NPM Ekosisteminde Yönlü Karmaşık Ağ Analizi

Öğretici ve Akademik Genişletilmiş Sürüm

Yusuf Talha ARABACI

22 Ekim 2025

Özet

Bu belge, NPM ekosistemindeki bağımlılık ilişkilerini ağ bilimi yaklaşımıyla inceleyen bir çalışmanın öğretici versiyonudur. Amaç, yazılım paketleri arasındaki yapısal ilişkilerin nasıl modellenebileceğini, ağ ölçümlerinin ne anlama geldiğini ve bu ölçümlerden elde edilen sonuçların sistemik risk açısından nasıl yorumlanabileceğini öğrenci düzeyinde açıklamaktır.

1 Giriş: Yazılım Ekosistemlerinde Bağımlılık ve Risk

Modern yazılım geliştirme süreçlerinde bir proje nadiren sıfırdan yazılır; çoğu zaman binlerce hazır bileşen (paket) birbirine bağımlı şekilde kullanılır. Bu durum geliştirmenin hızını artırsa da, sistemin kırılganlığını da artırır.

Açıklama

Bir paketteki hata veya kötü niyetli değişiklik, ona bağımlı olan tüm projelere yayılabilir. Bu zincirleme yayılım, **tedarik zinciri riski** olarak adlandırılır.

NPM (Node Package Manager) ekosistemi JavaScript dünyasında bu bağımlılık ilişkilerinin en yoğun olduğu alanlardan biridir. Çalışmamızın amacı, bu ekosistemi bir ağ (network) olarak düşünmek ve yapısal riskleri matematiksel olarak gözlemlemektir.

2 Temel Kavramlar ve Kuramsal Arka Plan

Bu bölümde ağ teorisinin temel kavramları sade bir dille açıklanmıştır.

Kavram

Düğüm (Node): Her bir paket, ağda bir noktadır.
Kenar (Edge): İki paket arasındaki "bağımlı → bağımlılık" yönlü bağlantısıdır.

2.1 Yönlü Ağ

Bir paket başka bir pakete bağımlıysa, bu ilişki yönlü bir bağ ile gösterilir. Örneğin:

`A` → `B` demek, "A paketi B'ye bağımlıdır" anlamına gelir.

2.2 In-Degree ve Out-Degree

- **In-degree:** Bu pakete kaç farklı paket bağımlı? (Popülerlik ve yayılım potansiyelini gösterir.)
- **Out-degree:** Bu paket kaç farklı pakete bağımlı? (Kırılganlık ve maruziyet düzeyini gösterir.)

Açıklama

Yüksek **in-degree** = "Birçok kişi bunu kullanıyor."
Yüksek **out-degree** = "Bu paket çok şeye bağımlı, biri bozulursa etkilenir."

2.3 Betweenness Merkeziyeti

Ağda iki nokta arasındaki en kısa yolların kaç tanesi bu paketten geçiyor? Bunu ölçer.

Açıklama

Bir paketin betweenness değeri yüksekse, o paket ağda *köprü* konumundadır. Silinirse iki grup arasındaki bağlantı kopar.

2.4 Bileşik Risk Skoru

Çeşitli metriklerin (in-degree, out-degree, betweenness) normalleştirilip birleştirilmesiyle elde edilir:

$$\text{Risk}(n) = 0.5\,\tilde{d}_{in}(n) + 0.2\,\tilde{d}_{out}(n) + 0.3\,\tilde{b}(n)$$

Böylece her paket için 0 ile 1 arasında bir risk değeri hesaplanabilir.

3 Veri Toplama ve Yöntem

3.1 Veri Kaynakları

NPM API ve npms.io servislerinden alınan paket verileriyle, her paketin bağımlılık listesi çıkarılmıştır.

3.2 Ağ Kurulumu

Elde edilen veriler Python'un `NetworkX` kütüphanesi ile yönlü bir graf yapısına dönüştürülmüştür. Her düğüm bir paketi, her kenar bir bağımlılığı temsil eder.

3.3 Ölçümlerin Hesaplanması

- Her düğüm için in-degree, out-degree ve betweenness değerleri hesaplanır.
- Bu değerler min–max yöntemiyle 0–1 aralığına normalize edilir.
- Ağırlıklı ortalama alınarak bileşik risk skoru bulunur.

Min–Max Normalizasyon

Bir sayıyı 0–1 aralığına getirmek için şu formül kullanılır:

$$\tilde{x} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Böylece tüm değişkenler aynı ölçeğe getirilir.

3.4 Sağlamlık Analizi

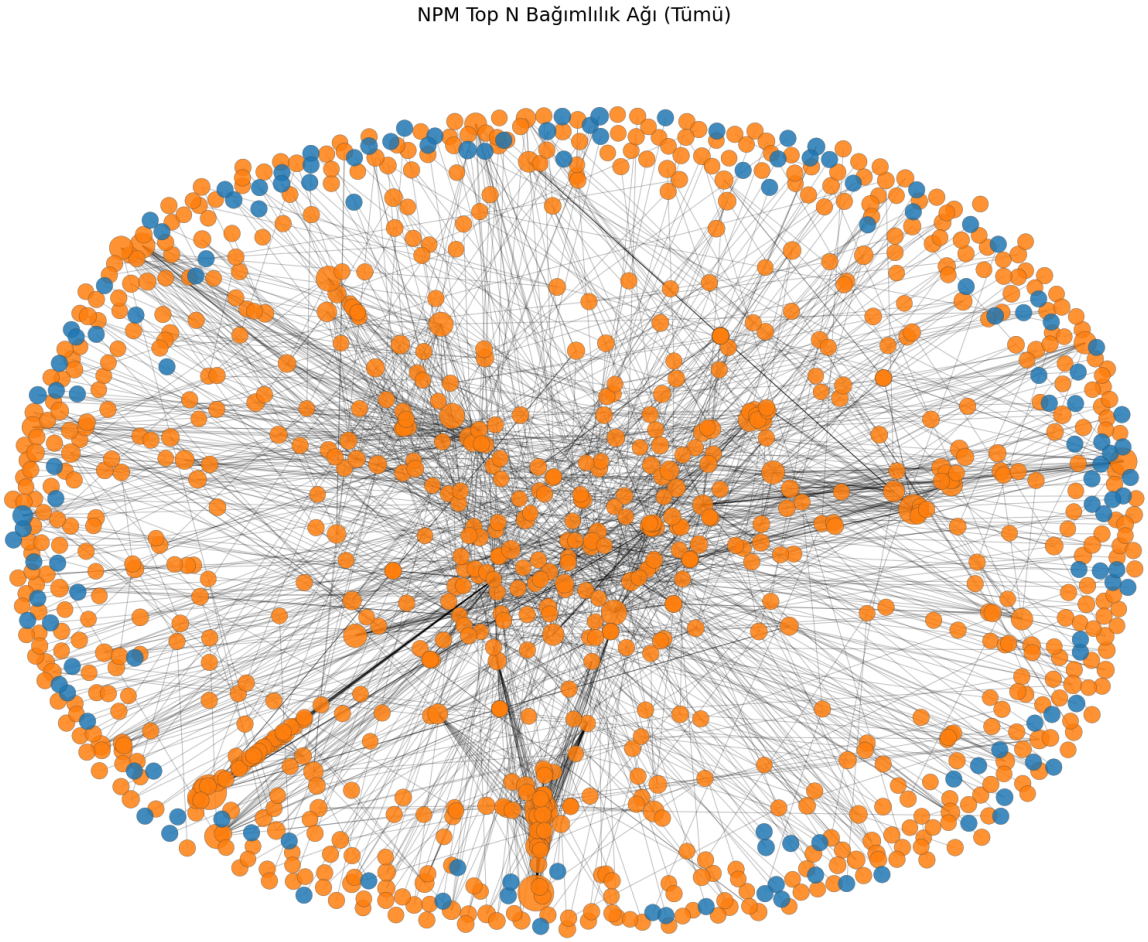
Yüksek riskli birkaç düğüm (paket) ağdan çıkarıldığında ağın bağlantı durumu tekrar ölçülür. Amaç, ağın kırılganlığını (robustluk düzeyini) gözlemlemektir.

Açıklama

Bir ağda birkaç kritik düğüm silindiğinde bile ağ bağlantısı korunabiliyorsa, o ağ **sağlamdır**. Ancak NPM ağı bu konuda zayıf görünmektedir.

4 Bulguların Görselleştirilmesi ve Yorumlanması

4.1 Ağın Genel Görünümü

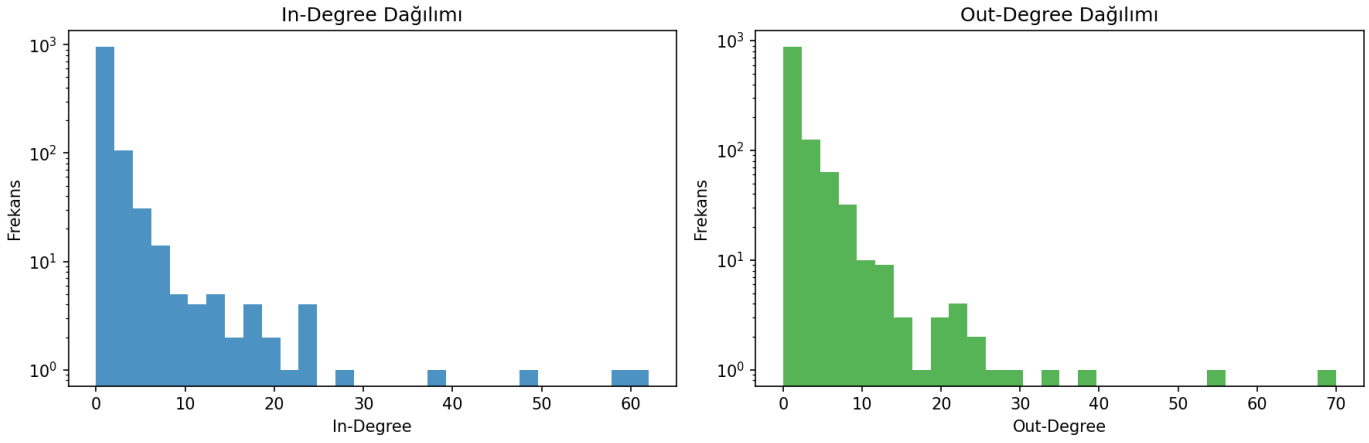


Şekil 1: NPM paketleri arasındaki bağlantı ağı. Turuncular çekirdek (Top N) paketleri gösterir.

Açıklama

Merkezdeki büyük düğümler en çok kullanılan paketlerdir. Bu düğümler ağın omurgasını oluşturur.

4.2 Derece Dağılımları

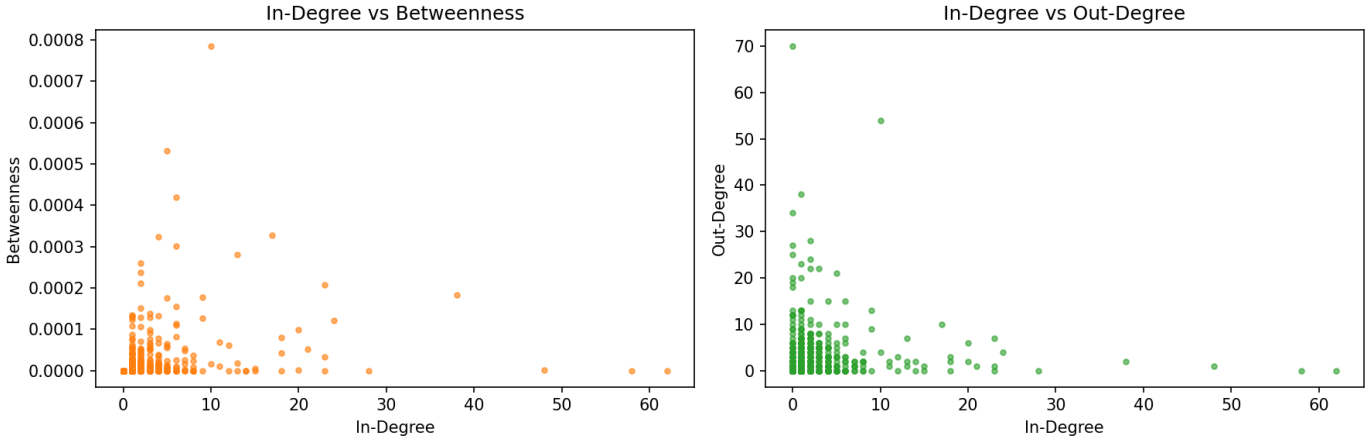


Şekil 2: In-degree ve Out-degree dağılımları (log ölçek).

Açıklama

Az sayıda çok büyük değere sahip paket vardır (örneğin Babel, TypeScript). Bu, ekosistemin dengesizliğini gösterir.

4.3 Korelasyon Analizi

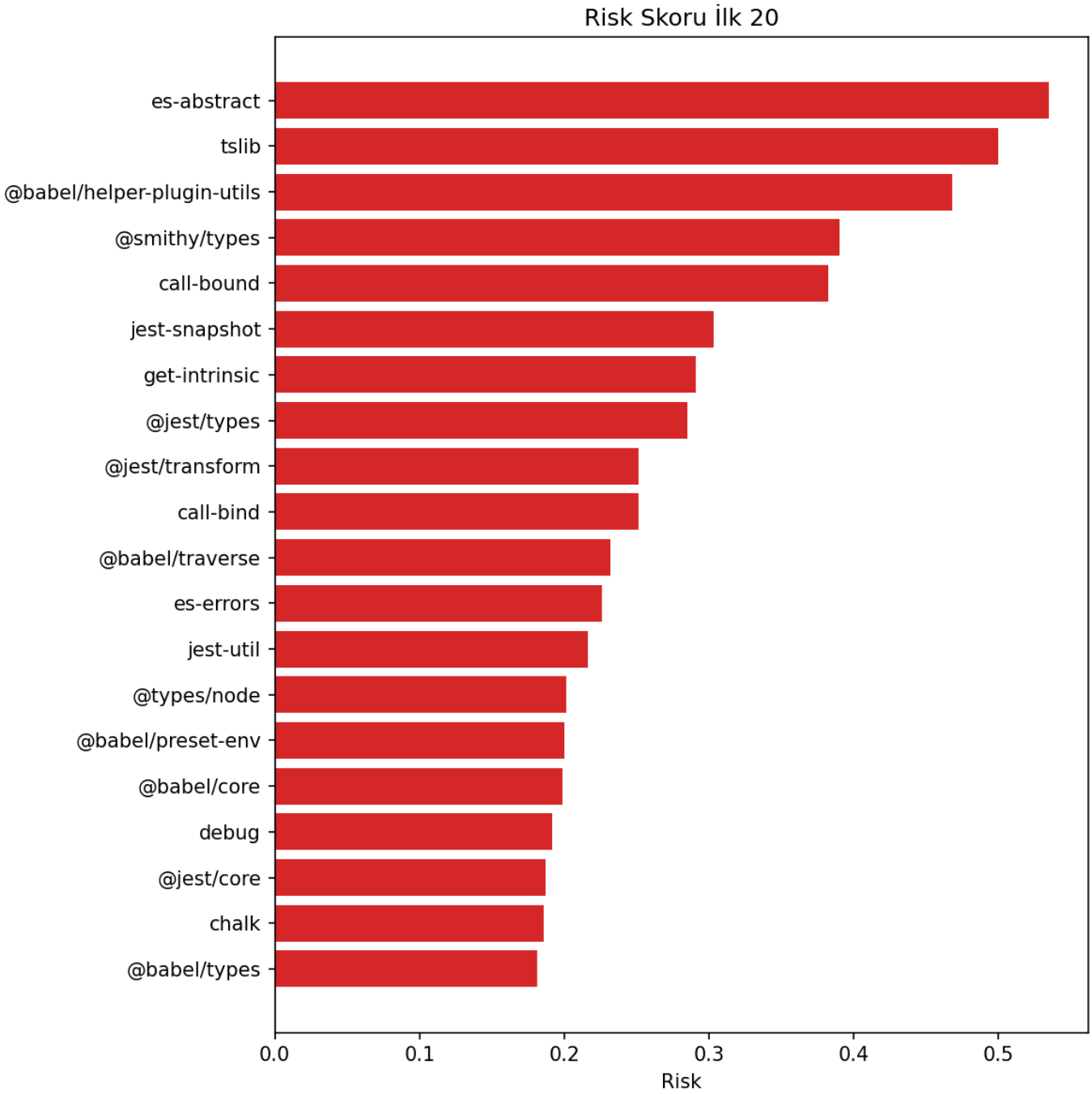


Şekil 3: Metrikler arası ilişkiler (in-degree–betweenness ve in-degree–out-degree).

Açıklama

Popüler (yüksek in-degree) paketler genellikle ağda köprü (yüksek betweenness) rolü de oynar.

4.4 Risk Skorları



Şekil 4: Bileşik risk skoruna göre en riskli 20 paket.

Tablo 1: Top 20 Risk Skoru

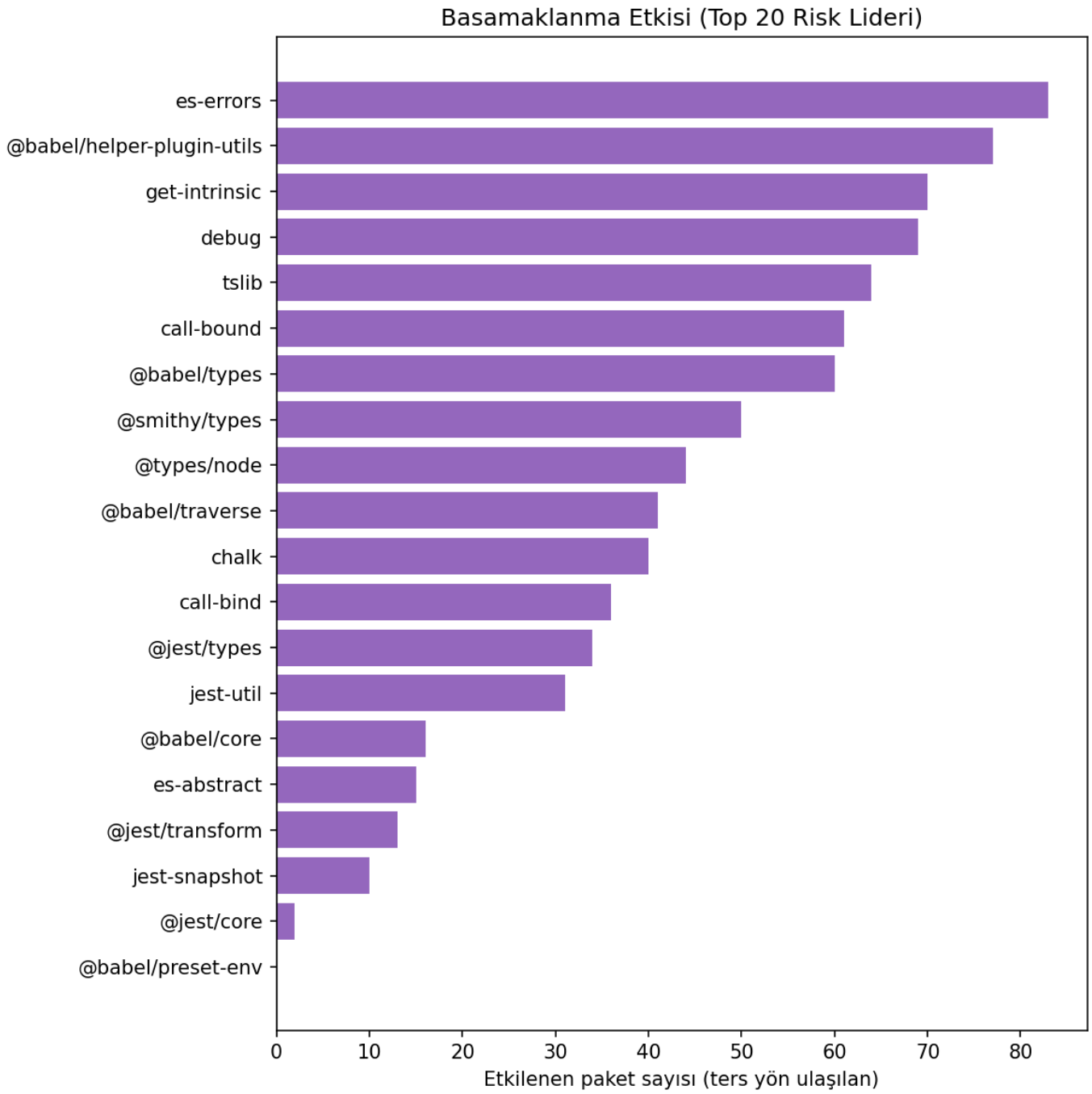
Paket	Risk	In-Degree	Out-Degree	Betweenness	TopN?
es-abstract	0.534931	10	54	0.000785	True
tslib	0.500000	62	0	0.000000	True
@babel/helper-plugin-utils	0.467742	58	0	0.000000	True
@smithy/types	0.390249	48	1	0.000001	True
call-bound	0.382129	38	2	0.000183	True
jest-snapshot	0.303511	5	21	0.000532	True
get-intrinsic	0.290993	17	10	0.000328	True
@jest/types	0.284735	23	7	0.000207	True
@jest/transform	0.251456	6	15	0.000419	True
call-bind	0.251171	24	4	0.000121	True

Paket	Risk	In-Degree	Out-Degree	Betweenness	TopN?
@babel/traverse	0.231984	13	7	0.000280	True
es-errors	0.225806	28	0	0.000000	True
jest-util	0.216164	20	6	0.000099	True
@types/node	0.201331	23	1	0.000034	True
@babel/preset-env	0.200000	0	70	0.000000	True
@babel/core	0.199075	4	15	0.000324	True
debug	0.191845	21	1	0.000051	True
@jest/core	0.187008	2	28	0.000238	True
chalk	0.185484	23	0	0.000000	True
@babel/types	0.181088	18	2	0.000079	True

Açıklama

En riskli paketler genellikle altyapı düzeyindeki küçük ama çok kullanılan modüllerdir (`es-abstract`, `tslib`, `babel`, vb.).

4.5 Kaskad Etkisi



Şekil 5: Riskli paketlerin neden olabileceği zincirleme etki boyutu.

Tablo 2: Basamaklanma Etkisi: Top 20 (Ters yonde etkilenebilecek paket sayisi)

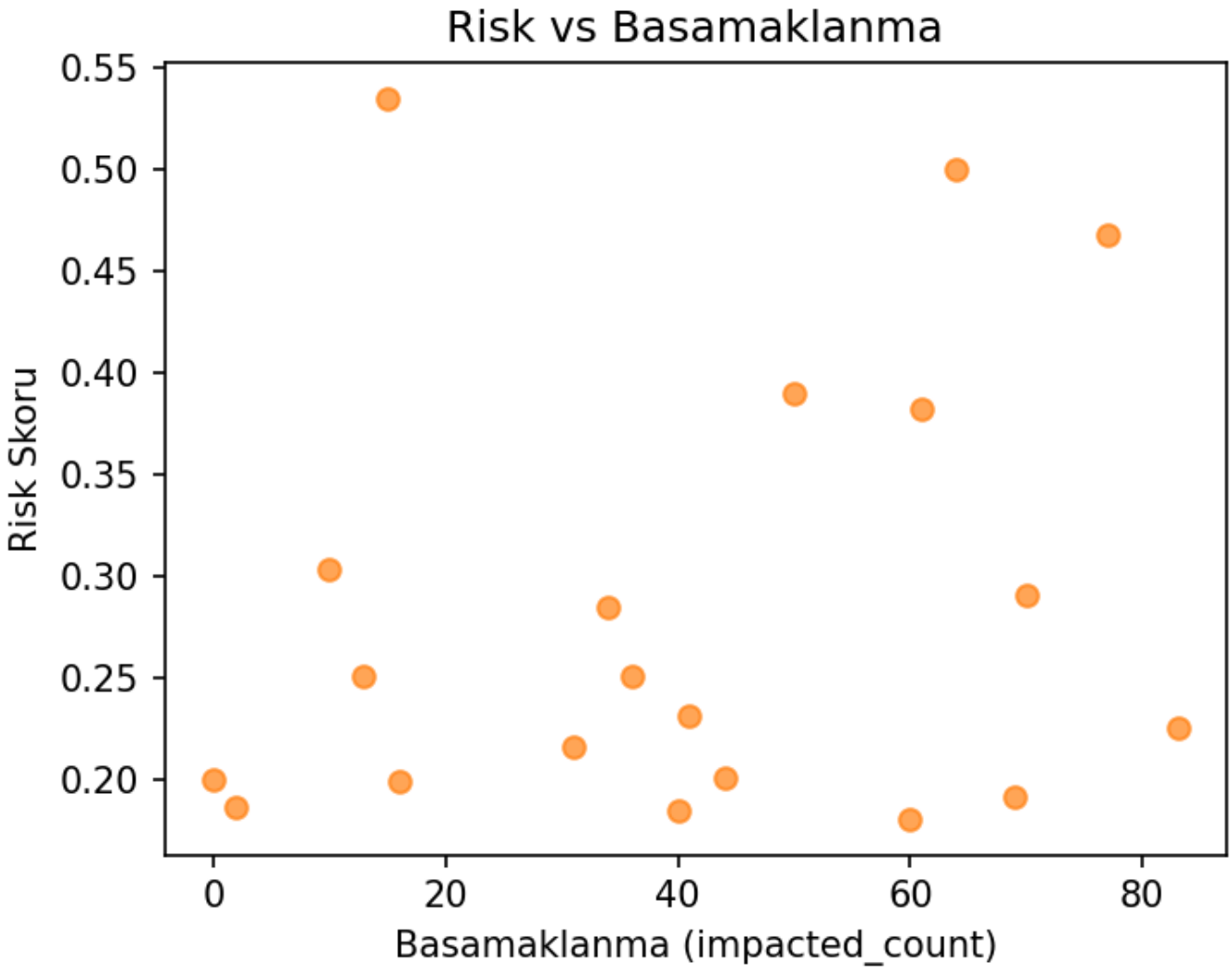
Paket	Etkilenen Paket Sayisi
es-errors	83
@babel/helper-plugin-utils	77
get-intrinsic	70
debug	69
tslib	64
call-bound	61
@babel/types	60
@smithy/types	50
@types/node	44
@babel/traverse	41

Paket	Etkilenen Paket Sayisi
chalk	40
call-bind	36
@jest/types	34
jest-util	31
@babel/core	16
es-abstract	15
@jest/transform	13
jest-snapshot	10
@jest/core	2
@babel/preset-env	0

Açıklama

Bazı küçük paketler beklenmedik derecede çok sayıda başka paketi etkileyebilir — bu, ağın karmaşıklığının sonucudur.

4.6 Sağlamlık Deneyi



Şekil 6: Risk skoru ve kaskad etkisi arasındaki ilişki.

Açıklama

İlişki doğrusal değildir. Bu, yalnızca bir metriğe bakarak sistemik riski anlamanın yetersiz olduğunu gösterir.

4.7 Köprü Kenarlar (Edge Betweenness)

Tablo 3: Edge Betweenness Ilk 10 (Yuksek kopru kenarlar)

U	V	Edge Betweenness
@jest/transform	babel-plugin-istanbul	0.000222
@jest/expect	jest-snapshot	0.000212
jest-snapshot	@jest/transform	0.000206
jest	@jest/core	0.000150
call-bound	get-intrinsic	0.000150
glob	jackspeak	0.000147
reflect.getprototypeof	which-builtin-type	0.000146
jackspeak	@isaacs/cliui	0.000140
babel-plugin-istanbul	test-exclude	0.000139
@babel/core	@babel/helper-compilation-targets	0.000138

Açıklama

Köprü kenarlar, alt ağları birbirine bağlayan kırılgan bağlantılardır. Bu bağlar koptuğunda ağ parçalanabilir.

5 Tartışma ve Değerlendirme

Bu analiz NPM ekosisteminde ağın aşırı merkezî olduğunu göstermiştir. Az sayıda paket ekosistemin çok büyük kısmını birbirine bağlamaktadır.

- **Avantaj:** Yönetim kolaylığı, yeniden kullanım.
- **Dezavantaj:** Yüksek sistemik risk ve kırılganlık.

Açıklama

Bir paketin bozulması yüzlerce projenin etkilenmesine yol açabilir. Bu nedenle güvenlik yalnızca kod düzeyinde değil, ağ düzeyinde de incelenmelidir.

5.1 Ağın Temel İstatistikleri

Tablo 4: Graf İstatistikleri (özet)

Ölçüt	Değer
Düğüm sayısı	1139
Kenar sayısı	2164
Bileşen sayısı (zayıf)	160
En büyük bileşen boyutu	853
Ortalama in-degree	1.8999
Ortalama out-degree	1.8999

Açıklama

Düğüm ve kenar sayıları ölçeği; bileşen sayısı parçalanma seviyesini; en büyük bileşen çekirdeğin büyüklüğünü gösterir.

6 Sınırlamalar ve Geliştirme Alanları

- Veriler yalnızca `dependencies` alanına dayalıdır, peer bağımlılıklar hariçtir.

- Betweenness hesaplaması örnekleme yöntemiyle yapılmıştır.
- Ağın görsel düzeni sezgiseldir; yerleşim algoritmaları sonucu etkileyebilir.

7 Sonuç ve Öğrenilenler

Bu çalışma, ağ biliminin yazılım güvenliği bağlamında nasıl kullanılabileceğini göstermektedir. Öğrenciler için çıkarılabilecek dersler şunlardır:

1. Yazılım güvenliği sadece kod taraması değil, **bağımlılık yapısının** incelenmesini de gerektirir.
2. Ağ analizi araçları (`NetworkX`, `Gephi`, vb.) bu tür karmaşık ilişkileri anlamada etkilidir.
3. "Küçük ama merkezi" paketlerin önemi, sistemik risk açısından büyüktür.

Ek A: Temel Terimler Sözlüğü

Terim	**Açıklama ve Örnek**
In-degree	Bu paketi kaç proje kullanıyor? Örn: `tslib` birçok proje tarafından kullanılır.
Out-degree	Bu paket kaç farklı pakete bağımlı? Örn: `react-scripts` onlarca modüle bağlıdır.
Betweenness	Ağdaki iki nokta arasındaki yolların ne kadarında bu paket var?
Kaskad Etkisi	Bir paketteki hatanın dolaylı olarak etkileyebileceği diğer paket sayısı.
Bileşik Risk Skoru	Yukarıdaki metriklerin ağırlıklı birleşimi (0–1 arası).