1- تحلیل likelihood smpling

در این روش evidence ها را تنظیم میکنیم و تنها برای باقی نود ها به ترتیب توپولوجیکال سورت یک مقدار رندوم بر حسب سایر نود ها تولید میکنیم.

در این روش سمپل های تولید شده چند متغیر ثابت دارند که به صورت رندوم مقداردهی نشده اند. برای همین به هر سمپل یک وزن میدهیم و وزن هر سمپل برابر احتمال رخ دادن evidence هاست. در نهایت وزن سمپل هایی که query در آن برقرار است را جمع میکنیم و تقسیم بر مجموع وزن سمپل ها میکنیم(در همه سمپل ها evidence برقرار است).

در تابع likelihood__sample ابتدا نود ها را بر اساس توپولوجیکال مرتب میکنیم تا ابتدا پرنت ها و سپس فرزند ها را مقدار دهی کنیم. سپس آرایه سمپل ها را initialize میکنیم.

سپس در یک حلقه 10000 تایی سمپل ها را تولید میکنیم. این حلقه به این صورت کار میکند:

```
for i in range(10000):
    value = [-1] * n
    w = 1
    for vertex in sort_vertex:
        if evidence[vertex] == -1:
            value[vertex] = sample_vertex(vertex, graph2[vertex], value, sort_vertex, probability)
        else:
            value[vertex] = evidence[vertex]
            w *= find_row(probability[vertex], value)
    samples.append([value, w])
```

ابتدا تمام مقدار ها با -1 initilize میشوند.(یعنی مقداری داده نشده). سپس در یک لوپ هر نود از شبکه بیز بررسی میشود. اگر این نود در evidence ها داده شده باشد، مقدار آن نود را در value مقدار تعیین شده در evidence میگذاریم و w را در احتمال رخ دادن آن مقدار ضرب میکنیم. در غیر این صورت با استفاده از تابع sample_vertex یک مقدار رندوم با توجه به سایر value های تعیین شده به آن نود میدهیم.

در آخر حلقه هم سپمل ایجاد شده را به آرایه اد میکنیم.

```
good_sample = 0
sum_sample = 0
for sample in samples:
    flag = 1
    for i in range(len(nodes)):
        if bool(values[i]) != samp
            flag = 0
            break
    sum_sample += sample[1]
    if flag == 1:
        good_sample += sample[1]
return good_sample / sum_sample
```

در بخش بعدی با استفاده از یک حلقه، مجموع سمپل هایی که در آن query برقرار بوده و مجموع کل سمپل ها محاسبه میشوند. و در نهایت با تقسیم این دو مقدار احتمال مورد نظر به دست میآید.

2- تحلیل gibbs smaple

در این روش evidence ها ثابت قرار داده میشوند و سایر نود ها به صورت رندم مقدار دهی میشوند.(initialize اولیه)

سپس برای تولید هر سمپل در این روش، یک حلقه روی نود هایی که evidence نیستند زده میشود.در هر نوبت حلقه ابتدا مقدار آن راس پاک شده و دوباره با توجه به سایر راس ها به آن یک مقدار رندوم داده میشود. این اتفاق برای سایر راس های غیر evidence هم رخ میدهد و وقتی همه راس ها تمام شدند یک sample تولید شده.

برای سمپل های بعدی از همین سمپل ایجاد شده استفاده میشود. به همین علت هر سمپل روی سمپل بعد تاثیر میگذارد و به مرور سمپل ما به واقعیت و رویدادی که بیشتر احتمال را دارد نزدیک تر میشود.

در تابع gibbs__sample دوباره ابتدا نود ها به ترتیب topological مرتب میشوند، آرایه سمپل و value ها initialize میشوند.

```
for i in range(n):
    if evidence[i] != -1:
        value[i] = evidence[i]
    else:
        if np.random.random() <
            value[i] = True
        else:
            value[i] = False
```

در حلقه اول، ابتدا نود هایی که در evidence هستند مقدار دهی میشوند، و سایر نود ها هم بر اساس یک احتمال رندوم 50/50 مقدار دهی میشوند.

```
for i in range(10000):
    new_value = [-1] * n
    for vertex in sort_vertex:
        if evidence[vertex] == -1:
            value[vertex] = -1
            value[vertex] = sample_vertex(vertex, graph2[vertex], value, sort_vertex, probability)
            new_value[vertex] = value[vertex]
        else:
            new_value[vertex] = value[vertex]
    samples.append(new_value)
```

در حلقه دوم با اندازه 10000، در هر حلقه یک سمپل تولید میشوند. روی راس ها یک حلقه میزنیم. اگر راس در evidence ها بود مقدار آن را مقدار تعیین شده در evidence قرار میدهیم و در غیر این صورت، مقدار آن راس را -1 (تعیین نشده) قرار میدهیم و با استفاده از تابع sample_vertex یک مقدار رندوم با توجه به سایر راس ها برای آن تولید میکنیم.

وقتی همه نود ها را بررسی کردیم، در آخر حلقه سمپل ایجاد شده رو به آرایه samples اد میکنیم.

```
good_sample = 0
for sample in samples:
    flag = 1
    for i in range(len(nodes)):
        if bool(values[i]) != sample[nodes[i]]:
            flag = 0
            break
    if flag == 1:
        good_sample += 1
return good_sample / len(samples)
```

در نهایت نیز با یک حلقه روی سمپل ها، تعداد سمپل هایی که در آن query برقرار است را میشماریم و تقسیم بر تعداد کل سمپل ها میکنیم و پاسخ را ریترن میکنیم.

بررسی کوئری ها

1- کوئری 6 از input1

gibbs: این روش در این کوئری دچار خطای زیادی شده. علت آن این است که مزیت اصلی این روش این است که نود ها با اثر گذاری روری یک دیگر سمپل ها را به مرور زمان بهبود میدهند. اما در کوئری 6، نود ها طوری انتخاب شده که از بین نود هایی که evidence نیستند(ثابت نیستند)، اثر گذاری ای روی یکدیگر ندارند. یعنی اکثر نود ها یا پرنت ندارند، یا پرنتی دارند که evidence است و مقدار ثابتی دارد و بهبود پیدا نمیکند. به همین علت سمپل ها بهبود پیدا نمیکنند و gibbs نتیجه درستی نمیدهد.

perior , rejection: این دو روش مشابه هم هستند تنها مزیت rejection این است که سمپل ناسازگار با evidence را زود تر تشخیص میدهد. تفاوت نتیجه این دو تابع هم به علت رندوم بودن سمپل هاست.

likelihood: در این روش سپمل هایی که گرفته میشوند، مفید تر هستند یعنی در perior و rejection از هر 1000 سمپل، مثلا 500 تای آن ها با evidence ها سازگارند اما در likelihood، تمام 1000 سمپل سازگار هستند. به همین علت معمولا این روش احتمال را بهتر از دو روش قبل حساب میکند.

2- کوئری 2 از input1

gibbs: این روش در این کوئری هم دچار خطا شده اما نسبت به قبلی کم تر است. علت آن این است که در این کوئری اثر گذار نود ها روی هم بیشتر است. در کوئری قبل تنها اثر گذاری از E به D بود. البته از آنجایی که D یک پرنت دیگر به نام C هم داشت و C، evidence بود، نود ها خیلی اثر گذاری روی هم نداشتند. اما در این کوئری B کاملا از A تاثیر میپذیرد و کمی بهبود پیدا میکند. برای همین کمی نتایج آن از قبلی بهتر است.

perior, rejection, likelihood: این سه روش هم منطقی مانند کوئری قبلی دارند. likelihood احتمالا از دوتای دیگر کمی دقیق تر است.

3- کوئری 3 از input1

gibbs: در این کوئری نود ها اثر گذاری زیادی روی هم دارند به همین علت این روش میتواند به مرور سمپل ها را بهبود دهد و به نتایج واقعی تری دست پیدا کند.

rejection, perior, likelihood: منطق مشابه با موارد قبل

4- کوئری 6 از input2

gibbs: در این روش به علت اثر گذاری مناسب نود ها روی هم gibbs به خوبی سمپل ها را بهبود میدهد و نتایج مناسبی میدهد.

perior, rejection: در این روش احتمال رخ دادن evidence ها بسیار پایین است(یک هزارم) به همین علت در روش های perior, rejection، از 10000 سمپل تولید شده تعداد کمی از آن ها با evidence های داده شده سازگار هستند(حدود 10 الی 20 تا) به همین علت جامعه آماری کوچک است و احتمال محاسبه شده دچار خطای زیادی است.

likelihood: در این روش با توجه به این که evidence ها ثابت هستند، تمام سمپل های تولید شده سازگار هستند وجامعه آماری مناسب است و احتمال دچار خطا نیست و جواب میدهد.

5- کوئری 6 از input3

gibbs: در این کوئری هم مانند کوئری 6 از input1 نود ها روی هم اثر گذاری ندارند و gibbsدچار خطا میشود.

perior, likelihood, rejection: منطق مشابه با کوئری 6 از input1.

تحلیل هر روش:

gibbs:

زمانی که شبکه داده شده کوچک باشد، پس ازمشخص شده evidence ها نود باقی مانده ممکن است اثر گذاری خیلی کمی روی هم داشته باشند یا اصلا نداشته باشند. در این صورت سمپل ها بهبود پیدا نمیکنند و gibbs دچار خطای زیادی میشود.

این روش برای مواقعی مناسب است که شبکه بزرگ است و ارتباطات زیاد و پیچیده بین نود ها وجود دارد. در این صورت این روش با بهبود هر سمپل، سمپل ها را به سمتی حرکت میدهد که به واقعیت نزدیک تر باشند و احتمال رخ دادن آن ها بیشتر باشد. بنابراین با تعداد سمپل های کم تری خیلی زود به نتیجه مناسب میرسد.

likelihood:

این روش برای زمانی مناسب است که احتمال رخ دادن evidence ها خیلی کم است. دراین موارد اگر از perior یا rejection استفاده کنیم از سمپل های تولید شده درصد خیلی کمی از آن ها با evidence ها سازگار هستند و اکثر سمپل ها بدون استفاده خواهند ماند.

perior:

از این روش زمانی میتوان استفاده کرد که شبکه بیز داده شده خیلی ساده باشد و احتمال رخ دادن evidence ها کم نباشد.(یا evidence ای نداشه باشیم.) این روش ساده و خام ترین روش نمونه گیری است و بهینه سازی نشده. بنابراین در اکثر موارد بهتر است از روش های بهینه تر مانند likelihood یا rejection استفاده کنیم.

rejection:

این روش در واقع بهبود یافته perior است. این روش برای زمانی که تعداد نود ها زیاد باشد به بهبود سرعت اجرا خیلی کمک میکند. زیرا ناسازگاری یک سمپل با evidence را خیلی زود تشخیص میدهد و محاسبات اضافی برای سایر نود ها انجام نمیشود.