Índice

1. Introducción 2

2. Reyes y gallinas. 3

3. El juego del Hex y núcleos 8

4. Pseudo-oraciones 11

Pseudo-oraciones, quasi-núcleos. Las matemáticas y el lenguaje.

Miguel Angel Mejía Galindo

5 de junio de 2025

1. Introducción

Hoy en día varios modelos de generación de texto, como *ChatGTP*, empiezan su proceso generativo con un input del usuario y mediante cálculos complejos (y el input) seleccionan una palabra para continuar el texto. Recursivamente toman ese resultado y vuelven a predecir qué palabra sigue; así hasta que su algoritmo les dice que acaben el proceso recursivo.

En términos muy simples, dado un texto ChatGTP sabe como continuar "bien" la siguiente palabra.

Con esta idea, podemos modelar este comportamiento con una digráfica (gráfica dirigida) donde los vértices son palabras y hay exactamente una flecha de ida y regreso entre dos palabras (el análogo de arista en digráficas), así lo que hacen estos modelos de texto generativo, es tomar un camino dirigido (que sería el input del usuario) y extenderlo (i.e. continuar el camino) un vértice a la vez.

Esta idea puede refinarse poniendo pesos a las flechas de la digráfica, sin embargo los modelos como ChatGTP hacen más que esto, pues asignan a cada output paso un valor diferente (a veces referido como token) y los valores del input cambian dependiendo del output anterior.

A continuación, daremos los conceptos, nociones, definiciones y teoremas que necesitamos para poder generar pseudo-oraciones, que serán algo mucho mas débil que lo que hace ChatGTP, pero más simple y sencillo de entender.

Para esto necesitamos un tipo especial de digráfica y un conjunto de vértices especial de una digráfica.

Este conjunto lo presentamos junto con el siguiente articulo *The King Chickens Theorems* [4] donde se introduce su definición dual[1], el conjunto rey.

[1] En teoría de gráficas, al estar usando digráficas con sus definiciones y teoremas, se le llama definición, teorema dual o simplemente dual a la definición, teorema que se obtiene por cambiar la dirección de las flechas en la definición o teorema.

2. Reyes y gallinas.

Maurer, el principal autor en [4], se interesó e inspiró por un artículo científico sobre dominancia en una parvada de aves de corral (gallinas), y junto algunos de sus alumnos, intentó caracterizar a la gallina (o gallinas) más dominante (dominantes) con la definición de conjunto rey en digráficas.

Un conjunto rey, es un conjunto de vértices de una digráfica que cumplen dos condiciones, la primera es que cualesquiera dos vértices diferentes dentro del conjunto no tienen flechas entre sí; lo segundo es que todo vértice fuera del conjunto rey es 2-dominado [2] por algún vértice del conjunto rey.

La idea que tenían era que los vértices del conjunto rey representaban gallinas que picaban a otras, o que picaban a alguien que pica a otra, además estas no se picaban entre sí, por eso la analogía reyes que tienen subordinados, a la vez que sus subordinados tienen subordinados. Además, un rey no manda sobre otro rey.

Un ejemplo de conjunto rey, son los vértices azules en la figura 1.

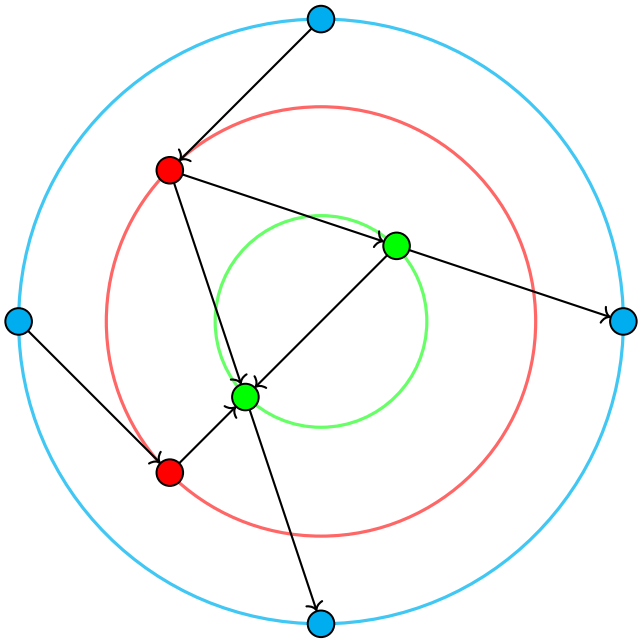

Figura 1: Conjunto rey (ejemplo)

La figura 1, contiene círculos concéntricos para para distinguir los tres tipos de vértices que se pueden clasificar de acuerdo al conjunto rey, los vértices del conjunto rey (azul), los subordinados por estos (rojos) y los que son subordinados de estos últimos y no son reyes (verdes).

[2] Un vértice x es 2-dominado, si existe un vértice y del conjunto rey que llega en máximo dos pasos al vértice x. Usando la notación de distancia en digráficas es $d(y, x) \leq 2$.

En la digráfica, uno puede observar que los vértices verdes tienen una flecha hacía a unos azules. Este es uno de los primeros problemas que rompe la noción de que un conjunto rey es dominante, ya que hay reyes que son dominados por subordinados. Esto no es un gran problema, si pensamos que son reyes locales en vez de globales y hay reyes mas locales que otros.

De cierta manera esto se compensa, ya que toda digráfica tiene al menos un conjunto rey. En su articulo, Maurer lo prueba para un caso particular de digráficas llamadas torneos.

Una demostración del resultado mas general se prueba para el concepto dual del conjunto rey llamado conjunto quasi-núcleo[3] (o simplemente quasi-núcleo) en [1], donde se profundiza en la definición de conjunto núcleo más allá de lo explicado en este escrito.

OJO: Los quasi-núcleos no necesariamente son únicos. Un ejemplo de ello viene en un artículo de Jacob y Meyniel [3], donde dan una familia de digráficas que satisface tener al menos tres quasi-núcleos.

Volviendo a Maurer, sus alumnos y las gallinas, en su artículo menciona varios resultados sobre los conjuntos reyes en los torneos, entre ellos menciona que en un torneo un conjunto rey consta de un solo elemento, y a medida que consideramos torneos con mas vértices, la probabilidad de que un vértice arbitrario sea un conjunto rey tiende a 1.

Este último resultado también implica que eventualmente, al aumentar la cantidad de vértices, todo vértice sera dominado por alguien y además sera un conjunto rey.

De esto, Maurer concluye que los torneos no son óptimos para modelar la dominancia de las gallinas y que se debe buscar otro tipo de digráficas que no permitan que todos los vértices se vuelvan eventualmente conjuntos reyes. Por ello propone los torneos lineales pero ese tema no nos concierne por ahora.

En nuestro caso usaremos un tipo especial de digráficas generadas por textos, llamadas digráficas de co-ocurrencia, donde los vértices son palabras y hay una flecha de la palabra x a la palabra y si x es la palabra anterior a y en un texto. Esta relación es la que llamamos de co-ocurrencia.

Una propiedad básica de estas digráficas es que siempre existe un camino desde el primer vértice (la primera palabra) hacía cualquier otro vértice (cualquier otra palabra).

Como ejemplo de este tipo de digráficas, considérese el texto Ün gato ve un gato". Entonces (sin distinguir mayúsculas de minúsculas) tenemos la figura 2:

[3]La definición dual del conjunto rey es el conjunto quasi-núcleo en este caso pedimos que $d(x, y) \leq 2$. En palabras es que todo vértice fuera del quasi-núcleo llega en dos pasos al quasi-núcleo.

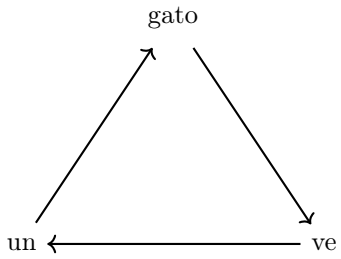

Figura 2: Ejemplo: Digráfica de co-ocurrencia

Si distinguimos mayúsculas y minúsculas tenemos la figura 3:

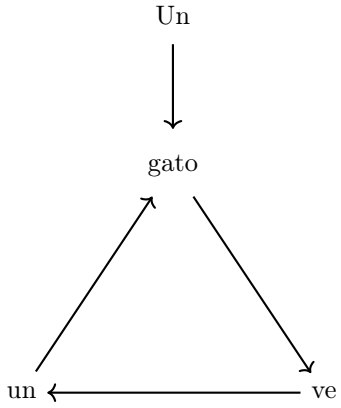

Figura 3: Ejemplo: Digráfica de co-ocurrencia

En general, uno puede pensar las digráficas generadas por un texto como una cadena (por la propiedad antes mencionada) y las generadas por varios textos como cadenas sobre cadenas.

Este tipo de digráficas con la propiedad mencionada, en **algunos textos** es llamada gráfica enraizada (rooted graph) según Wikipedia en [7].

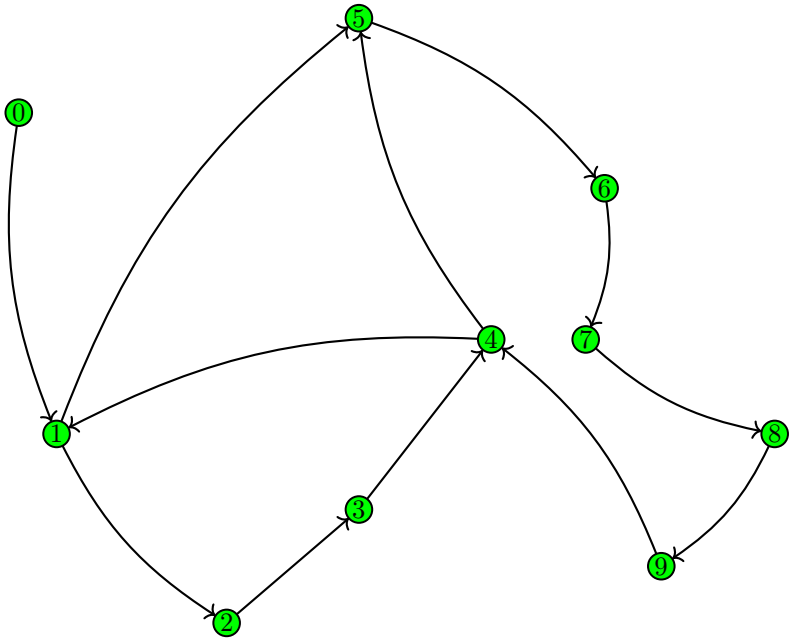

Figura 4: Digráfica de ejemplo

Ahora describimos una forma de calcular el quasi-núcleo usando de ejemplo la digráfica de la figura 4.

Empezamos reduciendo la digráfica, para esto vamos recursivamente quitando un vértice x y los vértices que tienen una flecha hacía x.

Tomando desde $x = 9$ y aplicando lo anterior tenemos esta sucesión de digráficas (figuras de 5 a 8):

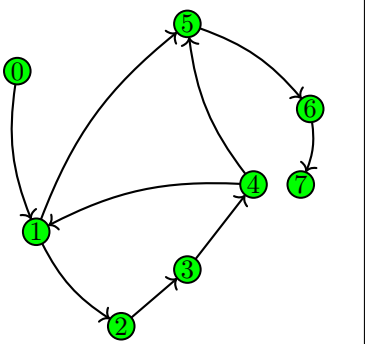

Figura 5: $x = 9$

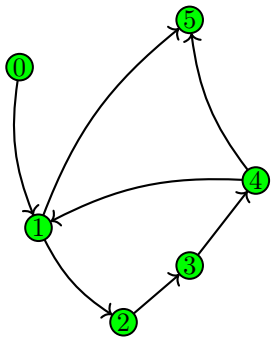

Figura 6: $x = 7$

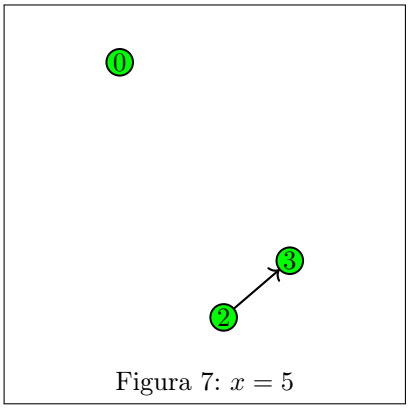

Figura 7: $x = 5$

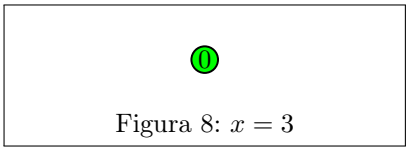

Figura 8: $x = 3$

El quasi-núcleo de esta última digráfica (figura 8), es $S = \{0\}$, ahora lo completamos a un quasi-núcleo de la digráfica de la figura 4.

Para esto, basta agregar a S los vértices x que quitamos tales que no tienen flechas y no les llegan flechas de algún vértice s en el quasi-núcleo.[4]

En forma de lista ponemos como se va completando el quasi-núcleo de la digráfica de la figura 4

$$\begin{aligned} & S = \{0\} \\ x = 3 \quad & S = \{0, 3\} \\ x = 5 \quad & S = \{0, 3, 5\} \\ x = 7 \quad & S = \{0, 3, 5, 7\} \\ x = 9 \quad & S = \{0, 3, 5, 7, 9\} \end{aligned}$$

Volviendo a representar los vértices azules como los vértices del quasi-núcleo y los demás como rojos y verdes dependiendo de su relación con el quasi-núcleo, tenemos lo siguiente

[4] Esto se puede ver en la demostración de Chvátal y Lovász en [1].

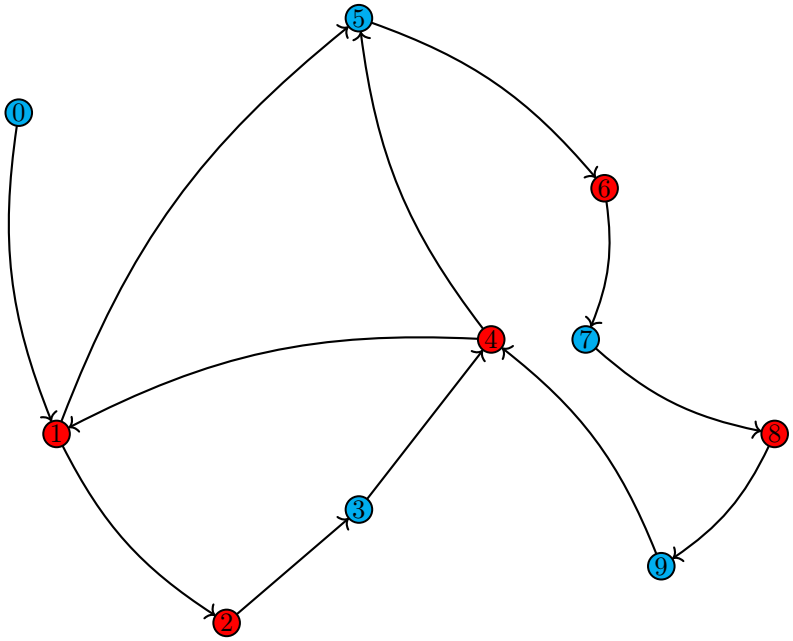

Figura 9: Digráfica con quasi-núcleo en azul

Comentario: Lo que se puede controlar sobre el quasi-núcleo a priori es que vértices no van a estar en el quasi-núcleo, pues en la demostración (en [1]) se puede ver que los vértices que llegan a un paso a un vértice eliminado no están en el quasi-núcleo de la digráfica reducida y no pueden estar en ningun quasi-núcleo que se construya desde la digráfica reducida.

Con solo esto ya podríamos hablar de cómo generar las pseudo-oraciones usando el quasi-núcleo, pero antes quisiera hablar sobre cómo se puede casi recuperar el concepto de unicidad usando conjuntos núcleos.

Nuestro primer ejemplo de un conjunto núcleo de hecho es el conjunto de vértices azules en la figura 9.

En términos de como clasificamos con colores los vértices de la digráfica, un núcleo es un quasi-núcleo sin vértices de color verde.

Ahora en la siguiente sección ahondamos en estos conjuntos introduciendolos con un juego.

3. El juego del Hex y núcleos

A veces las digráficas enraizadas, anteriormente mencionadas, se usan para modelar estados de juegos por turnos, por ejemplo para el ajedrez, un vértice es una etapa de alguna partida y hay una flecha de una etapa x a otra etapa y, si son turnos consecutivos.

En la sección anterior vimos que siempre las digráficas tienen quasi-núcleo,

pero hay un concepto mas fuerte que es el de conjunto núcleo[5], pues todo conjunto núcleo es un quasi-núcleo, pero no siempre pasa al revés.

Por otro lado, para las digráficas que son acíclicas tenemos un resultado muy fuerte, que nos dice que siempre tienen un conjunto núcleo y este además es único.[6]

Ahora un ejemplo de una digráfica acíclica enraizada son los posibles estados de un juego de Hex, creado por Piet Hein (según Martin Gardner en [2]), pues el juego de Hex (clásico) no permite que un jugador pueda repetir jugadas ni saltarse turnos y es finito, osea el juego siempre acaba en algún número (natural) de estados.

Con este resultado fuerte sabemos que la digráfica generada por los estados del juego del Hex tiene un único núcleo, este núcleo corresponde a los movimientos de un jugador, es decir hay un jugador que siempre gana si juega óptimamente.

Por esta razón normalmente cuando uno juega Hex, aparece la opción para el segundo jugador de tomar el primer movimiento del primer jugador como si fuera suyo.

Regresando a las digráficas de co-ocurrencia, resulta que no siempre son acíclicas (en el sentido de digráficas) como en los ejemplos anteriores. Sin embargo podemos construir una digráfica acíclica de esta digráfica de co-ocurrencia, llamada condensación de la digráfica.

Esta gráfica básicamente se construye tomando todos los vértices que se alcanzan mutuamente (i.e. hay un camino dirigido de uno al otro y viceversa) y los colapsamos a un vértice (normalmente llamada componente fuerte). Las flechas nos llevan de un vértice a otro tal que no hay un camino de regreso.

Un ejemplo visual es la figura 10, donde los rectángulos son vértices y las flechas son aquellas que no se quedan atrapadas en uno solo rectángulo.

[5] En este caso en vez de pedir $d(x, y) \leq 2$, pedimos $d(x, y) \leq 1$.

[6] Se dice que John von Neumann y Oskar Morgenstern lo demostraron en su libro [6], yo no encontré el resultado, probablemente lo enunciaron en otra notación por que la notación que usan es de hace ocho décadas.

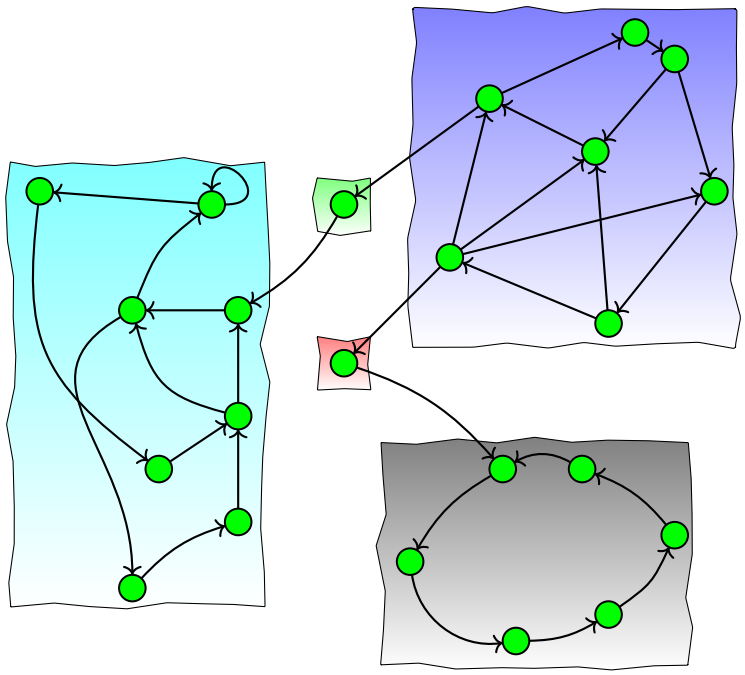

Figura 10: Condensación de una digráfica

Es bastante directo ver por qué la digráfica de condensación es acíclica mediante una demostración por contradicción y notando que siempre una condensación de una digráfica tiene un vértice del que que no salen flechas.

Por ejemplo, la condensación de la digráfica de la figura 3 es la siguiente

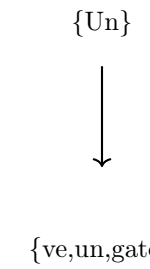

Figura 11: Ejemplo: Digráfica de condensación

Aquí el núcleo solo es el conjunto {ve, un, gato}.

4. Pseudo-oraciones

Ahora usando la condensación de la digráfica y el núcleo de esta digráfica, podemos construir a partir de una palabra en algún vértice, una pseudo-oración[7], que empieza en esa palabra y acaba en alguna palabra de un vértice en el núcleo.

En el ejemplo anterior se pueden generar las pseudo-oraciones

Un gato
Un gato ve
Un gato ve un gato
Un gato ve un gato ve un gato
⋮
ve un gato
un gato ve
gato ve un

El proceso para generar estas oraciones es tomar una palabra inicial y continuar el camino dentro de una componente fuerte hasta llegar a una palabra que en la digráfica de co-ocurrencia tiene una flecha hacía otra componente fuerte y repetir este algoritmo hasta donde se pueda (sabemos que si acaba pues las componentes fuertes son finitas en digráficas finitas).

La ventaja y desventaja del núcleo es que las oraciones acaban de dos formas con una palabra en el núcleo o en un ciclo, y además entre mas conectada este la digráfica (haya mas flechas entre palabras), la condensación de la gráfica se va haciendo mas chica.

[7] Con pseudo-oración me refiero a los textos en los que, cada par de palabras contiguas son congruentes gramaticalmente porque la digráfica de co-ocurrencia fue hecha para que se de esa relación, pero no necesariamente pasa esto para una cantidad de palabras arbitraria.

Para evitar tener digráficas triviales, otra forma de generar estos pseudo-oraciones es con el quasi-núcleo que tiene una dinámica a mi parecer mas interesante.

Recordemos que con un quasi-núcleo, podemos clasificar a los vértices en una digráfica en tres tipos; luego en la digráfica de co-ocurrencia toda palabra (posiblemente excepto la última), tiene una flecha hacía otra palabra.

Con esto (y en términos de como clasificamos los vértices con colores dada su relación con el quasi-núcleo) tenemos que dado un vértice de color verde siempre tiene al menos una flecha hacía un vértice rojo, un vértice rojo tiene una flecha siempre tiene al menos hacía un vértice azul, esto simplemente por que la digráfica tiene un quasi-núcleo; luego hay una probabilidad no cero de que un vértice azul tenga una flecha hacía un vértice rojo o verde, esto por que la digráfica de co-ocurrencia como la definimos, tiene la propiedad de que un vértice (exceptuando el último) tiene una flecha hacía algún otro vértice.

Visualmente la dinámica, comportamiento se ve como en la figura 12:

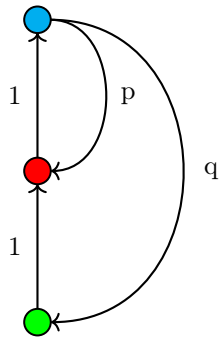

Figura 12: Dinámica del quasi-núcleo en digráficas de co-ocurrencia

Con p representando la probabilidad de que un vértice azul vaya a un vértice rojo y q la probabilidad de que ese vértice azul vaya a un vértice verde.

Los 1's significan que un vértice verde siempre tiene una flecha hacía un vértice rojo y un vértice rojo siempre tiene una flecha hacía un vértice azul.

Me gusta pensar esta dinámica como escalar una cadena de montañas. Los vértices azules son las puntas de las montañas, los rojos son las laderas y los verdes las bases de las montañas.

Y las probabilidades en este caso nos dicen que siempre podemos subir una montaña pero al tirarnos de la punta de la montaña podemos caer en una ladera o en una base.

Técnicamente, el concepto de conjunto núcleo acepta lazos, que son flechas que salen de un vértice y van a él mismo, pero para la dinámica anteriormente mencionada no afecta mucho más que para quedarse atrapado en un vértice.

Hasta ahora, hemos dicho las definiciones, teoremas que necesitamos para hablar de quasi-núcleos, núcleos y digráficas donde tienen ciertas propiedades que nos dan la dinámica anterior, hablemos ahora del código para generar estas

oraciones.

Para el código que ponemos aquí hacemos uso del lenguaje de programación python y algunas las librerías como networkx, re y matplotlib. Aparte de las usuales como numpy, random y math.

En la primera parte del código, creamos la digráfica de co-ocurrencia con la que vamos a trabajar, para esto usamos en especifico las librerías networkx, re de la siguiente manera

```
A = ["Un ejemplo", "Oracion ejemplo", "Un gato ve un ejemplo"]

def split_sentence(sentence):
    return [word for word in re.findall(r'\b\w+\b|[^\w\s]', sentence)]

Aa = [split_sentence(A[i]) for i in range(len(A))]

G = nx.DiGraph()

for i in range(len(A)):
    for j in range(len(Aa[i])-1):
        G.add_edge(Aa[i][j],Aa[i][j+1])

```

Listing 1: Generación de la digráfica

OJO: La forma de extraer palabras y caracteres especiales (tales como la lematización o extracción en bruto)[8] de una lista de textos influye en el tiempo en encontrar el quasi-núcleo.

Luego pasamos a la parte de encontrar un quasi-núcleo de la digráfica, en el código las siguientes dos funciones sirven para encontrar un quasi-núcleo de una digráfica:

La primera función construye un conjunto independiente maximal (al azar)

```
def find_max_independent_set(graph):
mis = set()
nodes = list(graph.nodes())

while nodes:
    node = nodes.pop()
    mis.add(node)
    nodes = [n for n in nodes if not (n in graph[node] or node in graph
    [n])]

return mis

```

Listing 2: Crear conjunto independiente maximal

Para la segunda función, en términos de como hemos estado coloreando las digráficas, verifica que los vértices se puedan repartir en los tres colores que hemos estado manejando

[8] La lematización de una palabra es su forma canónica (no esta, por ejemplo, en tiempo pasado) y la extracción en bruto me refiero a que se extrae de un texto, las cadenas de texto separadas por espacios.

```python
def dist_2(G, S):
    for node in set(G.nodes) - S:
        if all(nx.shortest_path_length(G, node, s) > 2 for s in S if nx
    .has_path(G, node, s)):
            return False
    return True

```

Listing 3: Verificar la propiedad de 2-absorbentes

Para esta parte de encontrar un quasi-núcleo juntamos estas dos funciones en un loop hasta encontrar un quasi-núcleo.

```python
def refine_set(G):
    S = find_max_independent_set(G)
    while not dist_2(G, S):
        S = find_max_independent_set(G)
    return S

```

Listing 4: Parte del código que busca el quasi-núcleo

Hay formas de optimizar este proceso, tomando una sub-digráfica de la digráfica de co-ocurrencia, por ejemplo una optimización posible es el siguiente código:

Reducimos la digráfica de co-ocurrencia (esta es la sub-digráfica)

```python
nodes = list(G.nodes())

NodesToElim = []

Rev_nodes = list(reversed(nodes))

H = G.copy()

for n in range(0,len(nodes)-5):
    v = Rev_nodes[n]
    Pred_v = G.predecessors(v)

    H_nodes = list(H.nodes())

    if v in H_nodes:
        NodesToElim.append(v)
        H.remove_node(v)
        H.remove_nodes_from(Pred_v)

H_nodes = list(H.nodes())
```

Listing 5: Reducción de la digráfica

Encontramos un quasi-núcleo de la digráfica reducida usando las funciones anteriormente mencionadas

```python
Qn = refine_set(H)

```

Listing 6: Quasi-núcleo de la digráfica reducida

Ahora acompletamos el quasi-núcleo de la digráfica reducida a una de la digráfica de co-ocurrencia

```
Inv_Original_NodesToElim = list(reversed(NodesToElim))

for n in range(0,len(NodesToElim)):

    Qn_copy = Qn.copy()

    if any(G.has_edge(Inv_Original_NodesToElim[n],s) or G.has_edge(s,
    Inv_Original_NodesToElim[n]) for s in Qn_copy):
        continue
    else:
        Qn.add(Inv_Original_NodesToElim[n])

```

Listing 7: Quasi-núcleo de la digráfica de co-ocurrencia

Con esto ya calculamos el quasi-núcleo de la digráfica de co-ocurrencia, pasamos a la parte del código que genera las pseudo-oraciones que es bastante básico.

Lo que hace este código es, tomar una palabra al azar de la digráfica, hacer un camino dirigido que empieza en esa palabra que sigue la dinámica que mencionamos junto con la figura 12.

```
N1 = set(v for v in set(G.nodes-Qn) if any(nx.shortest_path_length(G, v
    , s) == 1 for s in Qn if nx.has_path(G, v, s)))

N2 = set(v for v in set(G.nodes-Qn-N1) if any(nx.shortest_path_length(G
    , v, s) == 2 for s in Qn if nx.has_path(G, v, s)))

def SigNodo(current_node, quasi_kernel, graph):
    if current_node in quasi_kernel:
        for neighbor in graph[current_node]:
            if neighbor not in quasi_kernel:
                return neighbor
    elif current_node in N1:
        for neighbor in graph[current_node]:
            if neighbor in quasi_kernel:
                return neighbor
    elif current_node in N2:
        for neighbor in graph[current_node]:
            if neighbor in N1:
                return neighbor
    return None

for m in range(50):
  InCam = random.choice(nodeso)
  path = [InCam]
  for _ in range(50):
      next_node = SigNodo(InCam, Qn, G)
      if next_node is None:
          break
      path.append(next_node)
      InCam = next_node

  print(f'Camino dirigido: {" ".join(path)}')
```

Listing 8: Generación de pseudo-oraciones

Lo que es bastante básico del código es la función *SigNodo*, dado que no busca conservar la estructura gramática, aunque la digráfica de co-ocurrencia preserva la estructura gramática entre dos palabras (que tengan una flecha entre si).

Ahora cuatro ejemplos de la ejecución del código.

Tomando de la página [8], el volumen 3 capitulo 7, del libro de *'Frankenstein o el prometeo moderno'* [5].

• A gigantic monster drink deep grief which travellers , together with strength and composure ; let him feel , together with strength and composure ; let him feel

• He had belonged to cease but seen not die , together with strength and composure ; let him feel , together with strength and composure ; let him feel

• daily lost . "By the earth , together with strength and composure ; let him feel , together with strength and composure ; let him feel

• Greeks wept for them ! Covered with strength and composure ; let him feel , together with strength and composure ; let him feel

Una cosa a notar es que las pseudo-oraciones generadas parecen acabar en un bucle como las versiones tempranas de ChatGTP y ninguna es una oración que aparezca en el libro.

El código para generar pseudo-oraciones y las digráficas, se puede descargar en https://github.com/M1AnM3/Pseudo-texto.

Referencias

[1] V. Chvátal and L. Lovász. Every directed graph has a semi-kernel. In Claude Berge and Dijen Ray-Chaudhuri, editors, *Hypergraph Seminar*, pages 175–175, Berlin, Heidelberg, 1974. Springer Berlin Heidelberg.

[2] Martin Gardner. *The Scientific American book of mathematical puzzles and diversions.* New York, Simon and Schuster, 1959.

[3] H. Jacob and H. Meyniel. About quasi-kernels in a digraph. *Discrete Mathematics*, 154(1):279–280, 1996.

[4] Stephen B. Maurer. The king chicken theorems. *Mathematics Magazine*, 53:67–80, 1980.

[5] Mary Wollstonecraft Shelley. *Frankenstein; or, The Modern Prometheus.* Lackington, Hughes, Harding, Mavor, & Jones, London, 1818.

[6] John von Neumann, Oskar Morgenstern, and Ariel Rubinstein. *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition).* Princeton University Press, 1944.

[7] Wikipedia contributors. Rooted graph — Wikipedia, the free encyclopedia, 2024. [Online; accessed 25-June-2024].

[8] Wikisource. Frankenstein, or the modern prometheus (first edition, 1818)/volume 3/chapter 7 — wikisource,, 2018. [Online; accessed 23-June-2024].