dongu_yolu[0])

    pozisyon = nx.spring_layout(G, seed=42)
    nx.draw_networkx(G, pozisyon, with_labels=True, node_color='lightblue')

    nx.draw_networkx_edges(
        G, pozisyon,
        edgelist=list(zip(dongu_yolu[:-1], dongu_yolu[1:])),
        edge_color='red', width=2
    )
    plt.title("Hamilton Döngüsü (Kırmızı Kenarlar)")

    # --- YENİ EKLENEN SATIR ---
    plt.axis('off') # Eksen çizgilerini ve çerçeveyi kaldırır
    # -----

    plt.show()

def test_orneklerini_calistir():
    """Dodecahedral (Hamiltonian) ve Herschel (Hamiltonian olmayan) graflarla test eder."""
    dodecahedral_graf = {0: [1, 4, 5], 1: [0, 7, 2], 2: [1, 9, 3], 3: [2, 11, 4], 4: [3, 13,
0], 5: [0, 14, 6], 6: [5, 16, 7], 7: [6, 8, 1], 8: [7, 17, 9], 9: [8, 10, 2], 10: [9, 18,
11], 11: [10, 3, 12], 12: [11, 19, 13], 13: [12, 14, 4], 14: [13, 15, 5], 15: [14, 16, 19],
16: [6, 17, 15], 17: [16, 8, 18], 18: [10, 19, 17], 19: [18, 12, 15]}
    herschel_graf = {0: [1, 7, 9, 10], 1: [0, 2, 8], 2: [1, 3, 9], 3: [2, 4, 8], 4: [3, 5, 9,
10], 5: [4, 6, 8], 6: [5, 7, 10], 7: [0, 6, 8], 8: [1, 3, 5, 7], 9: [0, 2, 4], 10: [0, 4, 6]}

    print("Dodecahedral Grafı Test Ediliyor:")
    cozucu_dodec = hamilton_kisitlari_olustur_hizli(dodecahedral_graf)
    if cozucu_dodec:
        if cozucu_dodec.check() == sat:
            model = cozucu_dodec.model()
            print("Hamilton Döngüsü Bulundu!")
            donguyu_gorsellestir(dodecahedral_graf, model)
        else:
            print("Hamilton Döngüsü Tespit Edilmedi")

```



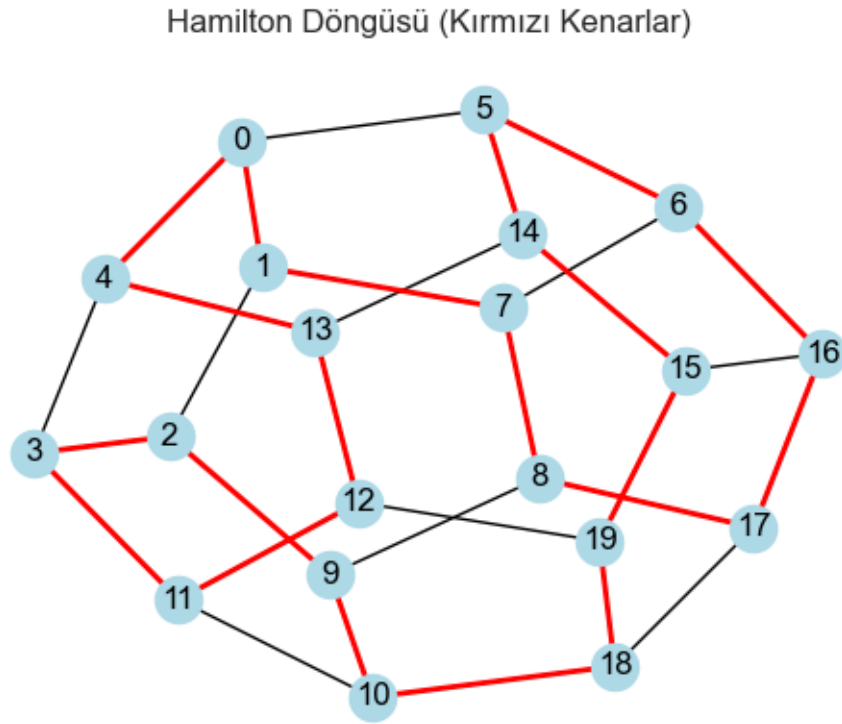
```

print("\nHerschel Grafı Test Ediliyor:")
cozucu_herschel = hamilton_kisitlari_olustur_hizli(herschel_graf)
if cozucu_herschel:
    sonuc = cozucu_herschel.check()
    print(f"Hamilton Döngüsü Var mı? {sonuc == sat}")

if __name__ == "__main__":
    test_orneklerini_calistir()

```

Liste 1: Hamilton Döngüsü Testi ve Grafi



Şekil 1: Hamilton Döngüsü Grafi

III. Keçeci Dizilimi ile Hamilton Döngüleri

```

# -*- coding: utf-8 -*-
"""

```

Bu betik, Z3 SMT çözücüsünü kullanarak bir graftaki Hamilton döngüsünü bulur ve görselleştirir. Graf görselleştirmesi NetworkX ve Matplotlib ile yapılırken, düğüm konumlandırması Keçeci Dizilim (Keçeci Layout) algoritması ile belirlenir.

Bu sürüm, Z3 için daha verimli bir modelleme tekniği kullanır.

```

from typing import Dict, List, Optional
import networkx as nx
import matplotlib.pyplot as plt

```



```

from z3 import Solver, Int, And, Distinct, sat, ModelRef, Implies, Or

# Kececilayout'un özel veya üçüncü taraf bir modül olduğu varsayılmaktadır.
try:
    import kececilayout as kl
except ImportError:
    print("Uyarı: kececilayout modülü bulunamadı. NetworkX'in circular_layout dizilimine geri dönülüyor.")
    def yedek_dizilim(G, **kwargs):
        return nx.circular_layout(G)
    kl = type('module', (object,), {'kececi_layout': yedek_dizilim})

def graf_gecerli_mi(graf: Dict[int, List[int]]) -> bool:
    """
    Graf yapısını doğrular; boş olmadığını, yönsüz (simetrik) olduğunu
    ve bağlı (connected) olduğunu kontrol eder.
    """
    if not graf:
        print("Hata: Graf boş.")
        return False
    for dugum, komsular in graf.items():
        for komsu in komsular:
            if dugum not in graf.get(komsu, []):
                print(f"Hata: {dugum} ve {komsu} arasında simetrik olmayan kenar.")
                return False
    ziyaret_edilenler = set()
    kuyruk = [next(iter(graf.keys()))]
    ziyaret_edilenler.add(kuyruk[0])
    while kuyruk:
        mevcut_dugum = kuyruk.pop(0)
        for komsu in graf.get(mevcut_dugum, []):
            if komsu not in ziyaret_edilenler:
                ziyaret_edilenler.add(komsu)
                kuyruk.append(komsu)
    if len(ziyaret_edilenler) != len(graf):
        print("Hata: Graf bağlı değil.")
        return False
    return True

def hamilton_kisitlari_olustur_hizli(graf: Dict[int, List[int]]) -> Optional[Solver]:
    """
    Bir Hamilton döngüsü bulmak için Z3'e yönelik verimli kısıtlar oluşturur.
    Bu model, her pozisyonda hangi düğümün olduğunu tanımlar.

    Args:
        graf: Komşuluk listesi olarak temsil edilen geçerli bir graf.

    Returns:
        Kısıtları içeren bir Z3 çözücü nesnesi veya graf geçersizse None.
    """
    if not graf_gecerli_mi(graf):
        return None

    dugum_sayisi = len(graf)
    tum_dugumler = list(graf.keys())
    cozucu = Solver()

```

```

# Adım 1: Döngüdeki her pozisyon için bir değişken oluştur.
# yol[i], döngünün i. pozisyonundaki düğümün ID'sidir.
yol = [Int(f'yol_{i}') for i in range(dugum_sayisi)]

# Kısıt 1: Yoldaki her eleman geçerli bir düğüm olmalıdır.
cozucu.add([And(y >= 0, y < dugum_sayisi) for y in yol])

# Kısıt 2: Her düğüm yolda yalnızca bir kez görünmelidir.
cozucu.add(Distinct(yol))

# Kısıt 3 (Bitişiklik): Yoldaki ardışık düğümler graf üzerinde komşu olmalıdır.
# "Eğer yol[i] 'd' düğümü ise, yol[i+1] 'd'nin komşularından biri olmalıdır."
for i in range(dugum_sayisi):
    for d in tum_dugumler:
        # Döngünün son elemanının ilk elemana bağlı olması gerekir.
        sonraki_pozisyon = (i + 1) % dugum_sayisi

        # d'nin komşuları boşsa, bu düğümden bir yol geçemez.
        if not graf[d]:
            komsuluk_kisiti = False
        else:
            komsuluk_kisiti = Or([yol[sonraki_pozisyon] == komsu for komsu in graf[d]])

        cozucu.add(Implies(yol[i] == d, komsuluk_kisiti))

# Kısıt 4 (Simetriyi Kırma): Aramayı hızlandırmak için başlangıç düğümünü sabitle.
cozucu.add(yol[0] == tum_dugumler[0])

return cozucu

def hamilton_dongusunu_gorsellestir(graf: Dict[int, List[int]], model: ModelRef) -> None:
    """
    Grafı görselleştirir ve bulunan Hamilton döngüsünü vurgular.
    """
    G = nx.from_dict_of_lists(graf)
    poz = kl.kececi_layout(G, primary_spacing=1.5, secondary_spacing=0.8)

    # Z3 modelinden döngü yolunu çıkar
    # Model zaten pozisyona göre sıralı düğümleri verir
    dugum_sayisi = len(graf)
    dongu_yolu_dugumleri = [model.eval(Int(f'yol_{i}')).as_long() for i in
range(dugum_sayisi)]

    # Görselleştirme için döngüyü kapat
    dongu_yolu_dugumleri.append(dongu_yolu_dugumleri[0])
    dongu_kenarlari = list(zip(dongu_yolu_dugumleri[:-1], dongu_yolu_dugumleri[1:]))

    nx.draw_networkx(G, poz, with_labels=True, node_color='lightblue', node_size=700,
font_weight='bold')
    nx.draw_networkx_edges(G, poz, edgelist=dongu_kenarlari, edge_color='red', width=2.5,
alpha=0.8)

    plt.title("Keçeci Dizilimi ile Hamilton Döngüsü (Hızlı Model)")
    plt.axis('off')
    plt.show()

def test_orneklerini_calistir():
    """

```

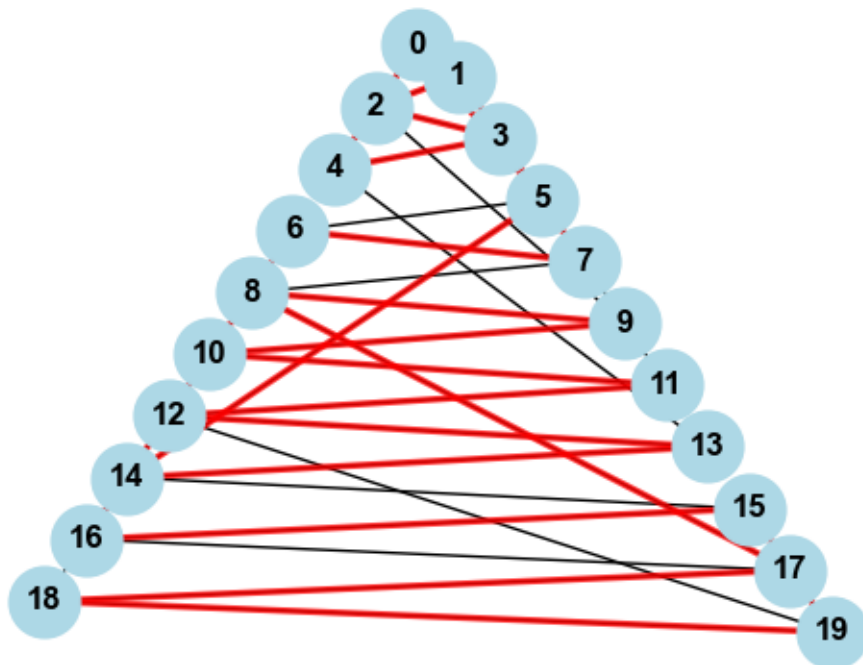
```
Test senaryolarını çalıştırır.
"""
dodecahedral_graf = {
    0: [1, 4, 5], 1: [0, 7, 2], 2: [1, 9, 3], 3: [2, 11, 4], 4: [3, 13, 0],
    5: [0, 14, 6], 6: [5, 16, 7], 7: [6, 8, 1], 8: [7, 17, 9], 9: [8, 10, 2],
    10: [9, 18, 11], 11: [10, 3, 12], 12: [11, 19, 13], 13: [12, 14, 4],
    14: [13, 15, 5], 15: [14, 16, 19], 16: [6, 17, 15], 17: [16, 8, 18],
    18: [10, 19, 17], 19: [18, 12, 15]
}

print("--- Dodecahedral Graf Test Ediliyor (Hızlı Model) ---")
cozucu = hamilton_kisitlari_olustur_hizli(dodecahedral_graf) # Hızlı fonksiyonu çağır
if cozucu:
    print("Z3 çözücüsü çalışıyor, bu işlem birkaç saniye sürebilir...")
    if cozucu.check() == sat:
        print("Hamilton döngüsü bulundu!")
        model = cozucu.model()
        hamilton_dongusunu_gorsellestir(dodecahedral_graf, model)
    else:
        print("Bu graf için Hamilton döngüsü bulunamadı.")

if __name__ == "__main__":
    test_orneklerini_calistir()
```

Liste 2: Keçeci dizilimi ile Hamilton döngüleri test kodu ve grafi

Keçeci Dizilimi ile Hamilton Döngüsü (Hızlı Model)



Şekil 2: Keçeci dizilimi ile Hamilton döngüsü grafi

Bu graf bize istediğimiz netliği verememiştir. Bu yüzden Keçeci Dizilimini bir adım daha ileriye taşıyorum.

```
# -*- coding: utf-8 -*-
"""
```

Bu betik, Z3 SMT çözücüsünü kullanarak bir graftaki Hamilton döngüsünü bulur ve görselleştirir.

Bu sürüm, mantıksal olarak doğru ve yüksek performanslı bir Z3 modellemesi kullanır ve görselleştirme için kececilayout modülünü entegre eder.

```
"""
```

```
from typing import Dict, List, Optional
import networkx as nx
import matplotlib.pyplot as plt
from z3 import Solver, Int, And, Distinct, sat, ModelRef, Implies, Or
```

```
# Kececilayout modülünü içe aktar
```

```
try:
```

```
    import kececilayout as kl
```

```
except ImportError:
```

```
    print("Uyarı: kececilayout modülü bulunamadı. NetworkX'in dairesel dizilimine geri dönülüyor.")
```

```
    kl = None
```

```
def graf_gecerli_mi(graf: Dict[int, List[int]]) -> bool:
```

```
    """Graf yapısını doğrular (simetri ve bağlantı kontrolü)."""
```

```
    if not graf:
```

```
        return False
```

```
    # Simetri kontrolü
```

```
    for dugum, komsular in graf.items():
```

```
        for komsu in komsular:
```

```
            if dugum not in graf.get(komsu, []):
```

```
                print(f"[Hata] Asimetrik kenar: {dugum} ↔ {komsu}")
```

```
                return False
```

```
    # Bağlantı kontrolü (BFS)
```

```
    ziyaret_edilenler = set()
```

```
    kuyruk = [next(iter(graf.keys()))]
```

```
    ziyaret_edilenler.add(kuyruk[0]) # Daha verimli BFS için burada ekle
```

```
    head = 0
```

```
    while head < len(kuyruk):
```

```
        mevcut = kuyruk[head]
```

```
        head += 1
```

```
        for komsu in graf[mevcut]:
```

```
            if komsu not in ziyaret_edilenler:
```

```
                ziyaret_edilenler.add(komsu)
```

```
                kuyruk.append(komsu)
```

```
    return len(ziyaret_edilenler) == len(graf)
```

```
def hamilton_kisitlari_olustur_hizli(graf: Dict[int, List[int]]) -> Optional[Solver]:
```

```
    """
```

Bir Hamilton döngüsü bulmak için Z3'e yönelik verimli ve doğru kısıtlar oluşturur.

Bu model, her pozisyonda hangi düğümün olduğunu tanımlar.

```
    """
```

```
    if not graf_gecerli_mi(graf):
```

```
        return None
```

```

dugum_sayisi = len(graf)
tum_dugumler = list(graf.keys())
cozucu = Solver()

# Adım 1: Döngüdeki her pozisyon için bir değişken oluştur ('yol').
# yol[i], döngünün i. pozisyonundaki düğümün ID'sidir.
yol = [Int(f'yol_{i}') for i in range(dugum_sayisi)]

# Kısıt 1: Yoldaki her eleman geçerli bir düğüm olmalıdır.
cozucu.add([And(y >= 0, y < dugum_sayisi) for y in yol])

# Kısıt 2 (DOĞRULUK): Her düğüm yolda yalnızca bir kez görünmelidir.
cozucu.add(Distinct(yol))

# Kısıt 3 (PERFORMANS): Yoldaki ardışık düğümler graf üzerinde komşu olmalıdır.
for i in range(dugum_sayisi):
    sonraki_pozisyon = (i + 1) % dugum_sayisi
    for d in tum_dugumler:
        komsuluk_kisiti = Or([yol[sonraki_pozisyon] == komsu for komsu in graf[d]])
        cozucu.add(Implies(yol[i] == d, komsuluk_kisiti))

# Kısıt 4 (Hızlandırma): Başlangıç düğümünü sabitleyerek simetriyi kır.
cozucu.add(yol[0] == tum_dugumler[0])

return cozucu

def donguyu_gorsellestir_kececi(graf: Dict[int, List[int]], model: ModelRef) -> None:
    """Kececi Layout ile Hamilton döngüsünü görselleştirir."""
    G = nx.from_dict_of_lists(graf)

    # Kececi Layout kullanılamıyorsa, NetworkX'in standart bir dizilimine geri dön
    if kl:
        pos = kl.kececi_layout(
            G,
            primary_spacing=1.5,
            secondary_spacing=0.8,
            primary_direction='left-to-right',
            secondary_start='up'
        )
    else:
        pos = nx.circular_layout(G)

    # Modelden döngü sırasını çıkar (yeni modele göre)
    dugum_sayisi = len(graf)
    dongu_yolu = [model.eval(Int(f'yol_{i}')).as_long() for i in range(dugum_sayisi)]
    dongu_yolu.append(dongu_yolu[0]) # Döngüyü kapat

    # Çizim
    plt.figure(figsize=(12, 8))
    nx.draw_networkx(
        G, pos,
        node_color='#1f78b4',
        node_size=800,
        font_color='white',
        font_weight='bold',
        with_labels=True
    )

```

```

# Hamilton döngüsünü vurgula
nx.draw_networkx_edges(
    G, pos,
    edgelist=list(zip(dongu_yolu[:-1], dongu_yolu[1:])),
    edge_color='#ff7f0e',
    width=3,
    alpha=0.8,
    style='dashed'
)

plt.title("Keçeci Layout ile Hamilton Döngüsü", pad=20, fontsize=14)
plt.axis('off')
plt.tight_layout()
plt.show()

def test_hamiltonian_graflari():
    """Test graf örneklerini çalıştırır."""
    gr_dodec = {0: [1, 4, 5], 1: [0, 7, 2], 2: [1, 9, 3], 3: [2, 11, 4], 4: [3, 13, 0], 5:
    [0, 14, 6], 6: [5, 16, 7], 7: [6, 8, 1], 8: [7, 17, 9], 9: [8, 10, 2], 10: [9, 18, 11], 11:
    [10, 3, 12], 12: [11, 19, 13], 13: [12, 14, 4], 14: [13, 15, 5], 15: [14, 16, 19], 16: [6,
    17, 15], 17: [16, 8, 18], 18: [10, 19, 17], 19: [18, 12, 15]}
    gr_herschel = {0: [1, 7, 9, 10], 1: [0, 2, 8], 2: [1, 3, 9], 3: [2, 4, 8], 4: [3, 5, 9,
    10], 5: [4, 6, 8], 6: [5, 7, 10], 7: [0, 6, 8], 8: [1, 3, 5, 7], 9: [0, 2, 4], 10: [0, 4, 6]}

    for isim, gr in [("Dodecahedral", gr_dodec), ("Herschel", gr_herschel)]:
        print(f"\n{isim} grafi test ediliyor...")
        # HIZLI VE DOĞRU MODELİ KULLAN
        cozucu = hamilton_kisitlari_olustur_hizli(gr)
        if not cozucu:
            print(" ⚠ Geçersiz graf yapısı!")
            continue

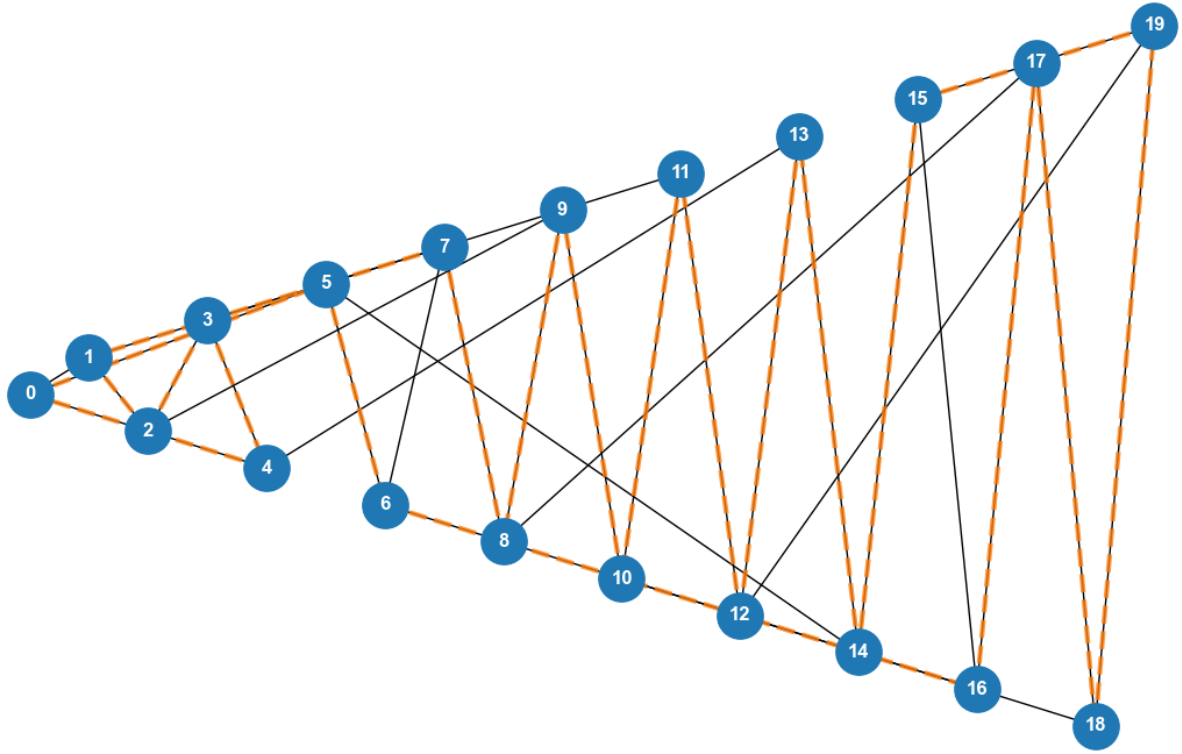
        if cozucu.check() == sat:
            print(" ✅ Hamilton döngüsü bulundu!")
            donguyu_gorsellestin_kececi(gr, cozucu.model())
        else:
            print(f" ❌ {isim} grafında Hamilton döngüsü yok")

if __name__ == "__main__":
    test_hamiltonian_graflari()

```

Liste 3: Keçeci dizilimi ile Hamilton döngüsü testi ve grafi

Keçeci Layout ile Hamilton Döngüsü



Şekil 3: Keçeci dizilimi ile Hamilton döngüsü grafi

Bu graf bize istediğimiz netliği verememiştir. Bu yüzden Keçeci Dizilimini bir adım daha ileriye taşıyorum.

```
# -*- coding: utf-8 -*-
"""
```

Z3 ve Gelişmiş Görselleştirme ile Hamilton Döngüsü Çözücü.

Bu modül, graflar oluşturmak, Z3 teoremi kanıtlayıcısını kullanarak Hamilton döngülerini kontrol etmek ve sonuçları Keçeci Dizilimi ile görselleştirmek için araçlar sağlar. Kararlı ve verimli bir Z3 formülasyonunu sağlam yardımcı fonksiyonlarla birleştirir.

```
"""
```

```
# --- Standart Kütüphane Modülleri ---
from functools import lru_cache, wraps
from typing import Dict, List, Optional, Any, Callable
```

```
# --- Üçüncü Parti Kütüphane Modülleri ---
import networkx as nx
from networkx.generators import small as nxs
import matplotlib.pyplot as plt
from z3 import Solver, Int, Or, And, Distinct, sat, Implies, ModelRef
```

```
# --- Yerel Kütüphane ---
```



```

try:
    import keccilayout as kl
except ImportError:
    print("Uyarı: 'keccilayout' modülü bulunamadı.")
    kl = None

# =====
# 1. GRAF YARDIMCI FONKSİYONLARI
# =====

def _grafi_nx_cevir(fonksiyon: Callable) -> Callable:
    """Sözlük tabanlı grafi otomatik olarak NetworkX nesnesine dönüştüren dekoratör."""
    @wraps(fonksiyon)
    def sarmalayici(graf_girdisi: Any, *args, **kwargs) -> Any:
        G = graf_girdisi if isinstance(graf_girdisi, nx.Graph) else nx.Graph(graf_girdisi)
        return fonksiyon(G, *args, **kwargs)
    return sarmalayici

@_grafi_nx_cevir
def graf_gecerli_mi(G: nx.Graph) -> bool:
    """Bir grafin boş olmadığını ve bağlı olup olmadığını NetworkX ile kontrol eder."""
    return G.number_of_nodes() > 0 and nx.is_connected(G)

def karmalanabilir_tuple_cevir(graf: Dict[int, List[int]]) -> tuple:
    """Bir graf sözlüğünü önbellekleme için karmalanabilir bir demete dönüştürür."""
    return tuple(sorted((k, tuple(sorted(v))) for k, v in graf.items()))

# =====
# 2. HAMILTON DÖNGÜSÜ ÇÖZÜCÜ (İYİLEŞTİRİLMİŞ FORMÜLASYON)
# =====

@lru_cache(maxsize=128)
def hamilton_cozucu_olustur(graf_demeti: tuple) -> Optional[Solver]:
    """
    Bir Hamilton döngüsü için Z3 kısıtları oluşturur ve önbelleğe alır.
    Daha verimli ve okunabilir olan 'Implies' tabanlı bir formülasyon kullanır.
    """
    graf = dict(graf_demeti)
    if not graf_gecerli_mi(graf):
        return None

    dugumler = sorted(graf.keys())
    dugum_sayisi = len(dugumler)

    if dugum_sayisi == 0:
        return None

    cozuclu = Solver()
    # pozisyon[i], döngünün i. pozisyonundaki düğümü temsil eder.
    pozisyon = [Int(f'pos_{i}') for i in range(dugum_sayisi)]

    # Kısıt 1: Her pozisyonda farklı ve geçerli bir düğüm olmalıdır.
    cozuclu.add(Distinct(pozisyon))
    for i in range(dugum_sayisi):
        cozuclu.add(Or([pozisyon[i] == dugum for dugum in dugumler]))

    # Kısıt 2 (İYİLEŞTİRİLMİŞ): Ardışık düğümler graf üzerinde komşu olmalıdır.
    for i in range(dugum_sayisi):

```

```

mevcut_dugum = pozisyon[i]
sonraki_dugum = pozisyon[(i + 1) % dugum_sayisi]

for dugum, komsular in graf.items():
    if not komsular: continue
    # "Eğer mevcut pozisyondaki düğüm 'dugum' ise, sonraki pozisyondaki düğüm
    # onun komşularından biri olmalıdır."
    komsuluk_kisiti = Or([sonraki_dugum == komsu for komsu in komsular])
    cozucu.add(Implies(mevcut_dugum == dugum, komsuluk_kisiti))

return cozucu

# =====
# 3. GÖRSELLEŞTİRME
# =====

def cozumu_gorsellestir(graf: Dict[int, List[int]], model: ModelRef) -> None:
    """Grafı ve bulunan döngüyü Keçeci Diziliminin 'curved' stiliyle görselleştirir."""
    G = nx.Graph(graf)

    dizilim_parametreleri = {'primary_spacing': 1.5, 'secondary_spacing': 0.8,
'primary_direction': 'left-to-right', 'secondary_start': 'up'}
    pozisyonlar = kl.kececi_layout(G, **dizilim_parametreleri)

    dugum_sayisi = G.number_of_nodes()
    dongu_yolu = [model.evaluate(Int(f'pos_{i}'))].as_long() for i in range(dugum_sayisi)]
    dongu_yolu.append(dongu_yolu[0])
    dongu_kenarlari = list(zip(dongu_yolu[:-1], dongu_yolu[1:]))

    fig, ax = plt.subplots(figsize=(12, 8))

    kl.draw_kececi(G, style='curved', ax=ax, node_color='#1f78b4', edge_color='grey',
alpha=0.6, **dizilim_parametreleri)

    nx.draw_networkx_edges(G, pozisyonlar, ax=ax, edgelist=dongu_kenarlari,
edge_color='#ff7f0e', width=3, style='dashed', connectionstyle='arc3,rad=0.2', arrows=True)

    nx.draw_networkx_nodes(G, pozisyonlar, ax=ax, node_color='#1f78b4', node_size=800)
    nx.draw_networkx_labels(G, pozisyonlar, ax=ax, font_color='white', font_weight='bold')

    ax.set_title("Keçeci Dizilimi ile Hamilton Döngüsü", pad=20, fontsize=14)
    plt.axis('off')
    plt.tight_layout()
    plt.show()

# =====
# 4. ANA ÇALIŞTIRMA VE TESTLER
# =====

def test_graflarini_calistir():
    """Önceden tanımlanmış graflar üzerinde Hamilton döngüsü testlerini çalıştırır."""
    gr_dodec = nx.to_dict_of_lists(nxs.dodecahedral_graph())

    gr_herschel = nx.to_dict_of_lists(nx.Graph([(0, 1), (0, 7), (0, 9), (0, 10), (1, 2), (1,
8), (2, 3), (2, 9), (3, 4), (3, 8), (4, 5), (4, 9), (4, 10), (5, 6), (5, 8), (6, 7), (6, 10),
(7, 8)]))

    for isim, gr in [("Dodecahedral", gr_dodec), ("Herschel", gr_herschel)]:

```

```

print(f"\n--- {isim} Grafı Test Ediliyor ---")
cozucu = hamilton_cozucu_olustur(karmalanabilir_tuple_cevir(gr))

if not cozucu:
    print(" ⚠ Geçersiz graf yapısı, atlanıyor.")
    continue

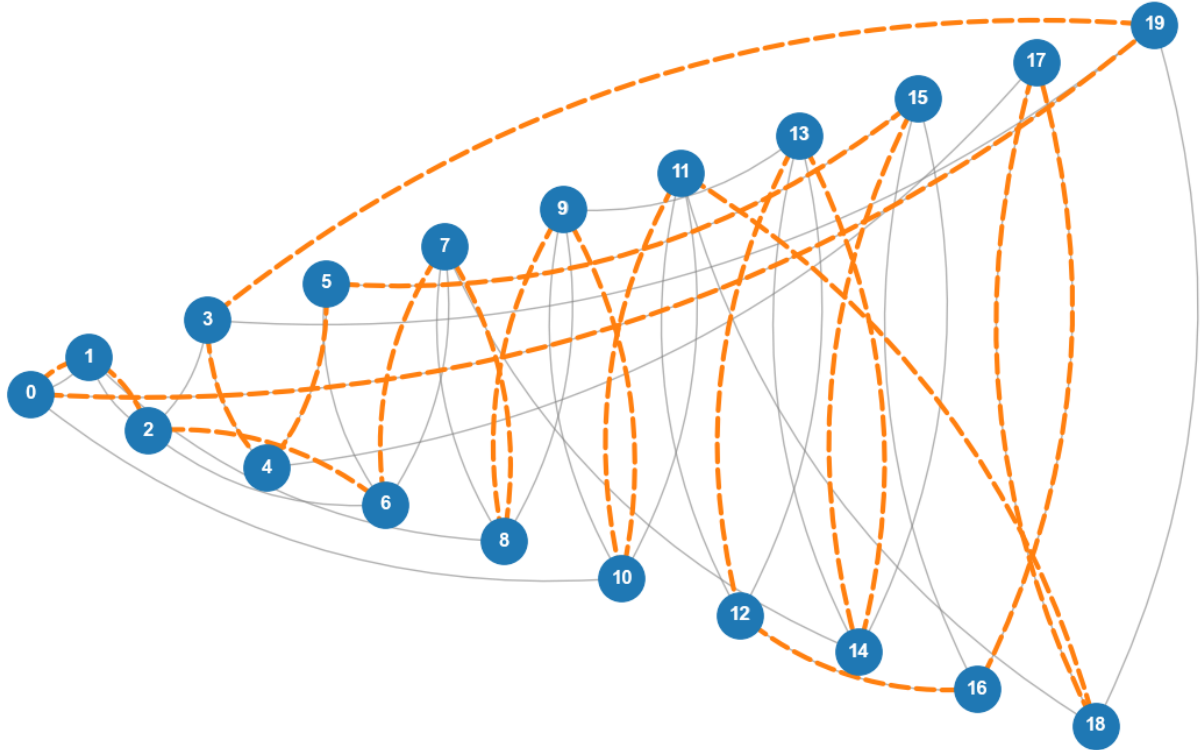
if cozucu.check() == sat:
    print(" ✅ Hamilton döngüsü BULUNDU!")
    cozum_gorsellestir(gr, cozucu.model())
else:
    print(f" ❌ {isim} grafında Hamilton döngüsü mevcut değil.")

if __name__ == "__main__":
    test_graflarini_calistir()

```

Liste 4: Keçeci dizilimi ile Hamilton döngüsü testi ve graf kodu

Keçeci Dizilimi ile Hamilton Döngüsü



Şekil 4: Keçeci dizilimi ile Hamilton döngüsü grafı

Bu Keçeci Dizilimi grafı bize istediğimizi verdi sâdece çizgilerin yönlerini göstermediği için onu da ekleyerek daha da anlaşılır olsun.

```
# -*- coding: utf-8 -*-
"""
```

Z3 ve Gelişmiş Görselleştirme ile Hamilton Döngüsü Çözücü.

Bu modül, graflar oluşturmak, Z3 teoremi kanıtlayıcısını kullanarak Hamilton döngülerini kontrol etmek ve sonuçları Keçeci Dizilimi ile görselleştirmek için araçlar sağlar. Kararlı ve verimli bir Z3 formülasyonunu sağlam yardımcı fonksiyonlarla birleştirir.

```
"""
```

```
# --- Standart Kütüphane Modülleri ---
```

```
from functools import lru_cache, wraps
from typing import Dict, List, Optional, Any, Callable
```

```
# --- Üçüncü Parti Kütüphane Modülleri ---
```

```
import networkx as nx
from networkx.generators import small as nxs
import matplotlib.pyplot as plt
from z3 import Solver, Int, Or, Distinct, sat, Implies, ModelRef
```

```
# --- Yerel Kütüphane ---
```

```
try:
    import kececilayout as kl
except ImportError:
    print("Uyarı: 'kececilayout' modülü bulunamadı.")
    kl = None
```

```
# =====
# 1. GRAF YARDIMCI FONKSİYONLARI
# =====
```

```
def _grafi_nx_cevir(fonksiyon: Callable) -> Callable:
    """Sözlük tabanlı grafi otomatik olarak NetworkX nesnesine dönüştüren dekoratör."""
    @wraps(fonksiyon)
    def sarmalayici(graf_girdisi: Any, *args, **kwargs) -> Any:
        G = graf_girdisi if isinstance(graf_girdisi, nx.Graph) else nx.Graph(graf_girdisi)
        return fonksiyon(G, *args, **kwargs)
    return sarmalayici
```

```
@_grafi_nx_cevir
def graf_gecerli_mi(G: nx.Graph) -> bool:
    """Bir grafin boş olmadığını ve bağlı olup olmadığını NetworkX ile kontrol eder."""
    return G.number_of_nodes() > 0 and nx.is_connected(G)
```

```
def karmalanabilir_tuple_cevir(graf: Dict[int, List[int]]) -> tuple:
    """Bir graf sözlüğünü önbellekleme için karmalanabilir bir demete dönüştürür."""
    return tuple(sorted((k, tuple(sorted(v))) for k, v in graf.items()))
```

```
# =====
# 2. HAMILTON DÖNGÜSÜ ÇÖZÜCÜ (İYİLEŞTİRİLMİŞ FORMÜLASYON)
# =====
```

```
@lru_cache(maxsize=128)
def hamilton_cozucu_olustur(graf_demeti: tuple) -> Optional[Solver]:
    """
```

Bir Hamilton döngüsü için Z3 kısıtları oluşturur ve önbelleğe alır.
Daha verimli ve okunabilir olan 'Implies' tabanlı bir formülasyon kullanır.

```
"""
```

```

graf = dict(graf_demeti)
if not graf_gecerli_mi(graf):
    return None

dugumler = sorted(graf.keys())
dugum_sayisi = len(dugumler)

if dugum_sayisi == 0:
    return None

cozucu = Solver()
# pozisyon[i], döngünün i. pozisyonundaki düğümü temsil eder.
pozisyon = [Int(f'pos_{i}') for i in range(dugum_sayisi)]

# Kısıt 1: Her pozisyonunda farklı ve geçerli bir düğüm olmalıdır.
cozucu.add(Distinct(pozisyon))
for i in range(dugum_sayisi):
    cozucu.add(Or([pozisyon[i] == dugum for dugum in dugumler]))

# Kısıt 2 (İYİLEŞTİRİLMİŞ): Ardışık düğümler graf üzerinde komşu olmalıdır.
for i in range(dugum_sayisi):
    mevcut_dugum = pozisyon[i]
    sonraki_dugum = pozisyon[(i + 1) % dugum_sayisi]

    for dugum, komsular in graf.items():
        if not komsular: continue
        komsuluk_kisiti = Or([sonraki_dugum == komşu for komşu in komsular])
        cozucu.add(Implies(mevcut_dugum == dugum, komsuluk_kisiti))

return cozucu

# =====
# 3. GÖRSELLEŞTİRME (OK İŞARETLERİ VE RENKLENDİRME İLE)
# =====

def cozumu_gorsellestir(graf: Dict[int, List[int]], model: ModelRef) -> None:
    """Grafı ve bulunan Hamilton döngüsünü yönlü, renkli oklarla görselleştirir."""
    G = nx.Graph(graf)

    dizilim_parametreleri = {'primary_spacing': 1.5, 'secondary_spacing': 0.8,
                             'primary_direction': 'left-to-right', 'secondary_start': 'up'}
    pozisyonlar = kl.kececi_layout(G, **dizilim_parametreleri)

    dugum_sayisi = G.number_of_nodes()
    dongu_yolu = [model.evaluate(Int(f'pos_{i}')).as_long() for i in range(dugum_sayisi)]
    dongu_yolu.append(dongu_yolu[0])
    dongu_kenarlari = list(zip(dongu_yolu[:-1], dongu_yolu[1:]))

    fig, ax = plt.subplots(figsize=(14, 9))

    kl.draw_kececi(G, style='curved', ax=ax, node_color='#1f78b4', edge_color='lightgray',
                  alpha=0.7, **dizilim_parametreleri)

    nx.draw_networkx_edges(G, pozisyonlar, ax=ax, edgelist=dongu_kenarlari,
                          edge_color='#e41a1c', width=2.5, style='solid', connectionstyle='arc3,rad=0.2', arrows=True,
                          arrowstyle='->', arrowsize=20)

    nx.draw_networkx_nodes(G, pozisyonlar, ax=ax, node_color='#1f78b4', node_size=800)

```

```

nx.draw_networkx_labels(G, pozisyonlar, ax=ax, font_color='white', font_weight='bold')

ax.set_title("Yönlü Oklarla Gösterilen Hamilton Döngüsü", pad=20, fontsize=16)
plt.axis('off')
plt.tight_layout()
plt.show()

# =====
# 4. ANA ÇALIŞTIRMA VE TESTLER
# =====

def test_senaryolarini_calistir():
    """Önceden tanımlanmış graflar üzerinde Hamilton döngüsü testlerini çalıştırır."""
    gr_dodec = nx.to_dict_of_lists(nx.dodecahedral_graph())

    herschel_kenarlari = [(0, 1), (0, 7), (0, 9), (0, 10), (1, 2), (1, 8), (2, 3), (2, 9),
(3, 4), (3, 8), (4, 5), (4, 9), (4, 10), (5, 6), (5, 8), (6, 7), (6, 10), (7, 8)]
    H = nx.Graph(herschel_kenarlari)
    H.add_nodes_from(range(11))
    gr_herschel = nx.to_dict_of_lists(H)

    for isim, gr in [("Dodecahedral", gr_dodec), ("Herschel", gr_herschel)]:
        print(f"\n--- {isim} Grafi Test Ediliyor ---")
        cozucu = hamilton_cozucu_olustur(karmalanabilir_tuple_cevir(gr))

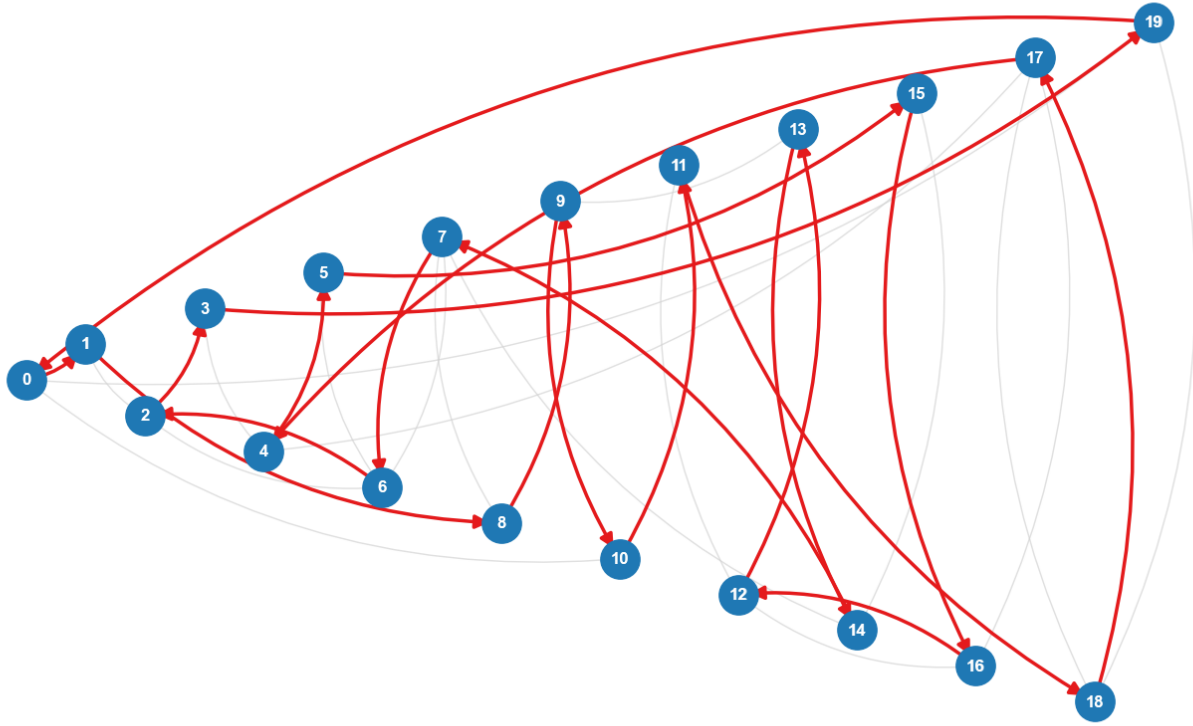
        if not cozucu:
            print(" ⚠ Geçersiz graf yapısı, atlanıyor.")
            continue

        if cozucu.check() == sat:
            print(" ✅ Hamilton döngüsü BULUNDU!")
            cozumu_gorsellestir(gr, cozucu.model())
        else:
            print(f" ❌ {isim} grafında Hamilton döngüsü mevcut değil.")

if __name__ == "__main__":
    test_senaryolarini_calistir()

```

Liste 5: Keçeci dizilimi ile Hamilton döngüsü testi ve graf kodu



Şekil 5: Keçeci dizilimi ile Hamilton döngüsü oklu grafi

İstersek renklendirerek daha da anlaşılır yapabiliriz.

```
# -*- coding: utf-8 -*-
"""
```

Z3 ve Gelişmiş Görselleştirme ile Hamilton Döngüsü Çözücü.

Bu modül, graflar oluşturmak, Z3 teoremi kanıtlayıcısını kullanarak Hamilton döngülerini kontrol etmek ve sonuçları Keçeci Dizilimi ile görselleştirmek için araçlar sağlar. Kararlı ve verimli bir Z3 formülasyonunu sağlam yardımcı fonksiyonlarla birleştirir.

```
"""
```

```
# --- Standart Kütüphane Modülleri ---
from functools import lru_cache, wraps
from typing import Dict, List, Optional, Any, Callable
```

```
# --- Üçüncü Parti Kütüphane Modülleri ---
import networkx as nx
import matplotlib.pyplot as plt
from networkx.generators import small as nxs
from z3 import Solver, Int, Or, Distinct, sat, Implies, ModelRef
```

```
# --- Yerel Kütüphane ---
try:
```



```

import kececilayout as kl
except ImportError:
    print("Uyarı: 'kececilayout' modülü bulunamadı.")
    kl = None

# =====
# 1. GRAF YARDIMCI FONKSİYONLARI
# =====

def _grafi_nx_cevir(fonksiyon: Callable) -> Callable:
    """Sözlük tabanlı grafi otomatik olarak NetworkX nesnesine dönüştüren dekoratör."""
    @wraps(fonksiyon)
    def sarmalayici(graf_girdisi: Any, *args, **kwargs) -> Any:
        G = graf_girdisi if isinstance(graf_girdisi, nx.Graph) else nx.Graph(graf_girdisi)
        return fonksiyon(G, *args, **kwargs)
    return sarmalayici

@_grafi_nx_cevir
def graf_gecerli_mi(G: nx.Graph) -> bool:
    """Bir grafin boş olmadığını ve bağlı olup olmadığını NetworkX ile kontrol eder."""
    return G.number_of_nodes() > 0 and nx.is_connected(G)

def karmalanabilir_tuple_cevir(graf: Dict[int, List[int]]) -> tuple:
    """Bir graf sözlüğünü önbellemek için karmalanabilir bir demete dönüştürür."""
    return tuple(sorted((k, tuple(sorted(v))) for k, v in graf.items()))

# =====
# 2. HAMILTON DÖNGÜSÜ ÇÖZÜCÜ (İYİLEŞTİRİLMİŞ FORMÜLASYON)
# =====

@lru_cache(maxsize=128)
def hamilton_cozucu_olustur(graf_demeti: tuple) -> Optional[Solver]:
    """
    Bir Hamilton döngüsü için Z3 kısıtları oluşturur ve önbellege alır.
    Daha verimli ve okunabilir olan 'Implies' tabanlı bir formülasyon kullanır.
    """
    graf = dict(graf_demeti)
    if not graf_gecerli_mi(graf):
        return None

    dugumler = sorted(graf.keys())
    dugum_sayisi = len(dugumler)

    if dugum_sayisi == 0:
        return None

    cozuclu = Solver()
    # pozisyon[i], döngünün i. pozisyonundaki düğümü temsil eder.
    pozisyon = [Int(f'pos_{i}') for i in range(dugum_sayisi)]

    # Kısıt 1: Her pozisyonda farklı ve geçerli bir düğüm olmalıdır.
    cozuclu.add(Distinct(pozisyon))
    for i in range(dugum_sayisi):
        cozuclu.add(Or([pozisyon[i] == dugum for dugum in dugumler]))

    # Kısıt 2 (İYİLEŞTİRİLMİŞ): Ardışık düğümler graf üzerinde komşu olmalıdır.
    for i in range(dugum_sayisi):
        mevcut_dugum = pozisyon[i]

```

```

sonraki_dugum = pozisyon[(i + 1) % dugum_sayisi]

for dugum, komsular in graf.items():
    if not komsular: continue
    komsuluk_kisiti = Or([sonraki_dugum == komsu for komsu in komsular])
    cozucu.add(Implies(mevcut_dugum == dugum, komsuluk_kisiti))

return cozucu

# =====
# 3. GÖRSELLEŞTİRME (GELİŞMİŞ RENKLENDİRME İLE)
# =====

def cozum_gorsellestir(graf: Dict[int, List[int]], model: ModelRef) -> None:
    """Döngüyü gradyan renkli oklarla ve konuma dayalı düğüm renkleriyle görselleştirir."""
    G = nx.Graph(graf)

    dizilim_parametreleri = {'primary_spacing': 1.5, 'secondary_spacing': 0.8,
    'primary_direction': 'left-to-right', 'secondary_start': 'up'}
    pozisyonlar = kl.kececi_layout(G, **dizilim_parametreleri)

    dugum_sayisi = G.number_of_nodes()
    dongu_yolu = [model.evaluate(Int(f'pos_{i}'))].as_long() for i in range(dugum_sayisi)]
    dongu_yolu.append(dongu_yolu[0])
    dongu_kenarlari = list(zip(dongu_yolu[:-1], dongu_yolu[1:]))

    fig, ax = plt.subplots(figsize=(14, 9))

    # Konuma dayalı düğüm renklerini hesapla
    renk_sol = '#2ca02c'; renk_sag = '#9467bd'; renk_merkez = '#7f7f7f'
    dugum_renkleri = [renk_sol if pozisyonlar[dugum][0] < -0.01 else renk_sag if
    pozisyonlar[dugum][0] > 0.01 else renk_merkez for dugum in G.nodes()]

    # Gradyan kenar renklerini oluşturun
    renk_haritasi = plt.cm.jet
    kenar_renkleri = [renk_haritasi(i / len(dongu_kenarlari)) for i in
    range(len(dongu_kenarlari))]

    # 1. Temel grafin kenarlarını soluk renkte çiz
    nx.draw_networkx_edges(G, pozisyonlar, ax=ax, edge_color='gainsboro', style='solid')

    # 2. Hamilton döngüsünü gradyan renkli ve yönlü oklarla çiz
    nx.draw_networkx_edges(G, pozisyonlar, ax=ax, edgelist=dongu_kenarlari,
    edge_color=kenar_renkleri, width=2.5, connectionstyle='arc3,rad=0.2', arrows=True,
    arrowstyle='->', arrowsize=20)

    # 3. Düğümleri ve etiketleri en üste çiz (kenarların altında kalmasınlar)
    nx.draw_networkx_nodes(G, pozisyonlar, ax=ax, node_color=dugum_renkleri, node_size=800)
    nx.draw_networkx_labels(G, pozisyonlar, ax=ax, font_color='white', font_weight='bold')

    ax.set_title("Gradyan Oklar ve Konumsal Düğümler ile Hamilton Döngüsü", pad=20,
    fontsize=16)
    plt.axis('off')
    plt.tight_layout()
    plt.show()

# =====
# 4. ANA ÇALIŞTIRMA VE TESTLER

```

```
# =====

def test_senaryolarini_calistir():
    """Önceden tanımlanmış graflar üzerinde Hamilton döngüsü testlerini çalıştırır."""
    gr_dodec = nx.to_dict_of_lists(nx.dodecahedral_graph())

    herschel_kenarlari = [(0, 1), (0, 7), (0, 9), (0, 10), (1, 2), (1, 8), (2, 3), (2, 9),
(3, 4), (3, 8), (4, 5), (4, 9), (4, 10), (5, 6), (5, 8), (6, 7), (6, 10), (7, 8)]
    H = nx.Graph(herschel_kenarlari); H.add_nodes_from(range(11))
    gr_herschel = nx.to_dict_of_lists(H)

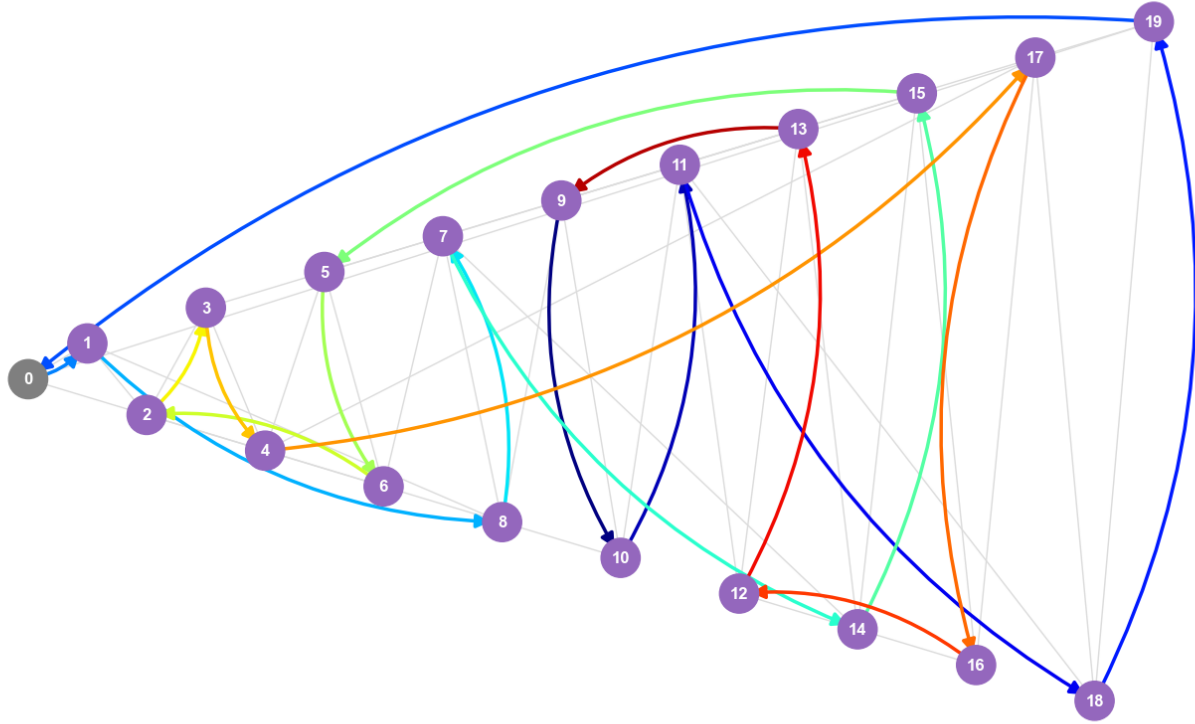
    for isim, gr in [("Dodecahedral", gr_dodec), ("Herschel", gr_herschel)]:
        print(f"\n--- {isim} Grafi Test Ediliyor ---")
        cozucu = hamilton_cozucu_olustur(karmalanabilir_tuple_cevir(gr))

        if not cozucu:
            print(" ⚠ Geçersiz graf yapısı, atlanıyor.")
            continue

        if cozucu.check() == sat:
            print(" ✅ Hamilton döngüsü BULUNDU!")
            cozumu_gorsellestir(gr, cozucu.model())
        else:
            print(f" ❌ {isim} grafında Hamilton döngüsü mevcut değil.")

if __name__ == "__main__":
    test_senaryolarini_calistir()
```

Liste 6: Keçeci Dizilimi ile Hamilton döngüsü testi ve graf kodu



Şekil 6: Keçeci dizilimi ile Hamilton döngüsü oklu grafi

```
# -*- coding: utf-8 -*-
"""
```

Z3 ve Gelişmiş Görselleştirme ile Hamilton Döngüsü Çözücü.

Bu modül, graflar oluşturmak, Z3 teoremi kanıtlayıcısını kullanarak Hamilton döngülerini kontrol etmek ve sonuçları Keçeci Dizilimi ile görselleştirmek için araçlar sağlar. Kararlı ve verimli bir Z3 formülasyonunu sağlam yardımcı fonksiyonlarla birleştirir.

```
"""
```

```
# --- Standart Kütüphane Modülleri ---
from functools import lru_cache, wraps
from typing import Dict, List, Optional, Any, Callable
```

```
# --- Üçüncü Parti Kütüphane Modülleri ---
import networkx as nx
import matplotlib.pyplot as plt
from networkx.generators import small as nxs
from z3 import Solver, Int, Or, Distinct, sat, Implies, ModelRef
```

```
# --- Yerel Kütüphane ---
try:
    import kececilayout as kl
except ImportError:
    print("Uyarı: 'kececilayout' modülü bulunamadı.")
    kl = None
```

```
# =====
# 1. GRAF YARDIMCI FONKSİYONLARI
# =====

def _grafi_nx_cevir(fonksiyon: Callable) -> Callable:
    """Sözlük tabanlı grafi otomatik olarak NetworkX nesnesine dönüştüren dekoratör."""
    @wraps(fonksiyon)
    def sarmalayici(graf_girdisi: Any, *args, **kwargs) -> Any:
        G = graf_girdisi if isinstance(graf_girdisi, nx.Graph) else nx.Graph(graf_girdisi)
        return fonksiyon(G, *args, **kwargs)
    return sarmalayici

@_grafi_nx_cevir
def graf_gecerli_mi(G: nx.Graph) -> bool:
    """Bir grafin boş olmadığını ve bağlı olup olmadığını NetworkX ile kontrol eder."""
    return G.number_of_nodes() > 0 and nx.is_connected(G)

def karmalanabilir_tuple_cevir(graf: Dict[int, List[int]]) -> tuple:
    """Bir graf sözlüğünü önbellemek için karmalanabilir bir demete dönüştürür."""
    return tuple(sorted((k, tuple(sorted(v))) for k, v in graf.items()))

# =====
# 2. HAMILTON DÖNGÜSÜ ÇÖZÜCÜ (İYİLEŞTİRİLMİŞ FORMÜLASYON)
# =====

@lru_cache(maxsize=128)
def hamilton_cozucu_olustur(graf_demeti: tuple) -> Optional[Solver]:
    """
    Bir Hamilton döngüsü için Z3 kısıtları oluşturur ve önbellege alır.
    Daha verimli ve okunabilir olan 'Implies' tabanlı bir formülasyon kullanır.
    """
    graf = dict(graf_demeti)
    if not graf_gecerli_mi(graf):
        return None

    dugumler = sorted(graf.keys())
    dugum_sayisi = len(dugumler)

    if dugum_sayisi == 0:
        return None

    cozuclu = Solver()
    # pozisyon[i], döngünün i. pozisyonundaki düğümü temsil eder.
    pozisyon = [Int(f'pos_{i}') for i in range(dugum_sayisi)]

    # Kısıt 1: Her pozisyonda farklı ve geçerli bir düğüm olmalıdır.
    cozuclu.add(Distinct(pozisyon))
    for i in range(dugum_sayisi):
        cozuclu.add(Or([pozisyon[i] == dugum for dugum in dugumler]))

    # Kısıt 2 (İYİLEŞTİRİLMİŞ): Ardışık düğümler graf üzerinde komşu olmalıdır.
    for i in range(dugum_sayisi):
        mevcut_dugum = pozisyon[i]
        sonraki_dugum = pozisyon[(i + 1) % dugum_sayisi]

        for dugum, komsular in graf.items():
            if not komsular: continue
```

```

komsuluk_kisiti = Or([sonraki_dugum == komsu for komsu in komsular])
cozucu.add(Implies(mevcut_dugum == dugum, komsuluk_kisiti))

return cozucu

# =====
# 3. GÖRSELLEŞTİRME (GELİŞMİŞ RENKLENDİRME İLE)
# =====

def cozumu_gorsellestir(graf: Dict[int, List[int]], model: ModelRef) -> None:
    """Döngüyü gradyan renkli oklarla ve konuma dayalı düğüm renkleriyle görselleştirir."""
    G = nx.Graph(graf)

    dizilim_parametreleri = {'primary_spacing': 1.5, 'secondary_spacing': 0.8,
'primary_direction': 'left-to-right', 'secondary_start': 'up'}
    pozisyonlar = kl.kececi_layout(G, **dizilim_parametreleri)

    dugum_sayisi = G.number_of_nodes()
    dongu_yolu = [model.evaluate(Int(f'pos_{i}'))].as_long() for i in range(dugum_sayisi)]
    dongu_yolu.append(dongu_yolu[0])
    dongu_kenarlari = list(zip(dongu_yolu[:-1], dongu_yolu[1:]))

    fig, ax = plt.subplots(figsize=(14, 9))

    # Konuma dayalı düğüm renklerini hesapla
    renk_sol = '#2ca02c'; renk_sag = '#9467bd'; renk_merkez = '#7f7f7f'
    dugum_renkleri = [renk_sol if pozisyonlar[dugum][0] < -0.01 else renk_sag if
    pozisyonlar[dugum][0] > 0.01 else renk_merkez for dugum in G.nodes()]

    # Gradyan kenar renklerini oluştur
    renk_haritasi = plt.cm.plasma
    kenar_renkleri = [renk_haritasi(i / len(dongu_kenarlari)) for i in
    range(len(dongu_kenarlari))]

    # 1. Temel grafin kenarlarını soluk renkte çiz
    nx.draw_networkx_edges(G, pozisyonlar, ax=ax, edge_color='gainsboro', style='solid')

    # 2. Düğümleri hesaplanan konuma dayalı renklerle çiz
    nx.draw_networkx_nodes(G, pozisyonlar, ax=ax, node_color=dugum_renkleri, node_size=800)

    # 3. Hamilton döngüsünü gradyan renkli ve yönlü oklarla çiz
    nx.draw_networkx_edges(G, pozisyonlar, ax=ax, edgelist=dongu_kenarlari,
    edge_color=kenar_renkleri, width=2.5, connectionstyle='arc3,rad=0.2', arrows=True,
    arrowstyle='->', arrowsize=20)

    # 4. Etiketleri en üste ekle
    nx.draw_networkx_labels(G, pozisyonlar, ax=ax, font_color='white', font_weight='bold')

    ax.set_title("Gradyan Oklar ve Konumsal Düğümler ile Hamilton Döngüsü", pad=20,
    fontsize=16)
    plt.axis('off')
    plt.tight_layout()
    plt.show()

# =====
# 4. ANA ÇALIŞTIRMA VE TESTLER
# =====

```

```

def test_senaryolarini_calistir():
    """Önceden tanımlanmış graflar üzerinde Hamilton döngüsü testlerini çalıştırır."""
    gr_dodec = nx.to_dict_of_lists(nx.dodecahedral_graph())

    herschel_kenarlari = [(0, 1), (0, 7), (0, 9), (0, 10), (1, 2), (1, 8), (2, 3), (2, 9),
(3, 4), (3, 8), (4, 5), (4, 9), (4, 10), (5, 6), (5, 8), (6, 7), (6, 10), (7, 8)]
    H = nx.Graph(herschel_kenarlari); H.add_nodes_from(range(11))
    gr_herschel = nx.to_dict_of_lists(H)

    for isim, gr in [("Dodecahedral", gr_dodec), ("Herschel", gr_herschel)]:
        print(f"\n--- {isim} Grafi Test Ediliyor ---")
        cozucu = hamilton_cozucu_olustur(karmalanabilir_tuple_cevir(gr))

        if not cozucu:
            print(" ⚠ Geçersiz graf yapısı, atlanıyor.")
            continue

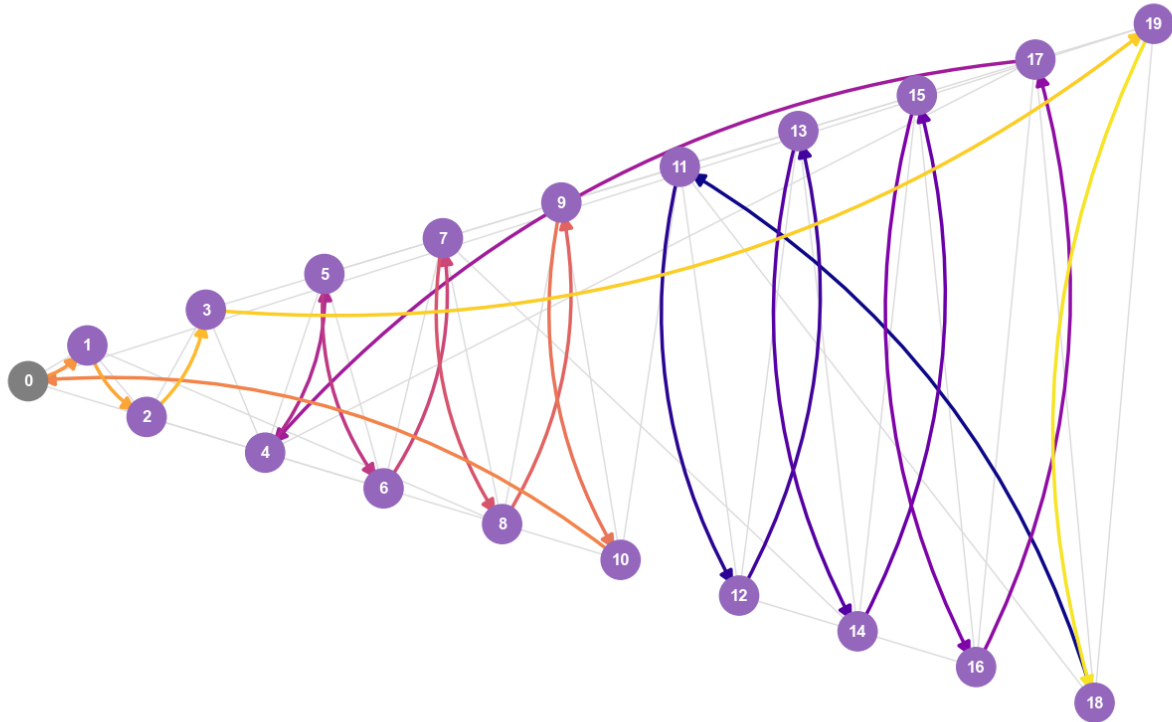
        if cozucu.check() == sat:
            print(" ✅ Hamilton döngüsü BULUNDU!")
            cozum_gorsellestir(gr, cozucu.model())
        else:
            print(f" ❌ {isim} grafında Hamilton döngüsü mevcut değil.")

if __name__ == "__main__":
    test_senaryolarini_calistir()

```

Liste 7: Keçeci dizilimi ile Hamilton döngüsü testi ve graf kodu

Gradyan Oklar ve Konumsal Döğümler ile Hamilton Döngüsü



Şekil 7: Keçeci dizilimi ile Hamilton döngüsü oklu grafi

IV. Sonuç

Bu çalışmada, **Z3 SMT Çözücüsü**'nün mantıksal gücü ile **Keçeci Dizilimi (Keçeci Layout)**'nin yapısal görselleştirme yetenekleri bir araya getirilerek, Hamilton döngüsü gibi karmaşık kombinatoryal problemler için uçtan uca bir çözüm ve analiz aracı oluşturulmuştur. Bu iki teknolojinin bir arada kullanılmasının getirdiği temel faydalar şunlardır:

I. Z3 Çözücü: Matematiksel Kesinlik ve Kanıt Motoru

Z3, problemin "nasıl" çözüleceğini adım adım programlamak yerine, problemin "ne" olduğunu (kurallarını ve kısıtlarını) bildirimsel olarak tanımlamamızı sağlar.

- **Garantili Doğruluk:** Z3, tanımlanan kısıtlara uyan bir çözüm bulduğunda, bu çözümün matematiksel olarak doğru olduğu garanti edilir. Bu, deneme-yanılma veya olasılıksal algoritmaların belirsizliğini ortadan kaldırır.
- **Var Olmama Kanıtı:** Bir çözüm bulmak kadar değerli olan bir diğer yeteneği, bir çözümün **var olmadığını** kesin olarak kanıtlayabilmesidir. Herschel grafi örneğinde gördüğümüz gibi, Z3 arama uzayını tamamen tüketerek bir Hamilton döngüsünün imkânsız olduğunu kanıtlar. Bu, sezgisel yöntemlerin sunamayacağı bir kesinliktir.

Özetle, Z3 projenin **mantıksal motoru** olarak görev yapar; bize sadece bir cevap değil, problemin doğasına dair kesin bir kanıt sunar.

II. Keçeci Dizilimi: Soyut Veriden Anlamalı Görsele

Z3'ün ürettiği çözüm, özünde bir düğüm sıralamasından ibarettir ($[0, 5, 14, 13, \dots]$). Bu, doğru olsa da tek başına bir anlam ifade etmez. Keçeci Dizilimi burada devreye girer:

- **Yapısal Netlik:** Rastgele veya kaotik dizilimlerin aksine, Keçeci Dizilimi grafi birincil ve ikincil eksenlere göre organize eder. Bu, grafin içsel simetrisini, katmanlarını ve yapısını ortaya çıkarır.
- **Sezgisel Anlayış:** Bu düzenli yapı üzerine çizilen Hamilton döngüsü, sadece bir yol olmaktan çıkar; akışı, yönü ve grafin geometrisiyle olan ilişkisi kolayca anlaşılır hâle gelir. Gradyan renkler ve yönlü oklar gibi eklemelerle bu anlayış daha da derinleşir.

Kısacası, Keçeci Dizilimi projenin **sezgisel görsel arayüzüdür**; Z3'ün ürettiği soyut ve ham veriyi, insan zihninin kolayca yorumlayabileceği anlamlı bir içgörüyeye dönüştürür.

Sinerji: Kanıt ve Sunumun Mükemmel Birleşimi

Bu iki aracın asıl gücü, birbirlerini tamamlamalarından doğar. Z3 **doğruluğu** sağlarken, Keçeci Dizilimi bu doğruluğun **anlaşılabilirliğini** sağlar. Biri olmadan diğeri eksik kalır: Anlaşılmaz bir doğru cevap veya potansiyel olarak yanlış olan güzel bir görsel, pratik bir değere sâhip değildir.

Nihâyetinde bu birleşim, zorlu bir hesaplama problemini, **doğrulanabilir ve net bir görsel anlayışa** dönüştürerek, hem geliştiriciler hem de araştırmacılar için güçlü bir analiz platformu sunar.

Conflicts of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

V. References

1. Cook, S. A. (1971). The Complexity of Theorem-Proving Procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing (pp. 151–158). Association for Computing Machinery. DOI: <https://doi.org/10.1145/800157.805047>
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press. (Chapter 34: NP-Completeness)
3. Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages & Computing*, 13(3), 259–290. DOI: <https://doi.org/10.1006/jvlc.2002.0237>
4. Di Battista, G., Eades, P., Tamassia, R., & Tollis, I. G. (1998). Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall.
5. Fruchterman, T. M. J., & Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11), 1129–1164. DOI: <https://doi.org/10.1002/spe.4380211102>
6. De Moura, L., & Bjørner, N. (2008). Z3: An Efficient SMT Solver. In Tools and Algorithms for the Construction and Analysis of Systems (pp. 337–340). Springer Berlin Heidelberg. DOI: https://doi.org/10.1007/978-3-540-78800-3_24
7. <https://github.com/Z3Prover/z3>
8. <https://z3prover.github.io/papers/programmingz3.html>
9. <https://microsoft.github.io/z3guide/>
10. Keçeci, M. (2025). The Keçeci Layout: A Deterministic Visualisation Framework for the Structural Analysis of Ordered Systems in Chemistry and Environmental Science. Open Science Articles (OSAs), Zenodo. <https://doi.org/10.5281/zenodo.16696713>
11. Keçeci, M. (2025). The Keçeci Layout: A Deterministic, Order-Preserving Visualization Algorithm for Structured Systems. Open Science Articles (OSAs), Zenodo. <https://doi.org/10.5281/zenodo.16526798>
12. Keçeci, M. (2025). Keçeci Deterministic Zigzag Layout. WorkflowHub. <https://doi.org/10.48546/workflowhub.document.31.1>
13. Keçeci, M. (2025). Keçeci Zigzag Layout Algorithm. Authorea. <https://doi.org/10.22541/au.175087581.16524538/v1>
14. Keçeci, M. (2025). The Keçeci Layout: A Structural Approach for Interdisciplinary Scientific Analysis. Open Science Articles (OSAs), Zenodo. <https://doi.org/10.5281/zenodo.15792684>

15. Keçeci, M. (2025). When Nodes Have an Order: The Keçeci Layout for Structured System Visualization. HAL open science. <https://hal.science/hal-05143155>;
<https://doi.org/10.13140/RG.2.2.19098.76484>
16. Keçeci, M. (2025). The Keçeci Layout: A Cross-Disciplinary Graphical Framework for Structural Analysis of Ordered Systems. Authorea. <https://doi.org/10.22541/au.175156702.26421899/v1>
17. Keçeci, M. (2025). Beyond Traditional Diagrams: The Keçeci Layout for Structural Thinking. Knowledge Commons. <https://doi.org/10.17613/v4w94-ak572>
18. Keçeci, M. (2025). The Keçeci Layout: A Structural Approach for Interdisciplinary Scientific Analysis. figshare. Journal contribution. <https://doi.org/10.6084/m9.figshare.29468135>
19. Keçeci, M. (2025, July 3). The Keçeci Layout: A Structural Approach for Interdisciplinary Scientific Analysis. OSF. <https://doi.org/10.17605/OSF.IO/9HTG3>
20. Keçeci, M. (2025). Beyond Topology: Deterministic and Order-Preserving Graph Visualization with the Keçeci Layout. WorkflowHub. <https://doi.org/10.48546/workflowhub.document.34.4>
21. Keçeci, M. (2025). A Graph-Theoretic Perspective on the Keçeci Layout: Structuring Cross-Disciplinary Inquiry. Preprints. <https://doi.org/10.20944/preprints202507.0589.v1>
22. Keçeci, M. (2025). Keçeci Layout. Open Science Articles (OSAs), Zenodo.
<https://doi.org/10.5281/zenodo.15314328>
23. Keçeci, M. (2025). kececilayout [Data set]. WorkflowHub.
<https://doi.org/10.48546/workflowhub.datafile.17.1>
24. Keçeci, M. (2025). Kececilayout. Open Science Articles (OSAs), Zenodo.
<https://doi.org/10.5281/zenodo.15313946>
25. <https://github.com/WhiteSymmetry/kececilayout>
26. <https://pypi.org/project/kececilayout/>
27. <https://anaconda.org/bilgi/kececilayout>
28. Keçeci, M. (2025). Graf Teorisi Eğitiminde Yeni Bir Araç: Z3 ve Keçeci Dizilimi ile Hamilton Probleminin İnteraktif Keşfi. Open Science Articles (OSAs), Zenodo.
<https://doi.org/10.5281/zenodo.16883657>
29. Keçeci, M. (2025). A Novel Tool for Graph Theory Education: Interactive Exploration of the Hamiltonian Problem with Z3 and the Keçeci Layout. Open Science Articles (OSAs), Zenodo.
<https://doi.org/10.5281/zenodo.16920991>
30. Keçeci, M. (2025). Graf Teorisi Eğitiminde Yeni Bir Araç: Z3 ve Keçeci Yerleşimi ile Hamilton Probleminin İnteraktif Keşfi. Open Fig Share Articles (OFSAs), figshare.
<https://doi.org/10.6084/m9.figshare.29958116>

31. Keçeci, M. (2025). Graf Teorisi Eğitiminde Yeni Bir Araç: Z3 ve Keçeci Layout ile Hamilton Probleminin İnteraktif Keşfi. Open Science Output Articles (OSOAs), OSF.
<https://doi.org/10.17605/OSF.IO/E23US>
32. Keçeci, M. (2025). Graf Teorisi Eğitiminde Z3 ve Keçeci Layout ile Hamilton Problemi. Open Science Knowledge Articles (OSKAs), Knowledge Commons. <https://doi.org/10.17613/g5r9k-ksb90>
33. Keçeci, M. (2025). Graf Teorisi Eğitiminde Z3 ve Keçeci Dizilimi ile Hamilton Problemi. Open Work Flow Articles (OWFAs), WorkflowHub. <https://doi.org/10.48546/workflowhub.document.45.2>
34. Keçeci, M. (2025). Z3 ve Keçeci Layout ile Hamilton Problemi. ResearchGate.
<https://doi.org/10.13140/RG.2.2.23316.97924>
35. Keçeci, M. (2025). Interactive Exploration of the Hamiltonian Problem with Z3 and the Keçeci Layout. Open Fig Share Articles (OFSAs), figshare. <https://doi.org/10.6084/m9.figshare.29959778>
36. Keçeci, M. (2025). An Interactive Tool for Graph Theory Education: Exploring the Hamiltonian Problem with Z3 and the Keçeci Layout. Open Science Output Articles (OSOAs), OSF.
<https://doi.org/10.17605/OSF.IO/HZU8Y>
37. Keçeci, M. (2025). The Hamiltonian Problem in Graph Theory Education: An Interactive Approach Using Z3 and the Keçeci Layout. Open Science Knowledge Articles (OSKAs), Knowledge Commons. <https://doi.org/10.17613/mvq42-h4262>
38. Keçeci, M. (2025). Solving the Hamiltonian Problem in Graph Theory Education with Z3 and the Keçeci Layout. Open Work Flow Articles (OWFAs), WorkflowHub.
<https://doi.org/10.48546/workflowhub.document.48.2>
39. Keçeci, M. (2025). Hamiltonian Problem with Z3 and the Keçeci Layout. ResearchGate.