Ecrin Nazar Aybay

21253043

SPAN MDS ALGORİTMASI İNCELEMESİ

Paylaştığınız kod ve algoritmadaki Span MDS (Minimum Dominating Set), dağıtık sistemlerde minimum baskın küme problemi için tasarlanmış bir algoritma uygulamasını gösteriyor. Bu algoritma, bir ağdaki düğümleri siyah (baskın kümede) ve gri (baskın kümeye komşu) olarak renklendirir.

Algoritmanın Temel Prensibi

1. Her düğüm başlangıçta beyaz renk alır ve bir "span" değerine sahiptir (komşu sayısı + 1)
2. Her turda, en yüksek span değerine sahip düğümler siyah olur
3. Siyah düğümlerin komşuları gri olur
4. Algoritma, tüm düğümler ya siyah ya da gri olduğunda tamamlanır

Mesaj Tipleri

Algoritma, düğümler arasında iletişim için şu mesaj tiplerini kullanıyor:

- **round**: Bir turun başladığını ve düğümün span değerini bildiren mesaj
- **ch_black**: Bir düğümün siyah olarak renklendirildiğini bildiren mesaj
- **ch_gray**: Bir düğümün gri olarak renklendirildiğini bildiren mesaj
- **undecide**: Bir düğümün karar veremediğini bildiren mesaj
- **no_change**: Bir düğümün renginin değişmediğini bildiren mesaj
- **confirm**: Bir düğümün mevcut rengini onaylayan mesaj (kodunuzdaki iyileştirme)

Düğüm Renkleri

- **Beyaz (WHITE)**: Başlangıç rengi, henüz karar verilmemiş
- **Siyah (BLACK)**: Baskın küme üyesi
- **Gri (GRAY)**: Baskın kümeye komşu

Kodun İşleyişi

1. **Başlangıç (init)**: Her düğüm beyaz renk ile başlar, komşularını belirler ve span değerini hesaplar.
2. **Çalıştırma (run)**: İlk turu başlatır.
3. **Tur Başlatma (start_round)**: Round mesajını yayınlar ve karar verme adımına geçer.
4. **Karar Verme (make_decision)**:
    - Eğer düğümün siyah bir komşusu varsa ve düğüm beyazsa, hemen gri olur
    - Eğer düğüm beyazsa ve en yüksek span değerine sahipse, siyah olur

- Aksi halde değişiklik yapmaz veya kararsız durumunu bildirir

5. **Onay Gönderme (send_confirmation)**: Mevcut rengini komşularına bildirir.
6. **Çakışmaları Çözme (resolve_conflicts)**:
   - Siyah-siyah çakışmalarını çözer (ID'si küçük olan siyah kalır)
   - Beyaz düğümler, siyah komşuları varsa gri olur
7. **Turu Bitirme (finish_round)**:
   - Span değerlerini günceller
   - İzole düğümleri siyaha dönüştürür
   - Algoritmanın bitişini kontrol eder
   - Maksimum tur sayısını kontrol eder
   - Bitmediyse yeni tur başlatır
8. **Mesaj Alma (on_receive)**: Gelen mesajları işler:
   - **ROUND**: Komşu span bilgilerini kaydeder
   - **CONFIRM**: Siyah-siyah çakışmalarını tespit eder
   - **CH_BLACK**: Komşu rengini siyah olarak günceller, beyaz düğüm hemen gri olur
   - **CH_GRAY**: Komşu rengini gri olarak günceller
   - **UNDECIDE/NO_CHANGE**: Komşuyu alınanlar listesine ekler

Kodunuzdaki İyileştirmeler

Orjinal algoritmaya ek olarak:

1. **Siyah-siyah çakışmalarını çözme**: İki komşu düğüm aynı anda siyah olursa, ID'si daha büyük olan gri olur
2. **Maksimum tur sınırı**: Sonsuz döngüyü önlemek için
3. **İzole düğümleri zorunlu siyah yapma**: Komşusu olmayan düğümler direkt siyah olur
4. **Onay mekanizması**: CONFIRM mesajı ile kararların netleştirilmesi

Simülasyon

Kodun son kısmında, 10x10 grid üzerinde rastgele konumlandırılmış düğümlerle bir ağ oluşturulup algoritma simüle edilmektedir:

1. Simülasyonun görsel olarak izlenmesi sağlanır
2. Toplam simülasyon süresi 200 birim zaman olarak belirlenir

Algoritma tamamlandığında, ağ üzerinde belirlenen düğümler minimum baskın kümeyi (veya ona yakın bir çözümü) oluşturur.

Kod Açıklaması

```
import random
import sys

sys.path.insert(1, '.')
from source import DawnSimVis
```

- `random`: Rasgele sayı üretmek için.

- `sys.path.insert`: Kodun `source` klasöründeki `DawnSimVis` modülünü kullanmak için klasör yolu ekleniyor.

- `DawnSimVis`: Görsel simülasyon ortamını sağlayan özel bir modül.

```
# Message types
ROUND = 'round'
CH_BLACK = 'ch_black'
CH_GRAY = 'ch_gray'
UNDECIDE = 'undecide'
NO_CHANGE = 'no_change'
CONFIRM = 'confirm'  # Onay mesajı ekledim

# Node colors
WHITE = 'white'
BLACK = 'black'
GRAY = 'gray'
```

Her düğüm, bu mesaj türlerini kullanarak diğer düğümlerle iletişim kurar. Renkler, düğümün durumunu temsil eder:

- **WHITE**: Henüz karar vermemiş.
- **BLACK**: Dominant düğüm (seçilen).
- **GRAY**: Komşusu dominant olduğu için elenen.

Node Sınıfı

init() — Başlatma

Her düğüm başlangıçta:

- Beyaz (WHITE) renkte başlar.
- Komşularının renk bilgilerini saklar.
- Komşuların sayısına göre **span** hesaplar.
- Görsel rengini ayarlar.

```
self.spans = len(self.curr_neighs) + 1
```

Bu değer, düğümün etki alanı büyüklüğünü (kendisi + komşuları) ifade eder.

change_visual_color() — Görsel Renk Güncelleme

Renk durumuna göre düğümün görsel rengi değişir:

- Beyaz: (1, 1, 1)
- Siyah: (0, 0, 0)
- Gri: (0.5, 0.5, 0.5)

run() ve start_round()

Algoritma başlatılır:

```
self.current_round = 1
self.start_round()
```

Her turda:

- Komşulara mesaj gönderilir (ROUND).
- 1 zaman birimi sonra make_decision çağrılır.

make_decision() — Karar Verme

```
def make_decision(self):
    if self.finished:
        return

    # Check if we already have a black neighbor
    has_black_neighbor = False
    for nid, color in self.neigh_cols.items():
        if color == BLACK:
            has_black_neighbor = True
            break

    if has_black_neighbor and self.color == WHITE:
        # Komşu siyahsa, biz gri olalım
        self.color = GRAY
        self.change_visual_color()

        pck = {
            'type': CH_GRAY,
            'sender': self.id,
            'round': self.current_round,
            'color': GRAY
        }
        self.send(DawnSimVis.BROADCAST_ADDR, pck)
        return

    # If we are white and have highest span, become black
    if self.color == WHITE and self.spans != 0:
        has_highest_span = True
        for nid in self.curr_neighs:
            if nid in self.received:
                neighbor_span = getattr(self, 'neighbor_spans',
{}).get(nid, 0)
                # Eşitlik durumunda düğüm ID'lerine bakarak belirle
                if self.spans < neighbor_span or (self.spans ==
neighbor_span and self.id < nid):
                    has_highest_span = False
                    break
```

```
        if has_highest_span:
            self.log(f"Node {self.id} has highest span ({self.spans}),
coloring BLACK")
            self.color = BLACK
            self.spans = 0
            self.change_visual_color()

            pck = {
                'type': CH_BLACK,
                'sender': self.id,
                'round': self.current_round,
                'color': BLACK
            }
            self.send(DawnSimVis.BROADCAST_ADDR, pck)
        else:
            pck = {
                'type': UNDECIDE,
                'sender': self.id,
                'round': self.current_round,
                'color': self.color
            }
            self.send(DawnSimVis.BROADCAST_ADDR, pck)
    else:
        # No change
        pck = {
            'type': NO_CHANGE,
            'sender': self.id,
            'round': self.current_round,
            'color': self.color
        }
        self.send(DawnSimVis.BROADCAST_ADDR, pck)

    # Onay fazına geç
    self.set_timer(1, self.send_confirmation)
```

Düğüm aşağıdaki kurallara göre karar verir:

 Eğer düğüm işini bitirmişse işlem yapmaz
 Siyah komşu olup olmadığını kontrol eder

- Siyah komşu varsa ve düğüm beyazsa, düğüm gri olur ve bunu bildirir

 Düğüm beyazsa ve span değeri sıfır değilse:

- Komşularıyla span değerlerini karşılaştırır
- Eşitlik durumunda ID'lere bakılır
- En yüksek span'e sahipse siyah olur ve bunu bildirir
- Değilse kararsız olduğunu bildirir

 Diğer durumlarda değişiklik olmadığını bildirir
 Onay fazına geçmek için zamanlayıcı ayarlar

send_confirmation() — Onay Mesajı

```
def send_confirmation(self):
    if self.finished:
        return

    # Send confirmation with current color
    pck = {
        'type': CONFIRM,
        'sender': self.id,
        'round': self.current_round,
        'color': self.color
    }
    self.send(DawnSimVis.BROADCAST_ADDR, pck)

    # Çakışmaları çözmek için zamanlayıcı ayarla
    self.set_timer(1, self.resolve_conflicts)
```

Kendi rengini komşulara tekrar bildirir (CONFIRM mesajı).

- Düğümün mevcut rengini doğrulayan bir onay mesajı gönderir
- Çakışmaları çözmek için zamanlayıcı ayarlar

resolve_conflicts() — Çakışmaları Çözme

```
def resolve_conflicts(self):
    if self.finished:
        return

    # Siyah-siyah çakışmalarını çöz
    if self.color == BLACK and len(self.black_conflicts) > 0:
        for conflict_id in self.black_conflicts:
            # ID'si küçük olan siyah kalır, diğeri gri olur
            if self.id > conflict_id:
                self.log(f"Node {self.id} changing to GRAY due to conflict
with node {conflict_id}")
                self.color = GRAY
                self.change_visual_color()

                # Değişikliği bildir
                pck = {
                    'type': CH_GRAY,
                    'sender': self.id,
                    'round': self.current_round,
                    'color': GRAY
                }
                self.send(DawnSimVis.BROADCAST_ADDR, pck)
                break

    # Phase 2: Check if neighbors are black
    if self.color == WHITE:
        for nid, col in self.neigh_cols.items():
            if col == BLACK:
                self.log(f"Node {self.id} changing to GRAY due to BLACK
neighbor {nid}")
                self.color = GRAY
                self.change_visual_color()

                pck = {
                    'type': CH_GRAY,
                    'sender': self.id,
                    'round': self.current_round,
```

```
                    'color': GRAY
                }
                self.send(DawnSimVis.BROADCAST_ADDR, pck)
                break

    # Turu bitir
    self.set_timer(1, self.finish_round)
```

Siyah düğümler arası çakışmaları çözer:

- İki komşu düğüm siyahsa, **ID'si büyük olan gri** olur.
- Ayrıca komşulardan biri siyahsa ve düğüm beyazsa, kendisi gri olur.
- Siyah-siyah çakışmalarını çözer:
- Eğer düğüm siyahsa ve çakışan siyah komşuları varsa
- ID'si büyük olan gri olur
- Tekrar beyaz düğümleri kontrol eder:
- Siyah komşu varsa gri olur
- Turu bitirmek için zamanlayıcı ayarlar

finish_round() — Tur Sonu İşlemleri

```
def finish_round(self):
    if self.finished:
        return

    # Update spans for next round
    for nid, col in self.neigh_cols.items():
        if col != WHITE:
            # Node is not white, remove from span count
            if nid not in self.lost_neighs:
                self.lost_neighs.add(nid)

    # Remove lost neighbors
    self.curr_neighs -= self.lost_neighs

    # İzole düğümleri zorla siyah yap
    if len(self.curr_neighs) == 0 and self.color == WHITE:
        self.color = BLACK
        self.change_visual_color()
        self.finished = True
        self.log(f"Isolated node {self.id} colored BLACK and FINISHED")

    # Maksimum tur kontrolü
    if self.current_round >= self.max_rounds:
        if self.color == WHITE:
            # Beyaz kalmış düğümleri zorla siyah yap
            self.color = BLACK
            self.change_visual_color()
            self.log(f"Node {self.id} forced BLACK after max rounds")
        self.finished = True

    # Algoritma bitişini kontrol et
    if self.color != WHITE:
```

```
        # Gri veya siyah düğümler için tüm komşuların renklenmesini kontrol
et
        all_colored = True
        for nid in self.curr_neighs:
            if self.neigh_cols.get(nid) == WHITE:
                all_colored = False
                break

        if all_colored:
            self.finished = True
            self.log(f"Node {self.id} FINISHED with color {self.color}")

    # Sonraki turu başlat
    if not self.finished:
        self.current_round += 1
        self.lost_neighs.clear()
        self.set_timer(1, self.start_round)
```

- Beyaz olmayan komşular span listesinden çıkarılır.
- İzole (yalnız) kalan düğüm **zorla siyah yapılır**.
- 20 tur sonra hala beyaz olan düğüm **zorla siyah yapılır**.
- Eğer komşularının hepsi gri/siyahsa, düğüm işlemeyi bitirir.

 Span değerlerini günceller:

- Beyaz olmayan komşuları "kayıp komşular" listesine ekler
- Kayıp komşuları aktif komşu listesinden çıkarır

 İzole düğümleri siyah yapar

 Maksimum tur sayısını kontrol eder:

- Maksimum tur sayısına ulaşıldığında kalan beyaz düğümleri siyah yapar

 Algoritmanın bitişini kontrol eder:

- Eğer düğüm beyaz değilse ve tüm komşuları renklendirilmişse işi bitti demektir

 Eğer algoritma bitmemişse:

- Tur sayacını artırır
- Kayıp komşuları temizler
- Bir sonraki turu başlatır

on_receive(pck) — Mesaj Alımı

```
def on_receive(self, pck):
    msg_type = pck.get('type')
    sender = pck.get('sender')
    sender_color = pck.get('color')

    # Komşunun rengini güncelle
    if sender_color and sender in self.curr_neighs:
        self.neigh_cols[sender] = sender_color

    if msg_type == ROUND:
        # Store neighbor span information for comparison
        if not hasattr(self, 'neighbor_spans'):
            self.neighbor_spans = {}
        self.neighbor_spans[sender] = pck.get('spans', 0)
        self.received.add(sender)

    elif msg_type == CONFIRM:
        # Onay mesajını işle
        if sender_color == BLACK and self.color == BLACK:
            # İki komşu siyah, çakışma var
            self.black_conflicts.add(sender)
            self.log(f"Black conflict detected with node {sender}")

    elif msg_type == CH_BLACK:
        # Update neighbor color to black
        self.neigh_cols[sender] = BLACK
        self.recvd_cols.add(BLACK)
        self.received.add(sender)

        # Beyaz düğüm için span güncelle
        if self.color == WHITE:
            self.spans -= 1

        # Beyaz düğüm ve komşu siyahsa, hemen gri ol
        if self.color == WHITE:
            self.log(f"Node {self.id} immediately changing to GRAY due to
BLACK neighbor {sender}")
            self.color = GRAY
            self.change_visual_color()

            pck = {
                'type': CH_GRAY,
                'sender': self.id,
                'round': self.current_round,
                'color': GRAY
            }
            self.send(DawnSimVis.BROADCAST_ADDR, pck)

    elif msg_type == UNDECIDE:
        # Kararsız komşu
        self.received.add(sender)

    elif msg_type == CH_GRAY:
        # Update neighbor color to gray
        self.neigh_cols[sender] = GRAY
        self.received.add(sender)

        # Beyaz düğüm için span güncelle
        if self.color == WHITE:
```

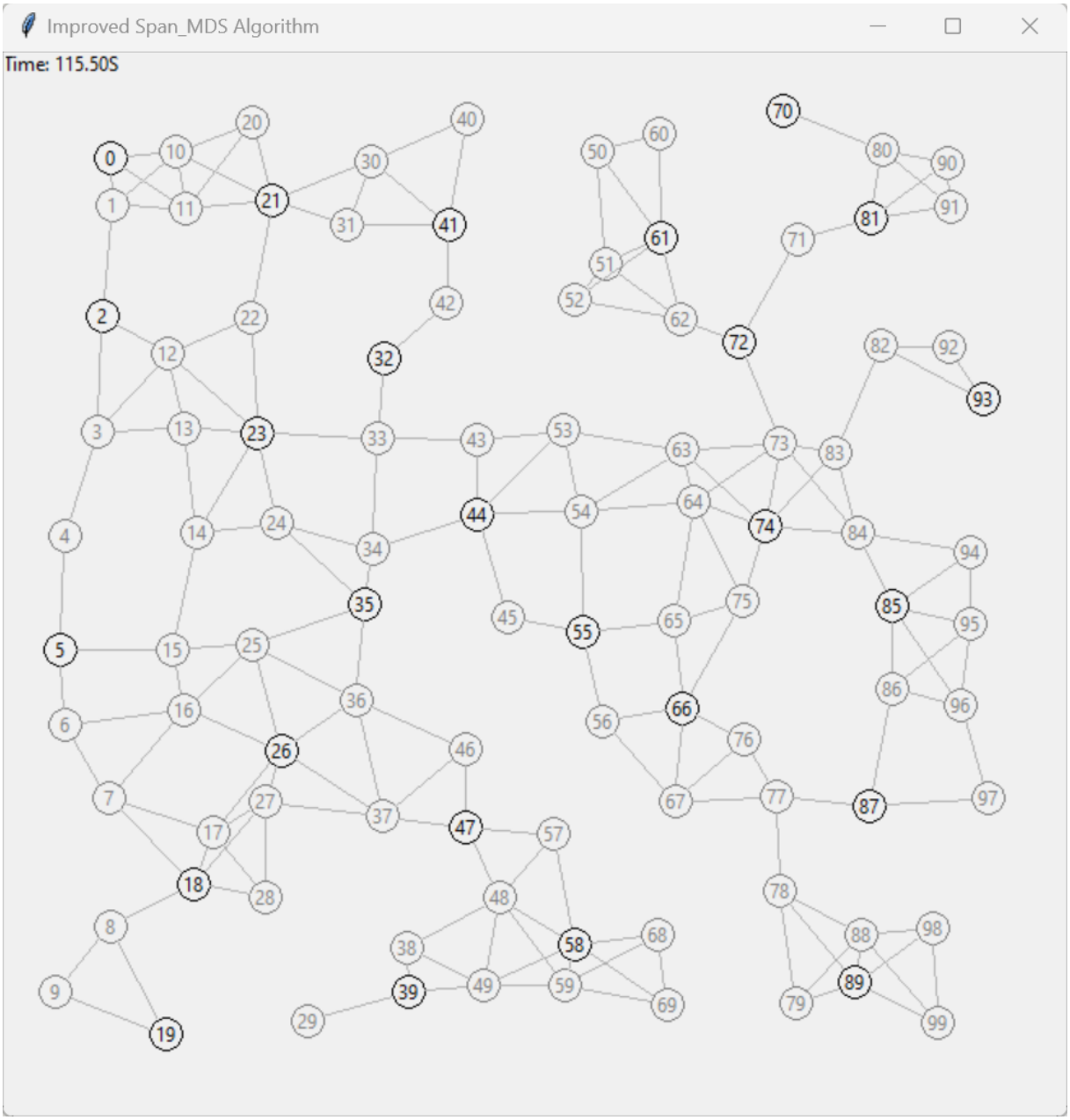
```
            self.spans -= 1

    elif msg_type == NO_CHANGE:
        # Değişmeyen komşu
        self.received.add(sender)
```

Gelen mesaj türüne göre:

- Komşu renk bilgileri güncellenir.
- CH_BLACK veya CH_GRAY ile düğüm karara varabilir.
- CONFIRM mesajlarıyla çakışmalar tespit edilir.
- UNDECIDE, NO_CHANGE vb. durumlar izlenir.

DawnSim Simülasyon Ekranı ve Terminal Örnek Kısım



```
Node #65 [  10.00000] Starting round 3
Node #71 [  10.00000] Starting round 3
Node #73 [  10.00000] Starting round 3
Node #80 [  10.00000] Starting round 3
Node #2  [  11.00000] Node 2 has highest span (3), coloring BLACK
Node #19 [  11.00000] Node 19 has highest span (2), coloring BLACK
Node #1  [  11.00002] Node 1 immediately changing to GRAY due to BLACK neighbor 2
Node #3  [  11.00002] Node 3 immediately changing to GRAY due to BLACK neighbor 2
Node #9  [  11.00002] Node 9 immediately changing to GRAY due to BLACK neighbor 19
Node #0  [  14.00000] Isolated node 0 colored BLACK and FINISHED
Node #0  [  14.00000] Node 0 FINISHED with color black
Node #2  [  14.00000] Node 2 FINISHED with color black
Node #3  [  14.00000] Node 3 FINISHED with color gray
Node #4  [  14.00000] Node 4 FINISHED with color gray
Node #8  [  14.00000] Node 8 FINISHED with color gray
Node #9  [  14.00000] Node 9 FINISHED with color gray
Node #12 [  14.00000] Node 12 FINISHED with color gray
Node #13 [  14.00000] Node 13 FINISHED with color gray
Node #19 [  14.00000] Node 19 FINISHED with color black
Node #1  [  15.00000] Starting round 4
Node #10 [  15.00000] Starting round 4
Node #11 [  15.00000] Starting round 4
Node #24 [  15.00000] Starting round 4
```