TEMat

Este trabajo colaboró con una conferencia plenaria durante el XVIII *Encuentro Nacional de Estudiantes de Matemáticas*, celebrado en Sevilla en julio de 2017.

ENEM '17

SEVILLA

Buscando triángulos en grafos muy grandes: un ejemplo de *property testing*

Alberto Espuny Díaz
University of Birmingham
axe673@bham.ac.uk

Resumen: El *property testing* hace referencia a un conjunto de técnicas algorítmicas que se han desarrollado para conseguir algoritmos decisionales que trabajen en tiempo sublineal en el tamaño de la entrada a cambio de perder precisión en la respuesta del algoritmo. En este artículo presentamos una breve discusión sobre el *property testing*, explicando el porqué de su utilidad y cómo valorar su eficiencia. En particular, nos centramos en algoritmos que comprueban propiedades en grafos, presentando los tres modelos más utilizados para comprobar estas propiedades y un ejemplo de cómo tratar una propiedad, la ausencia de triángulos, en cada uno de ellos.

Abstract: Property testing refers to a group of algorithmic techniques developed to achieve decision algorithms that work in sublinear time at the expense of some accuracy in the algorithms' output. In this paper, we present a brief discussion about property testing, explaining its usefulness and how to evaluate the efficiency of property testing algorithms. In particular, we consider algorithms that test graph properties. We present the three most common property testing models for graphs and, as an example, show how to test one property, triangle-freeness, in each of these models.

Palabras clave: algoritmos, grafos, *property testing*, libre de triángulos, regularidad, complejidad.

MSC2010: 05C85, 05D40, 68R10.

Recibido: 11 de septiembre de 2017.
Aceptado: 9 de septiembre de 2018.

Referencia: Espuny Díaz, Alberto. «Buscando triángulos en grafos muy grandes: un ejemplo de *property testing*». En: *TEMat*, 3 (2019), págs. 87-100. ISSN: 2530-9633. URL: https://temat.es/articulo/2019-p87.

Este trabajo se distribuye bajo una licencia Creative Commons Reconocimiento 4.0 Internacional
https://creativecommons.org/licenses/by/4.0/

1. Introducción

El desarrollo de los ordenadores y de los algoritmos que estos ejecutan ha permitido que a día de hoy podamos resolver en unos pocos segundos problemas que hace cien años habrían llevado meses de trabajo a un gran equipo de personas. Los problemas que se pueden resolver son muchos y muy variados, al igual que lo son las ideas subyacentes en los algoritmos. Sin embargo, y al contrario de una noción bastante extendida en la sociedad, no todos los problemas se pueden resolver rápido con nuestros ordenadores, y no es cuestión solo de conseguir ordenadores más potentes.

En general, podemos decir que un algoritmo es un proceso que, dada una entrada de datos (un conjunto de datos para el que queremos resolver un problema), sigue una serie de pasos hasta que resuelve el problema. En general, cuanto más grande sea el conjunto de entrada, mayor será el tiempo que tarda el algoritmo en resolver el problema, aunque sea solo porque necesita leer toda la entrada. Así, si el conjunto de entrada tiene tamaño n (en número de bits, aunque se acostumbra a utilizar el tamaño natural de la entrada), el tiempo que tarda en resolver el problema es una función de n, y esta función es a lo que llamamos la *complejidad temporal* del algoritmo. En general, nos interesa que los algoritmos resuelvan problemas para entradas de datos muy grandes (porque esas son las que nos cuesta mucho resolver a nosotros), de modo que nos interesa conocer cómo crece esta complejidad cuando n tiende a infinito.

Vamos a considerar un par de ejemplos. Supongamos que la entrada de datos es un grafo con n vértices; es muy razonable suponer que queremos conocer una propiedad del grafo de entrada. Por ejemplo, supongamos que queremos saber si el grafo es *bipartito*. En ese caso, es fácil demostrar que hay un algoritmo lineal que permite resolver el problema (lineal en el tamaño de la entrada, que en este caso no es n sino n más el número de aristas del grafo). Los algoritmos que tienen una complejidad lineal (y, en general, polinómica) se consideran «eficientes» (o «rápidos»).

El problema de saber si un grafo es bipartito es equivalente a saber si es 2-coloreable. En general, consideremos el problema de la *k-coloreabilidad* (es decir, queremos saber si los vértices del grafo se pueden colorear con k colores de manera que no haya dos vértices del mismo color unidos por una arista). Para cualquier valor de $k \geq 3$ se sabe que este problema es NP-completo [17], lo que quiere decir que los mejores algoritmos que conocemos para resolverlo tienen una complejidad *superpolinomial* (es decir, que crece más rápido que cualquier polinomio) en el tamaño de la entrada. De hecho, los algoritmos exactos que se conocen tienen complejidad exponencial. En general, se considera que los algoritmos que tienen una complejidad superpolinomial son «ineficientes». Los ejemplos de problemas que se comportan de este modo son muy variados, de manera que hay muchos problemas para los que un ordenador tardará mucho tiempo en dar una respuesta exacta si el conjunto de datos de la entrada es grande.

Consideremos ahora un algoritmo eficiente (cuya complejidad sea polinomial), pero supongamos que queremos estudiar una entrada de datos enorme. Por ejemplo, supongamos que queremos estudiar el grafo de internet, o de Facebook, o estudiar una propiedad en una base de datos enorme de una empresa. En estos casos, aunque el algoritmo sea eficiente, el simple hecho de leer toda la entrada de datos y procesarla ya supone un coste absurdo, además de que solo el almacenamiento de los datos ya supondría un problema (por el espacio de memoria que tienen los ordenadores).

Así pues, la pregunta natural es «¿qué podemos hacer en estos casos en los que queremos resolver un problema pero no tenemos suficientes recursos temporales?».

2. Complejidad y notación asintótica

El análisis de algoritmos es una tarea imprescindible para saber si su implementación es adecuada o si los recursos que necesitan para resolver un problema son excesivos. De este análisis se desprende la idea de que los algoritmos sean eficientes o no, y a partir de este estudio se han desarrollado muchos problemas en el área de la teoría de la computación.

En general, dado un algoritmo que resuelve un problema, el objetivo del análisis de algoritmos es determinar los recursos (temporales, espaciales o de cualquier otro tipo) que el algoritmo necesita para resolver el problema. La cantidad de recursos que necesita depende del tamaño de la entrada, pero distintas entradas con el mismo tamaño pueden necesitar distintos recursos. En ese caso, se toma una cota superior sobre

los recursos que necesita considerando el peor caso de entre todas las entradas con el mismo tamaño. Los recursos más estudiados habitualmente son el tiempo y el espacio, aunque en este artículo no nos centraremos en el espacio.

Definición 1. La **complejidad temporal** de un algoritmo para cada valor de n se define como el máximo número total de operaciones atómicas que realiza el ordenador sobre todas las entradas de tamaño n. ◂

Nótese que esta definición no tiene que tener una equivalencia exacta con el tiempo real que tarda el algoritmo. Este tiempo depende de muchos otros factores, como el ordenador empleado, la distribución de los datos en la memoria o el tipo de operaciones que el ordenador realiza. Sin embargo, por simplicidad, se suele trabajar en el *modelo de coste uniforme*, según el cual todas las operaciones que realiza el ordenador tienen el mismo coste constante (operaciones atómicas). Este modelo es bastante adecuado si se puede garantizar que el tiempo de todas las operaciones está acotado por alguna constante.

A la hora de estudiar la complejidad de algoritmos (y también el comportamiento de muchas estructuras combinatorias que dependen de un cierto parámetro de tamaño n) muchas veces es interesante saber cuál es su crecimiento asintótico en función del parámetro n cuando este tiende a infinito. Esto es especialmente útil cuando solo interesa saber cómo se comparan las complejidades de dos algoritmos diferentes. Para hablar de estos comportamientos asintóticos es útil emplear la *notación asintótica*.

Definición 2. Sean f y g dos funciones de n que toman solo valores no negativos.

- Decimos que $f(n) = \mathcal{O}(g(n))$ (y se lee que «f es **o grande** de g») si y solo si existen constantes $M \in \mathbb{R}$ y $n_0 \in \mathbb{N}$ tales que $f(n) \leq Mg(n)$ para todo $n \geq n_0$.
- Decimos que $f(n) = o(g(n))$ (y se lee que «f es **o pequeña** de g») si y solo si para toda constante $\delta > 0$ existe una constante $N \in \mathbb{N}$ tal que $f(n) \leq \delta g(n)$ para todo $n \geq N$. Equivalentemente, si g no se anula a partir de un cierto punto, podemos decir que $f(n) = o(g(n))$ si $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$.
- Decimos que $f(n) = \Omega(g(n))$ (y se lee que «f es **omega grande** de g») si y solo si existen constantes $M \in \mathbb{R}$ y $n_0 \in \mathbb{N}$ tales que $f(n) \geq Mg(n)$ para todo $n > n_0$. Equivalentemente, podemos decir que $f(n) = \Omega(g(n))$ si y solo si $g(n) = \mathcal{O}(f(n))$.
- Decimos que $f(n) = \Theta(g(n))$ (y se lee que «f es **theta** de g») si y solo si existen constantes $k_1, k_2 \in \mathbb{R}$ y $n_0 \in \mathbb{N}$ tales que $k_1 g(n) \leq f(n) \leq k_2 g(n)$ para todo $n \geq n_0$. Equivalentemente, decimos que $f(n) = \Theta(g(n))$ si y solo si $f(n) = \mathcal{O}(g(n))$ y $f(n) = \Omega(g(n))$. ◂

Nótese que aquí el uso de «=» es un abuso de notación. Se puede utilizar $\mathcal{O}(g(n))$ para denotar el conjunto de todas las funciones f que cumplen la definición 2 (y lo mismo se puede hacer para las otras notaciones presentadas). De este modo, el símbolo «=» indica que f pertenece a ese conjunto, no la igualdad.

Al trabajar con la complejidad (temporal, digamos) de un algoritmo es muy habitual utilizar este tipo de notación. Así, por ejemplo, se pueden programar algoritmos para determinar si un grafo $G = (V, E)$ es bipartito o no con complejidad $\mathcal{O}(|V| + |E|)$. En general, la notación o grande se utiliza para dar cotas superiores sobre la complejidad.

Es habitual hablar de la «complejidad» de un problema para referirnos a la complejidad de un algoritmo que resuelve el problema de manera óptima. Si sabemos que ningún algoritmo podrá resolver el problema con una complejidad menor que una determinada función, esta constituye una cota inferior, y podemos utilizar la notación omega grande para denotar esta cota.

3. *Property testing*

La idea básica del *property testing* surge como una respuesta natural a la pregunta final de la introducción: «¿Qué podemos hacer si queremos resolver un problema pero no tenemos suficientes recursos temporales?». Digamos que tenemos una cierta estructura de datos y queremos saber si cumple una cierta propiedad (por ejemplo, dada una función f como un conjunto de pares (x, y), donde $y = f(x)$, podemos querer saber si f es lineal o no). Nuestro objetivo es desarrollar algoritmos que nos permitan saber esto y cuya complejidad sea mejor que la de los algoritmos exactos. Como nuestros algoritmos exactos no son suficientemente

eficientes para nuestros objetivos, a cambio estamos dispuestos a sacrificar parte del conocimiento que queremos obtener; en particular, en lugar de saber si la propiedad se cumple o no, nos puede interesar saber si se cumple o está «lejos» de cumplirse, y nos podemos conformar con saber esto no de manera exacta, pero sí con una probabilidad suficientemente grande. Con esta premisa, nos gustaría desarrollar algoritmos *ultraeficientes*.

Para ser precisos, un algoritmo de *property testing* toma una decisión aproximada sobre si la estructura de datos presenta o está «lejos» de presentar la propiedad estudiada, donde «lejos» quiere decir que la entrada se debe modificar de un modo no negligible para que llegue a tener la propiedad. Para ello, al algoritmo se le proporciona un parámetro de *distancia*, ε, y el algoritmo deberá aceptar la entrada con probabilidad alta si presenta la propiedad, rechazarla con probabilidad alta si está ε-lejos de presentarla, y puede contestar cualquier cosa en el caso restante. Por probabilidad alta queremos decir probabilidad mayor que $2/3$, aunque esta es una convención, y se podría tomar cualquier probabilidad mayor que $1/2$. La definición de distancia se debe especificar en cada problema concreto.

Podemos observar que el solo hecho de leer toda la entrada ya supone un coste lineal para el algoritmo, de modo que no vamos a poder mejorar esto. En su lugar, lo que suponemos es que el algoritmo no lee la entrada, sino que los datos de entrada están almacenados en algún sitio y nuestro algoritmo tiene acceso a un *oráculo* que conoce estos datos. El algoritmo tiene permitido hacer *consultas* a este oráculo, de manera que podemos leer solo una parte pequeña de los datos, ahorrando así mucho tiempo al algoritmo. Nuestro objetivo general, ahora, es encontrar algoritmos con complejidad *sublineal*; en particular, el uso de esta teoría ha permitido desarrollar algoritmos que tienen una complejidad *independiente del tamaño de la entrada*, es decir, podríamos decir que tienen complejidad *constante*.

El *property testing* se comenzó a estudiar en los años 90, siendo utilizado implícitamente por Blum, Luby y Rubinfeld [7] y definido formalmente por primera vez por Rubinfeld y Sudan [25] en 1996. Más tarde, Goldreich, Goldwasser y Ron [12] extendieron la definición a un contexto más general y empezaron a estudiar propiedades de grafos. El estudio del *property testing* se ha hecho muy generalizado desde entonces y se han obtenido aplicaciones muy variadas, como algoritmos para trabajar con diferentes propiedades algebraicas de funciones, con funciones lógicas, propiedades geométricas, de lenguajes restringidos, de distribuciones, etc. Pero quizá el área en la que más énfasis se ha dado al *property testing* es en combinatoria y, en particular, en teoría de grafos. Nosotros nos vamos a concentrar en una de estas aplicaciones; para leer sobre otros muchos problemas, técnicas y aplicaciones se puede acudir al estudio de Ron [24], aunque también existen otros estudios menos enfocados a la parte combinatoria y numerosos artículos.

Dado un algoritmo cualquiera, el estudio de su complejidad es siempre necesario para saber si implementarlo vale la pena. En el caso del *property testing* vamos a considerar especialmente dos tipos de recursos: el tiempo y el número de consultas al oráculo. Podemos hablar así de complejidad de consultas.

Definición 3. La **complejidad de consultas** para cada valor de n de un algoritmo con acceso a un oráculo se define como el máximo número total de consultas que realiza el algoritmo sobre todas las entradas de tamaño n. ◂

En este artículo nos centramos en trabajar con grafos. Recordamos que un **grafo** se define como un par $G = (V, E)$ donde V es un conjunto de **vértices** y $E \subseteq V \times V$ es un conjunto de pares de vértices, que se denominan **aristas**. Por simplicidad, denotaremos una arista (u, v) como uv. Los vértices de un grafo se suelen representar mediante puntos, y las aristas, mediante líneas que unen los puntos. Dada una arista, decimos que es **incidente** a un vértice si uno de sus dos extremos es dicho vértice. El número de aristas incidentes a un vértice es su **grado**. Para un vértice v, decimos que un vértice u es **vecino** de v si $uv \in E$. Decimos que un grafo es **simple** si no hay aristas múltiples (dos o más aristas uniendo los mismo vértices) ni aristas que unan un vértice consigo mismo. Aquí nos centraremos solo en grafos simples.

Sea $G = (V, E)$ un grafo simple con $|V| = n$. A la hora de dar este grafo como una entrada para un algoritmo, los vértices se deben etiquetar, es decir, se les da un orden $v_1, v_2, \dots, v_n$. Las formas de dar las aristas pueden ser diferentes, y los algoritmos que se desarrollan dependen de esta estructura de datos.

Para trabajar con problemas de grafos en *property testing* se han definido varios modelos. Cada uno de ellos tiene sus particularidades y su utilidad, y es interesante en general estudiar problemas en los distintos modelos.

3.1. El modelo de grafos densos

Una de las estructuras de datos tradicionales para almacenar grafos es la que se denomina la **matriz de adyacencia**, una matriz cuadrada $A = (a_{i,j})_{i,j=1}^{n}$ de tamaño $n \times n$ en la que las filas y columnas se numeran con las etiquetas de los vértices, y la entrada $a_{i,j}$ toma el valor 1 si $v_i v_j \in E(G)$, y 0 en caso contrario.

Dada esta estructura de datos, las consultas que se le pueden hacer al oráculo son del tipo «¿hay una arista entre los vértices v_i y v_j del grafo?», y la respuesta del oráculo deberá ser «sí» o «no». A estas consultas las denominaremos **consultas de par de vértices**. La idea de distancia que emplearemos también tiene que ver con esta representación del grafo. Diremos que un grafo G está ε-lejos de tener una propiedad si hay que modificar por lo menos εn^2 entradas de la matriz de adyacencia para que G presente dicha propiedad, es decir, si hay que modificar $\varepsilon n^2/2$ aristas del grafo. Nótese que el número máximo de aristas que un grafo puede tener es $\binom{n}{2} \approx n^2/2$. Este modelo fue introducido por Goldreich, Goldwasser y Ron [12].

El motivo por el que este modelo se conoce como el de grafos densos tiene que ver con esta noción de distancia. Sea $\{G_n : n \in \mathbb{N}\}$ una secuencia de grafos tales que G_n tiene n vértices. Decimos que los grafos de la secuencia son **densos** (a partir de un índice n_0) si existe alguna constante δ tal que todos los grafos G_n con $n \geq n_0$ tienen al menos δn^2 aristas. Por comodidad, no hablaremos de secuencias de grafos, sino solo de grafos densos, pero siempre con la idea subyacente de que el tamaño de los grafos tiende a infinito. Es en este tipo de grafos en los que es más interesante estudiar propiedades en este modelo, ya que, por ejemplo, cualquier grafo no denso nunca estará ε-lejos de una propiedad que se pueda conseguir eliminando aristas, de modo que nuestros algoritmos no podrán determinar si la tiene o no.

3.2. El modelo de grado acotado

La otra estructura de datos tradicional para presentar los grafos se conoce como la **lista de incidencia**, y consiste en una lista de los vértices vecinos a v_i para cada $i \in \{1, \dots, n\}$. El orden en que aparecen estos vecinos puede ser arbitrario, e induce un doble etiquetado de las aristas del grafo. Si el grado de todos los vértices está uniformemente acotado por una cota d, esta estructura de datos presenta toda la información sobre las aristas del grafo de una forma bastante compacta.

Si tenemos esta estructura de datos, para cada vértice u las consultas que se le pueden hacer al oráculo son del tipo «¿cuál es el i-ésimo vecino de u?». La respuesta del oráculo deberá ser o bien un vértice v, que aparece en la i-ésima posición en la lista de incidencia de u, o bien un símbolo especial si el grado de u es menor que i. A estas consultas las denominaremos **consultas de vecindad**. La distancia que definimos también tiene que ver con la estructura de datos: diremos que un grafo está ε-lejos de tener una propiedad si hay que modificar más de εnd entradas de la lista de incidencia para que obtenga la propiedad. Nótese que el máximo número de aristas que puede tener el grafo, dada la cota sobre el grado, es $nd/2$.

Este modelo, introducido por Goldreich y Ron [13], es especialmente útil para grafos en los que el número de aristas es del orden de nd, es decir, aquellos en los que el grado medio del grafo es del mismo orden que el grado máximo. En particular, como veremos, esto es cierto si d es una constante.

3.3. El modelo de grafos generales

El modelo de grafos generales fue introducido por Kaufman, Krivelevich y Ron [18] después de que Parnas y Ron [22] propusieran separar el modelo de la estructura de datos utilizada e introdujesen un modelo intermedio. Así, sin conocer la estructura de datos empleada, el oráculo puede responder tanto a consultas de par de vértices como de vecindad. Además, a veces también se le permite responder a la consulta «¿cuál es el grado del vértice u?», aunque nosotros no utilizaremos este tipo de consultas.

Dado que la estructura de datos no se conoce, la distancia en este modelo no se mide respecto al máximo número de aristas que el grafo podría tener, sino respecto al número de aristas que tiene. Así, decimos que un grafo $G = (V, E)$ está ε-lejos de presentar una propiedad si hay que modificar al menos $\varepsilon|E|$ aristas para que la propiedad se dé.

El modelo de grafos generales es hasta ahora el menos estudiado de los tres. Presenta la ventaja de que se puede trabajar con todo tipo de grafos, pero el análisis de algoritmos en él puede llegar a ser muy

complejo. En cualquier caso, cabe insistir en que los modelos no son equivalentes. En este sentido, un mismo problema estudiado en modelos diferentes puede tener soluciones también diferentes. Veremos un ejemplo de esto más adelante.

4. El problema de la ausencia de triángulos

Sea $G = (V, E)$ un grafo. Sean $u, v, w \in V(G)$ tres vértices cualesquiera. Si $uv, uw, vw \in E(G)$, decimos que estas tres aristas forman un **triángulo**. Si G no tiene ningún triángulo, decimos que G está **libre de triángulos**. A la propiedad de que G esté libre de triángulos la llamamos **ausencia de triángulos**.

Supongamos que tenemos un grafo grande y queremos saber si está libre de triángulos. Un algoritmo exacto que comprueba si el grafo está libre de triángulos mediante una búsqueda en anchura tiene una complejidad lineal ($\mathcal{O}(|V| + |E|)$), de modo que es un problema que se resuelve de manera muy eficiente, pero ¿qué pasa si el tamaño del grafo es tal que un tiempo lineal no es suficiente, o que nuestro ordenador no tiene suficiente memoria para almacenar los datos? Vamos a intentar resolver este problema como un ejemplo de *property testing* en cada uno de los modelos.

4.1. Ausencia de triángulos en el modelo de grado acotado

El problema de la ausencia de triángulos en el modelo de grado acotado fue resuelto por Goldreich y Ron [13] en el mismo artículo en que introdujeron dicho modelo. La solución que dieron se basa en dar directamente un algoritmo bastante sencillo:

Algoritmo 1 (comprueba la ausencia de triángulos para grafos en el modelo de grado acotado).

```
1:  Entrada: tamaño del grafo n, grado máximo d y parámetro de distancia ε
2:  seleccionar uniforme e independientemente s = Θ(1/ε) vértices de G
3:  etiquetar los vértices seleccionados como v_1, v_2, ..., v_s
4:  para i = 1 hasta s hacer
5:      consultar todos los vecinos de v_i
6:      para cada vecino u de v_i hacer
7:          consultar todos los vecinos de u
8:          comprobar si algún vecino de u es vecino de v_i
9:          si sí, entonces
10:             rechazar
11:         fin si
12:     fin para
13: fin para
14: aceptar
```

Lo que hace este algoritmo es buscar triángulos de los que forme parte alguno de los vértices v_i seleccionados. Para cada $i \in \{1, \dots, s\}$, lo que hace el algoritmo es seleccionar todos los vecinos de v_i y, para cada uno de estos, comprobar si cierra un triángulo con alguno de los otros vecinos (línea 8). Si el algoritmo consigue encontrar un solo triángulo, entonces rechaza la entrada (es decir, el algoritmo afirma que el grafo está ε-lejos de la ausencia de triángulos); si no encuentra ningún triángulo, entonces acepta la entrada.

Teorema 4. *El algoritmo 1 es un algoritmo que comprueba la ausencia de triángulos en un grafo en el modelo de grado acotado con $\mathcal{O}\left(d^2/\varepsilon\right)$ consultas en tiempo $\mathcal{O}\left(d^3/\varepsilon\right)$.*

Demostración. Para demostrar este teorema hay que comprobar varias cosas: que el algoritmo es correcto (es decir, que dada una entrada cualquiera contesta de forma correcta con las probabilidades pertinentes) y que se cumplen las dos cotas de complejidad dadas.

En primer lugar, es muy fácil observar que si el grafo sobre el que se hacen consultas no tiene triángulos, el algoritmo nunca podrá encontrar ninguno y aceptará la entrada con probabilidad 1. Ahora concentrémonos en el bucle que empieza en la línea 6. En cada iteración el algoritmo realiza como mucho d consultas

para obtener los vecinos de v_i, y después d consultas para cada uno de ellos, lo que supone un total de como máximo $d^2 + d$ consultas. Dado que al bucle se entra $\Theta(1/\varepsilon)$ veces, está claro que el número total de consultas es $\mathcal{O}\left(d^2/\varepsilon\right)$. En cuando al tiempo que tarda el algoritmo en total, en el mismo bucle tenemos que comprobar si se cierran triángulos. Para esto, para cada uno de los vecinos u de v_i (de los que hay como máximo d) comparamos cada vecino de u con cada uno de los vecinos de v_i; esto supone un máximo de como mucho d^3 comparaciones. De este modo, la cota $\mathcal{O}\left(d^3/\varepsilon\right)$ sobre la complejidad temporal del algoritmo está clara.

Finalmente, queda comprobar que si el grafo que estudiamos está ε-lejos de la ausencia de triángulos, entonces el algoritmo 1 rechazará la entrada con probabilidad al menos 2/3. Por definición, si el grafo está ε-lejos de la ausencia de triángulos, entonces hay que realizar al menos εnd modificaciones sobre las aristas para convertirlo en un grafo libre de triángulos. Como añadir aristas nunca nos ayuda a eliminar triángulos, esto quiere decir que debemos eliminar al menos εnd aristas para eliminar todos los triángulos del grafo, lo cual significa que hay al menos εnd aristas del grafo que forman parte de triángulos. Ahora bien, como cada vértice tiene grado como máximo d, eso quiere decir que hay al menos εn vértices del grafo que forman parte de triángulos. Así, al elegir los s vértices se puede tomar una constante (implícita en la notación Θ) suficientemente grande como para que la probabilidad de que al menos uno de los s vértices seleccionados pertenezca a un triángulo sea al menos 2/3. Y, entonces, el algoritmo encontrará ese triángulo y rechazará la entrada. ∎

En particular, podemos calcular el valor de esa constante implícita. Como la muestra se toma de manera independiente, tenemos que la probabilidad de que cada uno de los vértices elegidos pertenezca a un triángulo es mayor o igual que ε, de modo que la probabilidad de que no pertenezca a ningún triángulo es menor o igual que $1 - \varepsilon$. Después de tomar s vértices, la probabilidad de que ninguno pertenezca a un triángulo es menor o igual que $(1 - \varepsilon)^s$, así que la probabilidad de que alguno de los vértices pertenezca a un triángulo es

$$\Pr(\text{algún vértice de la muestra pertenezca a un triángulo}) \geq 1 - (1 - \varepsilon)^s \geq 1 - \mathrm{e}^{-\varepsilon s}.$$

Como queremos que esta probabilidad sea de al menos 2/3, tenemos que

$$1 - \mathrm{e}^{-\varepsilon s} \geq \frac{2}{3} \iff e^{-\varepsilon s} \leq \frac{1}{3} \iff \varepsilon s \geq \log 3 \iff s \geq \frac{\log 3}{\varepsilon},$$

de modo que tomar $s = \log 3/\varepsilon$ es suficiente para que el algoritmo funcione.

Es interesante remarcar el hecho de que la cota que hemos obtenido sobre el número de consultas *no depende de n*, como tampoco lo hace la complejidad temporal del algoritmo. Así, podemos ir aumentando el tamaño del grafo tanto como queramos, y nuestro algoritmo siempre tardará el mismo tiempo en decidir.

Por otra parte, si queremos afinar más nuestra decisión, nos interesará variar el parámetro ε. Si el parámetro es muy pequeño podremos distinguir grafos que están más cerca según nuestra distancia, con lo cual podríamos decir que la respuesta del algoritmo es mejor en este sentido (distingue grafos que no distinguiría para valores de ε más grandes). Como queremos variar este parámetro, también es bueno saber cómo varía la complejidad del algoritmo respecto a él. Y lo que tenemos en este caso, en virtud del teorema anterior, es que la complejidad (tanto temporal como del número de consultas) es lineal en el inverso de ε, es decir, el algoritmo es bastante eficiente también en este sentido.

4.2. Ausencia de triángulos en el modelo de grafos densos: regularidad

Vamos a considerar ahora el mismo problema, pero trabajando en el modelo de grafos densos. Así, podemos suponer que los grafos que queremos estudiar tienen un número cuadrático de aristas. Para poder resolver bien el problema en este contexto vamos a necesitar una serie de definiciones y un resultado central en la combinatoria extremal de los últimos cincuenta años.

Dado un grafo $G = (V, E)$, sean A y B dos conjuntos no vacíos de vértices tales que $A \cap B = \varnothing$. Supongamos que $E(A, B)$ representa el conjunto de aristas cuyos vértices están uno en A y otro en B.

Definición 5. La **densidad** del par A, B se define como

$$d(A,B) = \frac{|E(A,B)|}{|A||B|}.$$

Definición 6. Se dice que el par A, B es γ**-regular**, para algún $\gamma \in [0,1]$, si para todo par de conjuntos $A' \subseteq A$, $B' \subseteq B$ con $|A'| \geq \gamma|A|$ y $|B'| \geq \gamma|B|$ se cumple que $|d(A',B') - d(A,B)| < \gamma$.

Una de las herramientas centrales en combinatoria extremal hoy en día, aunque también tiene muchas aplicaciones en otros campos, es el siguiente lema, debido a Szemerédi [26].

Lema 7 (lema de regularidad de Szemerédi). *Para cualquier $\ell_0 \in \mathbb{N}$ y cualquier $\gamma \in (0,1]$ existe un entero $M = M(\ell_0, \gamma)$ tal que todo grafo $G = (V,E)$ con $n \geq M$ vértices tiene una partición $\mathcal{A} = \{V_1, \dots, V_\ell\}$ de V con $||V_i| - |V_j|| \leq 1$ para todo $i, j \in \{1, \dots, \ell\}$, donde $\ell_0 \leq \ell \leq M$, tal que todos los pares (V_i, V_j) excepto como mucho $\gamma\binom{\ell}{2}$ son γ-regulares.*

La demostración de este resultado no es complicada, pero nos interesa más ver una de sus aplicaciones. Vamos a presentar un algoritmo muy simple que comprueba si un grafo cualquiera está libre de triángulos o está ε-lejos de estarlo, y vamos a utilizar el lema 7 para analizarlo.

Algoritmo 2 (comprueba la ausencia de triángulos para grafos en el modelo de grafos densos).

1: **Entrada:** tamaño del grafo n y parámetro de distancia ε
2: $M \leftarrow M(8/\varepsilon, \varepsilon/8)$
3: seleccionar uniforme e independientemente $s = \Theta\left(M^2/\varepsilon^3\right)$ vértices de G
4: consultar todos los pares de vértices
5: **si** encuentra un triángulo, **entonces**
6: rechazar
7: **en caso contrario**
8: aceptar
9: **fin si**

Teorema 8. *El algoritmo 2 es un algoritmo que comprueba la ausencia de triángulos en un grafo en el modelo de grafos densos con complejidad temporal y de consultas $\mathcal{O}\left(M^4/\varepsilon^6\right)$.*

Demostración. En el caso de que el grafo de la entrada no tenga ningún triángulo, está claro que el algoritmo 2 no podrá encontrar ninguno, así que dirá que el grafo está libre de triángulos con probabilidad 1. De este modo, debemos comprobar que el algoritmo es correcto cuando el grafo $G = (V,E)$ de la entrada está ε-lejos de la ausencia de triángulos.

Por el lema 7, tomando $\ell_0 = 8/\varepsilon$ y $\gamma = \varepsilon/8$, sabemos que existe una partición $\mathcal{A} = \{V_1, \dots, V_\ell\}$ de los vértices de G en ℓ partes, $\ell_0 \leq \ell \leq M = M(\varepsilon)$, con las propiedades dadas por el lema. Cada una de las partes tendrá tamaño $\lfloor n/\ell \rfloor$ o $\lceil n/\ell \rceil$; como trabajamos con problemas asintóticos, tenemos que n tiende a infinito y podemos despreciar los errores de redondeo, y trabajar solo con n/ℓ. Alternativamente, también se puede pensar que tomamos n múltiplo de ℓ para este desarrollo.

Si nos fijamos en una cualquiera de las partes de la partición, el número máximo de aristas que puede haber en su interior es $\binom{n/\ell}{2} \leq \frac{1}{2}(n/\ell)^2$, lo que supone un total de como máximo $\frac{\ell}{2}(n/\ell)^2 = \frac{1}{2\ell}n^2 \leq \frac{1}{2\ell_0}n^2 = \frac{\varepsilon}{16}n^2$ aristas. Si definimos un grafo G_1 como el grafo que se obtiene de G después de eliminar todas estas aristas, tenemos que G_1 está al menos $(\frac{15}{16}\varepsilon)$-lejos de estar libre de triángulos.

Consideremos ahora las parejas de la partición de G que no son regulares. Por el lema 7, hay como máximo $\frac{\varepsilon}{8}\binom{\ell}{2} \leq \frac{\varepsilon}{16}\ell^2$ parejas de estas, y cada pareja puede formar un total de como máximo $(n/\ell)^2$ aristas, de modo que en total hay como máximo $\frac{\varepsilon}{16}\ell^2(n/\ell)^2 = \frac{\varepsilon}{16}n^2$ aristas entre las parejas no regulares. Si definimos un grafo G_2 como el que se obtiene de G_1 después de eliminar todas las aristas que hay entre las parejas no regulares, tenemos que G_2 está al menos $(\frac{7}{8}\varepsilon)$-lejos de la ausencia de triángulos.

Consideremos, finalmente, las parejas (V_i, V_j) de la partición con $d(V_i, V_j) < \varepsilon/2$, es decir, aquellas en las que $|E(V_i, V_j)| < \frac{\varepsilon}{2}(n/\ell)^2$. El número de parejas con esta propiedad es como máximo $\binom{\ell}{2} \leq \ell^2/2$ (en el caso de que todas tengan la propiedad), de modo que el número total de aristas que hay entre todas estas parejas

de la partición es de como máximo $\frac{\varepsilon}{4}n^2$. Si definimos un grafo G_3 como el que se obtiene de G_2 después de eliminar todas estas aristas, tenemos que G_3 está al menos $(\frac{5}{8}\varepsilon)$-lejos de la ausencia de triángulos. En particular, como nuestro grafo aún tiene triángulos, tienen que existir tres índices i, j y k tales que $d(V_i, V_j), d(V_j, V_k), d(V_i, V_k) \geq \varepsilon/2$. Todo este proceso de eliminación de aristas se muestra en la figura 1.

Vamos a demostrar ahora que el hecho de que exista esta terna (V_i, V_j, V_k) tal que todas las parejas son $\frac{\varepsilon}{8}$-regulares y tienen densidad al menos $\varepsilon/2$ implica que en el grafo va a haber muchos triángulos, y que una muestra suficientemente grande conseguirá capturar alguno de estos. Sea $v \in V_i$ un vértice cualquiera. Sea $\Gamma_j(v)$ el conjunto de vecinos de v en V_j, y sea $\Gamma_k(v)$ el conjunto de vecinos de v en V_k. Vamos a decir que v es *útil* si $|\Gamma_j(v)| \geq \frac{\varepsilon}{4}\frac{n}{\ell}$ y $|\Gamma_k(v)| \geq \frac{\varepsilon}{4}\frac{n}{\ell}$. Como (V_j, V_k) es $\frac{\varepsilon}{8}$-regular, si v es útil entonces

$$|E\left(\Gamma_j(v), \Gamma_k(v)\right)| \geq \left(d\left(V_j, V_k\right) - \frac{\varepsilon}{8}\right)\left(\frac{\varepsilon}{4}\right)^2\left(\frac{n}{\ell}\right)^2 \geq \frac{3\varepsilon^3}{256\ell^2}n^2,$$

ya que $d\left(V_j, V_k\right) \geq \varepsilon/2$ en G_3. Teniendo en cuenta que $\ell \leq M$, esto quiere decir que si tenemos un vértice útil $v \in V_i$ y tomamos una muestra adicional de $\mathcal{O}\left(M^2/\varepsilon^3\right)$ pares de vértices, con una constante implícita suficientemente grande, entonces encontraremos uno de estos triángulos con probabilidad al menos 2/3. Y para esto basta, claramente, con tomar una muestra de $\mathcal{O}\left(M^2/\varepsilon^3\right)$ vértices y considerar todas las posibles parejas.

Finalmente, queda demostrar que el número de vértices útiles es suficientemente grande, de modo que una muestra también suficientemente grande podrá encontrar uno de ellos con probabilidad al menos 2/3. Para ello, consideremos un vértice $v \in V_i$ que no es útil. Diremos que v es *inútil en* j si $|\Gamma_j(v)| < \frac{\varepsilon}{4}\frac{n}{\ell}$, y que es *inútil en* k si $|\Gamma_k(v)| < \frac{\varepsilon}{4}\frac{n}{\ell}$. Podemos suponer sin pérdida de generalidad que el número de vértices inútiles en j es mayor o igual que el de vértices inútiles en k. Supongamos que al menos la mitad de los vértices no son útiles. Entonces, al menos la cuarta parte son inútiles en j. Sea V_i' el conjunto de dichos vértices, de modo que $|V_i'| > \gamma|V_i|$ ya que $\gamma = \frac{\varepsilon}{8} \leq \frac{1}{8}$, y sea $V_j' = V_j$. Entonces,

$$d\left(V_i', V_j'\right) \leq \frac{|V_i'|\frac{\varepsilon}{4}\frac{n}{\ell}}{|V_i'||V_j'|} = \frac{\varepsilon}{4} < \frac{3}{8}\varepsilon \leq d\left(V_i, V_j\right) - \gamma,$$

lo que contradice la regularidad del par (V_i, V_j). Por lo tanto, el número de vértices útiles en V_i es de al menos $\frac{n}{2\ell}$, y una muestra independiente de $\mathcal{O}(M)$ vértices contendrá uno de estos con probabilidad al menos 2/3.

Así, se puede tomar una muestra de $\mathcal{O}(M) + \mathcal{O}\left(M^2/\varepsilon^3\right) = \mathcal{O}\left(M^2/\varepsilon^3\right)$ vértices, que contendrá al menos un vértice útil y un par de vértices que formen un triángulo con el anterior con probabilidad al menos 2/3. Al consultar todos los pares de vértices, el algoritmo descubrirá el triángulo y rechazará la entrada, lo que demuestra que el algoritmo es correcto. El número de consultas es $\mathcal{O}\left(M^4/\varepsilon^6\right)$, ya que el número de posibles aristas es cuadrático en el número de vértices seleccionados. La complejidad temporal es la misma, ya que se puede buscar un triángulo en el grafo inducido por la muestra con una búsqueda en anchura, que tiene un coste lineal en el tamaño del grafo que se estudia. ∎

Como se desprende de la demostración, el teorema 8 establece que la complejidad, tanto temporal como de consultas, del algoritmo 2 es independiente del tamaño de la entrada, igual que en el caso del modelo de grado acotado. Sin embargo, el comportamiento con respecto a ε es mucho menos eficiente que en el caso anterior. Aunque el exponente de la cota sobre el número de consultas que da el teorema 8 se puede mejorar con un análisis más cuidadoso, la cota sobre M que da el lema de regularidad de Szemerédi es una torre de altura polinomial en el inverso de ε.

Definición 9. Denotamos por $T(n)$ la función **torre de altura** n, que se define de manera recursiva como $T(1) = 2$ y $T(n+1) = 2^{T(n)}$ para todo $n \geq 1$. ◂

La mejor cota superior sobre el valor de M que se obtiene del lema 7 es de la forma $T\left(\mathcal{O}(1/\varepsilon^2)\right)$, un número enorme que hace que este algoritmo no sea realmente aplicable para resolver este problema, ya que ningún ordenador tiene capacidad de memoria suficiente (aunque para grafos suficientemente grandes, sigue siendo mejor esta cota que cualquier otra que sea una función que tienda a infinito con n). Siendo esto una cota superior, nos podríamos plantear si no se puede mejorar utilizando otras técnicas. Sin embargo,

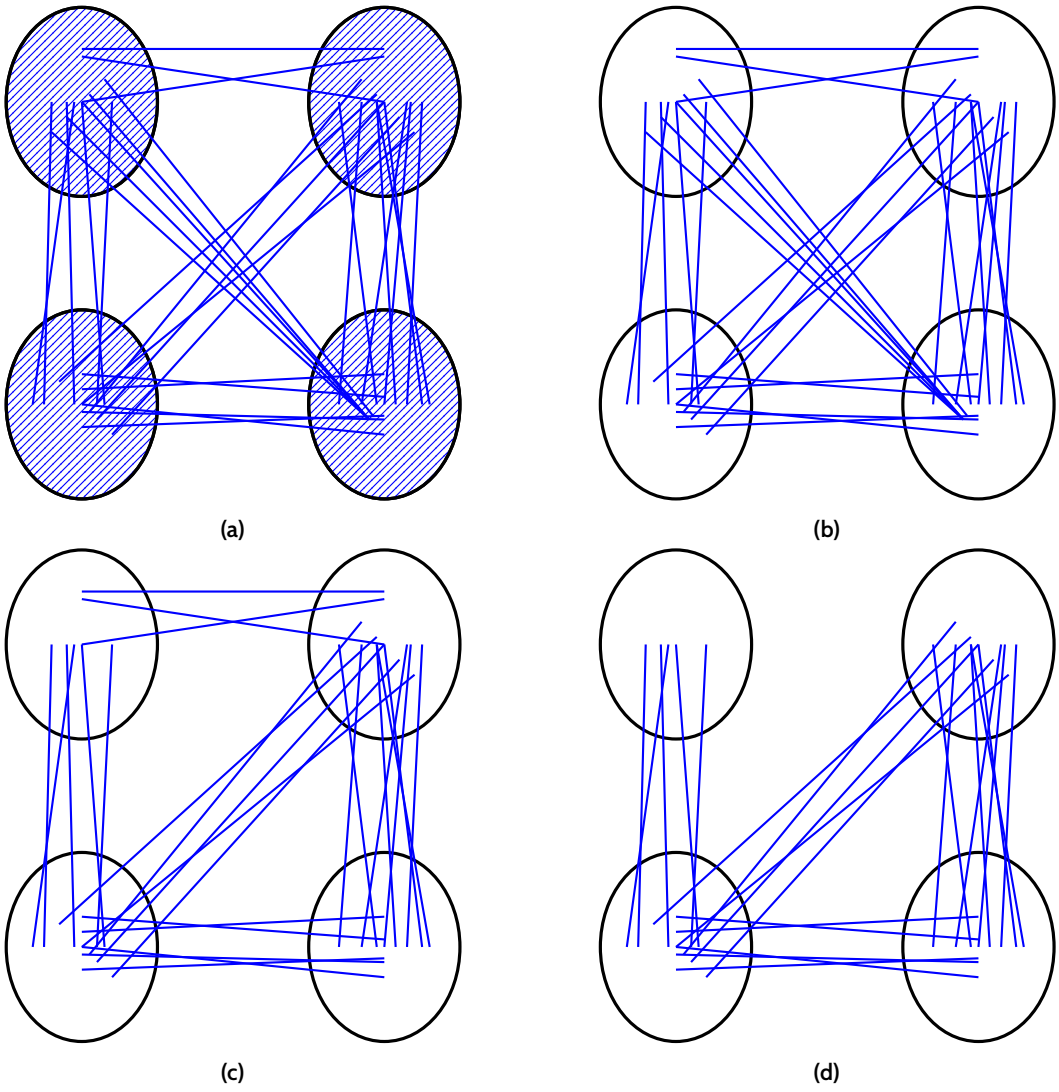

Figura 1: Representación del proceso de eliminación de aristas empleado en la demostración del teorema 8. La partición representada tiene cuatro conjuntos por simplicidad, pero el número de conjuntos dado por el lema 7 es mucho mayor. La figura 1a representa el grafo original G. La figura 1b representa el grafo G_1 obtenido tras eliminar las aristas dentro de cada conjunto de la partición. El grafo G_2, obtenido eliminando las aristas de pares no regulares, se representa en la figura 1c. La figura 1d representa el grafo G_3 final, obtenido tras eliminar también las aristas de parejas cuya densidad sea muy baja.

Alon [1] demostró que la complejidad de consultas de este problema tiene que ser superpolinómica, es decir, demostró que existen grafos que están ε-lejos de la ausencia de triángulos pero para los cuales no hay ningún algoritmo que trabaje con un número polinómico en $1/\varepsilon$ de consultas que pueda distinguirlos de grafos libres de triángulos. Este resultado fue completado más tarde por Alon y Shapira [5]. A pesar de la cota superpolinómica de Alon, sigue habiendo un salto muy grande entre las cotas inferior y superior conocidas. Cerrar este hueco es un problema abierto interesante.

El lema de regularidad de Szemerédi (lema 7) tiene muchas más aplicaciones en la teoría del *property testing* en el modelo de grafos densos. Se utilizó repetidamente para obtener resultados para algoritmos que comprueban diferentes propiedades en grafos, especialmente en la caracterización de propiedades de primer orden [2]. Por ejemplo, se demostró que la k-coloreabilidad es una propiedad que se puede comprobar con un número de consultas independiente del tamaño del grafo, a pesar de que este problema sea NP-completo en su versión exacta. Finalmente, Alon, Fischer, Newman y Shapira [3] demostraron un resultado muy general, estableciendo que una propiedad de grafos se puede comprobar en tiempo independiente del tamaño de la entrada en el modelo de grafos densos si y solo si comprobar dicha propiedad se puede reducir a comprobar si el grafo satisface una determinada partición de Szemerédi.

4.3. Ausencia de triángulos en el modelo de grafos generales: cotas

El problema de la ausencia de triángulos en el modelo de grafos generales fue estudiado por Alon, Kaufman, Krivelevich y Ron [4]. Lo que demostraron en este contexto es que no se puede resolver el problema con un número de consultas independiente de n. Vamos a ver aquí un resultado parcial.

Para poder trabajar con una distancia que depende del número de aristas del grafo, debemos tener la capacidad de trabajar con cada número posible de aristas o, al menos, con cada crecimiento asintótico de este número. Así, queremos encontrar cotas sobre la complejidad de los algoritmos que comprueban si el grafo de entrada está libre de triángulos o está ε-lejos de la ausencia de triángulos para cada crecimiento asintótico del número de aristas. Para hacer esto, vamos a trabajar con el grado medio del grafo y a establecer cotas para cada posible valor. El grado medio de un grafo $G = (V, E)$ con $|V| = n$ se define como $d = 2|E|/n$. Si consideramos valores de d como funciones de n, cada posible valor de d da lugar también a un único crecimiento asintótico de $|E|$ al tender n a infinito. Nótese que d es como máximo lineal en n, en cuyo caso estamos trabajando con grafos densos.

Teorema 10. *Cualquier algoritmo que compruebe la ausencia de triángulos en un grafo en el modelo de grafos generales debe realizar al menos $\Omega(\sqrt{n/d})$ consultas.*

Demostración. Para comprobar esto, basta con demostrar que existen dos familias de grafos $\mathcal{G}_1$ y $\mathcal{G}_2$ tales que todos los grafos de ambas tienen el mismo grado medio d y el mismo número de vértices, todos los grafos de $\mathcal{G}_1$ están libres de triángulos, todos los grafos de $\mathcal{G}_2$ están ε-lejos de la ausencia de triángulos, y cualquier algoritmo que realice $o(\sqrt{n/d})$ consultas sobre un grafo de cualquiera de las dos familias no será capaz de distinguir a cuál de las dos pertenece con probabilidad suficientemente alta.

Las dos familias que definimos son las siguientes. Cada familia se define con un único grafo sobre n vértices, y considerando todos los posibles etiquetados de los vértices. El grafo que define $\mathcal{G}_1$ consiste en un grafo bipartito completo sobre dos conjuntos de tamaño $\sqrt{nd/2}$, y el resto de vértices están aislados (es decir, no tienen ninguna arista incidente a ellos). El grafo que define $\mathcal{G}_2$ consiste en un grafo completo sobre $\sqrt{nd}$ vértices, y el resto de vértices están de nuevo aislados. Es muy fácil comprobar que el grado medio de cualquier grafo en $\mathcal{G}_1$ es exactamente d, mientras que el grado medio en $\mathcal{G}_2$ es $d(1 - \Theta(\sqrt{1/nd}))$, que asintóticamente crece igual que d, de modo que estas dos familias tienen un número de aristas comparable conforme n tiende a infinito. Además, es evidente que cualquier grafo en $\mathcal{G}_1$ está libre de triángulos, ya que es bipartito.

Hay que comprobar ahora que cualquier grafo en $\mathcal{G}_2$ está ε-lejos de la ausencia de triángulos, para algún $\varepsilon > 0$. Pero esto es una consecuencia del teorema de Mantel [19], que establece que el máximo número de aristas en un grafo libre de triángulos sobre k vértices es $\lfloor k^2/4 \rfloor$. Así, en nuestro caso, tenemos que eliminar casi $nd/4$ aristas, lo que supone que cualquier grafo en $\mathcal{G}_2$ está ε-lejos de la ausencia de triángulos para cualquier $\varepsilon < 1/4$.

Ahora, supongamos que un algoritmo trata de encontrar un triángulo en un grafo seleccionado uniformemente entre las dos familias con solo $o(\sqrt{n/d})$ consultas. Para ser capaz de distinguirlos necesitará encontrar uno de los vértices que tienen grado positivo, ya que en caso contrario solo habrá encontrado vértices aislados. Pero la probabilidad de encontrar uno cualquiera de estos vértices es $\Theta(\sqrt{d/n})$, de modo que al realizar todas las consultas la probabilidad de encontrar uno, por la desigualdad de Boole, es $o(\sqrt{n/d}\sqrt{d/n}) = o(1)$, es decir, tiende a cero conforme n tiende a infinito. Así pues, se deben realizar al menos $\Omega(\sqrt{n/d})$ consultas. ∎

La cota que da el teorema 10 es mejor para grafos poco densos, y va empeorando conforme d aumenta. En particular, si d es constante se observa una gran diferencia entre este resultado y el del teorema 4: en este caso tenemos una cota de $\Omega(\sqrt{n})$ sobre el número de consultas, mientras que en el modelo de grado acotado se podía resolver el problema con un número de consultas independiente de n.

Conforme d crece, la cota del teorema 10 es más débil. Sin embargo, Alon, Kaufman, Krivelevich y Ron [4] dan otras cotas para distintos valores de d que mejoran esta, utilizando técnicas más avanzadas que las presentadas aquí. En general, consiguen demostrar que para cualquier valor de $d = \mathcal{O}(n^{1-\nu(n)})$, con $\nu(n) = \frac{\log\log\log n+4}{\log\log n} = o(1)$, cualquier algoritmo que compruebe la ausencia de triángulos deberá realizar $\Omega(n^{1/3})$ consultas.

Por otra parte, en el mismo artículo también se dan cotas superiores a la complejidad, es decir, se demuestra que existen algoritmos que resuelven el problema ejecutando un determinado número de consultas. La cota general que se obtiene es de $\mathcal{O}(n^{6/7}\text{poly}\log(n))$. Así, aunque en el modelo de grafos generales no se pueden conseguir algoritmos que resuelvan el problema con un número de consultas independiente de n, sí se puede conseguir que lo hagan en tiempo sublineal, lo cual ya es una mejora considerable con respecto a los algoritmos exactos. Cerrar el espacio entre la cota superior y la inferior para determinar la complejidad real del problema es aún un problema abierto.

5. Conclusiones y otros problemas

Como se ha podido ver a lo largo del texto, el *property testing* permite desarrollar algoritmos para comprobar propiedades de estructuras de datos cuya complejidad es inferior a la de algoritmos exactos a cambio de sacrificar exactitud en la respuesta. Este sacrificio tiene una componente probabilística (la probabilidad de que el algoritmo dé una respuesta errónea no es nula, en la mayoría de casos) y determinista (la respuesta no se da sobre la propiedad, sino con respecto a una medida de distancia respecto a la propiedad). El desarrollo de estos algoritmos, además de un interés puramente teórico, también tiene muchas aplicaciones para resolver problemas en situaciones en las que los algoritmos tradicionales son demasiado lentos.

Hemos podido ver tres ejemplos de cómo tratar algoritmos en diferentes modelos para una determinada propiedad, y cómo el comportamiento de los algoritmos en cada modelo puede ser muy diferente. En general, las propiedades que se pueden estudiar son muy variadas. En el modelo de grafos densos existen resultados muy generales que engloban muchas de ellas, como ocurre con el caso del resultado de Alon, Fischer, Newman y Shapira [3]. También se puede considerar *property testing* aplicado a hipergrafos, y en este contexto destaca el resultado de Joos, Kim, Kühn y Osthus [16], que generaliza la clasificación de propiedades que se pueden estudiar con un número constante de consultas. En el caso del modelo de grado acotado, uno de los resultados más destacables se debe a Benjamini, Schramm y Shapira [6], quienes determinaron que toda propiedad cerrada por menores se puede estudiar con un número constante de consultas. Destacan también los resultados de Newman y Sohler [21]. El modelo de grafos generales es, sin duda, uno de los menos estudiados, y al trabajo de Alon, Kaufman, Krivelevich y Ron [4] se le pueden añadir resultados como el de Kaufman, Krivelevich y Ron [18], que estudiaron la propiedad de ser bipartito, o el de Espuny Díaz, Joos, Kühn y Osthus [8], que extendieron algunos resultados de Alon, Kaufman, Krivelevich y Ron [4] de la ausencia de triángulos a la ausencia de cualquier subgrafo fijo y también al caso de hipergrafos. En general, esta área aún tiene muchos problemas abiertos. Muchos de los problemas abiertos tienen que ver con determinar con exactitud la complejidad de los problemas decisionales en *property testing*.

Además de todo lo que se ha discutido hasta ahora, el *property testing* también se emplea para estudiar propiedades que no tienen nada que ver con grafos (se pueden ver muchos ejemplos en el estudio de

Ron [24]). Además de esto, existen algunas extensiones naturales del *property testing*. Una de ellas tiene que ver con la distribución subyacente a la hora de definir la distancia; en particular, si esta distribución es desconocida se habla de *distribution-free testing* [11, 12, 14, 15]. Otra de las extensiones es la que se conoce como *tolerant testing*, en la que el algoritmo recibe dos parámetros de distancia, ε_1 y ε_2, y debe distinguir entre el caso de que la entrada de datos esté ε_1-cerca de la propiedad o ε_2-lejos [9, 23]. Relacionado con lo anterior, también se pueden estudiar algoritmos que traten de aproximar la distancia de la entrada a una cierta propiedad [10, 20, 23]. En general, estudiar la relación entre los algoritmos de *property testing* y los algoritmos de aproximación clásicos (se sabe que son dos tipos de problemas completamente distintos, como se menciona en el estudio de Ron [24]) también puede ser un problema interesante.

Referencias

[1] Alon, Noga. «Testing subgraphs in large graphs». En: *Random Structures & Algorithms* 21.3-4 (2002). Random structures and algorithms (Poznan, 2001), págs. 359-370. issn: 1042-9832. https://doi.org/10.1002/rsa.10056.

[2] Alon, Noga; Fischer, Eldar; Krivelevich, Michael, y Szegedy, Mario. «Efficient testing of large graphs». En: *Combinatorica. An International Journal on Combinatorics and the Theory of Computing* 20.4 (2000), págs. 451-476. issn: 0209-9683. https://doi.org/10.1007/s004930070001.

[3] Alon, Noga; Fischer, Eldar; Newman, Ilan, y Shapira, Asaf. «A combinatorial characterization of the testable graph properties: it's all about regularity». En: *SIAM Journal on Computing* 39.1 (2009), págs. 143-167. issn: 0097-5397. https://doi.org/10.1137/060667177.

[4] Alon, Noga; Kaufman, Tali; Krivelevich, Michael, y Ron, Dana. «Testing triangle-freeness in general graphs». En: *SIAM Journal on Discrete Mathematics* 22.2 (2008), págs. 786-819. issn: 0895-4801. https://doi.org/10.1137/07067917X.

[5] Alon, Noga y Shapira, Asaf. «Testing subgraphs in directed graphs». En: *Journal of Computer and System Sciences* 69.3 (2004), págs. 353-382. issn: 0022-0000. https://doi.org/10.1016/j.jcss.2004.04.008.

[6] Benjamini, Itai; Schramm, Oded, y Shapira, Asaf. «Every minor-closed property of sparse graphs is testable». En: *Advances in Mathematics* 223.6 (2010), págs. 2200-2218. https://doi.org/10.1016/j.aim.2009.10.018.

[7] Blum, Manuel; Luby, Michael, y Rubinfeld, Ronitt. «Self-testing/correcting with applications to numerical problems». En: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (Baltimore, MD, 1990)*. Vol. 47. 3. 1993, págs. 549-595. https://doi.org/10.1016/0022-0000(93)90044-W.

[8] Espuny Díaz, Alberto; Joos, Felix; Kühn, Daniela, y Osthus, Deryk. «Edge correlations in random regular hypergraphs and applications to subgraph testing». En: *ArXiv e-prints* (mar. de 2018). arXiv: 1803.09223 [math.CO].

[9] Fischer, Eldar y Fortnow, Lance. «Tolerant versus intolerant testing for Boolean properties». En: *Theory of Computing. An Open Access Journal* 2 (2006), págs. 173-183. issn: 1557-2862. https://doi.org/10.4086/toc.2006.v002a009.

[10] Fischer, Eldar y Newman, Ilan. «Testing versus estimation of graph properties». En: *SIAM Journal on Computing* 37.2 (2007), págs. 482-501. issn: 0097-5397. https://doi.org/10.1137/060652324.

[11] Glasner, Dana y Servedio, Rocco A. «Distribution-free testing lower bounds for basic Boolean functions». En: *Theory of Computing. An Open Access Journal* 5 (2009), págs. 191-218. issn: 1557-2862. https://doi.org/10.4086/toc.2009.v005a010.

[12] Goldreich, Oded; Goldwasser, Shafi, y Ron, Dana. «Property testing and its connection to learning and approximation». En: *Journal of the ACM* 45.4 (1998), págs. 653-750. issn: 0004-5411. https://doi.org/10.1145/285055.285060.

[13] Goldreich, Oded y Ron, Dana. «Property testing in bounded degree graphs». En: *Algorithmica. An International Journal in Computer Science* 32.2 (2002), págs. 302-343. issn: 0178-4617. https://doi.org/10.1007/s00453-001-0078-7.

[14] Halevy, Shirley y Kushilevitz, Eyal. «Distribution-free property-testing». En: *SIAM Journal on Computing* 37.4 (2007), págs. 1107-1138. issn: 0097-5397. https://doi.org/10.1137/050645804.

[15] Halevy, Shirley y Kushilevitz, Eyal. «Distribution-free connectivity testing for sparse graphs». En: *Algorithmica. An International Journal in Computer Science* 51.1 (2008), págs. 24-48. issn: 0178-4617. https://doi.org/10.1007/s00453-007-9054-1.

[16] Joos, Felix; Kim, Jaehoon; Kühn, Daniela, y Osthus, Deryk. «A characterization of testable hypergraph properties». En: *ArXiv e-prints* (jul. de 2017). arXiv: 1707.03303 [math.CO].

[17] Karp, Richard M. «Reducibility among combinatorial problems». En: *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*. Springer US, 1972, págs. 85-103. https://doi.org/10.1007/978-1-4684-2001-2_9.

[18] Kaufman, Tali; Krivelevich, Michael, y Ron, Dana. «Tight bounds for testing bipartiteness in general graphs». En: *SIAM Journal on Computing* 33.6 (2004), págs. 1441-1483. issn: 0097-5397. https://doi.org/10.1137/S0097539703436424.

[19] Mantel, W. «Problem 28 (solution by H. Gouwentak, W. Mantel, J. Teixeira de Mattes, F. Schuh and W.A. Wythoff». En: *Wiskundige Opgaven* 10 (1907), págs. 60-61.

[20] Marko, Sharon y Ron, Dana. «Distance approximation in bounded-degree and general sparse graphs». En: *Approximation, randomization and combinatorial optimization.* Vol. 4110. Lecture Notes in Comput. Sci. Springer, Berlin, 2006, págs. 475-486. https://doi.org/10.1007/11830924_43.

[21] Newman, Ilan y Sohler, Christian. «Every property of hyperfinite graphs is testable». En: *SIAM Journal on Computing* 42 (2013), págs. 1095-1112. issn: 0097-5397. https://doi.org/10.1137/120890946.

[22] Parnas, Michal y Ron, Dana. «Testing the diameter of graphs». En: *Random Structures & Algorithms* 20.2 (2002), págs. 165-183. issn: 1042-9832. https://doi.org/10.1002/rsa.10013.abs.

[23] Parnas, Michal; Ron, Dana, y Rubinfeld, Ronitt. «Tolerant property testing and distance approximation». En: *Journal of Computer and System Sciences* 72.6 (2006), págs. 1012-1042. issn: 0022-0000. https://doi.org/10.1016/j.jcss.2006.03.002.

[24] Ron, Dana. «Algorithmic and analysis techniques in property testing». En: *Foundations and Trends® in Theoretical Computer Science* 5.2 (2009), front matter, 73-205. issn: 1551-305X. https://doi.org/10.1561/0400000029.

[25] Rubinfeld, Ronitt y Sudan, Madhu. «Robust characterizations of polynomials with applications to program testing». En: *SIAM Journal on Computing* 25.2 (1996), págs. 252-271. issn: 0097-5397. https://doi.org/10.1137/S0097539793255151.

[26] Szemerédi, Endre. «Regular partitions of graphs». En: *Problèmes combinatoires et théorie des graphes (Colloq. Internat. CNRS, Univ. Orsay, Orsay, 1976)*. Vol. 260. Colloq. Internat. CNRS. CNRS, Paris, 1978, págs. 399-401.