**Minimum Cost Relay Assignment Problemi için Heuristic Algoritma**

Algoritma temel olarak üç kısımdan oluşuyor.

* İlk kısımda feasible Steiner forest buluyoruz
* İkinci kısımda buldugumuz overlay pathler içinde violation durumu kötü olanları düzeltmeye çalışıyoruz.
* Üçüncü kısımda ise relay node replacement işini nodeların relay costlarını ve ne kadar source destinationı bagladıgını göz önüne alarak çözüyoruz.
* Algoritma input olarak aşağıdaki parametreleri alır:
  + Underlay networkü oluşturan connected bir graph (acyclic veya cyclic olabilir)
  + Set of source destination pairs
  + Set of underlay paths
  + Her bir vertex için relay cost ve energy consumption cost degeri
  + Her bir edge için energy consumption cost degeri

**Kısım-1 (Feasible Steiner Forest bulma ve Sleep Edilemeyenleri Ekleme)**:

DEADLINE: 24 Eylül Pazartesi

1. Underlay graph üzerinde MST algoritması koşarak min. spanning tree bulunur (Çok fazla edge oldugu durumlarda Prim’s algorithm koşmak daha mantıklı)
2. Source ve destination pairlerini birbirine baglayan pathler unique oldugundan o pathleri olusturan edgeler unremovable olarak işaretlenir. Bu aşamada bulunan subgraph bir “feasible Steiner forest”tır.
3. Removable olanlar treeden çıkarılır ve sleep edilemeyen linkler de eklenir (forest + sleep edilemeyenler)

**Kısım-2 (Violation Durumu Kötü Olanları İyileştirmeye Çalışmak):**

1. Step 3’te elde ettigimiz generic graph üzerinde (çünkü forest’a sleep edilemeyenler eklendikten sonra generic herhangi bir graph haline geldi) her bir source ve destination pairi için shortest path algoritması koşularak energy açısından minimum costlu olan path bulunur.
2. Her source ve destination pairi için buldugumuz minimum cost pathlerde ne kadar sharing olduguna bakılır; daha doğrusu bu source-destination pair’i için bulduğumuz path’in hem underlay path’lerle hem de diğer overlay path’lerle kaç tane ortak edge’i olduğuna bakılır. Eğer overlap eden edge sayısı bu source-destination pair’i için input olarak verilmiş olan üst limiti aşıyorsa bu pair üzerinde extra işlem yapmamız gerekir. Bu şekilde her pair kontrol edilir.

DEADLINE: 28 Eylül Cuma’ya kadar Step 5’in sonuna gelinecek

1. Step 3’te sleep edilemeyenleri de eklemiştik. O yüzden herhangi bir source destination pairi için exponential sayıda path olabilir. Eger minimum cost path overlapping açısından üst limiti aşıyorsa diğer pathlere bakmamız gerekir. Fakat bakacagımız path sayısında bir üst limit olmalı, tüm pathlere bakamayız.Bu üst limit algoritmaya input olarak verilebilir.
2. Overlap üst sınırını violate eden pairler arasında violation miktarı (overlap eden edge sayısı – üst limit) en az olan pair’i seçeriz. Çünkü bu pair’i violation yapmayacak hale getirmek görece en kolay seçenektir.
3. Bu pairi birbirine bağlayan birden fazla path varsa bunların arasından birini seçmeliyim. Her olası pathe bakmalıyım. Önce uyutulamayan edgelerin bulundugu pathe bakarım eger varsa, çünkü uyutulamayan edge’lerin her halukarda energy consumption’ı olacaktır, uyutulabilenleri remove etmek daha kolaydır. Olası her path için eğer bu pathi bu pair için overlay path olarak seçersem toplam violate eden **pair** sayısı ne olur diye bakarım. Durumu en çok hangi path iyileştiriyorsa o pathi bu pair için yeni overlay path olarak seçerim.
4. Bu işlemi violate eden her pair için yaparım (violating pairlerin bulundugu liste her seferinde güncellenmiş oluyor sabit bir liste degil). Violate eden pairler sayısı minimum olana kadar bu işleme devam edilir. Tüm violate eden pair’ler işlem gördükten sonra algoritma durur. Eğer herhangi bir pair’i o durumda iyileştiren alternatif bir path bulamamışsam, yine o pair çıkarılır.

Fatma 9.step’teki diğer olasılıkları düşünerek deneyecek. Deneysel olarak da hangisinin daha iyi sonuç verdiğine bakarak karar vereceğiz. Olabilecek alternatif çözümleri de buraya yazacak (denemeden önce)

Aşagıda step 9 ile ilgili 2 yöntem öneriyorum.

* Burda liste en az violation olanda en cok violation olana gore sort edilmelidir. En az violation’dan kasıt, (violation sayısı-üst limit) değeri en az olandır, yani iyileştirme ihtimalimizin en kolay olduğu pair’dir.
* Basit versiyon: Başlangıç olarak listenin üzerinden sadece bir kere geçerek violation durumunu düzeltip düzeltemeyecegime bakarım. Böylelilke algoritmayı olası bir infinite looptan kurtarmış olurum. Bu aşamada bir pairin durumunu düzeltmem başka pairleri de etkiliyor olabilir. Listenin size ını küçültecek bir seçim oluyorsa bu pair için bu pathi seçmeliyim. Yeni violate olanları listeme eklemiyorum fakat var olanlardan violationı düzelen varsa onu çıkartıyorum. Bir pair düzeldiginde listedeki başka bir pair de düzelmiş olabilir, onu da çıkarıyorum. Listenin sonuna geldigimde durmam gerekir.
* Zor versiyon: Yukarıdaki algoritma düzgün çalışırsa bu kez yeni violate edenleri de listeme ekleyecegim yani listem dinamik bir liste olacak yapacagım her seçimde listemden bazı pairler çıkacak ve muhtemelen yeni pairler eklenecektir, ama her zaman yapacagım bir seçimde eklenen pairlerin sayısı çıkarılan pairlerin sayısından küçük olacaktır. Böylelikle listenin boyutunun büyümesi ve infinite loopa girmeyi engellemiş olurum. Algoritmanın bu versiyonunda listenin üzerinden sadece bir kere geçmem yeterli olmaz çünkü liste dinamik bir liste. Listenin üzerinden bir kere tam bir geçiş yaptıgım halde listenin boyutu değişmiyorsa durmam gerekir.

1. En sonunda her edge'e bakarım. Bu edge kimsenin overlayi degilse ve uyutulabiliyorsa çozumden silerim.

DEADLINE: 1 Ekim Pazartesi

**Kısım-3 (Relay Node’ların Seçimi):**

1. relay cost / (# of pairs that are newly connected by means of this node) baz alınarak relay node seçilebilir. “Newly connected”ten kasıt (bu node varken bağlı olan toplam pair sayısı – bu node’u çıkardığımızda bağlı olan toplam pair sayısı) dır. Amacımız relay node’ların olabildiği kadar az cost’la olabildiği kadar çok pair’i birbirine bağlaması olduğu için bu metric’i seçeriz. Bağladığı toplam pair sayısından ziyade “newly connected” pair sayısını seçmemizin sebebi ise marjinal faydasını önemsememizdir. Mesela bir node çok az bir relay cost’la bir sürü pair’i birbirine bağlıyor olabilir ama belki de bu node’u relay’lerden biri olarak seçmediğimizde zaten bu pair’lerin hepsi birbirine bağlı oluyordur.
2. Eger bütün source ve dest. pairleri bağlı ise algoritma durur.DEADLINE: 5 Ekim Cuma

21 Eylül’de yapılan toplantıdan bir takım notlar:

1. “How to Select a Good Alternate Path in Large Peer to Peer Systems?” makalesinde IP level path information cok fazla oldugu icin storage kolaylıgı ve algoritmanın daha hızlı sonuçlanabilmesi için AS-level path information tutulacagından bahsediliyor. Earliest Divergence algoritması kullanılıyor. Önemli bir nokta ise As-level path disjointness ile Iplevel path disjointness arasında pozitif korelasyon olması, zaten paperın amacı da as disjoint as possible overlay pathler bulabilmek.
2. Bu paperda aynı zamanda distributed bir overlay path secimi var. Secilen bir overlay pathin IP levelda iken underlay pathle ne kadar disjoint olup olmadıgı bilgisi sadece local bir serverda var.
3. Bizim heuristic algoritmamızda ise tum link ve node costlarının centralized bir serverda bulundugunu varsayıyoruz, algoritmamız da bu centralized serverda koşacaktır. Bu durumu distributed olarak degistirmeyi düsünebiliriz.