به نام خدا

پروژه دوم شبیه‌سازی کامپیوتری

مهدی معصوم پور 98243055 محمدحسین ارسلان 98243005

فاز اول

ابتدا به رسم یک مثال پرداخته‌ایم که نشان می‌دهد ممکن است گراف ما connected نباشد و فاقد راس ایزوله هم باشد. سپس بخشی از خواسته دوم که به پیاده‌سازی فرمولهای ارائه شده در قسمت 3 و 6 بود پرداختیم و این فرمولها را در توابعی مجزا پیاده سازی کردیم.

```
def reliability_isolated(graph, p):
    degrees=graph.degree()
    sigma = 0
    for node , degree in degrees:
        sigma = sigma + np.power(p,degree)
    ans = sigma * (1-p)
    return np.power(np.e,-ans)
```

این تابع در خروجی خود به ما قابلیت اطمینان یک گراف که هر راس آن با احتمال خرابی p فعالیت می‌کند را برمی‌گرداند که در ادامه مثالی با گراف اردوش-رنی اجرا کرده‌ایم و مقدار فرمول 3 برای این گراف با احتمال ارائه شده برابر با 0.6 بود.

```
G = nx.erdos_renyi_graph(10,0.5)
reliability_isolated(G,0.5)
```

```
0.604166020765368
```

برای فرمول شماره 6 باید ابتدا متغیر ϕ را برای گراف مدنظر بدست آورده و سپس با کمک عبارت $1 - \phi$ تاب‌آوری را برای گراف بدست آورد. برای محاسبات ϕ نیاز به محاسبه مجموعه‌ای به نام Cut-set بودیم که به عنوان مثال Cut-setهای دوتایی به این معنا هست که چند زیرگراف 2 راسی وجود دارد که اگر آن را از یک گراف همبند حذف کنیم موجب ناهمبندی گراف مذکور می‌شود.

برای محاسبه Cut-setها بدین صورت عمل می‌کنیم که با گرفتن یک گراف، مجموعه‌ای برمی‌گردانیم که در اندیس iام تعداد Cut-setهای iتایی آن گراف را برمی‌گرداند.

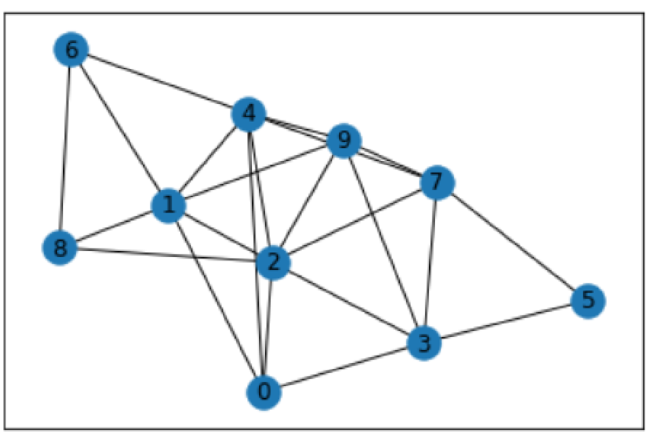
```python
def calculate_cutset(graph):
    vertices = set(graph.nodes())
    ans = [0] * len(vertices)
    if nx.is_connected(graph):
        ans = numberOfCutsets(graph)
    return ans


def numberOfCutsets(graph):
    vertices = set(graph.nodes())
    ans = [0] * len(vertices)
    for i in range(len(vertices)-1):
        subsets = list(itertools.combinations(vertices,i))
        for subset in subsets:
            graphcopy = copy.deepcopy(graph)
            graphcopy.remove_nodes_from(subset)
            if not (nx.is_connected(graphcopy)) :
                ans[i] = ans[i] + 1
#                   print(subset)
    return ans
```

با اجرای مثالی به بررسی صحت کد پرداختیم.

```python
G = nx.erdos_renyi_graph(10,0.5)
nx.draw_networkx(G)
print(calculate_cutset(G))
```

```
[0, 0, 1, 10, 41, 84, 95, 62, 22, 0]
```



حال با به دست داشتن اطلاعات مجموعه‌های Cut-set می‌توانیم به ساخت مدل تحلیلی شماره 5 و درنهایت شماره 6 بپردازیم که به‌صورت زیر و به زبان ریاضی پیاده‌سازی شده است.

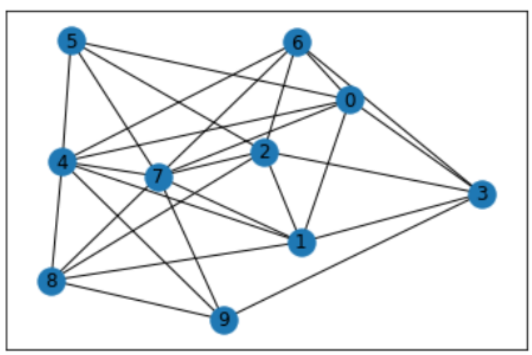
```
def residualConnectivity(graph , p):
    ci = calculate_cutset(graph)
    fi = 0
    multiplier = np.power(1-p,len(ci))
    for i in range(len(ci)):
        fi = fi + ci[i]*multiplier
        multiplier = multiplier * p/(1-p)
    return fi

def reliabilityRC(graph , p):
    return 1 - residualConnectivity(graph , p)
```

حال به کمک گراف اردوش-رنی یک گراف به عنوان مثال استفاده کرده‌ایم و در ابتدا تاب‌آوری(فرمول 6) و سپس قابلیت اطمینان (فرمول 3) را محاسبه کرده‌ایم (به صورت تحلیلی) و به مقادیری تقریبا نزدیک به هم رسیدیم.

```
G = nx.erdos_renyi_graph(10,0.5)
nx.draw_networkx(G)
print(reliabilityRC(G,0.5))
print(reliability_isolated(G,0.5))
```

```
0.9052734375
0.870513618882206
```



حال به سراغ خواسته اول می‌رویم و شبیه‌سازی مونت‌کارلو را برای گرافهای متنوع و احتمالات متفاوت انجام می‌دهیم در اولین تابعی که می‌بینید بسته به نوع گراف، گراف را ساخته و شبیه‌سازی را انجام می‌دهیم و در ساختمان داده Dictionary به نام result مقدار شبیه‌سازی را (به کمک تابع reliability_simulation) به همراه مقادیر فرمول تحلیلی 6 و 3 (به ترتیب) ذخیره می‌کنیم.

```
def reliability_simulation(graph , p):
    vertices = list(graph.nodes())
    todel=[]
    for i in range(len(vertices)):
        h = np.random.sample()
        if(h < p):
            todel.append(i)
    graph.remove_nodes_from(todel)
    if len(list(graph.nodes())) == 0:
        return 0.0
    if nx.is_connected(graph):
        return 1.0
    return 0.0

def montCarloForNodeElimination(iterate=10,
graph_type='ER',p_remove=0.01,n=10,p=0.5,m =3):
    result =
{"reliabilityCalcultedByRC":0.0,"reliabilityCalcultedByisolat
ed":0.0,"simulation":{}}

    for i in range(iterate):
        if graph_type=='ER':
            graph =  nx.erdos_renyi_graph(n,p)
            while not nx.is_connected(graph):
                graph =  nx.erdos_renyi_graph(n,p)
        elif graph_type=='WS':
            graph = nx.watts_strogatz_graph(n , m, p)
            while not nx.is_connected(graph):
```

```
                graph =  nx.watts_strogatz_graph(n , m, p)
        elif graph_type=='BA':
            graph = nx.barabasi_albert_graph(n,m)
            while not nx.is_connected(graph):
                graph =  nx.barabasi_albert_graph(n,m)
        else:
            print('this type of graph is not impelemented!')
            return result
        graphcopy = copy.deepcopy(graph)
        result["simulation"]["iter"+str(i+1)]=reliability_sim
ulation(graphcopy,p_remove)
        graphcopy = copy.deepcopy(graph)
        result["reliabilityCalcultedByRC"]=result["reliabilit
yCalcultedByRC"]+reliabilityRC(graphcopy,p_remove)
        graphcopy = copy.deepcopy(graph)
        result["reliabilityCalcultedByisolated"]=
result["reliabilityCalcultedByisolated"]+reliability_isolated
(graphcopy,p_remove)

    result["reliabilityCalcultedByisolated"]=
result["reliabilityCalcultedByisolated"]/iterate
    result["reliabilityCalcultedByRC"]=result["reliabilityCal
cultedByRC"]/iterate
    return result
```

حال که این تابع شبیه‌سازی (که البته در دل خودش مدل‌های تحلیلی را هم به ما برمی‌گرداند) در دست داریم به سراغ انجام خواسته اول می‌رویم که از ما انتظار داشت با احتمال‌های متفاوتی اجرا کرده و رسم نمودار داشته باشیم به ازای گراف‌های متفاوت.

در تابعی به نام simForDiffP شبیه‌سازی مونت کارلو را با احتمالات متفاوت (که بعنوان پارامتر یک احتمال پایه و گام تعیین میکنیم که پله پله به احتمال پایه می‌افزاید و شبیه‌سازی میک‌کند) طراحی کرده‌ایم.

```
def simFordiffP(iterate=1000,
graph_type='ER',gap=0.05,n=10,p=0.5,m =3):
    result={}
    for i in range(int(1/gap)):
        this =
montCarloForNodeElimination(iterate,graph_type,i*gap,n,p,m)
        std =
pd.Series(this.get('simulation')).aggregate('std')
        Simulation =
pd.Series(this.get('simulation')).aggregate('mean')
        this['simulation'] = Simulation
        this["error"]=std/np.sqrt(iterate)
        result[str(round(i*gap,2))]=this
    return result
```

مقادیر شبیه‌سازی و همچنین خطای حاصل نیز محاسبه شده و به کاربر ارائه می‌گردد.

حال ابتدا شبیه‌سازی ها را برای گراف‌های متفاوت انجام داده و رسم نمودارها را انجام داده‌ایم و سپس به مقایسه این سه گراف در پایان شبیه‌سازی ها پرداخته‌ایم.

شبیه سازی ER و نمودار مرتبط

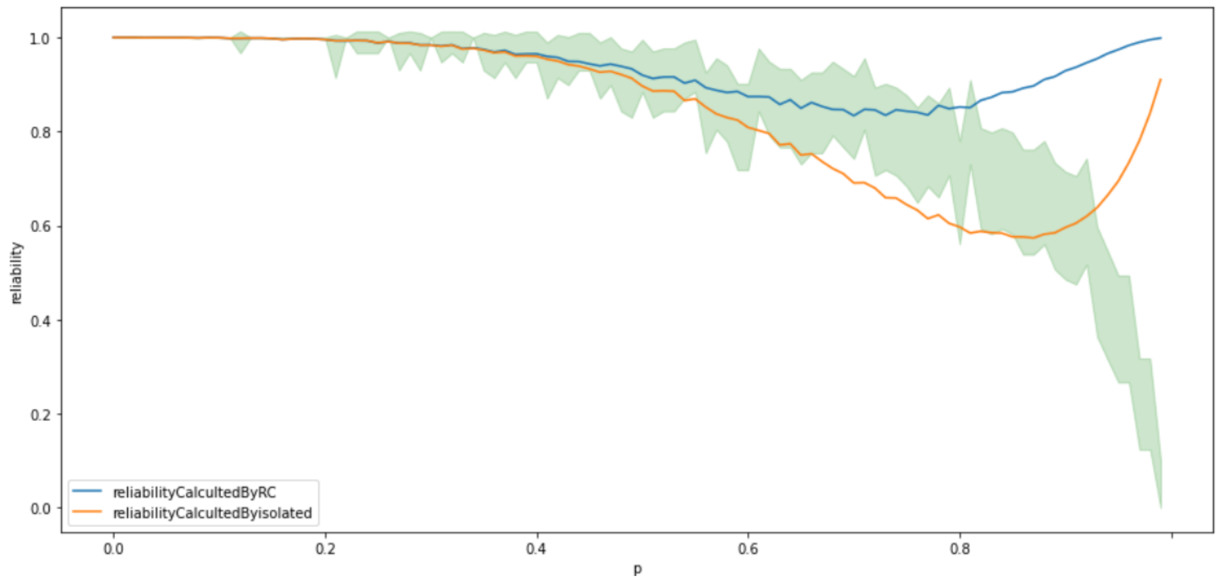
برای گراف اردوش رنی شبیه‌سازی مدنظر را انجام داده و سپس به رسم نمودار reliability بر حسب احتمال پرداخته‌ایم و در نمودار با دو منحنی نارنجی و آبی رنگ که به ترتیب مدل تحلیلی 3 و 6 هستند مواجه می‌شوید و یک منحنی بازه‌ای که برابر با مقادیر شبیه‌سازی و کران بالا و پایین با اطمینان 99 درصد (کمک گرفتن از توزیع نرمال استاندارد و خطای شبیه‌سازی) وجود دارد که صحت مدل‌های تحلیلی را می‌سنجد.

ER

```
result = simFordiffP(100,'ER',0.01,10,0.7,5)
DfER = pd.DataFrame(result)
DfER=DfER.T
```

```
plt = DfER[["reliabilityCalcultedByRC","reliabilityCalcultedByisolated"]].plot(figsize=(15,7),xlabel="p",ylabel="reliability")
z = 2.33
#99.01 percentage confidence
plt.fill_between(DfER.index,DfER['simulation']-z*DfER['error'],DfER['simulation']+z*DfER['error'],color='green',alpha=0.2)
```

نمودار شبیه‌سازی ER



در ادامه به طور مشابه برای گراف‌های BA و WS نیز این مراحل را انجام داده‌ایم.

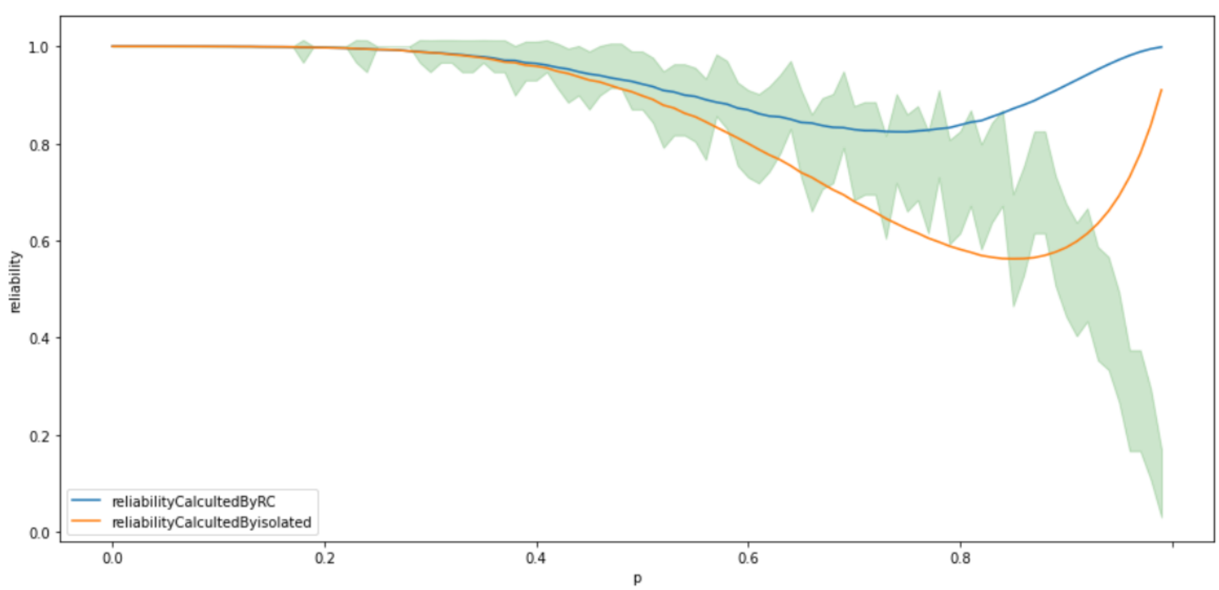
شبیه سازی WS و نمودار مرتبط

WS

```
result = simFordiffP(100,'WS',0.01,10,0.7,6)
DfWS = pd.DataFrame(result)
DfWS = DfWS.T
```

```
plt = DfWS[["reliabilityCalcultedByRC","reliabilityCalcultedByisolated"]].plot(figsize=(15,7),xlabel="p",ylabel="reliability")
z = 2.33
#99.01 percentage confidence
plt.fill_between(DfWS.index,DfWS['simulation']-z*DfWS['error'],DfWS['simulation']+z*DfWS['error'],color='green',alpha=0.2)
```

نمودار شبیه سازی WS



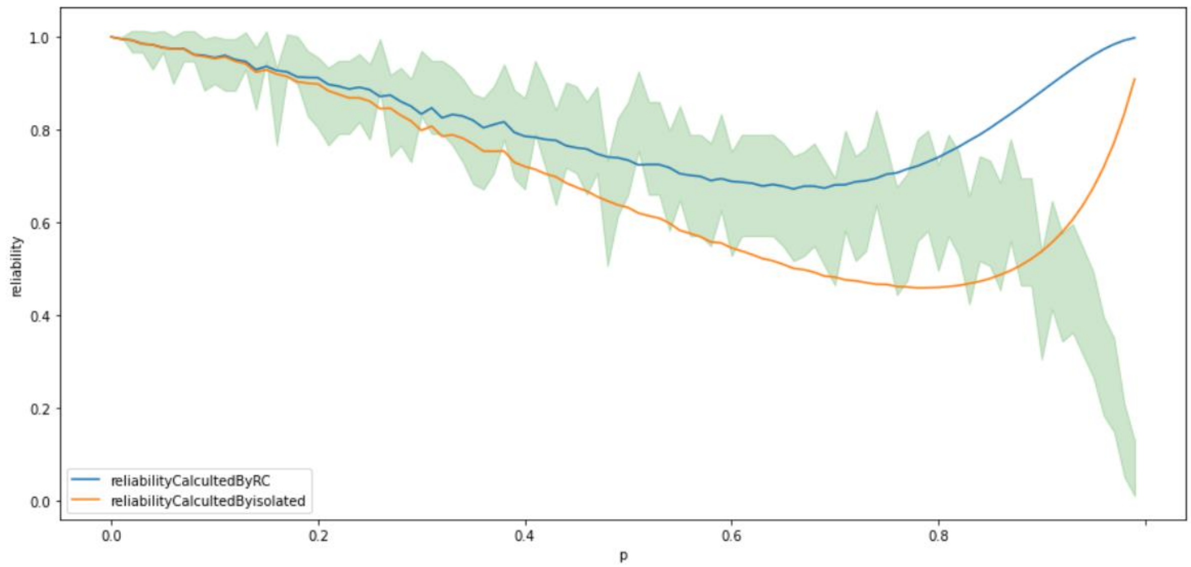
شبیه‌سازی BA و نمودار مرتبط

BA

```
result = simFordiffP(100,'BA',0.01,10,0.7,6)
DfBA = pd.DataFrame(result)
DfBA = DfBA.T
```

```
plt = DfBA[["reliabilityCalcultedByRC","reliabilityCalcultedByisolated"]].plot(figsize=(15,7),xlabel="p",ylabel="reliability")
z = 2.33
#99.01 percentage confidence
plt.fill_between(DfBA.index,DfBA['simulation']-z*DfBA['error'],DfBA['simulation']+z*DfBA['error'],color='green',alpha=0.2)
```

نمودار شبیه‌سازی BA



مقایسه و بررسی

اینجا آمده‌ایم و درصد تطابق مدل‌های تحلیلی در بازه تعیین شده به کمک شبیه‌سازی را محاسبه کرده‌ایم و در این مورد به این نتیجه می‌رسیم که گراف‌هایی از نوع BA بهترین عملکرد (تطابق مدل تحلیلی با شبیه‌سازی) و گراف‌های نوع WS بدترین عملکرد را داشته‌اند.

ER

```
z = 2.33
A=(DfER["reliabilityCalcultedByRC"] <=
(DfER["simulation"]+z*DfER["error"]))
B = (DfER["reliabilityCalcultedByRC"] >= (DfER["simulation"]-
z*DfER["error"] ))
DfER['isinintervelreliabilityCalcultedByRC'] = A&B
A=(DfER["reliabilityCalcultedByisolated"] <=
(DfER["simulation"]+z*DfER["error"]))
B = (DfER["reliabilityCalcultedByisolated"] >=
(DfER["simulation"]-z*DfER["error"] ))
DfER['isinintervalreliabilityCalcultedByisolated'] =A&B
```

نمایش درصد تطابق

```
print("the value calculated by reliabilityCalcultedByRC
function in ER graphs is
"+f"{(DfER['isinintervelreliabilityCalcultedByRC'].mean()*100
)}"+" percentage of time in confidence interval")
print("the value calculated by reliabilityCalcultedByisolated
function in ER graphs is
"+f"{(DfER['isinintervalreliabilityCalcultedByisolated'].mean
()*100)}"+" percentage of time in confidence interval")
```

```
the value calculated by reliabilityCalcultedByRC function in ER graphs is 57.99999999999999 percentage of time in confidence in
terval
the value calculated by reliabilityCalcultedByisolated function in ER graphs is 49.0 percentage of time in confidence interval
```

WS

```
z = 2.33
A=(DfWS["reliabilityCalcultedByRC"] <=
(DfWS["simulation"]+z*DfWS["error"]))
B = (DfWS["reliabilityCalcultedByRC"] >= (DfWS["simulation"]-
z*DfWS["error"] ))
DfWS['isinintervelreliabilityCalcultedByRC'] = A&B
A=(DfWS["reliabilityCalcultedByisolated"] <=
(DfWS["simulation"]+z*DfWS["error"]))
B = (DfWS["reliabilityCalcultedByisolated"] >=
(DfWS["simulation"]-z*DfWS["error"] ))
DfWS['isinintervalreliabilityCalcultedByisolated'] =A&B
```

نمایش درصد تطابق

```
print("the value calculated by reliabilityCalcultedByRC
function in small world graphs is
"+f"{(DfWS['isinintervelreliabilityCalcultedByRC'].mean()*100
)}"+" percentage of time in confidence interval")
print("the value calculated by reliabilityCalcultedByisolated
function in small world graphs is
"+f"{(DfWS['isinintervalreliabilityCalcultedByisolated'].mean
()*100)}"+" percentage of time in confidence interval")
```

```
the value calculated by reliabilityCalcultedByRC function in small world graphs is 54.0 percentage of time in confidence interv
al
the value calculated by reliabilityCalcultedByisolated function in small world graphs is 45.0 percentage of time in confidence
interval
```

BA

```
z = 2.33
A=(DfBA["reliabilityCalcultedByRC"] <=
(DfBA["simulation"]+z*DfBA["error"]))
B = (DfBA["reliabilityCalcultedByRC"] >= (DfBA["simulation"]-
z*DfBA["error"] ))
DfBA['isinintervelreliabilityCalcultedByRC'] = A&B
A=(DfBA["reliabilityCalcultedByisolated"] <=
(DfBA["simulation"]+z*DfBA["error"]))
B = (DfBA["reliabilityCalcultedByisolated"] >=
(DfBA["simulation"]-z*DfBA["error"] ))
DfBA['isinintervalreliabilityCalcultedByisolated'] =A&B
```

نمایش درصد تطابق

```
print("the value calculated by reliabilityCalcultedByRC
function in BA graphs is
"+f"{(DfBA['isinintervelreliabilityCalcultedByRC'].mean()*100
)}"+" percentage of time in confidence interval")
print("the value calculated by reliabilityCalcultedByisolated
function in BA graphs is
"+f"{(DfBA['isinintervalreliabilityCalcultedByisolated'].mean
()*100)}"+" percentage of time in confidence interval")
```

```
the value calculated by reliabilityCalcultedByRC function in BA graphs is 73.0 percentage of time in confidence interval
the value calculated by reliabilityCalcultedByisolated function in BA graphs is 50.0 percentage of time in confidence interval
```

فاز دوم

در این فاز قصد داریم تاب‌آوری شبکه را در حالی که با مدلی پویا از خرابی گره‌ها همراه هستیم بررسی کنیم و 5 سوال مربوطه را که در انتهای این بخش آمده است پاسخ دهیم.

اولین تابعی که برای این بخش پیاده‌سازی کرده‌ایم تابعی به نام random می‌باشد. این تابع بسته به ورودی‌هایش، یکی از سه توزیع نمایی، پارتو و یکنواخت را پردازش می‌کند.

```
def random(graph,alpha=3,rtype='exp',max_value=10):
    expected_LI = []
    if rtype == 'exp':
        expected_LI = max_value *
np.random.exponential(scale=1/2,size=(graph.number_of_nodes()
))
    elif rtype == 'pareto':
        expected_LI= max_value * lomax.rvs(alpha,
size=(graph.number_of_nodes()))
    elif rtype == 'uniform':
        expected_LI = max_value *
np.random.sample(size=(graph.number_of_nodes()))
    return expected_LI
```

توابع بعدی مرتبط با ویژگی‌هایی از شبکه است که در توضیحات این فاز به ما ارائه شده است از جمله L_i, s_i, T. روند کلی این است که هر سری که از عمر شبکه می‌گذرد، یک مقدار به T که بقای گره ها در شبکه متصل است اضافه می‌شود، از L_i گره‌های حاضر در شبکه یکی کم می‌شود (درصورتی که این پارامتر به صفر برسد، گره خراب می‌شود)، به یک متغیر sهم یکی اضافه می‌شود که مرتبط با جستجوی آن گره برای افزودن یک همسایه است و اگر این متغیر که هرسری یکی زیاد می‌شود به s_i مرتبط به آن گره رسید علاوه بر اینکه صفر می‌شود، یک یال هم از یکی از گره‌های غیرمجاور گره i به آن وصل می‌کند. اعمال مذکور به ترتیب در توابع update_iter و rewire صورت می‌گیرد.

تابع زیر هم کمک می‌کند تا گره‌های موجود در شبکه متصل را یافته، به زمان بقای آن‌ها در شبکه (T) یکی اضافه می‌کنیم.

```
def calculate_T(graph,T):
    for node , degree in graph.degree():
        if degree != 0:
            T[node] = T[node]+1
    return T
```

```python
def update_iter(graph,lifetime,searched):
    lifetime = lifetime-1
    searched = searched+1
    for i in range(len(lifetime)):
        if lifetime[i]<0:
            if i in graph:
                graph.remove_node(i)
    return lifetime,searched
```

```python
def rewire(graph,searchtime,searched):
    for i in range(len(searchtime)):
        if i in graph:
            if searchtime[i] <= searched[i] :
                searched[i]=0
                non_neighbors=nx.non_neighbors(graph,i)
                try:
                    j=next(non_neighbors)
                    while j not in graph:
                        j=next(non_neighbors)
                    graph.add_edge(i,j)
                except StopIteration:
                    continue
    return searched
```

در سوالات این بخش از پروژه از متغیر E[T] زیاد نام‌برده شده است، به کمک تابع زیر که مخصوص بخش **الف** پرسش‌ها می‌باشد، این متوسط را محاسبه کرده و نسبت E[T] برای هر گره را به درجه‌اش محاسبه می‌کنیم و در آرایه ET بعنوان خروجی برمی‌گردانیم و در رسم استفاده می‌کنیم.

```
def get_ET(graph,T):
    ET= np.zeros((graph.number_of_nodes()))
    countdegree = np.zeros((graph.number_of_nodes()))
    for i in range(len(graph.nodes())):
        ET[i]=0
    for i in range(len(graph.nodes())):
        countdegree[graph.degree[i]] =
countdegree[graph.degree[i]] +1
        ET[graph.degree[i]] =  ET[graph.degree[i]] + T[i]
    index=[]
    for i in range(len(graph.nodes())):
        index.append("degree"+str(i))
        if countdegree[i] != 0:
            ET[i] =  ET[i]/countdegree[i]

    return ET
```

سپس دو تابع داریم که یکی T را به ازای هر گره برمی‌گرداند و دیگری T نسبت به k یا به نوعی از تابع قبلی کمک می‌گیرد.

```
def
get_T(iteration,active,lifetype,n,p,maxlife,maxEs,alpha=3):
    Elf={}
    Es={}
    ET={}
    G1 = nx.erdos_renyi_graph(n,p)
    for i in range(iteration):
        G=copy.deepcopy(G1)
```

```
        lifetime = random(G,alpha =
alpha,rtype=lifetype,max_value=maxlife)
        searchtime =
random(G,rtype='uniform',max_value=maxEs)
        Elf['iter'+str(i)]=np.mean(lifetime)
        Es['iter'+str(i)]=np.mean(searchtime)
        s = np.zeros((G.number_of_nodes()))
        T = np.zeros((G.number_of_nodes()))
        while not is_ended(lifetime):
            T = calculate_T(G,T)
            if active:
                s = rewire(G,searchtime,s)
            lifetime,s = update_iter(G,lifetime,s)
        ET['iter'+str(i)]=T
    return {"Elf":Elf , "Es":Es , "ET":ET}
```

```
def
get_T_k(iteration,active,lifetype,n,p,maxlife,maxEs,alpha=3):
    Elf={}
    Es={}
    ET={}
    G1 = nx.erdos_renyi_graph(n,p)
    for i in range(iteration):
        G=copy.deepcopy(G1)
        lifetime = random(G,alpha =
alpha,rtype=lifetype,max_value=maxlife)
        searchtime =
random(G,rtype='uniform',max_value=maxEs)
        Elf['iter'+str(i)]=np.mean(lifetime)
        Es['iter'+str(i)]=np.mean(searchtime)
        s = np.zeros((G.number_of_nodes()))
        T = np.zeros((G.number_of_nodes()))
```

```
        while not is_ended(lifetime):
            T = calculate_T(G,T)
            if active:
                s = rewire(G,searchtime,s)
            lifetime,s = update_iter(G,lifetime,s)
        ET['iter'+str(i)]=get_ET(G1,T)

    return {"Elf":Elf , "Es":Es , "ET":ET}
```

سپس در مثالی به سراغ ساخت فرمول 8 رفته‌ایم و به نوعی مقادیر مرتبط با 4 گراف را به‌دست آورده و در جدولی نمایش داده‌ایم که تصویرش در ادامه آمده است.

```
this = get_T(4,True,'exp',50,0.5,10000,1000,3)
df=pd.DataFrame(this)
```

```
df['ET']
```

```
iter0    [295.0, 12447.0, 119.0, 4504.0, 2433.0, 1425.0...
iter1    [1418.0, 858.0, 2881.0, 6771.0, 5669.0, 839.0,...
iter2    [4210.0, 12661.0, 1439.0, 14060.0, 2149.0, 506...
iter3    [938.0, 7925.0, 12311.0, 7124.0, 1795.0, 9632....
Name: ET, dtype: object
```

```
df['phi']=df['Elf']/pd.DataFrame(df['ET'].tolist(),index=df.index).mean(axis=1)
df
```

	Elf	Es	ET	phi
iter0	4193.777459	529.904408	[295.0, 12447.0, 119.0, 4504.0, 2433.0, 1425.0...	1.044377
iter1	4790.076842	475.912192	[1418.0, 858.0, 2881.0, 6771.0, 5669.0, 839.0,...	1.036082
iter2	4631.758959	466.976894	[4210.0, 12661.0, 1439.0, 14060.0, 2149.0, 506...	1.016811
iter3	4502.412028	479.508226	[938.0, 7925.0, 12311.0, 7124.0, 1795.0, 9632....	1.009356

سپس در تابعی که در ادامه آمده است، به ازای احتمالات متفاوت که به سبک یک فاز تغییرات پله‌ای احتمال را صورت داده‌ایم، بخش **الف** را به ازای 4 توزیع مدنظر خواسته پروژه (نمایی (فعال / غیرفعال) و پارتو (فعال/غیرفعال)) پیش برده‌ایم.

```
def get_tdiffp(iteration,n,gap,Esearch,maxlife):
    result={}

    for i in range(int(1/gap)):
```

```
        this =
get_T_k(iteration,False,'exp',n,i*gap,maxlife,Esearch)
        df=pd.DataFrame(this)
        df=pd.DataFrame(pd.DataFrame(df['ET'].to_list()).repl
ace(0,np.nan).mean(skipna=True))
        result[str(n*round(i*gap,2))]={}
        result[str(n*round(i*gap,2))]["exp_inactive"]=df[0]
        this =
get_T_k(iteration,True,'exp',n,i*gap,maxlife,Esearch)
        df=pd.DataFrame(this)
        df=pd.DataFrame(pd.DataFrame(df['ET'].to_list()).repl
ace(0,np.nan).mean(skipna=True))
        result[str(n*round(i*gap,2))]["exp_active"]=df[0]
        this =
get_T_k(iteration,False,'pareto',n,i*gap,maxlife,Esearch)
        df=pd.DataFrame(this)
        df=pd.DataFrame(pd.DataFrame(df['ET'].to_list()).repl
ace(0,np.nan).mean(skipna=True))
        result[str(n*round(i*gap,2))]["pareto_inactive"]=df[0
]
        this =
get_T_k(iteration,True,'pareto',n,i*gap,maxlife,Esearch)
        df=pd.DataFrame(this)
        df=pd.DataFrame(pd.DataFrame(df['ET'].to_list()).repl
ace(0,np.nan).mean(skipna=True))
        result[str(n*round(i*gap,2))]["pareto_active"]=df[0]
    result = pd.DataFrame(result)
    result
    return result
```

در تابع بعدی هم عملکردی مشابه داریم اما این بار با زمان جستجوی متفاوت تا بتوانیم مورد **ب** را پوشش دهیم و البته بجای 4 حالت فقط دو حالت فعال را در نظر گرفته‌ایم چرا که در حالت غیرفعال جستجو معنایی ندارد.

```
Def get_tdiffs(iteration,n,p,gap,maxlife):
    result={}

    for I in range(1,int(0.5/gap)):
        this =
get_T(iteration,True,’exp’,n,p,maxlife,maxlife*(i*gap),3)
        df=pd.DataFrame(this)
        df=np.array(df[‘ET’].tolist(),dtype=object).mean().me
an()
        result[str(round(maxlife*i*gap,2))]={}
        result[str(round(maxlife*i*gap,2))][“exp_active”]=df
        this =
get_T(iteration,True,’pareto’,n,p,maxlife,maxlife*(i*gap),3)
        df=pd.DataFrame(this)
        df=np.array(df[‘ET’].tolist(),dtype=object).mean().me
an()
        result[str(round(maxlife*i*gap,2))][“pareto_active”]=
df
    return pd.DataFrame(result)
```

همچنین باید این عمل را یکبار دیگر هم انجام دهیم و اینبار مقادیر فرمول 8 را هم به ازای sهای مختلف بدست بیاوریم و چون در بخش **ج** هم باید برحسب E[s] کار کنیم، فقط دو حالت فعال را درنظر می‌گیریم.

```
def get_phidiffs(iteration,n,p,gap,maxlife):
    result={}
    for i in range(1,int(0.5/gap)):
        result[str(round(50*i*gap,2))]={}
        this =
get_T(iteration,True,'exp',n,p,maxlife,maxlife*(i*gap),3)
        df=pd.DataFrame(this)
```

```
        df['ET']=pd.DataFrame(df['ET'].tolist(),index=df.inde
x).mean(axis=1)
        df['phi']=df['Elf']/df['ET']
        result[str(round(50*i*gap,2))]["exp_active"]=
df['phi']
        this =
get_T(iteration,True,'exp',n,p,maxlife,maxlife*(i*gap),3)
        df=pd.DataFrame(this)
        df['ET']=pd.DataFrame(df['ET'].tolist(),index=df.inde
x).mean(axis=1)
        df['phi']=df['Elf']/df['ET']
        result[str(round(50*i*gap,2))]["pareto_active"]=
df['phi']
    return pd.DataFrame(result)
```

E[T] per k

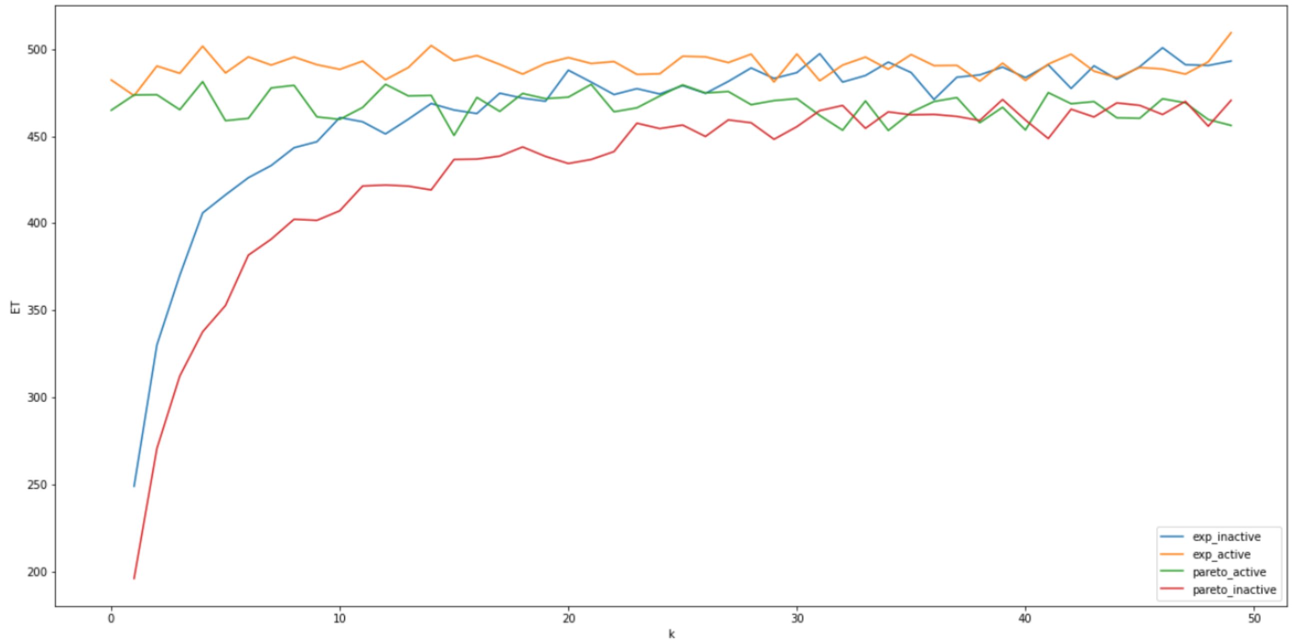
حال با شرایط زیر یعنی 500 تکرار با 50 گره، چهار حالت ارائه شده در مطالب بالاتر را ایجاد کرده و به کمک رسم نمودار به نمایش می‌گذاریم.

```
res1 = get_tdiffp(500,50,0.05,10,1000)
```

```
df = pd.DataFrame()
df['exp_inactive']=pd.DataFrame(pd.DataFrame(res1.iloc[0].tolist()).mean(axis=0,skipna=True))
df['exp_active']=pd.DataFrame(pd.DataFrame(res1.iloc[1].tolist()).mean(axis=0,skipna=True))
df['pareto_inactive']=pd.DataFrame(pd.DataFrame(res1.iloc[2].tolist()).mean(axis=0,skipna=True))
df['pareto_active']=pd.DataFrame(pd.DataFrame(res1.iloc[3].tolist()).mean(axis=0,skipna=True))
```

```
fig = pltt.figure(figsize=(20,10))
pltt.plot(df['exp_inactive'],label='exp_inactive')
pltt.plot(df['exp_active'],label='exp_active')
pltt.plot(df['pareto_active'],label='pareto_active')
pltt.plot(df['pareto_inactive'],label='pareto_inactive')
pltt.legend(loc='best')
pltt.xlabel('k')
pltt.ylabel('ET')
pltt.show()
```

که حاصل آن نمودار زیر می‌باشد.



بنا بر نمودار بالا، نتیجه می‌شود که وقتی ما در حالت غیرفعال قرار داریم و درجات پایین هم در دست هست، متوسط بقای گره‌ها خیلی کم است به نسبت حالاتی که فعال هستیم (یعنی بعد جستجو صورت می‌گیرد) یا غیرفعال هستیم و درجات بالایی داریم. نکته جالب توجه این است که در درجات بالا، حالت غیرفعال نمایی، متوسط بقای بهتری گاها نسبت به سه حالت دیگر تجربه می‌کند.

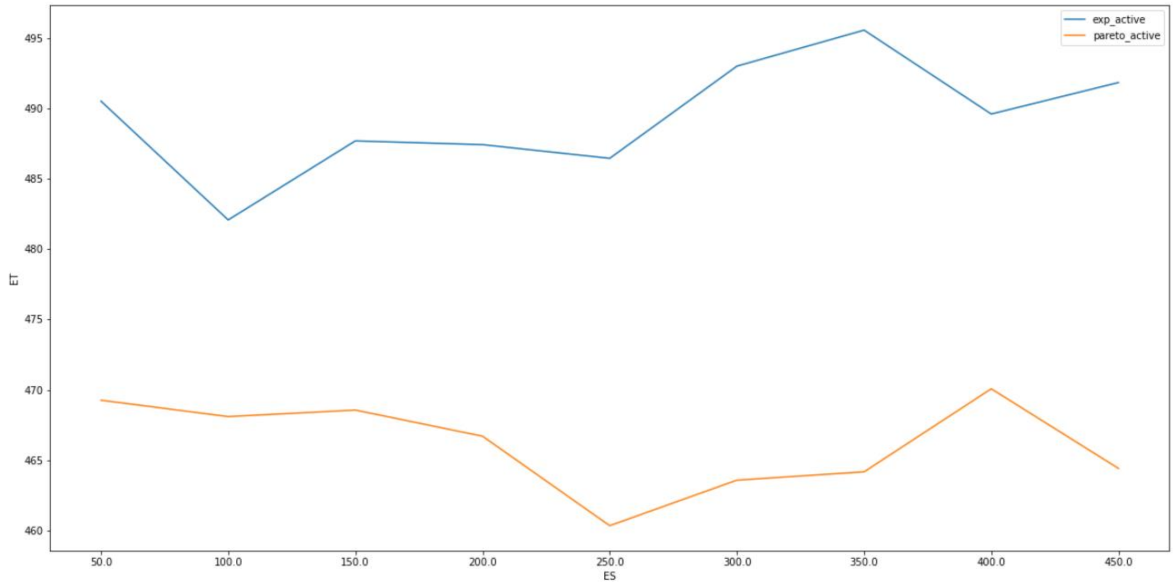
E[T] per E[s]

اینبار برای بخش **ب** و پس از توضیحات توابع مربوطه، با پارامترهای زیر اجرا می‌کنیم و نمودار در ادامه رسم می‌شود.

```
res1 = get_tdiffs(500,50,0.5,0.05,1000)
```

```
res1.T.plot(figsize=(20,10))
pltt.legend(loc='best')
pltt.xlabel('ES')
pltt.ylabel('ET')
```

می‌بینیم که متوسط بقای گره‌ها در حالت نمایی عملکرد بهتری از پارتو دارد.



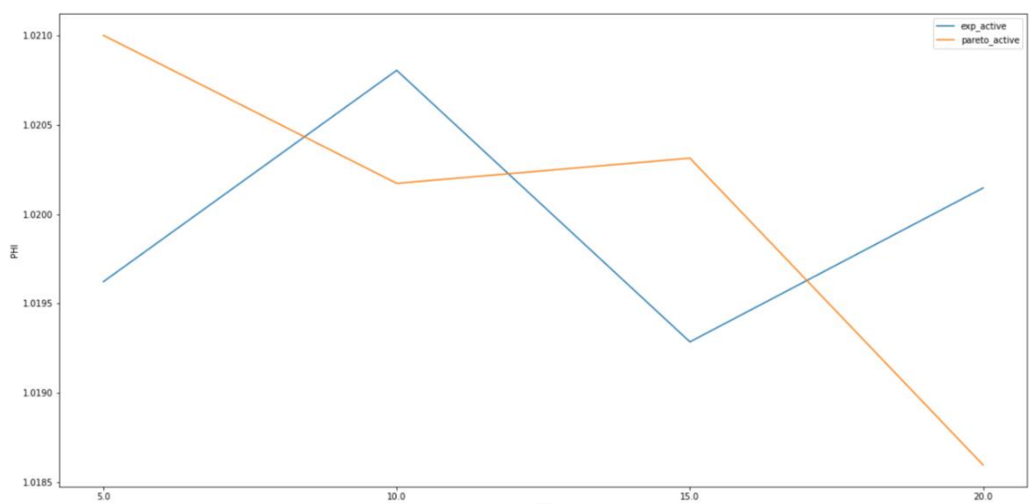
φ Per E[s]

```
res1 = get_phidiffs(500,50,0.5,0.1,1000)
```

```
df = pd.DataFrame()
df['exp_active']=pd.DataFrame(pd.DataFrame(res1.iloc[0].tolist(),index=res1.columns).mean(axis=1))
df['pareto_active']=pd.DataFrame(pd.DataFrame(res1.iloc[1].tolist(),index=res1.columns).mean(axis=1))
```

```
fig = pltt.figure(figsize=(20,10))
pltt.plot(df['exp_active'],label='exp_active')
pltt.plot(df['pareto_active'],label='pareto_active')
pltt.legend(loc='best')
pltt.xlabel('ES')
pltt.ylabel('PHI')
pltt.show()
```

با نمودار زیر که عملکردی پریشان دارد نمی‌توان تحلیل دقیق و صحیحی داشت، که البته منطقی هم هست همانطور که در فرمول شماره می‌بینیم که E[s] هم در صورت و هم در مخرج معادله قرار دارد.



α effect on φ and E[T]

با مقادیر فرض شده زیر این مرحله را پیش برده و نمودار E[T] برحسب **α** را رسم می‌کنیم.

```
result={}
iteration=500
n=10
p=0.5
maxlife=1000
for i in range(1,10):
    this = get_T(iteration,False,'pareto',n,p,maxlife,10,i)
    df=pd.DataFrame(this)
    df['ET']=pd.DataFrame(df['ET'].tolist(),index=df.index).mean(axis=1)
    df['phi']=df['Elf']/df['ET']
    result[str(i)]={}
    result[str(i)]["pareto_inactive"]= df[['phi','ET']]
    this = get_T(iteration,True,'pareto',n,p,maxlife,10,i)
    df=pd.DataFrame(this)
    df['ET']=pd.DataFrame(df['ET'].tolist(),index=df.index).mean(axis=1)
    df['phi']=df['Elf']/df['ET']
    result[str(i)]["pareto_active"]= df[['phi','ET']]
result = pd.DataFrame(result)
```

```
res1 = result.applymap(lambda x:x['ET'])
```

```
res2=result.applymap(lambda x:x['phi'])
```

```
df = pd.DataFrame()
df['exp_active']=pd.DataFrame(pd.DataFrame(res1.iloc[0].tolist(),index=res1.columns).mean(axis=1))
df['pareto_active']=pd.DataFrame(pd.DataFrame(res1.iloc[1].tolist(),index=res1.columns).mean(axis=1))
```

```
fig = pltt.figure(figsize=(20,10))
pltt.plot(df['exp_active'],label='exp_active')
pltt.plot(df['pareto_active'],label='pareto_active')
pltt.legend(loc='best')
pltt.xlabel('alpha')
pltt.ylabel('ET')
pltt.show()
```

انتظار می‌رود به دلیل حضور آلفا در مخرج کسر سازنده متوسط طول عمر، با افزایش این مقدار ما کاهش $E[L_i]$ و در نتیجه کاهش متغیر $E[T]$ برای ثابت ماندن φ را باید شاهد باشیم.

نمودار این مورد را که در ادامه آورده‌ایم پی به درستی فرضیه بالا می‌بریم.