# Reporte 4:
# Diagramas de Voronoi

Jorge Torres

6 de marzo de 2022

## 1. Objetivo

En esta práctica se genera un conjunto de celdas de Voronoi, también llamadas mosaicos de Dirichlet, con el método computacional descrito por P. J. Green y R. Sibson [1]. Se determina el efecto que tiene el variar la densidad de semillas en cinco niveles ($k = 25, 50, 75, 100, 125$), en la probabilidad de que una segunda grieta llegue a tocar una primera, es decir, fracturando la pieza dos veces con posiciones iniciales generadas independientemente al azar, sobre varias réplicas. Se analizan los resultados y se visualizan en la sección 3.

## 2. Desarrollo

El código que se describe en esta práctica está basado en el desarrollado por E. Schaeffer [2], y se puede encontrar en su totalidad en el repositorio de J. Torres en GitHub.

El primer paso consiste en definir los parámetros iniciales de operación, a partir de los cuales se generan las celdas de Voronoi. Estos consisten en un tamaño de matriz de 100, cinco niveles para la densidad inicial de semillas y 200 repeticiones del experimento para cada una de ellas (ver código 1).

Código 1: Parámetros

```
n = 100
seed = [25, 50, 75, 100, 125]
runs = 200
```

En el código 2 se define la función `creacion()`, en donde colocan las semillas en el plano de manera aleatoria, mientras que con la función `celda()` se determinan las distancias euclideanas más cercanas a cada semilla, definiendo así la celda de Voronoi para dicha semilla.

Código 2: Ubicación de semillas y celdas

```
def creacion():
    semillas = []
    for s in range(k):
        while True:
            x, y = randint(0, n - 1), randint(0, n - 1)
            if (x, y) not in semillas:
                semillas.append((x, y))
                break
    return semillas

def celda(pos):
    if pos in semillas:
        return semillas.index(pos)
    x, y = pos % n, pos // n
    cercano = None
    menor = n * sqrt(2)
    for i in range(k):
        (xs, ys) = semillas[i]
        dx, dy = x - xs, y - ys
        dist = sqrt(dx**2 + dy**2)
        if dist < menor:
            cercano, menor = i, dist
    return cercano
```

Al tener las celdas definidas, se puede proseguir a determinar, de manera aleatoria, la posición inicial de la grieta con la función `inicio()` descrita en el código 3.

Código 3: Ubicación inicial de la grieta

```
def inicio():
    direccion = randint(0, 3)
    if direccion == 0:
        return (0, randint(0, n - 1))
    elif direccion == 1:
        return (randint(0, n - 1), 0)
    elif direccion == 2:
        return (randint(0, n - 1), n - 1)
    else:
        return (n - 1, randint(0, n - 1))
```

Para propagar la grieta se tiene en cuenta que es más probable que se propague a lo largo de una frontera de celda, mientras que en el interior la probabilidad va disminuyendo gradualmente. En el código 4 se propaga una grieta en color negro hasta que toca un borde o hasta que le es demasiado difícil seguir propagándose al interior de una celda.

Código 4: Propagación de la primera grieta

```
def propaga_n():
    prob, dificil = 0.9, 0.8
    grieta_n = voronoi.copy()
    g = grieta_n.load()
    (xn, yn) = inicio()
    negro = (0, 0, 0)
    while True:
        g[xn, yn] = negro
        frontera, interior = [], []
        for v in vecinos:
            (dx, dy) = v
            vx, vy = xn + dx, yn + dy
            if vx >= 0 and vx < n and vy >= 0 and vy < n:
               if g[vx, vy] != negro:
                    if vor[vx, vy] == vor[xn, yn]:
                        interior.append(v)
                    else:
                        frontera.append(v)
        elegido = None
        if len(frontera) > 0:
            elegido = choice(frontera)
            prob = 1
        elif len(interior) > 0:
            elegido = choice(interior)
            prob *= dificil
        if elegido is not None:
            (dx, dy) = elegido
            xn, yn = xn + dx, yn + dy
        else:
            break
    return grieta_n
```

Se puede generar una segunda grieta (en color blanco) con la función `inicio()` del código 3. Ésta se propagaría de la misma manera que la primera, con la excepción de que se detiene por completo al hacer contacto con la primera grieta. Este comportamiento se puede observar en el código 5.

Código 5: Propagación de la segunda grieta

```
def propaga_b():
    prob, dificil = 0.9, 0.8
    grieta_b = propaga_n()
    g = grieta_b.load()
    (xb, yb) = inicio()
    blanco = (255, 255, 255)
    negro = (0, 0, 0)
    while True:
        g[xb, yb] = blanco
        frontera, interior = [], []
        for v in vecinos:
            (dx, dy) = v
```

```
            vx, vy = xb + dx, yb + dy
            if vx >= 0 and vx < n and vy >= 0 and vy < n:
               if g[vx, vy] != blanco:
                   if vor[vx, vy] == vor[xb, yb]:
                       interior.append(v)
                   else:
                       frontera.append(v)
        elegido = None
        if len(frontera) > 0:
            elegido = choice(frontera)
            prob = 1
        elif len(interior) > 0:
            elegido = choice(interior)
            prob *= dificil
        if elegido is not None:
            (dx, dy) = elegido
            xb, yb = xb + dx, yb + dy
            if g[xb, yb] == negro:
                tocaron.append(1)
                break
        else:
            break
    return grieta_b
```

En el código 6 se inicializan las funciones descritas anteriormente. El experimento se repite 200 veces para cada una de las densidades iniciales de semillas. Se lleva un conteo de las veces que se tocaron y se calcula la probabilidad de que se toquen basada en esos resultados utilizando la ecuación 1,

$$P = \frac{N_T}{R} \times 100 \tag{1}$$

donde $N_T$ es la cantidad de veces que se tocaron y $R$ es la cantidad de iteraciones.

Código 6: Iteraciones del experimento

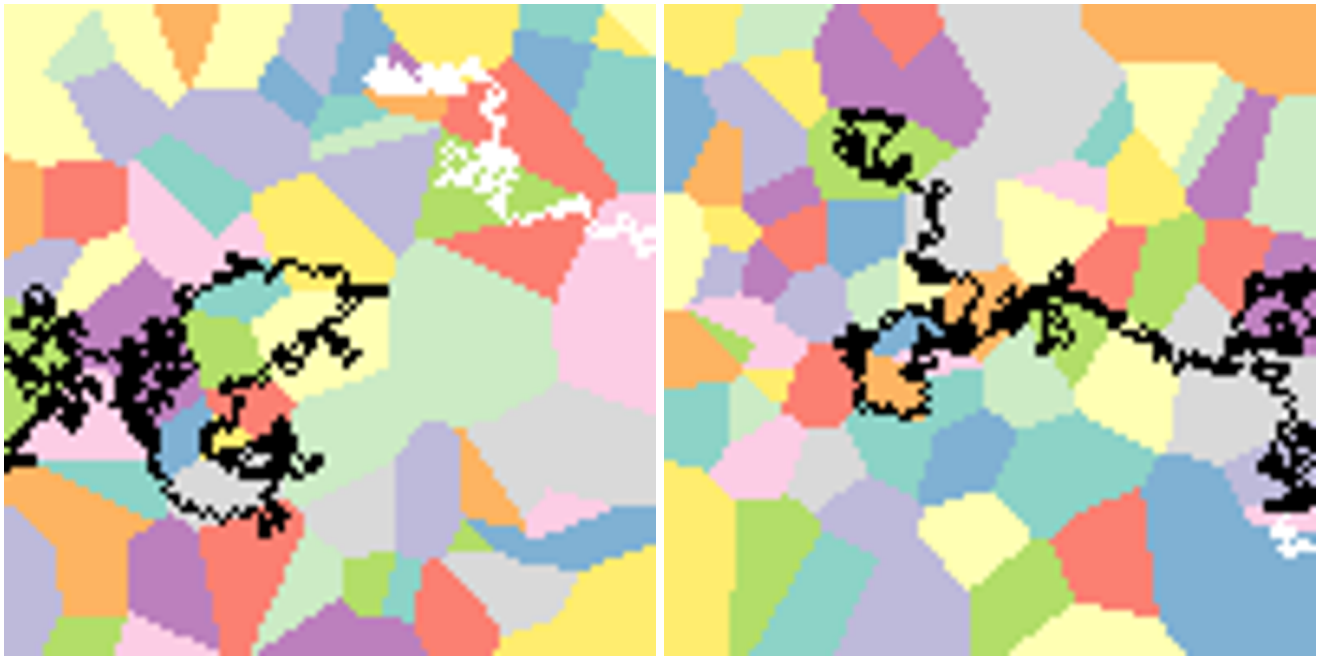
```
for k in seed:
    tocaron = []
    for r in range(runs):
        semillas = creacion()
        celdas = [celda(i) for i in range(n * n)]
        voronoi = Image.new('RGB', (n, n))
        vor = voronoi.load()
        c = sns.color_palette("Set3", k).as_hex()
        for i in range(n * n):
            vor[i % n, i // n] = ImageColor.getrgb(c[celdas.pop(0)])
        limite, vecinos = n, []
        for dx in range(-1, 2):
            for dy in range(-1, 2):
                if dx != 0 or dy != 0:
                    vecinos.append((dx, dy))
        propaga_b()
    pr = (len(tocaron)/runs)*100
    resultado.append(pr)
```

## 3. Resultados

Para una mejor visualización de la generación de las celdas y las grietas, en la figura 1 se exponen un par de ejemplos de éstas. Uno donde las grietas no se tocan (figura 1a), y otro donde las grietas se tocan (figura 1b). Por otro lado, en la figura 2 se muestran las probabilidades calculadas de que las grietas se toquen para los cinco distintos niveles de densidad de semillas.

## 4. Conclusiones

Observando la gráfica de la figura 2, podría aparentar que la probabilidad de que las grietas se toquen aumenta conforme se aumenta también la densidad de semillas. Debido a la naturaleza aleatoria del sistema, sería difícil decir con certeza el por qué esto es así, sin embargo podría deberse a que la cantidad de fronteras aumenta con la cantidad de semillas, además de que el tamaño promedio de celda disminuye, lo cual sería propicio para que las



(a) Las grietas no se tocan.

(b) Las grietas se tocan.

Figura 1: Ejemplos de grietas.

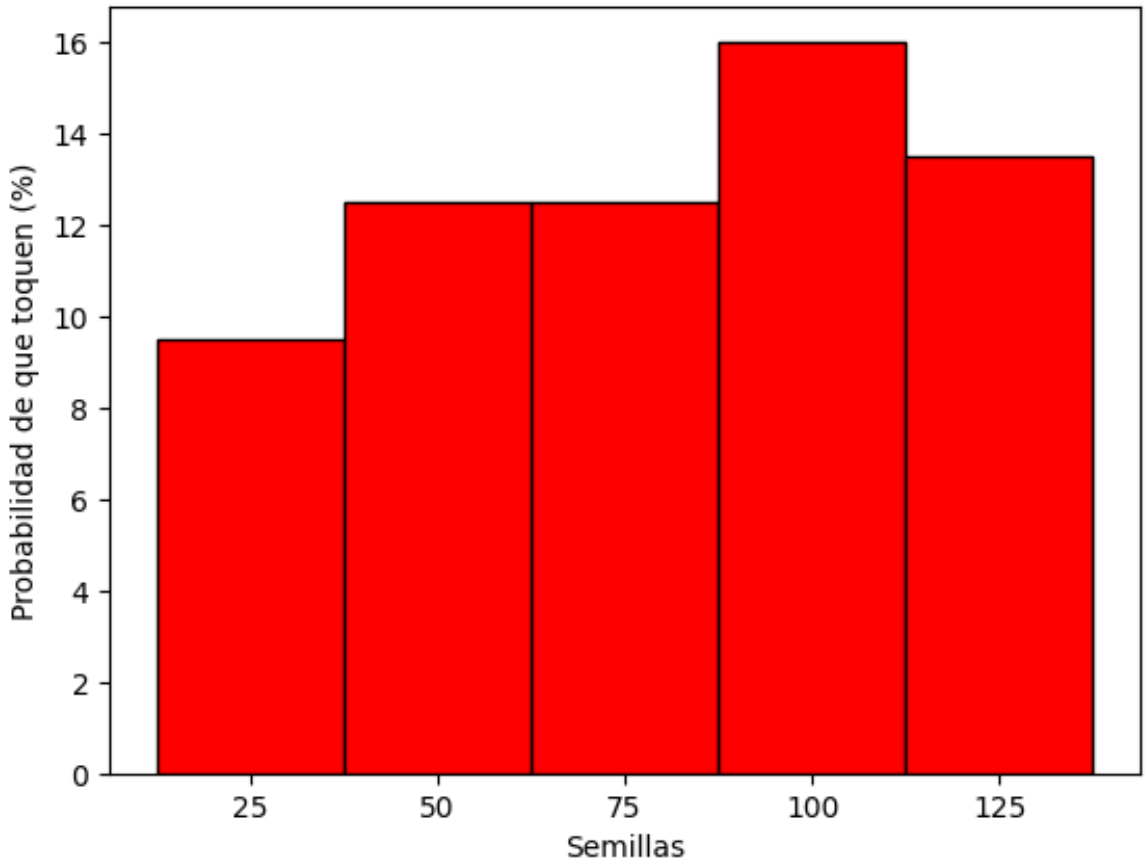



Figura 2: Probabilidades de que las grietas se toquen para los cinco niveles de densidad de semillas.

grietas coincidan en un lugar. Sin embargo, la probabilidad disminuye ligeramente en el último nivel, lo cual hace pensar que existe un límite para la propagación de las grietas. La determinación de dicho límite se encuentra fuera del rango de estudio de ésta práctica.

# Referencias


[1] P. J. Green and R. Sibson. Computing dirichlet tessellations in the plane. *The Computer Journal*, 21(2):168–173, 1978. doi: https://doi.org/10.1093/comjnl/21.2.168.

[2] E. Schaeffer. Voronoidiagrams. *Repositorio, GitHub*, 2019. URL https://github.com/satuelisa/Simulation/tree/master/VoronoiDiagrams.